La criptografia que et cal saber

... perquè serà l'última eina de protecció de l'individu.

Cristina Pérez Solà
Jordi Herrera Joancomartí

Aquesta obra és una obra derivada dels materials didàctics escrits pels mateixos autors per a la Universitat Oberta de Catalunya, també publicats amb una llicència CC BY-SA v.3.0.

Segona edició, Abril 2023 (amb modificacions de Febrer de 2026)

Versió	**Data**	**Canvis**
v0.1	23/04/2022	Versió inicial
v0.2	23/04/2023	Primera versió publicada a Internet
v0.2.1	04/02/2026	Correcció d’errades tipogràfiques

Taula 1: Històric de versions

Índex

I Conceptes bàsics

I

Conceptes bàsics

1. Introducció a la criptografia

En aquest capítol es presenten, d'una banda, els fonaments de la criptografia i, d'altra banda, es realitza un repàs històric de la criptografia premoderna.

Pel que fa als fonaments de la criptografia, descriurem els conceptes clau d'aquesta ciència, que farem servir al llarg del llibre per anar presentant les diferents tècniques que es fan servir en criptografia.

En relació amb el repàs històric, veurem com va sorgir la criptografia i quines tècniques es feien servir des dels seus orígens fins a l'inici de la criptografia moderna. La resta del llibre se centrarà precisament en descriure diversos aspectes de la criptografia moderna que, com veurem, ha evolucionat molt des de les seves arrels.

1.1 Conceptes bàsics

> La **criptografia** és la ciència que estudia l'escriptura de secrets, amb l'objectiu d'ocultar el missatge que s'escriu.

Etimològicament, la paraula prové del grec i sorgeix de la unió de dos conceptes: *kryptós*, que vol dir secret i *graphein*, que vol dir escriptura. Els orígens de l'escriptura secreta es remunten a fa més de 4000 anys, però en aquells moments la criptografia es trobava lluny de considerar-se una ciència. A mig camí entre art i joc d'enigmes, civilitzacions com l'antic Egipte van desenvolupar els primers escrits on es transformava el missatge original. Es considera però que la criptografia com a ciència no va començar a desenvolupar-se fins a mitjans del segle XX, amb les contribucions realitzades per Claude E. Shannon.

La **criptoànalisi** és la ciència que se centra en trencar les tècniques que desenvolupa la criptografia, ja sigui per a descobrir el text amagat darrere un text xifrat o bé per a demostrar les febleses d'un determinat esquema criptogràfic.

Així doncs, la criptoanàlisi és indispensable per a l'avenç de la criptografia, ja que s'encarrega d'avaluar la seguretat dels criptosistemes que aquesta desenvolupa. Tot i que el mot criptoanàlisi és bastant recent, tenim constància d'una criptoanàlisis realitzada al segle IX per un matemàtic àrab, Al-Kindi.

El terme general **criptologia** es fa servir per englobar tant criptografia com cripto-anàlisi.

En aquest llibre, ens centrarem en descriure les tècniques i algorismes que es fan servir per ocultar informació, és a dir, en la criptografia. Tot i així, en aquest capítol farem una petita introducció a la criptoanàlisi, per tal d'oferir unes nocions bàsiques dels models amb els quals s'avalua habitualment la seguretat dels esquemes criptogràfics.

Tradicionalment, la criptografia es basava únicament en protegir la **confidencialitat** dels missatges.

La **confidencialitat** és una propietat que garanteix que la informació no es fa pública a persones no autoritzades.

Els sistemes criptogràfics han evolucionat molt des dels seus orígens, i actualment poden oferir altres garanties, més enllà de la confidencialitat. Sovint, l'ús de la criptografia ens permet també garantir la integritat dels missatges o fins i tot el no-repudi.

La **integritat** és la propietat que garanteix que la informació no ha estat modificada.

Els sistemes que ofereixen integritat permeten detectar si hi ha hagut una modificació de la informació.

El **no-repudi** és la propietat que garanteix que l'autor d'una determinada acció no pugui negar haver-la realitzat.

Per tal de simplificar les explicacions, en criptografia es fan servir uns personatges ficticis, que acostumen a interpretar sempre els mateixos papers. Aquests personatges van ser creats per Ron

Rivest, Adi Shamir i Leonard Adleman, i el seu seu ús es troba molt extés.[1] L'Alice (A) i en Bob (B) són els dos personatges més populars i acostumen a ser dos usuaris que volen intercanviar algun missatge. L'Eve (E) és un atacant passiu, que pot escoltar les comunicacions entre l'Alice i en Bob, però no modificar-les. Mallory (M) és un atacant actiu, que pot escoltar les comunicacions entre l'Alice i en Bob, i també modificar el contingut de la transmissió.

Anem doncs a descriure l'escenari tradicional en què s'aplica la criptografia fent servir els personatges que acabem de presentar. En l'escenari bàsic, l'Alice vol enviar un missatge a en Bob a través d'un canal insegur. Com que el canal és insegur, l'Eve pot escoltar la comunicació entre l'Alice i en Bob. Amb aquest plantejament, l'Alice desitja enviar un missatge, m, a en Bob garantint-ne la confidencialitat. Per fer-ho l'Alice aplica un *algorisme de xifrat*, E, al text que vol enviar (anomenat *text en clar*) fent servir una determinada *clau*, k. El resultat d'aplicar l'algorisme de xifrat sobre el text en clar és el *text xifrat*, c, que és el que s'enviarà a través del canal insegur. En Bob, quan rebi el missatge xifrat, c, procedirà a aplicar un *algorisme de desxifrat*, D, al text xifrat fent servir la mateixa clau, k, obtenint el text en clar original, m. Per tal que l'esquema pugui aplicar-se, serà necessari doncs que l'Alice i en Bob disposin d'una clau compartida, k, que hauran hagut de comunicar-se anteriorment a través d'algun canal segur (potser fins i tot trobant-se físicament). L'Eve podrà recuperar el text xifrat de la comunicació c, però al no conéixer el valor de la clau, no serà capaç de recuperar-ne el text en clar corresponent.

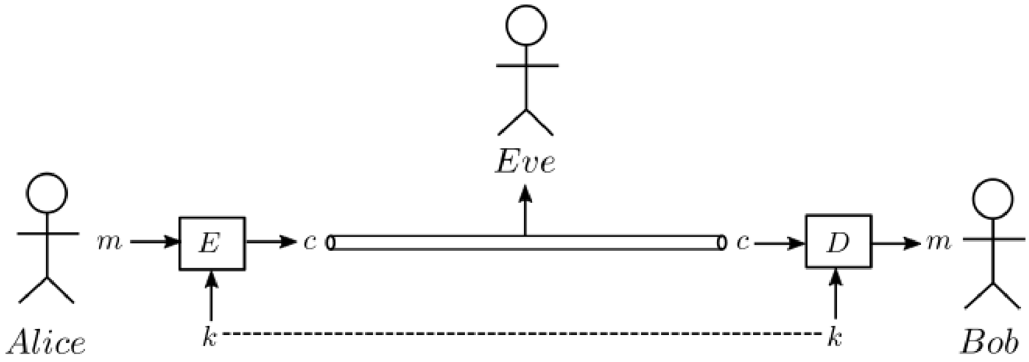


Figura 1.1: Escenari bàsic d'aplicació de la criptografia en les comunicacions entre dos usuaris.

Més formalment, direm que un criposistema queda definit per cinc paràmetres:

- El conjunt de possibles *textos en clar*, $\mathfrak{M}$
- El conjunt de possibles *textos xifrats*, $\mathfrak{C}$
- El conjunt de possibles *claus*, $\mathfrak{K}$
- E, una *funció de xifrat*, que detalla per a cada possible clau $k \in \mathfrak{K}$ i missatge $m \in \mathfrak{M}$, quin és el corresponent text xifrat $c \in \mathfrak{C}$.
- D, una *funció de desxifrat*, que realitza el procés invers de la funció de xifrat, és a dir, una funció tal que $D_k(E_k(m)) = m$, per a tot $m \in \mathfrak{M}$ i $k \in \mathfrak{K}$.

A partir d'aquest escenari bàsic, els escenaris en els quals s'aplica la criptografia avui en dia són molt diversos i variats, alguns dels quals no s'assemblen gens a l'escenari tradicional. Així, per exemple, la criptografia ens permet crear sistemes de credencials anònimes, que serviran per autenticar-se de manera anònima; sistemes de compartició de secrets, on caldrà la col·laboració d'n parts d'un conjunt d'm per recuperar el secret; criptomonedes, que oferiran mètodes de pagament

[1] Rivest, Shamir i Adleman van crear els personatges de l'Alice i en Bob a l'article "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*", publicat l'any 1978.

totalment descentralitzats i segurs; i protocols de computació multipart, on diverses entitats podran col·laborar per calcular funcions sobre dades confidencials.

1.1.1 Introducció a la criptoanàlisi

La criptoanàlisi se centra en analitzar els criptosistemes, amb l'objectiu d'avaluar-ne la seva seguretat. Depenent de si l'anàlisi es focalitza en l'algorisme, la implementació o el sistema complet que l'integra, distingim diferents atacs que el criptoanalista pot intentar realitzar contra un esquema criptogràfic.

> Els **atacs clàssics** intenten recuperar un text en clar a partir d'un text xifrat o bé recuperar una clau.

Existeixen diferents escenaris o models en els quals avaluar els criptosistemes, en funció de la informació de la qual disposa el criptoanalista per trencar els esquemes:

- En el model de **només text xifrat** (o COA, de l'anglès, *ciphertext-only attack*) l'atacant només disposa d'un conjunt de textos xifrats.
- En el model de **text en clar conegut** (o KPA, de l'anglès, *known-plaintext attack*), l'atacant disposa d'un conjunt de textos en clar i els seus corresponents textos xifrats.
- En el model de **text en clar escollit** (o CPA, de l'anglès, *chosen-plaintext attack*), el criptoanalista pot obtenir els textos xifrats corresponents a un conjunt de textos en clar seleccionats per ell mateix.
- En el model de **text xifrat escollit** (o CCA, de l'anglès, *chosen-ciphertext attack*), el criptoanalista pot obtenir els textos en clar corresponents a un conjunt de textos xifrats seleccionats per ell mateix.

Els models de text en clar i text xifrat escollit assumeixen normalment que el criptoanalista tria una única vegada el conjunt de textos en clar (respectivament, textos xifrats) i pot demanar-ne els corresponents textos xifrats (respectivament, en clar). Una variant d'aquests models, coneguda com a model **adaptatiu** de text en clar/xifrat escollit (respectivament, CPA2 i CCA2), permet al criptoanalista anar demanant els corresponents textos xifrats/en clar successivament, modificant els textos que demana en funció de les respostes que ha rebut fins al moment.

Avui en dia gairebé tots els criptogràfs assumeixen el principi de Kerckhoffs:

> El principi de **Kerckhoffs** afirma que, per a què un criptosistema pugui considerar-se segur, aquest ho ha de ser encara que l'atacant conegui tots els detalls del criptosistema, exceptuant-ne la clau.

És a dir, s'assumeix que l'atacant o el criptoanalista disposa de l'especificació completa de l'algorisme a trencar. Auguste Kerchoffs va formular aquest principi al segle XIX, i actualment, la versió més extesa del seu principi afirma que la seguretat d'un criptosistema ha de dependre únicament de la clau.

Tot i això, en productes criptogràfics comercials sovint es fa cas omís d'aquest principi i s'opta per l'alternativa, la seguretat per ofuscació (en anglès, *security through obscurity*). En aquest paradigma, la seguretat dels sistemes es basa en amagar els detalls sobre l'algorisme de xifrat, amb l'objectiu de dificultar-ne, suposadament, la criptoanàlisi. A la pràctica, però, normalment aquests detalls s'acaben fent públics igualment, de manera que amagar l'algorisme és contraproduent ja que únicament en dificulta l'avaluació de la seva seguretat. Alguns exemples de l'adopció d'aquest paradigma són en els algorismes xifrat de telefonia mòbil GSM, que es van intentar mantenir ocults sense èxit, o en el sistema de DRM dels DVDs, on calia pagar una llicència i signar un acord de no revel·lació per tal de tenir accés als detalls de l'algorisme.

Més enllà dels atacs clàssics, que consideren únicament l'algorisme utilitzat, existeixen també atacs de canal lateral i atacs d'enginyeria social.

Els **atacs de canal lateral** (en anglès, *side-channel attacks*) es basen en atacar un criptosistema a través d'informació extreta d'una implementació física.

Hi ha diferents classes d'atacs de canal lateral, depenent de la informació que s'extreu de la implementació per a realitzar l'atac. Així, els atacs de sincronització (en anglès, *timing attacks*) analitzen el temps que es tarda en realitzar diferents càlculs; els atacs de monitoreig d'energia estudien el consum energètic que té el dispositiu durant l'operació; el atacs electromagnètics mesuren les fugues de radiació electromagnètica; els atacs acústics tenen en compte el so que es produeix al realitzar els càlculs, etc.[2]

Més enllà dels atacs als algorismes i a les implementacions dels criptosistemes, els sistemes d'informació en general són susceptibles també de patir atacs d'enginyeria social.

Els **atacs d'enginyeria social** es basen en manipular als usuaris d'un sistema per tal d'obtenir informació que ens permeti trencar-ne la seguretat.

Així, els atacs d'enginyeria social es realitzen interactuant amb els usuaris, i sovint inclouen l'engany d'aquests per tal d'obtenir dades confidencials. Per exemple, un atacant pot intentar trucar a un usuari, fent-se passar per un tècnic informàtic i sol·licitant la clau de xifratge per tal de realitzar, suposadament, alguna comprovació. Evidentment, la criptografia poc té a fer amb aquests tipus d'atacs i, per aquest motiu, són dels més estesos i dels més perillosos.

1.2 Una mica d'història

Es diu que la història de la criptologia[3] començà l'any 1900 abans de Crist, amb uns escrits realitzats a la tomba de Khnumhotep II, un monarca de l'Alt Egipte. Als escrits trobats a la tomba s'hi troben alguns jeroglífics inusuals, que l'escribà va escriure enlloc d'altres més comuns, suposadament amb l'objectiu de dignificar el text. Tot i que en aquest cas no hi havia intenció d'ocultar el missatge, els escrits suposen el primer cas en la història on hi havia una tranformació deliverada del text que

[2] Per a un exemple concret d'atac de monitoreig d'energia al criptosistema RSA podeu consultar el Capítol 7 del llibre *Understanding cryptography*, de C. Paar i J. Pelzl.

[3] Una lectura recomanada per aprofundir en la història de la criptologia és el llibre *The codebreakers*, de David Khan.

s'escrivia.

També a l'Antic Egipte apareixen els primers escrits amb la intenció, ara sí, d'ocultar el missatge escrit. Es creu que l'objectiu era dotar el textos de cert aire de misteri i màgia, de manera que cridessin l'atenció del lector i que aquest s'entretingués desxifrant-los, com si fos un joc o un puzzle.

Uns quants segles després, l'ús de la criptografia va prendre un altre rumb i va començar-se a fer servir per ocultar missages amb contingut crític en temps de guerra. Els espartans, potència militar de l'antiga Grècia, van començar a fer servir, d'una banda, sistemes esteganogràfics i, d'altra banda, van inventar la primera xifra de transposició coneguda, l'escítala.

Pel que fa a l'esteganografia,[4] els primers usos que se'n coneixen daten de l'any 440 a.C.: Histiaeus va rapar el cap d'un dels seus servents per tatuar-hi un missatge, deixant que el cabell del servent tornés a créixer abans d'enviar-lo a Aristagoras, el receptor del misatge. Així, si l'esclau era capturat per l'enemic durant el viatge, el fet que l'esclau transportava un missatge romandria ocult. També en aquella època, Demaratus va enviar un missatge escrit en un parell de tabletes de cera, marcant el missatge a la fusta que quedava sota la cera i cobrint les tauletes de nou de cera. Així, si les tauletes eren interceptades, una revisió superficial de les mateixes no revel·laria que incorporaven un missatge ocult.

Pel que fa a la criptografia, els espartans són coneguts també per la utilització del primer sistema de criptografia militar, l'escítala, que descriurem posteriorment en l'apartat de xifres de transposició. Es creu que l'escítala va ser el primer aparell utilitzat per la criptografia. Thucydides, un historiador grec, recull l'ús d'aquest aparell per a xifrar un missatge dels èfors (uns magistrats de l'antiga Grècia) al general espartà Pausanius.

El primer ús conegut d'un criptosistema de substitució és atribuït als romans i, en concret, a Juli Cèsar, que el feia servir per escriure a Ciceró i d'altres amics. En els següents apartats descriurem també en detall aquesta xifra, així com les seves febleses.

Els primers textos on es parla de criptoanàlisi són atribuïts als àrabs. Al-Kindi, filòsof i matemàtic àrab del segle IX d.C., va descriure com utilitzar el fet que la freqüència d'aparició de les lletres de l'alfabet en un idioma determinat no és uniforme per trencar criptosistemes.

Ja al segle XIV, l'italià Leon Battista Alberti, va ser el primer occidental en documentar tècniques de criptoanàlisi i va crear el primer xifrat de substitució polialfabètic, la xifra d'Alberti.

Uns quants segles després, al 1883, Auguste Kerckhoffs, criptògraf d'origen holandés, va publicar un llibre sobre criptografia militar, on donava consells pràctics per al disseny de criptosistemes. Un d'aquests consells afirmava que un criptosistema havia de ser segur encara que l'atacant en conegués tots els detalls, a excepció de la clau feta servir per a xifrar. Aquest consell va rebre una àmplia acceptació i va acabant-se convertint en el principi Kerckhoffs, principi el qual la gran majoria de criptògrafs actuals respecten i segueixen.

L'any 1948 el matemàtic nordamericà Claude Elwood Shannon va crear els fonaments de la teoria de la informació. L'any següent, al 1949, ell mateix va publicar l'article *Communication Theory of Secrecy Systems*, que assentava les bases de la criptografia com a ciència i inaugurava la criptografia moderna. Entre moltes altres contribucions, Shannon va definir els conceptes de secret perfecte, va demostrar que la xifra de Vernam podia oferir aquest tipus de secret i va introduir el concepte de

[4] L'**esteganografia** és la pràctica que amaga un missatge dins d'un altre missatge, amb la intenció d'ocultar el primer. Així, per exemple, hom pot intentar amagar un missatge de text en una imatge, fent servir els bits menys significatius de cada píxel per tal de modificar al mínim la visualització de la imatge.

redundància.

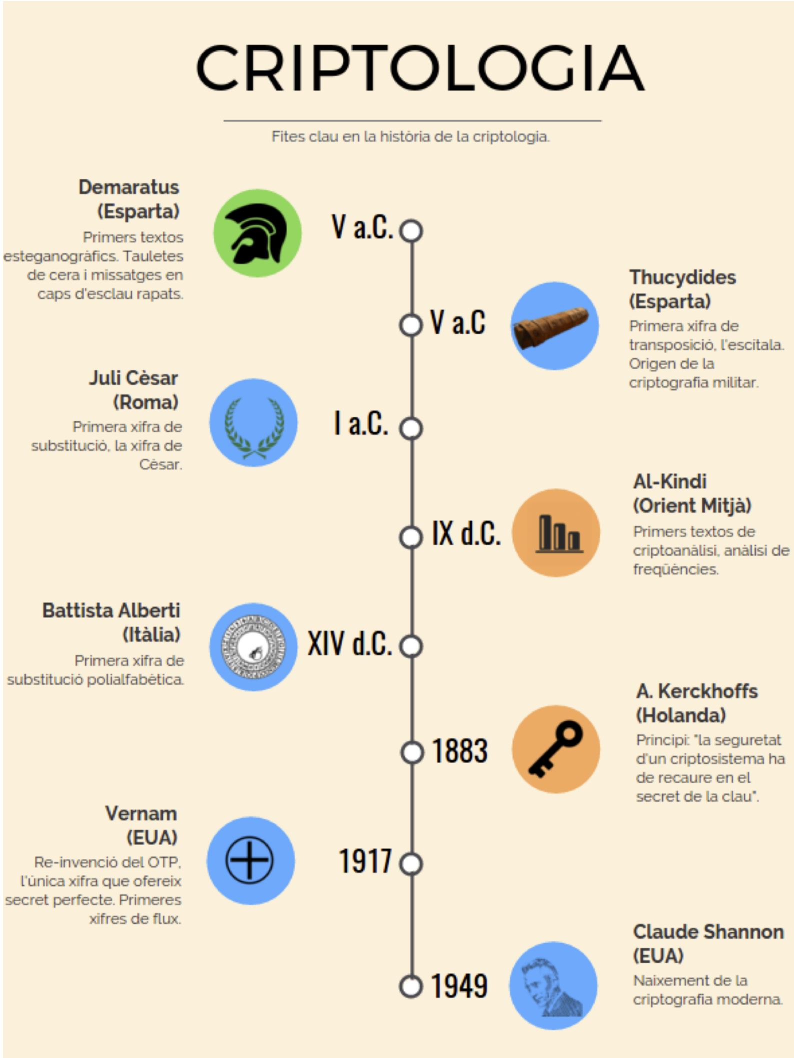


Figura 1.2: Línia de temps amb le fites clau de la criptologia pre-moderna.

A continuació descriurem els dos tipus de criptosistemes utilitzats en la criptografia història, les xifres de transposició i les xifres de substitució, i en presentarem alguns exemples concrets.

1.2.1 Xifres de transposició

> Les xifres de **transposició** es basen en canviar l'ordre dels caràcters del text en clar d'entrada per tal de generar el text xifrat.

És a dir, les xifres de transposició reordenen el text d'entrada, de manera que el text en clar és una permutació dels caràcters del text xifrat.

Escítala

Els espartans (al segle V a.C.) feien servir un criptosistema de transposició conegut pel nom d'escítala. La clau de xifrat era un pal o bastó d'un determinat gruix.

Per a xifrar, s'enrotllava una tira de papir al voltant del bastó i s'escrivia el missatge en sentit longitudinal, és a dir, seguint la direcció del propi bastó. Després, es desenrotllava la tira de papir, obtenint el missatge xifrat que podia ser enviat al receptor. Per tant, el gruix del bastó representava la clau compartida.

Al rebre la tira de papir, el receptor, que també disposava d'un bastó del mateix gruix que el de l'emissor, procedia a enrotllar la tira al voltant del bastó i podia així llegir el missatge original enviat.

La tira de papir, per si sola, era difícil de llegir, ja que contenia les mateixes lletres que el missatge en clar però desordenades per l'efecte de desenrotllar el papir. A més, si no es disposava d'un bastó del gruix adequat, el resultat d'enrotllar el papir al bastó no revel·lava el missatge original.

Exemple 1.1 Exemple de xifra amb escítala

Xifrem el missatge `THESEARESPARTASWALLS` fent servir una escítala. Suposem que el gruix del bastó utilitzat com a clau permet escriure quatre línies de text i que la longitud del bastó limita cada línia a cinc caràcters. Aleshores, el missatge quedaria escrit en quatre línies que serien:

```
THESE
ARESP
ARTAS
WALLS
```

Al desenrotllar el papir del bastó, el missatge que quedaria escrit en la tira de papir (i que correspondria al missatge xifrat) seria: `TAAWHRRAEETLSSALEPSS`.

Noteu com, efectivament, les lletres del missatge en clar han quedat desordenades, ocultant així el missatge original.

Exercici 1.1 Xifreu el missatge `THESEARESPARTASWALLS` fent servir una escítala amb un gruix de bastó que permeti escriure cinc línies de text i una longitud que permeti escriure quatre caràcters per línia.

1.2.2 Xifres de substitució

En contraposició a les xifres de transposició, les xifres de substitució no desordenen el text en clar per tal de xifrar, sinó que substitueixen les lletres del text en clar per altres símbols. Depenent de la tècnica utilitzada per realitzar les substitucions, distingirem entre xifres de substitució simple, polialfabètica i homofònica.

Substitució simple

La xifra de substitució simple és un dels mètodes més senzills per a xifrar text.

La xifra de **substitució simple** consisteix a substituir cada lletra individual del missatge en clar per una altra lletra.

La clau feta servir per xifrar és, aleshores, una taula que indica per cada lletra de l'alfabet d'entrada, quina és la seva corresponent lletra de l'alfabet xifrat.

El procediment a realitzar per xifrar consisteix a buscar cada lletra del text en clar a la taula utilitzada com a clau i substituir-la per la lletra indicada. Per a desxifrar, se segueix el mateix procediment, fent servir ara la taula en sentit invers.

La mida de l'espai de claus (és a dir, el número de possibles taules que podem crear indicant correspondències entre lletres) ve donada per les mides dels alfabets en clar i xifrat. Així, per exemple, si fem servir un alfabet de 26 caràcters tant per al text en clar com per al text xifrat, l'espai de claus té una mida de:

$$|\mathscr{K}| = 26 \cdot 25 \cdot 24 \cdot \ldots \cdot 1 = 26!$$

ja que, per al primer caràcter de l'alfabet en clar, podem triar 26 possibles lletres xifrades; per al segon caràcter, en podrem triar 25 (les 26 disponibles excepte la que ja hem triat per al primer caràcter); etc.

L'espai de claus de les xifres de substitució simple pot semblar prou gran per oferir un nivell de seguretat adequat. Tot i així, aquestes xifres són en realitat molt fàcils de trencar, en part perquè preserven la freqüència d'aparició de les lletres. En efecte, si una determinada lletra del text en clar x queda xifrada sempre per una lletra de l'alfabet xifrat y, la freqüència d'aparició de la lletra y en el text xifrat serà exactament la mateixa que la freqüència d'aparició d'x en el text en clar. Atès que les freqüències d'aparició de les lletres en els textos escrits presenten marcades diferències, quan els textos tenen certa longitud és fàcil identificar algunes lletres del text xifrat i acabar desxifrant el missatge sense conéixer la clau feta servir per xifrar.

La Figura 1.3 mostra les freqüències d'aparició mitjanes de les lletres de l'alfabet en textos escrits en català:

Es diu que Juli Cèsar va fer servir una variant de la xifra de substitució simple per escriure a Ciceró i d'altres amics. La variant que feia servir Cèsar xifrava cada lletra de l'alfabet en clar per la lletra que es troba tres posicions després en l'alfabet. Així, Cèsar feia servir les següents correspondències:

$$\begin{array}{c} \mathrm{A} \rightarrow \mathrm{D} \\ \mathrm{B} \rightarrow \mathrm{E} \\ \mathrm{C} \rightarrow \mathrm{F} \\ \mathrm{D} \rightarrow \mathrm{G} \\ \mathrm{E} \rightarrow \mathrm{H} \\ \ldots \\ \mathrm{X} \rightarrow \mathrm{A} \\ \mathrm{Y} \rightarrow \mathrm{B} \\ \mathrm{Z} \rightarrow \mathrm{C} \end{array}$$

Una generalització immediata de l'esquema que feia servir Cèsar resulta de xifrar cada lletra per la que es troba k posicions després en l'alfabet, on k pot ser qualsevol valor en $[0,25]$ (en comptes de fixar $k=3$).[5] Aquesta generalització és el que es coneix habitualment com a **xifra de Cèsar**.

[5] El nebot de Cèsar, Augustus, feia servir una variant de la xifra de Cèsar amb $k=1$.

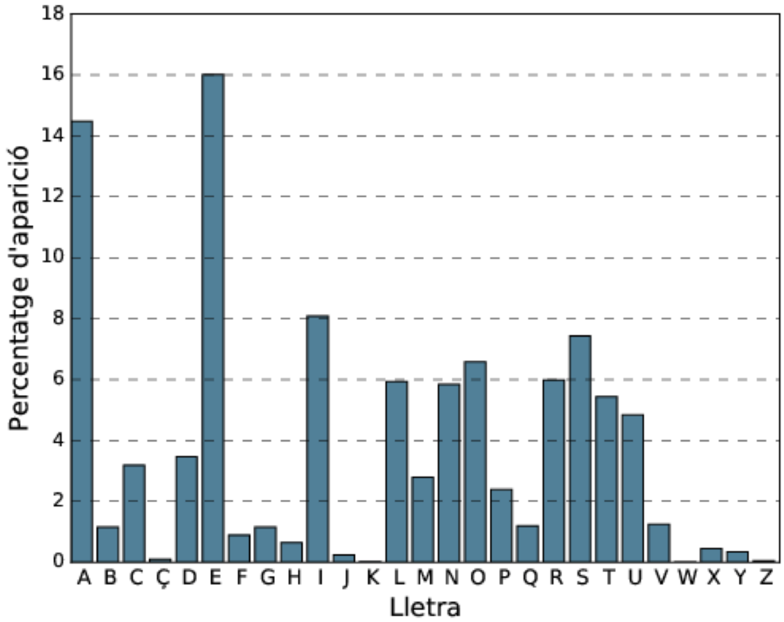


Figura 1.3: Freqüències d'aparició de les lletres en català.

Si assignem a cada lletra de l'alfabet una representació numèrica, on la A és representada pel 0, la B per l'1, etc., aleshores podem definir formalment la funció de xifrat de cada lletra del missatge com a:

$$E(x) = x + k \mod 26$$

on k és la clau secreta que comparteixen l'emissor i el receptor.

Simètricament, la funció de desxifrat és:

$$D(y) = y - k \mod 26$$

Exemple 1.2 Exemple de xifra de Cèsar

Volem xifrar el missatge $m =$ `THEDIEISCAST` fent servir la xifra de Cèsar original, amb $k = 3$. Procedim doncs a substituir cada lletra del missatge en clar per la lletra que es troba tres posicions després a l'alfabet, obtenint el missatge xifrat:

$$c = \texttt{WKHGLHLVFDVW}$$

Si volem fer servir la formulació matemàtica, convertirem primer el missatge m en una seqüència d'enters:

$$m' = 19\ \ 7\ \ 4\ \ 3\ \ 8\ \ 4\ \ 8\ \ 18\ \ 2\ \ 0\ \ 18\ \ 19$$

Sumarem $k = 3$ a cada valor, reduint el resultat mòdul 26 (noteu que en aquest cas concret, no cal reduir cap valor ja que tots són inferiors a 26):

$$c' = 22\ \ 10\ \ 7\ \ 6\ \ 11\ \ 7\ \ 11\ \ 21\ \ 5\ \ 3\ \ 21\ \ 22$$

i finalment convertirem la seqüència xifrada a cadena de caràcters, obtenint el text xifrat c:

$$c = \texttt{WKHGLHLVFDVW}$$

Exercici 1.2 Desxifreu el missatge `XADKTIWTCPBTDUWDCDGBDGTIWPCXUTPGSTPIW` sabent que ha estat xifrat amb una xifra de Cèsar amb $k = 15$.

Tant la xifra de substitució simple com la xifra de Cèsar són xifres de substitució monoalfabètiques:

> Les xifres de **substitució monoalfabètiques** es caracteritzen per fer servir una substitució de caràcters fixa, on una mateixa lletra del text en clar sempre correspondrà a la mateixa lletra del text xifrat, independentment de la posició que ocupi la lletra en el text en clar.

Substitució polialfabètica

Les xifres de substitució polialfabètiques van aparéixer bastants anys després que les xifres monoalfabètiques. Es creu que la primera xifra polialfabètica va ser creada per Leon Battista Alberti, sobre l'any 1467. De totes maneres, alguns historiadors argüeixen que les xifres polialfabètiques van ser ideades per Al Kindi molt abans (sobre l'any 800). La variant més popular de la xifra polialfabètica és atrïbuida a Blaise de Vigenère (tot i que ell no en va ser l'inventor) i es coneguda com a xifra de Vigenère.

> Les xifres de **substitució polialfabètiques** es caracteritzen per fer servir múltiples alfabets de substitució, fent que una mateixa lletra del text en clar pugui quedar xifrada amb diferents lletres, depenent de la posició que aquesta ocupi en el text en clar.

La **xifra de Vigenère** és una xifra de substitució polialfabètica periòdica, on es combinen diferents xifres de Cèsar. El període n ve determinat per la mida (en caràcters) de la clau de xifrat de Vigenère, i cada lletra individual de la clau es fa servir com a clau d'una xifra de Cèsar. Així, per a un missatge $m = m_1, m_2, \cdots, m_l$, una clau $k = k_1, k_2, \cdots, k_n$ i un alfabet de 26 caràcters, la funció de xifrat és:

$$E(m_i) = m_i + k_{i \; mod \; n} \quad \mod 26$$

De manera similar, la funció de desxifrat és:

$$D(c_i) = c_i - k_{i \; mod \; n} \quad \mod 26$$

Exemple 1.3 Exemple de xifra de Vigenère

Suposem que volem xifrar el missatge

$$m = \texttt{VIGENERECIPHERWASCREATEDBYGIOVANBATTISTA}$$

amb la clau:

$$k = \texttt{ENEGIV}$$

Procedim a convertir tant el missatge com la clau a la seva representació numèrica, i a calcular la representació numèrica de la lletra xifrada corresponent a cada lletra en clar (sumant els valors mòdul 26). Finalment, convertim la seqüència numèrica a caràcters i obtenim el missatge xifrat:

V	I	G	E	N	E	R	E	C	I	P	H	E	R	W	A	S	C	R	E
21	8	6	4	13	4	17	4	2	8	15	7	4	17	22	0	18	2	17	4
E	N	E	G	I	V	E	N	E	G	I	V	E	N	E	G	I	V	E	N
4	13	4	6	8	21	4	13	4	6	8	21	4	13	4	6	8	21	4	13
25	21	10	10	21	25	21	17	6	14	23	2	8	4	0	6	0	23	21	17
Z	V	K	K	V	Z	V	R	G	O	X	C	I	E	A	G	A	X	V	R

A	T	E	D	B	Y	G	I	O	V	A	N	B	A	T	T	I	S	T	A
0	19	4	3	1	24	6	8	14	21	0	13	1	0	19	19	8	18	19	0
E	G	I	V	E	N	E	G	I	V	E	N	E	G	I	V	E	N	E	G
4	6	8	21	4	13	4	6	8	21	4	13	4	6	8	21	4	13	4	6
4	25	12	24	5	11	10	14	22	16	4	0	5	6	1	14	12	5	23	6
E	Z	M	Y	F	L	K	O	W	Q	E	A	F	G	B	O	M	F	X	G

El missatge xifrat resultant és doncs:

$$c = \texttt{ZVKKVZVRGOXCIEAGAXVREZMYFLKOWQEAFGBOMFXG}$$

Exercici 1.3 Xifreu el missatge `USINGASERIESOFINTERWOVENCAESARCIPHERS` amb Vigenere, fent servir com a clau `KASISKI`.

Amb les xifres polialfabètiques s'aconsegueix que una mateixa lletra del text en clar no sempre quedi xifrada per la mateixa lletra, dificultant l'anàlisi de freqüències.

Un cas especialment interessant de xifra polialfabètica és la **xifra de Vernam**.

La **xifra de Vernam** és una xifra polialfabètica on el número d'alfabets que codifica la clau és igual o major al número de caràcters del text en clar a xifrar.

Quan es fa servir adequadament, amb claus aleatòries i d'un sol ús, la xifra de Vernam ofereix secret perfecte. De fet, la xifra de Vernam és l'única xifra coneguda, encara avui, que ofereix aquesta propietat.[6]

La xifra de Vernam és coneix també, en anglès, com a *one-time pad*. El nom prové dels primers usos del xifrat, on les claus es distribuien als espies en llibretes de paper (a vegades de paper altament imflamable), el que permetia fer servir la clau una vegada i destruir després el full de paper que contenia aquella clau.

Substitució homofònica

Una altra alternativa per tal d'evitar revel·lar les freqüències d'aparició de les lletres en el text xifrat és la que presenten les xifres homofòniques.

> La xifra de **substitució homofònica** permet substituir cada lletra del missatge en clar per un conjunt de lletres de l'alfabet xifrat.

Així doncs, a diferència de les xifres de substitució simple, on una lletra de l'alfabet en clar correspon a una única lletra de l'alfabet xifrat, en les xifres homofòniques una lletra del text en clar pot correspondre a vàries lletres de l'alfabet xifrat. Això fa que l'alfabet xifrat hagi de tenir més caràcters que l'alfabet en clar.

Per tal d'aconseguir amagar les freqüències d'aparició de les lletres, el que fan les xifres de substitució homofòniques és assignar més alternatives de xifrat a les lletres de l'alfabet en clar que apareixen més sovint, de manera que les freqüències d'aparició de les lletres en el text xifrat s'assemblin el màxim possible.

Exemple 1.4 Exemple de xifra homofònica

Suposem que volem xifrar el missatge `THEBEALEPAPERS` fent servir subsitució homofònica amb la següent clau:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
j	B	N	P	s	T	i	S	q	l	e	h	D	W	R	f	E	d	w	y	O	M	a	X	t	Z
g			Q	z	u		H	U			p	k	L	m			A	K	x	r		v			
J			o	C			c	V					I	Y			F	b	n						
				G																					

i tenint en compte que si disposem de més d'una alternativa per a xifrar una lletra, seleccionarem aleatòriament la lletra a xifrar d'entre les alternatives.

Noteu que, en aquest cas, l'alfabet del text en clar està format per 26 caràcters (les lletres de la A a la Z en majúscula, sense incloure la Ç), mentre que l'alfabet xifrat disposa de 52 caràcters (les lletres tant en majúscula com en minúscula).

[6]**Secret perfecte:** Claude Shannon va definir les mesures amb les quals s'avalua el nivell de secret que ofereix una determinada xifra. Informalment, diem que un criptosistema ofereix secret perfecte si el text xifrat no ofereix cap informació sobre el text en clar.

Així, un possible text xifrat seria `yHCBsjpGfgfzdw`, que correspondria a seleccionar la lletra `y` d'entre les tres alternatives per a xifrar `T` (`y`, `x` i `n`); la lletra `H` d'entre les tres alternatives per a xifrar `H` (`S`, `H` i `c`); etc.

Per a desxifrar seguiríem el procés invers, buscant les lletres de l'alfabet xifrat a la taula i extraient-ne la corresponent lletra en clar. En aquest cas, el desxifrat és únic. És a dir, per a un mateix text en clar, podem generar diferents textos xifrats. En canvi, per a un text xifrat, només hi haurà un únic text en clar.

Exercici 1.4 Genereu 5 textos xifrats diferents corresponent al missatge `THEBEALEPAPERS` fent servir la xifra de substitució homofònica amb la següent clau:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
j	B	N	P	s	T	i	S	q	l	e	h	D	W	R	f	E	d	w	y	O	M	a	X	t	Z
g			Q	z	u		H	U			p	k	L	m			A	K	x	r		v			
J			o	C			c	V					I	Y			F	b	n						
				G																					

Quina informació en pot extreure un criptoanalista que tingui accés als 5 textos xifrats (i sàpiga que es tracta d'un xifrat homofònic)?

La **xifra de Beale** és una xifra homofònica que feia servir com a clau la declaració d'independència dels Estats Units d'Amèrica.

La història diu que Thomas J. Beale va enterrar un tresor d'una expedició de miners que havien fet fortuna a les mines de l'oest llunyà a la dècada de 1820. El tresor, format per or, plata i joies, tindria actualment un valor d'uns 43 milions de dòlars. Beale va crear un conjunt de tres criptogrames que descrivien, respectivament, la localització, el contingut i els noms dels propietaris del tresor enterrat, i va deixar una capsa de ferro amb els criptogrames a un taverner anomenat Robert Morriss. Beale va desaparéixer, i el taverner va donar la capsa amb els criptogrames a un amic just abans de morir. L'amic, del qual no se'n coneix el nom, va aconseguir desxifrar el segon dels criptogrames fent servir un criptosistema homofònic amb la declaració d'independència dels Estats Units d'Amèrica com a clau. Per desxifrar el criptograma, l'amic va numerar cadascuna de les paraules de la declaració i va anar substituint cada número del text xifrat per la lletra inicial de la paraula que es trobava en la posició descrita pel número.

Es diu que l'amic no va ser capaç de trencar els altres dos criptogrames, motiu pel qual, l'any 1885, decideix fer pública la història i els criptogrames, amb l'esperança que algú altre pogués trencar-los. Des de llavors, hi ha hagut múltiples intents sense èxit de trencar els dos criptogrames restants.

De fet, les teories actuals apunten a què la història és en realitat un engany. Els arguments principals que en qüestionen la seva veracitat són que el text en clar del segon dels criptogrames fa servir paraules que no existien quan suposadament es van crear els criptogrames i que les característiques estadístiques dels dos criptogrames restants no semblen coincidir amb les que s'esperaria d'un text en anglès.

1.3 Resum

En aquest capítol hem presentat els conceptes bàsics relacionats amb la criptografia i hem descrit les fites històriques clau pel que fa al seu desenvolupament, tot introduint els criptosistemes que es van anar dissenyant durant l'era de la criptografia precientífica.

Anomenem **criptografia** a la ciència que estudia l'escriptura de secrets. En canvi, la **criptoànalisi** és la ciència que se centra en trencar les tècniques que desenvolupa la criptografia. Ambdues ciències treballen paral·lelament, de manera que els avenços d'una ajuden a avançar l'altra. Fem servir el mot general **criptologia** per englobar tant criptografia com criptoanàlisi.

Podem agrupar les xifres històriques en dos grans grups segons la tècnica que fan servir per xifrar: les xifres de **transposició** i les xifres de **substitució**. Les xifres de transposició modifiquen l'ordre dels caràcters del text en clar per generar el text xifrat. En canvi, les xifres de substitució canvien els caràcters del text en clar per altres caràcters.

1.4 Solucions dels exercicis

Exercici 1.1:

Tenint en compte les mides del bastó, procediríem a escriure el missatge longitudinalment:

```
THES
EARE
SPAR
TASW
ALLS
```

El missatge xifrat resultant seria, per tant: `TESTAHAPALERASLSERWS`.

Exercici 1.2:

En primer lloc convertim les lletres del missatge en la seva representació numèrica:

```
23 0 3 10 19 8 22 19 2 15 1 19 3 20 22 3 2 3 6 1 3 6 19 8 22 15 2
23 20 19 15 6 18 19 15 8 22
```

Seguidament, calculem $x-15 \mod 26$ per cada valor x de la representació numèrica de les lletres:

```
8 11 14 21 4 19 7 4 13 0 12 4 14 5 7 14 13 14 17 12 14 17 4 19 7 0 13
8 5 4 0 17 3 4 0 19 7
```

Finalment, recuperem el missatge en clar, convertint la seqüència numèrica de nou a lletres: `ILOVETHENAMEOFHONORMORETHANIFEARDEATH`

Exercici 1.3:

Convertim tant el missatge com la clau a la seva representació numèrica, i calculem el text en clar sumant els dos valors mòdul 26:

U	S	I	N	G	A	S	E	R	I	E	S	O	F	I	N	T	E	R	W	O	V	E	N	C	A	E	S	A	R	C	I	P	H	E	R	S
20	18	8	13	6	0	18	4	17	8	4	18	14	5	8	13	19	4	17	22	14	21	4	13	2	0	4	18	0	17	2	8	15	7	4	17	18
K	A	S	I	S	K	I	K	A	S	I	S	K	I	K	A	S	I	S	K	I	K	A	S	I	S	K	I	K	A	S	I	S	K	I	K	A
10	0	18	8	18	10	8	10	0	18	8	18	10	8	10	0	18	8	18	10	8	10	0	18	8	18	10	8	10	0	18	8	18	10	8	10	0
4	18	0	21	24	10	0	14	17	0	12	10	24	13	18	13	11	12	9	6	22	5	4	5	10	18	14	0	10	17	20	16	7	17	12	1	18
E	S	A	V	Y	K	A	O	R	A	M	K	Y	N	S	N	L	M	J	G	W	F	E	F	K	S	O	A	K	R	U	Q	H	R	M	B	S

El text xifrat resultant és doncs `ESAVYKAORAMKYNSNLMJGWFEFKSOAKRUQHRMBS`.

Exercici 1.4:

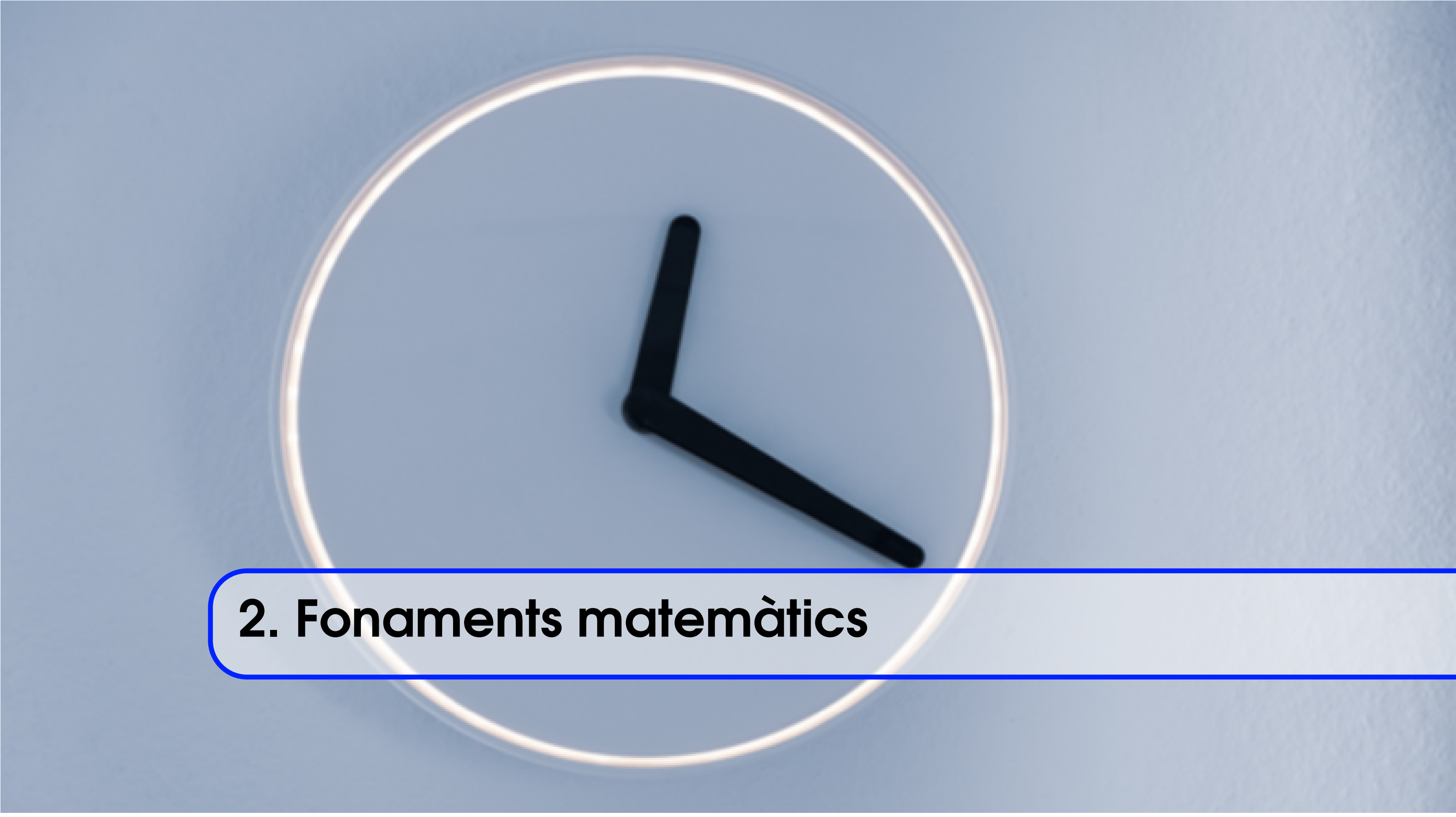
Cinc possibles textos xifrats són:

- `nHCBsJpsfjfGdK`
- `xHGBsJpCfJfCAb`
- `ycCBsjpzfgfsAw`
- `nczBzgpCfgfzFK`
- `xHsBCjpsfJfsFb`

Noteu que la solució no és única. A primer cop d'ull, un criptoanalista pot deduir que, amb probabilitat molt alta, les lletres xifrades `B`, `p` i `f` corresponen a lletres del text en pla que només tenen una única lletra xifrada assignada.

1.5 Bibliografia

Shannon, Claude E. (1949). *Communication theory of secrecy systems*. The Bell system technical journal, 28(4), 656-715.



2. Fonaments matemàtics

La criptografia és una disciplina amb un fort contingut matemàtic per diferents motius. Per una banda, la matemàtica permet mesurar de forma precisa la quantitat d'informació que conté un missatge i, per tant, també pot mesurar si quan el xifrem la quantitat d'informació que revela és menor o no en revela cap, de manera que ens dóna una mesura de la qualitat del sistema de xifrat que estem utilitzant. D'altra banda, l'explicitació de les funcions de xifrat i desxifrat de les quals hem parlat en l'anterior capítol sovint es realitza utilitzant funcions i expressions matemàtiques. Per aquest motiu, tenir uns bons coneixements de matemàtiques és fonamental per poder entedre el correcte funcionament de la criptografia.

En aquest capítol es proporcionen conceptes bàsics d'aritmètica modular així com algunes propietats dels nombres primers, necessaris en els criptosistemes de clau pública. D'altra banda, tal i com veurem al llarg d'aquest llibre, els criptosistemes de clau pública, així com algunes funcions que s'utilitzen en diferents protocols criptogràfics, basen la seva seguretat en problemes matemàtics difícils de resoldre. És per això que per entendre el grau de seguretat d'aquests sistemes és important comprendre quins són els problemes matemàtics que hi ha al darrere i quina dificultat té la seva resolució. En aquest capítol s'enuncien els problemes matemàtics més utilitzats en criptografia i se'n discuteix la seva complexitat.

Per últim, és important destacar que aquest capítol no pretén en cap cas proporcionar explicacions i demostracions formals dels conceptes matemàtics i menys encara presentar-ne una visió complerta. L'objectiu d'aquest capítol és dotar al lector de les eines necessàries per a entendre els criptosistemes i protocols que es descriuran al llarg del llibre. Per a aquells lectors que vulguin aprofundir en les nocions matemàtiques que es presenten en aquest capítol es recomana la lectura de les referències bibliogràfiques que s'indiquen al llarg del text.

2.1 Aritmètica modular

Normalment, en la nostra activitat quotidiana treballem amb els nombres reals amb els quals sabem realitzar tot un seguit de càlculs com ara sumes, restes, divisions, multiplicacions, exponenciacions, arrels quadrades, etc. Ara bé, una de les característiques dels nombres reals és que n'hi ha infinits, de manera que la seva representació en un ordinador és impossible. Una possibilitat de resoldre aquest problema és utilitzant conjunts que tinguin un nombre finit d'elements, podent-los representar tots sense cap problema.

L'aritmètica modular és una part de la matemàtica que permet definir tant aquest tipus de conjunts amb un nombre finit d'elements com també les operacions que permeten operar amb els elements d'aquest conjunt, assegurant que l'operació de dos elements del conjunt continuarà proporcionant un altre element del conjunt.

2.1.1 Estructures algebraiques: grups, anells i cossos

Des d'un punt de vista informal, podem definir una estructura algebraica com un conjunt d'elements i unes operacions associades que permeten operar amb els elements del conjunt. Depenent de les operacions que definim sobre el conjunt, quantes en definim i quines propietats tinguin, podem classificar l'estructura algebraica en diferents tipus, els més coneguts dels quals són els grups, els anells i els cossos.[1]

Per exemple, si prenem el conjunt dels nombres enters, que es representen per la lletra $\mathbb{Z}$, el qual té infinits elements $\mathbb{Z} = \{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$ i hi definim l'operació suma tal i com la coneixem, l'estructura algebraica resultant, que podem denotar per $(\mathbb{Z},+)$, és un grup. Això és així donada la següent definició.

Definició 2.1 Un **grup** és una estructura algebraica en què l'operació definida compleix la propietat associativa i, a més, el conjunt sobre el qual està definida l'operació conté l'element neutre i l'element invers d'aquesta operació, que anomenarem invers additiu o oposat.

Per exemple, si prenem tres valors enters qualssevol, com ara el -3, el -1 i el 2, efectivament, veiem que compleixen les propietats anteriors. Per la propietat associativa, tenim que $((-3)+(-1))+2 = (-3)+((-1)+2)$. D'altra banda, l'element neutre de la suma (aquell que sumat amb qualsevol valor dóna ell mateix) pertany als enters, ja que com sabem el neutre de la suma és el 0. L'element invers respecte la suma (aquell que sumat amb un element dóna el neutre de la suma) és el mateix valor canviat de signe, que també pertany al conjunt de nombres enters. Evidentment, aquest exemple concret no és cap demostració que $(\mathbb{Z},+)$ és un grup però ens dóna una exemplificació dels conceptes de propietat associativa, element neutre i element invers.

D'altra banda, també podem assegurar que l'estructura algebraica dels nombres naturals amb la suma, $(\mathbb{N},+)$ no és un grup ja que si bé la suma sobre els naturals si que té la propietat associativa i l'element neutre és el 0, que sí que pertany als naturals, l'element invers per la suma de cada element d'$\mathbb{N}$ no pertany a aquest conjunt, ja que els naturals només comprenen nombres positius (i el 0) i els inversos per la suma d'aquests valors són nombres negatius.

Una altra propietat interessant de les estructures algebraiques és la commutativitat. Un grup

[1] Malgrat que les operacions que es poden definir en una estructura algebraica poden ser tan complicades com es vulguin, a llarg d'aquest text ens restringirem a les dues operacions habituals de suma, $+$, i producte, $\cdot$, tal i com les coneixem habitualment.

s'anomena commutatiu si l'operació que hi ha definida és commutativa, és a dir, donats dos elements del conjunt a i b es compleix que $a+b=b+a$.

De la mateixa manera que hem definit una estructura algebraica amb una operació, en podem definir d'altres amb dues operacions diferents. Per exemple, la següent estructura algebraica està formada pels nombres reals amb les operacions de suma i producte: $(\mathbb{R},+,\cdot)$. En aquest cas, podem caracteritzar-les en funció de les propietats que presentin cada una de les operacions, fet que proporciona la definició d'anell i de cos.

Definició 2.2 Un **anell** és una estructura algebraica amb dues operacions on una d'elles presenta estructura de grup commutatiu, l'altra operació compleix la propietat associativa i, a més, ambdues propietats compleixen la distributivitat d'una respecte l'altra.

Exemple 2.1

L'estructura algebraica $(\mathbb{Z},+,\cdot)$ és un anell ja que com hem comentat anteriorment $(\mathbb{Z},+)$ és un grup, a més és commutatiu i es compleix la propietat distributiva de la suma respecte el producte, és a dir[a], $\forall a,b,c \in \mathbb{Z}$, $a\cdot(b+c)=(a\cdot b)+(a\cdot c)$.

[a]Recordeu que el símbol $\forall$ es llegeix com "per a qualsevol element".

Dels elements d'una estructura algebraica n'hi ha alguns d'especialment rellevants, com ara l'element neutre de cada una de les operacions. Són aquells que operats amb qualsevol element del grup no n'afecten el seu resultat (és a dir, el 0 per a la suma i l'1 per al producte). Com hem vist, en la definició de grup s'exigia que l'element neutre de la suma estigués contingut en el conjunt d'elements. Ara bé, en un anell no s'ha indicat cap condició sobre l'existència o no del neutre del producte. Per tant, podem enunciar la següent definició:

Definició 2.3 Un **anell amb unitat** és un anell que conté el neutre respecte el producte.

Exemple 2.2 $(\mathbb{Z},+,\cdot)$ és un anell amb unitat ja que $1\in\mathbb{Z}$ i 1 és el neutre del producte, perquè compleix que $\forall a\in\mathbb{Z}$, $a\cdot 1=1\cdot a=a$.

L'element unitat en un anell és important perquè ens permet definir el concepte d'element invers.

Definició 2.4 Donat un anell amb unitat, direm que un element a és **invertible** si existeix un altre element b tal que $a\cdot b=b\cdot a=1$, on 1 és l'element unitat.

Amb la definició d'element invertible, ja podem definir l'estructura algebraica més important que hi ha, el cos.

Definició 2.5 Una estructura algebraica és un **cos**, quan aquesta és un anell amb unitat on qualsevol element, llevat de l'element neutre de la suma, és invertible.

Notació 2.1. *Utilitzarem l'asterisc per denotar el subconjunt d'elements invertibles. Per exemple,* $\mathbb{Z}^*=\{1,-1\}$ *ja que són els únics elements que tenen invers. D'altra banda,* $\mathbb{R}^*=\mathbb{R}\setminus\{0\}$ *ja que tots els reals llevat del zero són invertibles.*

Així, l'anell $(\mathbb{Z},+,\cdot)$ tot i ser un anell amb unitat no és un cos perquè no tots els elements tenen invers. Per exemple, l'invers de 2 és $\frac{1}{2}$, i aquesta fracció no pertany als $\mathbb{Z}$. De fet, els únics elements que tenen inversos en $\mathbb{Z}$ són l'1 i el -1. D'altra banda, els nombres reals amb la suma i el producte $(\mathbb{R},+,\cdot)$ si que són un cos perquè tots els elements, llevat del 0, tenen invers en els $\mathbb{R}$.

2.1.2 Divisibilitat als enters

Com acabem de veure, els enters amb la suma i el producte són un anell, però no tenen estructura de cos perquè no tots els elements tenen invers respecte el producte. Aquest fet fa que quan calculem una divisió entre dos nombres enters el resultat no sempre sigui un nombre enter. Ara bé, el que si que podem fer és caracteritzar els elements de l'equació resultant d'una divisió entera. Aquesta caracterització la proporciona el següent teorema.[2]

Teorema 2.1 Donats dos elements $a,b \in \mathbb{Z}$ qualssevol (amb $b \neq 0$), $\exists\, q,r \in \mathbb{Z}$ únics, tals que $a = b\cdot q + r$, on $0 \leq r < |b|$

El que ens indica aquest teorema és que si dividim l'element a per un element b tenim com a resultat un quocient, q, i un residu, r. A més, com ja sabem, el residu sempre serà més petit que b ja que si no ho fos podríem continuar dividint i tindríem un quocient $q+1$, i això ho podríem realitzar de forma repetida fins que el residu sigui més petit que b.

Una vegada caracteritzats els elements de la divisió entera, podem definir el concepte de divisibilitat als enters.

Definició 2.6 Donats $a,b \in \mathbb{Z}$, diem que b divideix a si i només si $\exists\, q \in \mathbb{Z}$ tal que $a = b\cdot q$. Ho denotarem per $b|a$.

Un dels entrebancs que sovint hi ha amb el concepte de divisibilitat és la multiplicitat de definicions que en són equivalents. Així, que l'element b divideixi a l'element a és equivalent a dir qualsevol de les següents expressions:

- b és factor d'a
- b és divisor d'a
- a és divisible per b
- a és múltiple de b

La noció de divisibilitat és important perquè ens permet definir altres eines com ara el màxim comú divisor o caracteritzar alguns nombres, com ara els nombres primers.

Definició 2.7 Donats dos elements $a,b \in \mathbb{Z}$ direm que d és el màxim comú divisor d'a i de b si d divideix tant a com b i donat qualsevol altre valor c que també divideixi a i b, tenim que $c < d$. Denotarem el màxim comú divisor com $\gcd(a,b) = d$ (de l'anglès, *greatest common divisor*).

Dels nostres estudis previs en matemàtiques molt probablement el càlcul del màxim comú divisor el sabem fer a partir de la descomposició dels nombres a i b en factors primers i prendre'n els comuns amb els menors exponents. És a dir, si volem calcular el $\gcd(16,28)$, com que sabem que $16 = 2^4\cdot 1$ i $28 = 2^2\cdot 7\cdot 1$ podem concloure que el $\gcd(16,28) = 2^2 = 4$. Ara bé, aquest sistema per calcular el màxim comú divisor no és gens eficient perquè implica haver de factoritzar els nombres pels quals volem calcular-ne el màxim comú divisor. L'operació de factoritzar, com veurem més endavant en aquest mateix capítol, és un procés molt poc eficient computacionalment, per això

[2] Recordeu que el símbol $\exists$ es llegeix com "Existeix".

es fa servir l'Algorisme d'Euclides que permet calcular el màxim comú divisor de forma eficient, independentment de la mida del nombres. L'Algorisme d'Euclides es basa en el següent teorema:

Teorema 2.2 — Teorema d'Euclides. Siguin $a,b,q,r \in \mathbb{Z}$ tals que $a = b \cdot q + r$ aleshores $\gcd(a,b) = \gcd(b,r)$.

Aquest teorema ens indica que podem calcular el màxim comú divisor de dos valors, a i b, calculant el màxim comú divisor de dos valors diferents, b i r, els quals són tots dos més petits que els anteriors, $b < a$ i $r < b$. Així, calcular el màxim comú divisor consistirà en fer un càlcul recursiu en el que es van dividint els nombres entre ells fins arribar a la condició final, que es concreta en el fet que $\gcd(x,0) = x, \forall x \in \mathbb{Z}$.

Exemple 2.3 Càlcul de màxim comú divisor utilitzant l'Algorisme d'Euclides

Si volem calcular el màxim comú divisor de 2756 i 2621 podem realitzar les següents divisions successives:

$2756 = 2621 \cdot 1 + 135$
$2621 = 135 \cdot 19 + 56$
$135 = 56 \cdot 2 + 23$
$56 = 23 \cdot 2 + 10$
$23 = 10 \cdot 2 + 3$
$10 = 3 \cdot 3 + 1$
$3 = 1 \cdot 3 + 0$

D'aquestes divisions i en base al Teorema d'Euclides tenim que:
$\gcd(2756,2621) = \gcd(2621,135) = \gcd(135,56) = \gcd(56,23) = \gcd(23,10) = \gcd(10,3) = \gcd(3,1)$ Per tant, com que l'últim residu no nul és el 1, tenim que $\gcd(2756,2621) = 1$

Exercici 2.1 Calcula el màxim comú divisor de 35 i 48.

Una vegada definit el màxim comú divisor de dos nombres, podem donar la definició de nombres coprimers.

Definició 2.8 Dos elements a i b s'anomenen coprimers quan el $\gcd(a,b) = 1$.

Un altre teorema important que ens servirà més endavant per a calcular inversos modulars, és la Identitat de Bézout, que permet expressar el màxim comú divisor de dos elements com a combinació lineal dels mateixos.

Teorema 2.3 — Identitat de Bézout. Siguin $a,b \in \mathbb{Z}$ tals que $\gcd(a,b) = d$. Aleshores, existeixen uns únics valors $\lambda, \mu \in \mathbb{Z}$ tals que $\lambda a + \mu b = d$.

Tot i que la Identitat de Bézout només ens indica l'existència d'aquests dos valors, podem utilitzar el càlcul del màxim comú divisor amb l'Algorisme d'Euclides per calcular-ne exactament els valors λ i μ, tal i com es mostra en el següent exemple.

Exemple 2.4 Càlcul dels coeficients de la Identitat de Bézout

El càlcul dels coeficients de la Identitat de Bézout es pot realitzar utilitzant la descomposició que en resulta de l'Algorisme d'Euclides. Així, si volem calcular el coeficients de la Identitat de Bézout per als valors 2756 i 2621, primer farem el càlcul de les divisions successives de l'Algorisme d'Euclides:

$2756 = 2621 \cdot 1 + 135$
$2621 = 135 \cdot 19 + 56$
$135 = 56 \cdot 2 + 23$
$56 = 23 \cdot 2 + 10$
$23 = 10 \cdot 2 + 3$
$10 = 3 \cdot 3 + 1$

Posteriorment, en cada equació n'aïllarem el residu:

$2756 - (2621 \cdot 1) = 135$
$2621 - (135 \cdot 19) = 56$
$135 - (56 \cdot 2) = 23$
$56 - (23 \cdot 2) = 10$
$23 - (10 \cdot 2) = 3$
$10 - (3 \cdot 3) = 1$

i finalment substituirem en cada equació el valor corresponent per acabar obtenint-ne una sola amb els valors 2756 i 2621:

$1 = 10 - (3 \cdot 3) = 10 - ((23 - (10 \cdot 2)) \cdot 3) = (10 \cdot 7) - (23 \cdot 3) = ((56 - (23 \cdot 2)) \cdot 7) - (23 \cdot 3) = (56 \cdot 7) - (23 \cdot 17) = (56 \cdot 7) - ((135 - (56 \cdot 2)) \cdot 17) = (56 \cdot 41) - (135 \cdot 17) = ((2621 - (135 \cdot 19)) \cdot 41 - (135 \cdot 17) = (2621 \cdot 41) - (135 \cdot 769) = (2621 \cdot 41) - ((2756 - (2621 \cdot 1)) \cdot 796) = (2621 \cdot 837) - (2756 \cdot 796)$

Així, tenim que

$1 = 2621 \cdot 837 + 2756 \cdot (-796)$

i per tant els coeficients de la Identitat de Bézout per a 2756 i 2621 són -796 i 837 respectivament.

Exercici 2.2 Calcula els coeficients de la indentitat de Bezout de 35 i 48.

Com ja hem indicat anteriorment, a més del màxim comú divisor, el concepte de divisibilitat també ens permet definir els nombres primers.

Definició 2.9 Direm que un nombre $p \in \mathbb{N}$, amb $p > 1$, és primer si només és divisible per ell mateix i per 1.

L'últim concepte relacionat amb la divisibilitat als enters que definirem és la funció fi d'Euler, la qual, com veurem més endavant, és la base del funcionament de l'algorisme de xifrat RSA.

Definició 2.10 — Funció fi d'Euler. La funció fi d'Euler d'un valor natural n, $\phi(n)$, es defineix com el cardinal del conjunt de nombres coprimers amb n més petits que n. És a dir:

$$\phi(n) = \#\{a,\ \ 0 \le a < n,\ \ \text{tal que}\ \ \gcd(a,n) = 1\}$$

Exemple 2.5 Càlcul de funció fi d'Euler

Si volem calcular el valor de $\phi(10)$ seguint la definició d'aquesta funció, ens caldrà calcular tots els nombres més petits que 10 que són coprimers amb 10, és a dir, que el $\gcd(x,10) = 1$. Si els calculem resultarà que són els següents $\{1,3,7,9\}$, per tant, el valor de la funció fi d'Euler serà el nombre d'elements d'aquest conjunt, és a dir 4.

$$\phi(10) = \#\{1,3,7,9\} = 4$$

Més enllà de calcular tots els elements coprimers amb n i comptar-los hi ha tècniques més eficients per calcular la funció fi d'Euler. La més eficient que es coneix consisteix a descompondre el valor n en factors primers. Un cop descompost el valor, s'utilitzen les següents propietats:

- Si p és un nombre primer, $\phi(p) = p-1$. Això és fàcil de veure perquè tots els elements més petits que p seran coprimers amb p donat que no tenen cap factor en comú, justament pel fet que p és primer.
- Si $n = p \cdot q$ tals que p i q són coprimers, aleshores $\phi(n) = \phi(p) \cdot \phi(q)$.
- Si n és una potència d'un primer, $n = p^k$, aleshores $\phi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$

Com podem veure, malgrat que aquesta sigui la millor manera de calcular la funció d'Euler, donat que implica descompondre el valor en factors primers, no és una tasca computacionalment eficient quan els valor del nombre és molt elevat.

Exercici 2.3 Calcula quin és el valor de $\phi(527)$.

2.1.3 Aritmètica modular amb enters

En el primer aparat d'aquest capítol hem vist quines són les propietats que ha de tenir una estructura algebraica per a ser un grup, un anell o un cos. Com ja hem comentat, els cossos són estructures algebraiques molt versàtils gràcies a les propietats que presenten les seves operacions. Els exemples d'anells o cossos que hem vist en l'apartat anterior, i els que coneixem normalment, són exemples on el conjunt d'elements és un conjunt infinit. Així, el conjunt dels enters amb la suma i el producte $(\mathbb{Z},+,\cdot)$ és un anell, però els enters és un conjunt d'elements infinit. Igualment, el conjunt dels reals amb la suma i el producte $(\mathbb{R},+,\cdot)$ és un cos, però, de nou, els reals són un conjunt infinit. Per tant, ens podem preguntar si podem crear estructures algebraiques que siguin anells o cossos, però que tinguin un nombre finit d'elements. I la resposta a aquesta pregunta és afirmativa.

Definició 2.11 Definirem el conjunt dels enters mòdul n, per a $n \geq 2$, i el denotarem per $\mathbb{Z}_n$, com tots els nombres enters entre 0 i $n-1$, és a dir $\mathbb{Z}_n = \{0,1,\cdots,n-1\}$.

Com és evident, els enters mòdul n és un conjunt finit, ja que conté exactament n elements. Per tant, si aconseguim definir una operació suma i una operació producte que tinguin les propietats que hem enumerat en anteriors apartats, això ens permetrà construir anells i cossos amb un nombre finit d'elements. Per definir tant la suma com el producte a $\mathbb{Z}_n$ utilitzarem la definició de suma i producte d'enters que ja coneixem. Ara bé, caldrà anar en compte perquè és important que les operacions siguin operacions internes, és a dir, quan operem dos elements d'un conjunt cal que el resultat sigui un element del mateix conjunt. Això amb els conjunts en els que estem acostumats a treballar ja passa, perquè si sumem (o multipliquem) dos elements enters en dóna un enter i si sumem (o multipliquem) dos nombres reals, el resultat també és un nombre real. Ara bé, si prenem la suma tal i com la coneixem i considerem ara el conjunt $\mathbb{Z}_5 = \{0,1,2,3,4\}$ veurem que cal anar amb compte perquè si fem la suma de dos elements de $\mathbb{Z}_5$, per exemple $4+4$ el resultat és 8, que no és un element del conjunt $\mathbb{Z}_5$. Per tal de resoldre aquest problema el que farem és pensar $\mathbb{Z}_n$ com una reducció de tots els enters. Si tenim una manera per "reduir" qualsevol enter a un valor de $\mathbb{Z}_n$ ja haurem aconseguit l'objectiu, perquè per una banda tenim definides la suma i el producte de manera que el resultat és un enter, i amb l'eina de reducció, podríem reduir el resultat de l'operació a un element de $\mathbb{Z}_n$.

Per obtenir aquesta funció de reducció de tots els enters a $\mathbb{Z}_n$ només ens cal recuperar el teorema de divisió entera que hem vist anteriorment. Efectivament, si volem reduir un element enter a a un enter entre 0 i $n-1$ només cal fer la divisió entera de a entre n. Aquesta divisió tindrà un residu únic. A més, com que hem dividit per n, aquest residu serà un valor entre 0 i $n-1$ que és justament el que ens interessa. També direm que reduïm l'element a mòdul n, i escriurem $a \pmod n$.

Si tornem ara a l'exemple dels enters mòdul 5, que recordem que és el conjunt format per $\mathbb{Z}_5 = \{0,1,2,3,4\}$, veiem que la suma que proposàvem $4+4$ donava com a resultat 8. Ara bé, si reduïm el 8 tal i com hem descrit anteriorment, tenim que la divisió entera de 8 entre 5 dóna com a quocient 1 i de residu 3. Per tant, podem concloure que el 8 equival a un 3 a $\mathbb{Z}_5$ i que per tant, la suma que teníem ens queda $4+4=8=3$ a $\mathbb{Z}_5$. De la mateixa manera que hem pogut definir la suma, podem fer el mateix amb el producte. Així $3\cdot 4 = 2$ a $\mathbb{Z}_5$ ja que si dividim 12 (que és el resultat de 3 per 4) entre 5 tenim 2 de residu. [3]

Exercici 2.4 Quants elements té el conjunt $\mathbb{Z}_{25}$?

Una vegada definit el conjunt dels enters mòdul n i les operacions de suma i producte dins d'aquest conjunt, ja podem enunciar el següent teorema:

Teorema 2.4 L'estructura algebraica $(\mathbb{Z}_n,+,\cdot)$ amb la suma i el producte tal i com els hem definit anteriorment i per a qualsevol valor $n \geq 2$ és un anell commutatiu amb unitat.

D'aquesta manera, hem pogut definir un anell sobre un conjunt d'elements finits, com és el cas de $\mathbb{Z}_n$. Per fer operacions de suma i producte a $\mathbb{Z}_n$ ens caldrà únicament operar de forma normal amb els enters i un cop obtingut el resultat final reduir-lo al mòdul on treballem. A més, aquesta reducció al mòdul la podem fer al final dels càlculs o en qualsevol moment, per exemple, per simplificar els valors amb els que estem treballant.

Exemple 2.6 Càlculs en anells modulars

Si volem saber el valor de l'expressió $5\cdot(4+14)-3\cdot 8$ a $\mathbb{Z}_{10}$ podem fer el següents càlculs:

$$5\cdot(4+14)-3\cdot 8 \pmod{10}$$

$$5\cdot(18)-24 \pmod{10}$$
$$90-24 \pmod{10}$$

[3] Les equacions a $\mathbb{Z}_n$ s'anomenen equacions modulars. Per indicar l'equació modular $4+4$ a $\mathbb{Z}_5$ escriurem el següent: $4+4=3 \pmod 5$.

$$66 \pmod{10}$$

$$6 \pmod{10}$$

on l'últim pas prové del fet que el residu de dividir 66 entre 10 és 6.

Fixeu-vos que una altra manera de realitzar el càlcul és que en el primer pas, haguéssim reduït tant el 18 com el 24 mòdul 10, això és hagués proporcionat l'expressió
$5\cdot(8)-4 \pmod{10}$
més fàcil de gestionar per la mida dels valors. Aquesta expressió també hagués proporcionat el mateix resultat final, ja que
$5\cdot(8)-4 \pmod{10} = 40-4 \pmod{10} = 36 \pmod{10} = 6 \pmod{10}$

Fixeu-vos que fins ara només ens hem referit a operar amb sumes, restes i multiplicacions. Això és així perquè, fins al moment, hem pogut assegurar que l'estructura algebraica que hem construït és un anell. Però en un anell no necessàriament tots els elements tenen invers pel producte. Arribats a aquest punt, ens podem preguntar si l'estructura algebraica $(\mathbb{Z}_n,+,\cdot)$, a més d'un anell és també un cos.

Pel Teorema 2.4, $(\mathbb{Z}_n,+,\cdot)$ és un anell commutatiu amb unitat. P er veure si $(\mathbb{Z}_n,+,\cdot)$ és un cos, atenent-nos a la definició de cos que hem donat, només ens cal comprovar dos fets. El primer és que aquest anell té unitat. I el segon, que tot element de l'anell, llevat del neutre de la suma, és invertible. La primera comprovació és trivial, ja que sabem que l'element 1, que és la unitat del producte, sempre pertany a $\mathbb{Z}_n$ (ja que hem dit que $n \geq 2$). Ara bé, la segona propietat no sempre és certa, i dependrà del tipus de valor n.

Teorema 2.5 L'estructura algebraica $(\mathbb{Z}_p,+,\cdot)$ és un cos si i només si el valor p és un nombre primer.

Aquest teorema ens indica que, per exemple, l'estructura algebraica $(\mathbb{Z}_{17},+,\cdot)$ és un cos perquè 17 és un nombre primer. Per tant, qualsevol element a $\mathbb{Z}_{17}$, que recordem que està format pels elements $\{0,1,2,3,4,\cdots,15,16\}$, té invers pel producte. El fet que qualsevol element tingui invers és molt rellevant perquè permet fer divisions amb elements d'aquest conjunt. En efecte, si volem calcular $\frac{2}{3}$ només hem de saber quan val l'invers de 3, és a dir 3^{-1} i multiplicar aquest valor per 2.

Inversos modulars

Com acabem de veure, l'últim teorema de l'apartat anterior ens indica que l'estructura algebraica $(\mathbb{Z}_p,+,\cdot)$ és un cos si p és primer. Per tant, sabem que per a qualsevol element de $\mathbb{Z}_p$, llevat del zero, podem calcular-ne el seu valor invers. Vegem com fer-ho.

En primer lloc, és important recordar la definició d'element invers. Per exemple, si volem calcular l'invers de 3 a $\mathbb{Z}_{17}$ sabem que estem buscant un valor que multiplicat per 3 valgui 1 a $\mathbb{Z}_{17}$. A més, com que sabem que $\mathbb{Z}_{17}$ és un cos sabem que l'invers de 3 ha de pertànyer a $\mathbb{Z}_{17}$, per tant ha de ser un valor del conjunt $\{0,1,2,3,4,\cdots,15,16\}$. Si multipliquem cada un d'aquests elements d'aquest conjunt per 3, un dels productes ens donarà 1. En efecte, si fem el producte $3\cdot 6 = 18$ veiem que el resultat, reduït a mòdul 17, és 1. Així doncs, l'invers de 3 mòdul 17 serà 6.

Evidentment, aquest sistema que acabem de descriure no és bo per calcular inversos modulars en cas que el valor de p sigui molt gran, ja que ens requeriria fer molts productes. Una manera eficient de calcular inversos és utilitzant la Identitat de Bézout.

Fixeu-vos que si volem calcular l'invers de $x \in \mathbb{Z}_p$, on p és primer, com que x és més petit que p i p és primer, tenim que $\gcd(x, p) = 1$, ja que si el màxim comú divisor d no fos 1, p no seria primer perquè es podria dividir per d. Ara bé, si $\gcd(x, p) = 1$, per la Identitat de Bézout, sabem que existeixen dos elements λ i μ tals que $x \cdot \lambda + p \cdot \mu = 1$. Però fixeu-vos que si aquesta equació l'expressem modularment a $\mathbb{Z}_p$ ens queda $x \cdot \lambda + p \cdot \mu = 1 \pmod p$ i si la reduïm modularment obtenim l'equació equivalent $x \cdot \lambda = 1 \pmod p$ donat que p a $\mathbb{Z}_p$ val 0 (ja que el residu de dividir p entre p és 0). Per tant, el valor que multiplicat per x dóna 1 mòdul p és justament λ. Dit en altres paraules, un dels coeficients de la Identitat de Bézout és el que ens proporciona l'invers modular.

Exemple 2.7 Càlcul d'invers modular

Si volem calcular l'invers de 9 mòdul 11 calcularem els coeficients de la Identitat de Bézout tal i com hem mostrat en exemples anteriors, utilitzant l'algorisme de les divisions successives:
$11 = 9 \cdot 1 + 2$
$9 = 2 \cdot 4 + 1$
De la segona equació tenim $1 = 9 - (2 \cdot 4)$ i de la primera equació $2 = 11 - (9 \cdot 1)$. Si les combinem ens queda: $1 = 9 - (11 - (9 \cdot 1) \cdot 4) = 9 - (11 \cdot 4) + (9 \cdot 4) = 5 \cdot 9 - 11 \cdot 4$. Per tant, els coeficients de la Identitat de Bézout de 9 i 11 són 5 i -4 respectivament ja que $\gcd(9, 11) = 1 = 5 \cdot 9 + 11 \cdot (-4)$. Per tant, si reduïm aquesta equació mòdul 11 ens queda $5 \cdot 9 \pmod{11} = 1$, és a dir, l'invers de 9 mòdul 11 és 5. Això és fàcil de comprovar, perquè $9 \cdot 5 = 45$ i el residu de dividir 45 per 11 és, efectivament, 1.

Exercici 2.5 Troba l'invers de 7 a $\mathbb{Z}_{37}$

Exercici 2.6 Realitza els següents càlculs a $\mathbb{Z}_{37}$

- $20 + 20$
- $20 \cdot 4$
- 20^2
- $\frac{20}{7}$

Exercici 2.7 Per què l'estructura algebraica $(\mathbb{Z}_{37}, +, \cdot)$ és un cos?

El Teorema d'Euler

L'aritmètica modular conté innombrables resultats, que tot i ser molt interessants queden fóra de l'abast d'aquest llibre. En aquest apartat ens centrarem únicament amb el Teorema d'Euler, que és la pedra angular del funcionament de l'algorisme de clau pública RSA.

Teorema 2.6 — Teorema d'Euler. Sigui n un nombre natural i $\phi(n)$ la seva funció fi d'Euler. Si $\gcd(x, n) = 1$, aleshores:

$$x^{\phi(n)} = 1 \bmod n$$

Aquest teorema ens indica que quan estem en un anell modular, qualsevol valor coprimer amb el mòdul elevat a la funció d'Euler del mòdul és igual a la identitat. La importància d'aquest teorema en l'RSA és que permet demostrar, com veurem més endavant, que quan xifrem un missatge i posteriorment el desxifrem, el resultat que obtenim és el text en clar original.

Una altra implicació del Teorema d'Euclides és el següent resultat que ens permet calcular inversos modulars per mitjà d'exponenciacions:

Proposició 2.1 Sigui $x \in \mathbb{Z}_n$ tal que $\gcd(x,n) = 1$, l'invers d'x a $\mathbb{Z}_n$ és $x^{\phi(n)-1}$.

Demostració: La demostració d'aquesta proposició és immediata fent servir el Teorema d'Euclides. Efectivament si multipliquem x per $x^{\phi(n)-1}$ tenim $x^{\phi(n)}$ que val 1 a $\mathbb{Z}_n$ pel Teorema d'Euclides, cosa que prova que són inversos. ∎

Exemple 2.8 Càlcul d'invers modular

Si volem calcular l'invers de 2 a $\mathbb{Z}_{11}$, com que $\gcd(2,11) = 1$, podem calcular la funció d'Euler del mòdul $\phi(11) = 10$ i posteriorment calcular la següent potència:

$$2^{10-1} = 2^9 = 512 = 6 \pmod{11}$$

Per tant, l'invers de 2 mòdul 11 val 6.

Elements primitius

Quan treballem amb cossos finits, el fet que el conjunt d'elements que tenim sigui finit junt amb el fet que les operacions entre elements han de ser internes, ens trobem en situacions que no es donen quan treballem amb conjunts de mida infinita. Un exemple d'aquest cas el trobem en la definició d'ordre d'un element.

Definició 2.12 L'**ordre d'un element** $a \in \mathbb{Z}_n$ és el mínim exponent $i \in \mathbb{N}; i > 0$ tal que $a^i = 1$ mòdul n.

Exemple 2.9 Ordre d'un element

L'ordre de l'element 5 a $\mathbb{Z}_{42}$ és 6 ja que $5^1 = 5,\ 5^2 = 25,\ 5^3 = 41,\ 5^4 = 37,\ 5^5 = 17,\ 5^6 = 1$.

Una vegada definit el concepte d'ordre, podem definir el concepte d'element primitiu.

Definició 2.13 Un element $g \in \mathbb{Z}_n$ és un **element primitiu** si té ordre $\phi(n)$.

Fixeu-vos que en el cas que el mòdul del nostre conjunt sigui un nombre primer p, tenim que l'element g serà primitiu a $\mathbb{Z}_p$ si té ordre $\phi(p)$. Com que p es primer sabem que $\phi(p) = p-1$, per tant, l'ordre de g serà $p-1$. Això vol dir que si prenem g i anem calculant potències ens generarà $p-1$ elements diferents. Ara bé, fixeu-vos que $\mathbb{Z}_p$ té p elements diferents i, en concret, llevat del 0 tots són invertibles (ja que $\mathbb{Z}_p$ és un cos). Per tant, $\mathbb{Z}_p^*$ té $p-1$ elements i podem concloure que les $p-1$ potències diferents de g generen tots els elements invertibles de $\mathbb{Z}_p$. Per exemple, si prenem $\mathbb{Z}_7 = \{0,1,2,3,4,5,6\}$, veiem que els seus elements invertibles són $\mathbb{Z}_7^* = \{1,2,3,4,5,6\}$. Tenim que 3 és un element primitiu, i podem comprovar que les seves potències poden generar tots els elements invertibles: $3^1 = 3,\ 3^2 = 2,\ 3^3 = 6,\ 3^4 = 4,\ 3^5 = 5,\ 3^6 = 1$.

Sistemes d'equacions modulars

En apartats anteriors hem vist la resolució d'equacions modulars utilitzant els mecanismes estàndards de resolució d'equacions però tenint en compte que el conjunt on treballem és $\mathbb{Z}_n$. Ara bé, es pot donar el cas que ens interessi resoldre un sistema d'equacions definides sobre diferents mòduls, com per exemple:

$$\begin{cases} 3x+5=0 \pmod{11} \\ 3x-2=0 \pmod{5} \end{cases}$$

En aquest cas, el Teorema xinès dels residus ens proporciona informació sobre l'existència d'una solució.

Teorema 2.7 — Teorema xinès dels residus. Siguin n_1 i n_2 dos elements naturals tals que $\gcd(n_1, n_2) = 1$ aleshores, el sistema d'equacions modulars:

$$\begin{cases} x = a_1 \pmod{n_1} \\ x = a_2 \pmod{n_2} \end{cases}$$

té una única solució mòdul $n = n_1 \cdot n_2$ definida per l'equació:

$$x = \lambda \cdot n_2 \cdot a_1 + \mu \cdot n_1 \cdot a_2 \pmod{n}$$

on λ i μ són els coeficients de la Identitat de Bézout $\mu \cdot n_1 + \lambda \cdot n_2 = 1$.

Exemple 2.10 Resolució d'un sistema d'equacions modular

Si volem resoldre el sistema d'equacions següent:

$$\begin{cases} 3x+5=0 \pmod{11} \\ 3x-2=0 \pmod{5} \end{cases}$$

En primer lloc ens cal expressar les equacions en el format adequat. És a dir:

$$\begin{cases} x=2 \pmod{11} \\ x=4 \pmod{5} \end{cases}$$

A continuació, calcularem els elements de la Identitat de Bézout de 11 i 5, que són $\mu = 1$ i $\lambda = -2$, ja que $1 \cdot 11 + (-2) \cdot 5 = 1$. Per tant, la solució serà:

$$x = \lambda \cdot n_2 \cdot a_1 + \mu \cdot n_1 \cdot a_2 \pmod{n} = -2 \cdot 5 \cdot 2 + 1 \cdot 11 \cdot 4 \pmod{55} = 24 \pmod{55}$$

Residus quadràtics i arrels quadrades modulars

Quan treballem amb estructures algebraiques finites, una de les operacions que també ens interessarà realitzar són arrels quadrades. Com veurem en aquest apartat, no tots els elements tindran arrels quadrades i, a més, en cas que en tinguin en poden tenir més de dues.

Definició 2.14 Sigui p un nombre primer. Direm que $y \in \mathbb{Z}_p$ és un residu quadràtic si existeix un valor $x \in \mathbb{Z}_p$ tal que $x^2 = y$. En cas que no existeixi aquest valor, direm que y no és un residu quadràtic.

En el cos $\mathbb{Z}_p$, hi ha el mateix nombre d'elements que són residu quadràtic que elements que no ho són, és a dir un total de $\frac{p-1}{2}$ elements. A més, hi ha una manera fàcil de calcular si un element és un residu quadràtic aplicant la següent expressió:

$$y^{\frac{p-1}{2}} = \begin{cases} 1 & \text{si } y \text{ és un residu quadràtic} \\ -1 & \text{si } y \text{ no és un residu quadràtic} \end{cases}$$

Un cop sabem que un valor és un residu quadràtic, podem calcular-ne les seves arrel quadrades, ja que sabem que existeixen. Amb la fórmula anterior, és molt simple comprovar si un valor és un residu quadràtic o no ho és. Ara bé, calcular-ne les arrels quadrades suposa una mica més de feina. De tota manera, per a nosaltres ens serà suficient saber que existeixen algorismes eficients[4] que poden calcular arrels quadrades d'un residu quadràtic a $\mathbb{Z}_p$ encara que el valor p sigui molt gran, això sí, sempre que p sigui un nombre primer.

Quan deixem els nombres primers com a base de la nostra estructura algebraica i adoptem elements que són producte de primers, les coses es compliquen.

Proposició 2.2 Siguin p i q dos nombres primers i n el seu producte, $n = p \cdot q$. Aleshores, a $\mathbb{Z}_n$ hi ha exactament $\frac{\phi(n)}{4}$ residus quadràtics i cada un d'ells té exactament quatre arrels quadrades.

Un punt important a tenir en compte és que si un element y és residu quadràtic a $\mathbb{Z}_n$ i $n = p \cdot q$, aleshores y també és residu quadràtic a $\mathbb{Z}_p$ i a $\mathbb{Z}_q$. Aquest fet ens proporciona un sistema per calcular arrels quadrades d'un residu quadràtic y a $\mathbb{Z}_n$, ja que per calcular-les només ens caldrà calcular les arrels quadrades de y a $\mathbb{Z}_p$ i a $\mathbb{Z}_q$ i combinar-les.

Exemple 2.11 Càlcul d'arrels quadrades a $\mathbb{Z}_n$ amb $n = p \cdot q$ producte de dos primers

Calculem les arrels quadrades de 4 a $\mathbb{Z}_{15}$.

Donat que 15 és producte de dos primers, sabem que 4 té 4 arrels quadrades a $\mathbb{Z}_{15}$, que denotarem per x_1, x_2, x_3, x_4. Per calcular-les, calcularem primer les arrels quadrades de 4 a $\mathbb{Z}_3$ i a $\mathbb{Z}_5$.

Aquest cas és molt simple perquè sabem que, en els reals, les arrels de 4 són 2 i -2. Per tant, les arrels quadrades de 4 a $\mathbb{Z}_3$ seran els valors $y_1 = 2$, $y_2 = 1$ i les arrels quadrades de 4 a $\mathbb{Z}_5$ seran els valors $z_1 = 2$, $z_2 = 3$

Ara bé, sabem que les arrels quadrades de 4 a $\mathbb{Z}_{15}$ que busquem també ho han de ser a $\mathbb{Z}_3$ i a $\mathbb{Z}_5$. Això fa que puguem plantejar el següent sistema d'equacions:

[4] L'algorisme de Tonelli-Shank permet, en temps polinomial, calcular arrels quadrades d'un residu quadràtic a $\mathbb{Z}_p$, per a qualsevol valor p primer.

$$\begin{cases} x = y_i \pmod 3 \\ x = z_i \pmod 5 \end{cases}$$

Si ens hi fixem, aquí tenim un sistema d'equacions modulars que podem resoldre amb el Teorema xinès dels residus, tal i com hem explicat en l'apartat anterior. En aquest cas, com que els coeficients de la Identitat de Bézout per 3 i 5 valen 2 i -1, respectivament, ja que $1 = 2{\cdot}3 - 1{\cdot}5$, la solució del sistema amb el Teorema xinès dels residus ens queda:

$$m = 2{\cdot}3{\cdot}z_i - 1{\cdot}5{\cdot}y_i$$

Per tant, només ens queda substituir els valors de y_i i z_i per $i = \{1,2\}$ per trobar les quatre arrels quadrades de 4 a $\mathbb{Z}_{15}$ que seran $\{7,2,13,8\}$.

Hem vist que decidir si un element és un residu quadràtic a $\mathbb{Z}_p$ és fàcil en el cas que p sigui un nombre primer. Ara bé, decidir-ho a $\mathbb{Z}_n$ amb n producte de dos primers és un problema computacionalment intractable. Com també ho és el càlcul de les arrels quadrades. Si ens fixem amb l'exemple anterior, per calcular les arrels quadrades x_i hem hagut de calcular primer les y_i i les z_i per posteriorment combinar-les. Ara bé, això ho hem pogut fer perquè coneixíem la factorització del mòdul, en aquest cas sabíem que $n = 3 \cdot 5$. Ara bé, si no coneixem la descomposició del mòdul, no podrem calcular les arrels quadrades. De fet, el següent teorema mostra l'equivalència del càlcul d'arrels quadrades i la factorització del mòdul.

Teorema 2.8 Sigui $n = p \cdot q$, on p i q són primers imparells diferents. Si x i y són arrels quadrades essencialment diferents d'un element de $\mathbb{Z}_n$ aleshores $\gcd(x+y,n)$ és un dels dos factors p o q.

Dit d'una altra manera, el teorema anterior ens indica que saber calcular arrels quadrades a $\mathbb{Z}_n$ és el mateix que saber calcular la factorització del nombre n.

Exemple 2.12 Equivalència entre arrels quadrades i factorització

Suposem que volem calcular la factorització del valor $n = 925219$ però no tenim un algorisme per factortizar-lo. D'altra banda, sabem que les quatre arrels quadrades de 524422 a $\mathbb{Z}_{925219}$ valen $\{272635, 402576, 522643, 652584\}$.

Si prenem dues d'aquestes arrels quadrades que siguin essencialment diferents, per exemple, $\{272635, 402576\}$, i calculem $\gcd(272635+402576, 925219) = \gcd(675211, 925219) = 947$ podem veure que efectivament 947 és un dels primers que formen el valor 925219 ja que si els dividim $\frac{925219}{947} = 977$ la seva divisió és exacta i ens proporciona l'altre factor primer: $925219 = 947 \cdot 977$.

El concepte arrels essencialment diferents fa referència al fet que les dues arrels no poden ser l'inversa una de l'altra. És a dir, si ens hi fixem, per exemple, les arrels $\{272635, 652584\}$ compleixen que $652584 = -272635 \pmod{925219}$. Això fa que si haguéssim agafat aquestes dues arrels per fer el càlcul del màxim comú divisor no haguéssim aconseguit cap resultat ja que

$652584 + 272635 = 0 \pmod{925219}$.

Noteu que hem pogut factoritzar un valor "únicament" calculant un màxim comú divisor, una operació que és computacionalment simple fent servir l'Algorisme d'Euclides. El truc ha estat que tenim totes les arrels quadrades d'un element.

2.1.4 Aritmètica modular amb polinomis

En els apartats anteriors hem vist una manera de construir cossos finits, en concret cossos finits que tinguin un nombre primer d'elements. La pregunta que ens podríem fer ara és si podem crear cossos finits on la quantitat total d'elements no sigui un nombre primer. La resposta a aquesta pregunta és afirmativa i a continuació veurem com és possible crear cossos finits on el nombre total d'elements sigui una potència d'un nombre primer.

Quan en l'apartat anterior hem parlat d'estructures algebraiques no hem fet menció d'una altra estructura algebraica força coneguda que també és una anell. Aquesta estructura és l'anell de polinomis amb coeficients als reals, que denotarem per $(\mathbb{R}[x], +, \cdot)$. Com ja sabem, un element $a(x) \in \mathbb{R}[x]$ és un element del tipus $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_sx^s$ on $a_i \in \mathbb{R}$ i $a_s \neq 0$ (s és el grau d'$a(x)$). Amb els polinomis sabem perfectament sumar-los (sumant les components del mateix grau) i també multiplicar-los (component a component). A més, igual com teníem amb la divisió dels enters, en el cas que el grau d'un polinomi $a(x)$ sigui més gran que el grau del polinomi $b(x)$ també podem dividir el polinomi $a(x)$ entre el polinomi $b(x)$ i obtindrem dos polinomis $q(x)$ i $r(x)$ on es complirà que $a(x) = b(x) \cdot q(x) + r(x)$ i a més el grau de $r(x)$ és més petit estricte que el grau de $b(x)$.

Exemple 2.13 Operacions bàsiques amb polinomis

Donats els polinomis:
$a(x) = 3 + \frac{1}{2}x + 2x^2$
$b(x) = 1 + x$

Suma de dos polinomis:
$a(x) + b(x) = 3 + \frac{1}{2}x + 2x^2 + 1 + x = 4 + \frac{3}{2}x + 2x^2$

Producte de dos polinomis:
$a(x) \cdot b(x) = (3 + \frac{1}{2}x + 2x^2) \cdot (1 + x) = 3 + \frac{7}{2}x + \frac{5}{2}x^2 + 2x^3$

Divisió de polinomis:
$a(x) = q(x) \cdot b(x) + r(x) = (2x - \frac{3}{2}) \cdot (x + 1) + \frac{9}{2}$

Amb els anells de polinomis podem definir algorismes anàlegs als que hem vist per als enters. Així podrem calcular el màxim comú divisor de dos polinomis o els coeficients de la Identitat de Bézout.

Com és evident, el nombre d'elements de $(\mathbb{R}[x], +, \cdot)$ és infinit ja que el nombre d'elements d'$\mathbb{R}$ ho és. Ara bé, podríem intentar limitar el nombre de coeficients a utilitzar canviant el conjunt $\mathbb{R}$ pel conjunt $\mathbb{Z}_n$. Noteu que $(\mathbb{Z}_n[x],+,\cdot)$ continua tenint infinits elements perquè tot i que hem limitat el nombre de coeficients que podem triar en el polinomi (ara només poden ser enters entre el 0 i $n-1$) continuem tenint el grau del polinomi il·limitat. Per tant, ens cal que el nostre conjunt, a més de tenir el nombre de coeficients limitat, també tingui el grau del polinomi limitat.

Per limitar el grau del polinomi podem utilitzar un mecanisme anàleg al que hem fet als enters. Prenem tots els polinomis de $\mathbb{Z}_n[x]$ i els dividim per un polinomi de grau fixat k. Com que la divisió d'un polinomi de grau qualsevol per un polinomi de grau k sempre ens donarà com a residu un polinomi de grau més petit o igual que $k-1$, si prenem tots els residus d'aquesta divisio haurem aconseguit especificar tots els polinomis de $\mathbb{Z}_n[x]$ que tenen grau com a molt $k-1$, és a dir elements del tipus $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1}$ on $a_i \in \mathbb{Z}_n$. Si ens hi fixem, aquí sí que el nombre total d'elements d'aquest conjunt és un nombre finit, i concretament valdrà n^k. Aquest conjunt el denotarem com $\mathbb{Z}_n[x]/a(x)$ i seran els polinomis amb coeficients a $\mathbb{Z}_n$ mòdul el polinomi $a(x)$.

Així doncs, podem prendre el conjunt $\mathbb{Z}_n[x]/a(x)$ i definir-hi una suma i un producte estàndards de polinomis. Si volem que les operacions siguin internes, és a dir que la suma i productes d'elements de $\mathbb{Z}_n[x]/a(x)$ continuïn estan en el mateix conjunt haurem de fer el mateix que fèiem en els enters, és a dir reduir el resultat modularment.

Teorema 2.9 Donat un nombre p primer i un polinomi $m(x) \in \mathbb{Z}_p$, l'estructura algebraica $(\mathbb{Z}_p/m(x),+,\cdot)$, amb la suma i productes de polinomis modulars és un anell commutatiu amb unitat.

Exemple 2.14 Operacions a $(\mathbb{Z}_2[x]/(x^3+x+1),+,\cdot)$

Com que estem a $\mathbb{Z}_2$ i el polinomi és de grau 3 el conjunt tindrà un total de 2^3 elements que seran els següents:
$\mathbb{Z}_2[x]/(x^3+x+1) = \{0,1,x,x+1,x^2,x^2+1,x^2+x,x^2+x+1\}$

Donats els elements:
$a(x) = x^2+x+1$
$b(x) = x+1$

Suma de dos elements:
$a(x)+b(x) = x^2+x+1+x+1 = x^2+2x+2 = x^2$, ja que estem a $\mathbb{Z}_2$ i $2 = 0 \pmod 2$.

Producte de dos elements:
$a(x)\cdot b(x) = (x^2+x+1)\cdot(x+1) = x^3+2x^2+2x+1 = x^3+1 = x$, on l'última igualtat s'obté de dividir el polinomi x^3+1 entre x^3+x+1 i quedar-se amb el residu, que val x.

Exercici 2.8 Quants elements té el conjunt $\mathbb{Z}_2[x]/(x^6+x+1)$?

Exercici 2.9 Realitza els següents càlculs a $\mathbb{Z}_3[x]/(x^2+x+1)$

- $(x+1)+(2x+1)$
- $(x+1)\cdot(x+3)$
- $\frac{x+1}{2x+2}$

En el cas dels nombres enters hem vist que el concepte de nombre primer ens servia per definir estructures algebraiques a $\mathbb{Z}_p$ que fossin cossos. Per tal de poder tenir la mateixa equivalència en els polinomis ens caldrà tenir un concepte similar al de nombre primer però per als polinomis. És el concepte de polinomi irreductible.

Definició 2.15 Un polinomi $a(x) \in \mathbb{Z}_p[x]$ és **irreductible** a $\mathbb{Z}_p$ si al descomposar-lo com $a(x) = b(x)\cdot c(x)$ amb $b(x), c(x) \in \mathbb{Z}_p[x]$ aleshores $b(x)$ o $c(x)$ són constants, és a dir $b(x) \in \mathbb{Z}_p$ o $c(x) \in \mathbb{Z}_p$.

Amb aquesta definició ja estem en condició de poder definir cossos finits de mida p^k.

Teorema 2.10 Donat un nombre p primer i un polinomi $m(x) \in \mathbb{Z}_p$ irreductible a $\mathbb{Z}_p$ i de grau k, aleshores l'estructura algebraica $(\mathbb{Z}_p[x]/m(x), +, \cdot)$, amb la suma i productes de polinomis modulars és un cos finit amb p^k elements.

Fixem-nos que si l'estructura algebraica $(\mathbb{Z}_p[x]/m(x), +, \cdot)$ és un cos, podem calcular l'invers de qualsevol dels seus elements. Per fer-ho, simplement utilitzarem les mateixes tècniques que hem descrit per als enters, però en aquest cas operant amb polinomis.

Exemple 2.15 Càlcul d'inversos amb polinomis

Suposem l'estructura algebraica $(\mathbb{Z}_2[x]/(x^3+x+1), +, \cdot)$. Com que x^3+x+1 és irreductible a $\mathbb{Z}_2$, aquesta estructura algebraica és un cos. Calculem l'invers de l'element $a(x) = x^2+x+1$, és a dir, hem de trobar el polinomi $b(x) \in \mathbb{Z}_2[x]/(x^3+x+1)$ tal que $a(x)\cdot b(x) = 1$. Per fer-ho, hem de calcular els elements de la Identitat de Bézout entre $a(x)$ i x^3+x+1 que és el mòdul on estem treballant.

Si calculem les divisions successives de l'Algorisme d'Euclides per trobar el màxim comú divisor obtenim:
$x^3+x+1 = (x^2+x+1)\cdot(x+1)+x$
$x^2+x+1 = (x)\cdot(x+1)+1$
tenim que el $\gcd(x^3+x+1, x^2+x+1) = 1$, com ja sabíem. Ara si aïllem els dos residus de cada equació:
$1 = x^2+x+1-(x)\cdot(x+1)$
$x = x^3+x+1-(x^2+x+1)\cdot(x+1)$

A partir d'aquestes equacions, podem calcular els coeficients de la Identitat de Bézout, igual que hem fet en els enters. Fixeu-vos, que com que els coeficients del polinomi són elements de $\mathbb{Z}_2$ no tenim en compte el signe ja que $-1 = 1 \pmod 2$. Així obtenim que:
$1 = (x^3+x+1)\cdot(x+1)+(x^2+x+1)\cdot(x^2)$

Si prenem aquesta equació mòdul x^3+x+1 tindrem que el primer terme val 0 i per tant ens queda:
$1 = (x^2+x+1)\cdot(x^2)$
és a dir, l'invers de l'element x^2+x+1 que buscàvem és el polinomi x^2. Ho podem comprovar fent el producte de x^2+x+1 per x^2 i veient que si el dividim pel mòdul, x^3+x+1, el residu ens dona 1.

2.2 Nombres primers

Els nombres primers han estat estudiats abastament ja que són la base de tots els nombres, donat que qualsevol nombre enter es pot descompondre de manera única com a producte de primers. Tot i això, hi ha moltes propietats d'aquest tipus de nombres que encara no s'han pogut sistematitzar i grans qüestions obertes de la matemàtica giren al voltant dels nombres primers.

Com ja hem indicat anteriorment en la Definició 2.9, un **nombre primer** és aquell enter positiu >1 que només es pot dividir per ell mateix i per la unitat.

Primers de Mersenne

Els primers del tipus 2^p-1 s'anomenen primers de Mersenne. Aquests tipus de nombres són primers només en el cas que p sigui primer, però no es cert que qualsevol valor p primer generi un nombre 2^p-1 primer. Per exemple, per $p=2$ sí que es compleix ja que $2^2-1=3$ que és primer. Però per $p=11$ no és cert, perquè $2^{11}-1=2047$ que no és primer.

Els nombres primers s'estudien des de l'antiguitat i ja Euclides, 300 anys a.C. va demostrar que hi havia infinits nombres primers. Tot i això, i malgrat els diferents estudis sobre nombres primers, encara no s'ha pogut establir una fórmula que permeti donar la seqüència de nombres primers, i per tant, la única forma de trobar-los és anar generant nombres i comprovar si són primers o no ho són. En aquest sentit, el primer més gran que s'ha trobat (al gener del 2016) és el nombre $2^{74,207,281}-1$ que és un primer de Mersenne de més de 22 milions de xifres.

Tot i que no es coneix quina és la seqüència de nombres primers, sí que hi ha alguns resultats que permeten tenir estimacions sobre el nombre de primers que hi ha en un interval. Per exemple, el teorema dels nombres primers ens en dóna una aproximació.

Teorema 2.11 — Teorema dels nombres primers. Sigui $\pi(n)$ el nombre de primers més petits que un valor n, aleshores, es compleix que:

$$\lim_{n\to\infty} \frac{\pi(n)}{\frac{n}{ln(n)}} = 1$$

És a dir, podem aproximar el nombre de primers més petits que n calculant-ne el seu logaritme. En la següent taula es pot veure la diferència entre aquesta aproximació i el nombre real de primers que existeixen per a valors petits on s'han pogut calcular tots els nombres primers i comptar-los.

n	$\pi(n)$	$\frac{n}{ln(n)}$
10	4	4.34
100	25	21.7
1000	168	144.8
10^6	78498	72382
10^9	50847478	48254942

Si ens fixem en la taula, tot i que la diferència entre l'aproximació i el nombre real de primers sembla que cada vegada sigui més distant, si calculem el quocient dels dos valors podem veure que efectivament cada vegada és més proper a 1.

A l'hora de trobar nombres primers, el teorema que acabem d'enunciar ens dóna una estimació de la probabilitat que, donat un nombre aleatori qualsevol, aquest sigui primer. Efectivament, utilitzant l'aproximació que ens proporciona aquest teorema, sabem que trobar un nombre primer entre n i $2n$ té una probabilitat de $p = \frac{2}{ln(2n)} - \frac{1}{ln(n)}$ ja que el nombre total de primers en l'interval serà $\frac{2n}{ln(2n)} - \frac{n}{ln(n)}$ i això caldrà dividir-ho pel nombre d'elements de l'interval, és a dir n. Per posar aquests valors en context, si fem els càlculs, veurem que la probabilitat que donat un valor triat aleatòriament entre $1 \cdot 10^{20}$ i $2 \cdot 10^{20}$ sigui primer és d'un 2%. Per tant, si el procés per aconseguir un primer passa per seleccionar un valor a l'atzar, mirar si és primer i sinó buscar-ne un altre, clarament el cost de mirar si un nombre és primer cal que sigui computacionalment reduït si volem ser eficients en la generació de nombres primers.

2.2.1 Tests de primalitat

Com veurem al llarg del llibre, per a diferents criptosistemes i protocols criptogràfics, és necessari poder disposar de nombres primers molt grans. A l'hora de generar-los, donat que no tenim una fórmula que ens en doni la seqüència, el procés que es realitza consisteix a seleccionar un nombre aleatori molt gran i verificar-ne si és primer o no.

Per tal de comprovar si un nombre és primer s'utilitzen els test de primalitat, que no són més que algorismes que reben com a entrada un nombre i proporcionen com a sortida informació sobre la condició de primer del nombre donat. De test de primalitat n'hi ha de dos tipus: deterministes i probabilístics.

Un **test de primalitat determinista** és aquell que donat un nombre natural ens indica, de manera inequívoca, si és o no un nombre primer.

Una manera de determinar la primalitat d'un nombre seria fent-ne la descomposició en nombres primers. Si aquesta descomposició retorna més d'un factor diferent al propi nombre la conclusió és que el nombre no és primer i, en cas que els factors retornats siguin el propi nombre i l'1 es determinarà que el nombre sí que és primer.

Per exemple, podem utilitzar l'algorisme de factorització per prova de divisions, com el que es mostra en el següent codi en SAGE[5], que ens retornarà un sol valor (el nombre proporcionat) en

[5] El SAGE és un programari matemàtic molt potent de codi obert basat en Python. Podeu descarregar-vos-el de `http://www.sagemath.org/`. Tots els fragments de codi que es mostren en aquest capítol són codificats en SAGE.

cas que aquest sigui primer.

```
def trial_division(n):
    if n < 2:
        return []
    prime_factors = []
    for p in primes_first_n(int(n**0.5)):
        if p*p > n: break
        while n % p == 0:
            prime_factors.append(p)
            n //= p
    if n > 1:
        prime_factors.append(n)
    return prime_factors
```

Noteu que el bucle extern només prova primers p fins a $\sqrt{n}$. No cal provar divisors més grans que $\sqrt{n}$, perquè si n té un divisor més gran que $\sqrt{n}$, aleshores l'altre divisor serà més petit que $\sqrt{n}$.

Evidentment aquest algorisme no és gens eficient, i per tant, per a verificar la primalitat de nombres molt grans és totalment desaconsellable. En l'actualitat, l'algorisme més eficient que proporciona un test de primalitat determinista és el proposat pels matemàtics indis Agrawal, Kayal i Saxena l'any 2002. Malgrat ser el test determinista més eficient, no s'utilitza a la pràctica ja que per a valors elevats els temps de resposta són massa grans.

Per tal d'obtenir test de primalitat amb una complexitat suficientment baixa per a poder generar nombres primers prou grans en un interval de temps prou petit cal recórrer als tests de primalitat probabilístics.

Un **test de primalitat probabilístic** és aquell que donat un nombre natural ens indica si és primer amb una certa probabilitat.

Així doncs, un test de primalitat probabilístic ens pot indicar que un nombre és primer sense que realment ho sigui. De fet un test de primalitat probabilístic ens retornara el que es coneix com a pseudo-primer. El gran avantatge dels tests probabilístics és que són extremadament eficients i, a més, es pot determinar el valor de la probabilitat amb el que es poden equivocar i fer-lo tant petit com es vulgui.

Test de primalitat de Fermat

El test de primalitat de Fermat és un test de primalitat probabilístic basat en el teorema petit de Fermat. El teorema petit de Fermat és un cas particular del Teorema d'Euler que hem vist en apartats anteriors.

Teorema 2.12 — Teorema petit de Fermat. Sigui p un nombre primer, aleshores $a^{p-1} = 1 \pmod{p}$ per a qualsevol valor a tal que $1 \leq a < p$.

En base a aquest teorema, podem definir el test de primalitat de Fermat de la següent manera:

```
def Fermat_test(n,k):
    if n <= 1:
        return str(n) + ' no és primer'
    if n <= 3:
        return str(n) + ' és primer'
    for i in range(k):
        a = randint(2,n-2)
        if (a^(n-1))%n != 1:
            return str(n) + ' no és primer'
    p = numerical_approx(1-(1/2)^k)
    return str(n) + ' és primer amb probabilitat ' + str(p)
```

Nombres de Carmichael

Els nombres de Carmichael són nombres extremadament rars. Un nombre de Carmichael n, tot i no ser primer, compleix la congruència de Fermat per a tots els valors a, tals que $1 \leq a < p$ i $\gcd(n,a) = 1$. Per aquest motiu, el test de primalitat Fermat aplicat al nombre de Carmichael $n = 340561$ (que no és primer, ja que $340561 = 13 \cdot 17 \cdot 23 \cdot 67$) ens pot arribar a donar que és un nombre primer amb probabilitat 0,999.

Fixeu-vos que la idea és prendre el nombre que volem analitzar i assignar-lo com al mòdul de l'equació. Pel teorema petit de Fermat sabem que si el nombre és primer, l'equació modular sempre ens donarà 1. Ara bé, si el nombre no és primer l'equació modular pot donar 1 o pot donar un valor diferent de 1. A més, si n no és primer, en general, la meitat dels valors a més petits que n complirà l'equació i l'altra meitat no. Això ens porta a assegurar que si anem triant valors a diferents la probabilitat que l'equació sigui certa sense que n sigui primer és cada vegada més petita. En particular, la probabilitat es redueix a la meitat. Per aquest motiu, si repetim el test de Fermat k vegades i ens indica que el valor n és primer, la probabilitat que aquest ho sigui serà $1 - (\frac{1}{2})^k$.

Test de primalitat de Miller-Rabin

El test de primalitat de Miller-Rabin és un test que combina la condició del teorema petit de Fermat amb la particularitat dels residus quadràtics en aritmètica modular. Tal i com hem vist en l'Apartat 2.1.3, en el cas que n és primer l'equació $x^2 = 1 \pmod{n}$ té únicament dues solucions, mentre que si n no és primer, en té quatre. Així, aquest fet, juntament amb el teorema petit de fermat es poden unir creant el test de primalitat de Miller-Rabin implementat en el següent algorisme:

```
def Miller_Rabin_test(n,k):
    tmp = n-1
    s = 0
    while tmp%2 == 0:
        tmp = tmp // 2
        s= s +1
    r = (n-1) / (2^s)
    for i in range(k):
        a = randint(2,n-2)
        y = a^r%n
        if (y != 1) and (y != n-1):
            j = 1
            while (j >= (s-1)) and (y != (n-1)):
                y = (y^2)%n
                if y==1:
                    return str(n) + ' no és primer'
                j = j+1
            if y != n-1:
```

```
            return str(n) + ' no és primer'
    p = numerical_approx(1-(1/4)^k)
    return str(n) + ' és primer amb probabilitat ' + str(p)
```

L'avantatge d'aquest test respecte el de Fermat és que no està afectat pels nombres de Carmichael. A més, a cada iteració la probabilitat d'errar disminueix en 1/4 en comptes d'1/2 aconseguint una probabilitat més alta d'encertar un primer amb menys iteracions que el test de Fermat. En l'actualitat, per la seva eficiència, aquest test, o alguna de les seves variants, és un dels més utilitzats en les aplicacions criptogràfiques.

2.3 Problemes matemàtics difícils

Com veurem al llarg d'aquest llibre, la seguretat dels algorismes criptogràfics que es fan servir avui en dia recau en la dificultat que un atacant pugui realitzar els càlculs necessaris per trencar el criptosistema. Així, tal i com hem definit anteriorment, la seguretat de la majoria dels criptosistemes moderns és una seguretat computacional i no pas una seguretat teòrica.

Per tal de poder definir problemes que siguin difícils de resoldre per un atacant, ens caldrà primer definir què vol dir que un problema sigui difícil des d'un punt de vista computacional. A continuació, repassarem diferents funcions matemàtiques que presenten certa unidireccionalitat en el sentit que el seu càlcul en una direcció és molt simple però el càlcul de la seva inversa és molt complicat, fet que s'utilitza en el disseny de criptosistemes i protocols criptogràfics.

2.3.1 Complexitat d'un algorisme

La teoria de la complexitat algorísmica és molt complexa en si mateixa, de manera que en aquest apartat només en donarem unes nocions molt bàsiques.

La complexitat de càlcul d'un algorisme es mesura pel temps T que requereix la seva execució i s'expressa com a funció de la mida n de l'entrada de l'algorisme. Més que fer servir complexitats exactes que s'expressarien com a $f(n)$, se solen emprar ordres de magnitud, és a dir $\mathscr{O}(g(n))$, de tal manera que $f(n) = \mathscr{O}(g(n))$ vol dir que hi ha constants c i n_0 tals que:

$$f(n) \leq c|g(n)| \text{ ; per a } n \geq n_0$$

El propòsit que hi ha en fer servir ordres de magnitud és que l'explicitació de $g(n)$ sigui més simple que $f(n)$.

Exemple 2.16 Ordre de la complexitat

Si la complexitat exacta d'un algorisme és $f(n) = 36n + 10$, podem escriure que $f(n) = \mathscr{O}(n)$, ja que

$$36n + 10 \leq 37n \text{ ; per a } n \geq 10$$

Per la mateixa raó, si $f(n)$ és un polinomi de grau t en n, podem escriure $f(n) = \mathscr{O}(n^t)$.

Un **algorisme polinòmic** és aquell que el seu temps d'execució és $T = \mathscr{O}(n^t)$ per a alguna constant t. En el cas que $t = 0$ direm que l'algorisme és constant, lineal si $t = 1$, quadràtic si $t = 2$, etc.

Un **algorisme exponencial** és aquell que el seu temps d'execució és $T = \mathscr{O}(t^{h(n)})$ per a alguna constant t i un polinomi $h(n)$.

Per a valors d'n grans, les diferents classes de complexitat impliquen temps molts diferents d'execució. Per exemple, si suposem una màquina capaç d'executar 10^{12} instruccions per segon, la taula següent ens mostra els temps d'execució per a les classes d'algorismes que acabem de definir.

Classe	Complexitat	Operacions per $n = 10^{12}$	Temps
Constant	$\mathscr{O}(1)$	1	10^{-12} segons
Lineal	$\mathscr{O}(n)$	10^{12}	1 segon
Quadràtic	$\mathscr{O}(n^2)$	10^{24}	31.709 anys
Cúbic	$\mathscr{O}(n^3)$	10^{36}	$3,17 \cdot 10^{16}$ anys
Exponencial	$\mathscr{O}(2^n)$	$10^{3 \cdot 10^{11}}$	$10^{2,999 \cdot 10^{11}}$ milions d'anys.

2.3.2 Producte de primers i factorització d'enters

Un dels problemes matemàtics difícils és la factorització d'enters. Si tenim dos nombres primers p i q és molt fàcil calcular els seu producte $n = p \cdot q$. Això és així independentment de la mida dels valors p i q perquè els algorismes que realitzen la multiplicació d'enters són algorismes molt eficients i per tant, la mida dels nombres afecta molt poc al temps de resolució del producte.

Ara bé, donat un valor n que sabem que és producte de dos primers, és molt difícil trobar quins són aquests dos primers, és a dir, factoritzar-los, en el cas que n sigui prou gran.

Podeu veure la diferència entre aquestes dues operacions executant les següents comandes de SAGE.

```
p=next_prime(2^100)
q=next_prime(2^101)
print 'Temps per multiplicar', p, ' per ', q,':'
time n=p*q
print 'Temps per factoritzar', n, ':'
time factor(n)
```

D'algorismes per a la factorització d'enters n'hi ha de diferents tipus i la seva complexitat depèn de la forma dels primers que generen el valor a factoritzar. Si ens centrem en algorismes de factorització genèrics, per a valors de p i q propers i grans però sense cap caracterització concreta, el millor algorisme de factorització que es coneix s'anomena garbell sobre el cos de nombres generalitzat, en anglès *general number field sieve (GNFS)* i té una complexitat $\mathscr{O}(e^{(\sqrt[3]{\frac{64}{9}}+\mathscr{O}(1))(\ln(n))^{\frac{1}{3}}(\ln(\ln(n)))^{\frac{2}{3}}})$

Com veurem en els propers capítols, la dificultat en la factorització d'enters de mida molt gran és la base de la seguretat del criptosistema RSA.

2.3.3 Exponenciació i logaritme discret

Els enters modulars, $\mathbb{Z}_p$ amb p primer, són un grup multiplicatiu cíclic. Això vol dir que podem trobar un element g que s'anomena generador, tal que les seves potències generen tots els elements de $\mathbb{Z}_p$, llevat del zero. Per exemple, si considerem $\mathbb{Z}_7 = \{0,1,2,3,4,5,6\}$ veiem que el 3 és un generador perquè $\mathbb{Z}_7 = \{0, 1 = 3^6, 2 = 3^2, 3 = 3^1, 4 = 3^4, 5 = 3^5, 6 = 3^3\}$. A més, la distribució de les potències del generador entre els elements de $\mathbb{Z}_p$ és una distribució uniforme.

En aquest entorn és on trobem un altre dels problemes matemàtics que s'utilitzen en cripotografia. De nou, tenim una operació molt fàcil de realitzar: donats dos elements $x, y \in \mathbb{Z}_p$ calcular la seva potència, és a dir $z = x^y \pmod p$. Aquesta operació és molt eficient de realitzar perquè l'únic càlcul que realitzem són productes, amb una divisió per reduir el resultat al mòdul desitjat. Ara bé, l'operació inversa, la que donats dos elements $x, z \in \mathbb{Z}_p$ permet trobar l'element y tal que $z = x^y \pmod p$ és un problema pel qual no se'n coneix cap algorisme eficient. Fixeu-vos que aquest càlcul és l'equivalent a calcular el logaritme de z en base x, però en els enters mòdul p. Per això, aquest problema se'l coneix com a problema del logaritme discret.

El problema del logaritme discret és la base de la seguretat de diferents esquemes i protocols criptogràfics, els més coneguts dels quals són l'intercanvi de claus de Diffie i Hellman i el criptosistema d'ElGamal.

2.3.4 Quadrats i arrels quadrades modulars

Un altre problema problema matemàtic que s'utilitza en criptografia és el càlcul d'arrels quadrades en un anell multiplicatiu sobre $\mathbb{Z}_n$ quan el valor n és un producte de dos primers p i q.

Com ja hem comentat anteriorment, multiplicar dos nombres és molt ràpid, de manera que calcular el quadrat d'un nombre ha de ser forçosament també molt ràpid, perquè és el producte d'un nombre per ell mateix. A més, si una vegada hem fet el producte en volem calcular el seu equivalent mòdul n, això també és molt ràpid, perquè només cal que dividim el resultat pel mòdul i ens quedem amb el residu. Per tant, calcular quadrats a $\mathbb{Z}_n$ és pot fer de manera molt eficient. Ara bé, l'operació inversa, és a dir, donat un element $x \in \mathbb{Z}_n$ trobar-ne l'arrel quadrada (l'element y tal que $x = y^2 \pmod n$), és una operació molt costosa. De fet, tal i com s'enuncia en el Teorema 2.8, aquest problema és equivalent a factoritzar ja que per calcular les arrels d'un valor a $\mathbb{Z}_n$, on $n = pq$, ja hem vist que ens caldrà trobar les arrels quadrades a $\mathbb{Z}_p$ i a $\mathbb{Z}_q$, i per tant, ens cal factoritzar n.

2.4 Resum

En aquest capítol hem presentat els conceptes matemàtics bàsics utilitzats en criptografia, centrats en l'aritmètica modular. Conceptes com la divisibilitat de nombres enters, el màxim comú divisor o el Teorema d'Euler són cabdals per poder comprendre les operacions criptogràfiques que realitzen els algorismes de xifrat. Així, l'aritmètica modular és la base que ens permetrà entendre el funcionament de la majoria dels criptosistemes utilitzats en l'actualitat, tant els de criptografia de clau simètirca, com l'AES, com els de clau pública com l'RSA o ElGamal.

Ja centrats en la criptografia de clau pública, és important tenir clares les característiques dels nombres primers, ja que aquests acostumen a ser la matèria prima en la que es basen els criptosistemes. Aquests valors sovint formen part de les claus o dels paràmetres dels esquemes i per tant conèixer-ne la seva distribució i comprendre com es poden obtenir és vital per poder implementar qualsevol criptosistema de clau pública.

Finalment, i continuant amb la criptografia de clau pública, hem analitzat diferents problemes matemàtics que tenen una asimetria en la seva resolució i que es fan servir en criptosistemes de clau pública. Com hem vist, aquest problemes presenten una simplicitat d'execució quan els mirem en un sentit però són d'una dificultat extrema quan els volem realitzar en el sentit invers. Per exemple, multiplicar dos primers grans p i q pot ser immediat mentre que trobar-ne els que composen un nombre n pot implicar càlculs de milers d'anys. És important conèixer quins són aquests problemes matemàtics per tal d'entendre quines són les bases de seguretat dels criptosistemes que els fan servir.

2.5 Solucions dels exercicis

Exercici 2.1:

$\gcd(35,48) = 1$ ja que si calculem les divisions successives tenim: $48 = 35 \cdot 1 + 13$
$35 = 13 \cdot 2 + 9$
$13 = 9 \cdot 1 + 4$
$9 = 4 \cdot 2 + 1$
$4 = 4 \cdot 1 + 0$
i l'últim residu no nul és l'1.

Exercici ??:

Per calcular el coeficients de la indentitat de Bézout utilitzarem les igualtats de l'Algorisme d'Euclides de l'exercici anterior i n'aïllarem el residu:
$48 - 35 = 13$
$35 - (13 \cdot 2) = 9$
$13 - 9 = 4$
$9 - (4 \cdot 2) = 1$
i substituirem en cada equació el valor corresponent per acabar obtenint-ne una sola amb els valors 35 i 48:
$1 = 9 - (4 \cdot 2) = 9 - ((13 - 9) \cdot 2) = (3 \cdot 9) - (13 \cdot 2) = (3(35 - (13 \cdot 2)) - (13 \cdot 2) = (35 \cdot 3) - (8 \cdot 13) = (3 \cdot 35) - (8(48 - 35))) = (11 \cdot 35) - (8 \cdot 48)$
Així, tenim que
$1 = 11 \cdot 35 + (-8) \cdot 48$
i per tant els coeficients de la Identitat de Bézout per a 35 i 48 són 11 i -8 respectivament.

Exercici 2.3:

$\phi(527) = \phi(17 \cdot 31) = \phi(17) \cdot \phi(31) = (17 - 1) \cdot (31 - 1) = 16 \cdot 30 = 480.$

Exercici 2.4:

El conjunt $\mathbb{Z}_{25}$ està format per tots els residus de dividir per 25, per tant tindrà 25 elements que són $\mathbb{Z}_{25} = \{0,1,2,3,4,\cdots,22,23,24\}$

Exercici 2.7:

L'estructura algebraica $(\mathbb{Z}_{37},+,\cdot)$ és un cos ja que el nombre 37 és primer. Això fa que tots els elements més petits que 37, és a dir tots els elements inclosos en $\mathbb{Z}_{37}$, siguin coprimers amb 37 i per tant tinguin invers, condició necessària i suficient perquè $(\mathbb{Z}_{37},+,\cdot)$ sigui un cos.

Exercici 2.5:

L'invers de 7 a $\mathbb{Z}_{37}$ val 16 ja que $7 \cdot 16 = 112 = 1 \pmod{37}$. El valor 16 el podem calcular de diferents maneres. Per exemple, calculant $7^{\phi(37)-1} = 7^{35} = 16 \pmod{37}$ o bé calculant els coeficients de la Identitat de Bézout de 7 i 37, és a dir $7 \cdot 16 + 37 \cdot (-3)$.

Exercici 2.6:

Realitza els següents càlculs a $\mathbb{Z}_{37}$

- $20 + 20 = 40 = 3 \pmod{37}$
- $20 \cdot 4 = 80 = 6 \pmod{37}$

- $20^2 = 400 = 30 \pmod{37}$
- $\frac{20}{7} = 20 \cdot 16 = 320 = 24 \pmod{37}$, ja que 16 és l'invers de 7 tal i com hem calculat anteriorment.

Exercici 2.8:

El conjunt $\mathbb{Z}_2/(x^6+x+1)$ té un total de $2^6 = 64$ elements ja que són tots els polinomis de grau 5 amb coeficients binaris.

Exercici 2.9:

Realitza els següents càlculs a $\mathbb{Z}_3/(x^2+x+1)$

- $(x+1)+(2x+1) = 2 \pmod{(x^2+x+1)}$
- $(x+1)\cdot(x+3) = x^2+x = 2 \pmod{(x^2+x+1)}$
- $\frac{x+1}{2x+2} = (x+1)\cdot x = 2 \pmod{(x^2+x+1)}$, ja que $x^2+x+1 = 0$ per ser el mòdul i per tant: $x^2+x = 2$ ja que els coeficients del polinomi són de $\mathbb{Z}_3$.

2.6 Bibliografia

Shanks, D. (1973). *Five number-theoretic algorithms.* In Proceedings of the Second Manitoba Conference on Numerical Mathematics (Winnipeg).

II Criptografia de clau simètrica

3. Les xifres de flux

Ja hem vist en anteriors capítols que un criptosistema incondicionalment segur necessita tants bits de clau com bits de text a xifrar. El xifratge de Vernam és el criptosistema que aconsegueix aquesta seguretat incondicional, però el preu que en paga és la ineficiència del xifratge. Aquesta ineficiència recau justament en el fet que la clau ha de tenir la mateixa longitud del text a xifrar. Això comporta que la longitud de les claus sigui molt gran i, per tant, sigui més difícil guardar-les en secret. A més, es dóna la paradoxa que si tenim un canal segur per intercanviar les claus, aleshores també podem utilitzar-lo per intercanviar els missatges, ja que tenen la mateixa longitud.

Les **xifres de flux** sorgeixen com una aproximació optimitzada al xifratge de Vernam. La idea és construir una clau suficientment llarga, com a mínim de la longitud del missatge, a partir d'una clau inicial curta. Això s'aconsegueix utilitzant el que s'anomena un generador pseudoaleatori. Aquest generador expandeix una clau petita, anomenada llavor, obtenint-ne una de molt més llarga. L'operació d'expansió cal que tingui certes característiques ja que la seqüència que en resultarà és la que s'utilitzarà per xifrar el text en clar. Caldrà doncs, veure quines propietats hauran de complir les esmentades seqüències i estudiar quin tipus de generadors hi ha per obtenir-les.

Les xifres de flux són xifres de clau compartida ja que la llavor (que es fa servir per obtenir la seqüència xifrant) és utilitzada tant per a xifrar com per a desxifrar, i és per tant compartida entre l'emissor i el receptor.

Una alternativa al xifratge de flux és el que s'anomena xifratge de bloc. Aquest xifratge s'inclou també dins dels critosistemes de clau compartida ja que la clau que s'utilitza per a xifrar i desxifrar és la mateixa i la comparteixen emissor i receptor. La diferència bàsica entre el xifratge en flux i el xifratge de bloc és la utilització de memòria en els algorismes de xifratge. En el Capítol 4 es tracten amb més detall les xifres de bloc.

Com veurem en aquest capítol, el xifrat de flux utilitza una clau diferent per cada bit d'informació. Aquesta clau depèn de l'estat inicial del generador, però també de l'estat del generador en el moment de xifrar un bit concret. Per tant, dos bits iguals es poden xifrar de maneres diferents depenent de l'estat en què es trobi el generador. En el proper capítol veurem que en el xifratge en bloc això no passa. Les xifres en bloc actuen sense memòria, i per tant el text xifrat només depèn del text en clar i de la clau.

3.1 Criptografia de clau simètrica o compartida

En aquest capítol introduirem les xifres de clau compartida.

Les **xifres de clau simètrica o compartida** són aquelles en els quals l'emissor i el receptor comparteixen una mateixa clau, que és la que utilitzen per xifrar i desxifrar missatges.

És a dir, en les xifres de clau compartida, la clau que s'utilitza per xifrar és la mateixa que es fa servir per desxifrar i per tant, en qualsevol moment, l'emissor pot passar a fer de receptor i a l'inrevés, utilitzant sempre les mateixes claus.

Per les seves característiques, les xifres de clau compartida no poden oferir la propietat de no-repudi. Com veurem més endavant, existeixen altres construccions que sí que ens permetran oferir aquesta propietat.

Els dos tipus de xifres de clau simètrica més utilitzats són les xifres de flux i les de bloc. La diferència entre els dos tipus de xifres radica en com es processa la informació: a les xifres de flux la informació es xifra bit a bit, és a dir, els bits es xifren de manera invididual, mentre que els criptosistemes de bloc xifren un bloc sencer de n bits alhora.

3.2 Definició de les xifres de flux

D'una manera esquemàtica, un **criptosistema de flux** es pot expressar tal com mostra la Figura 3.1.

Tant l'emissor com el receptor disposen d'una mateixa clau c (anomenada llavor del generador), i d'un mateix algorisme determinista Alg, anomenat **generador pseudoaleatori**. Al proporcionar la clau k com a entrada a l'algorisme, aquest dóna com a sortida una seqüència s que s'anomena **seqüència xifrant**.

Per tal de xifrar el missatge, l'emissor va sumant cada bit del missatge m amb cada bit de la seqüència xifrant s, obtenint el missatge xifrat y. Quan el receptor rep el missatge xifrat y, utilitza el mateix algorisme determinista Alg i la clau k, que comparteix amb l'emissor, per tal d'obtenir la mateixa seqüència xifrant. Així sumant bit a bit el missatge que li arriba y, amb la seqüència resultant de l'algorisme s, obté el text en clar m enviat per l'emissor. Al llarg de tot aquest capítol les seqüències amb les que treballarem seran binàries i les operacions a les que ens referirem seran totes mòdul 2.[1]

Per tal que aquest criptosistema sigui segur, és bàsic que la seqüència xifrant no sigui coneguda, és a dir que en cap moment es pugui saber quin serà el següent bit de sortida. Idealment, el que es necessita per a la seguretat incondicional és que la clau, en aquest cas la seqüència xifrant, sigui

[1] De forma equivalent, podem pensar la suma mòdul 2 com una XOR.

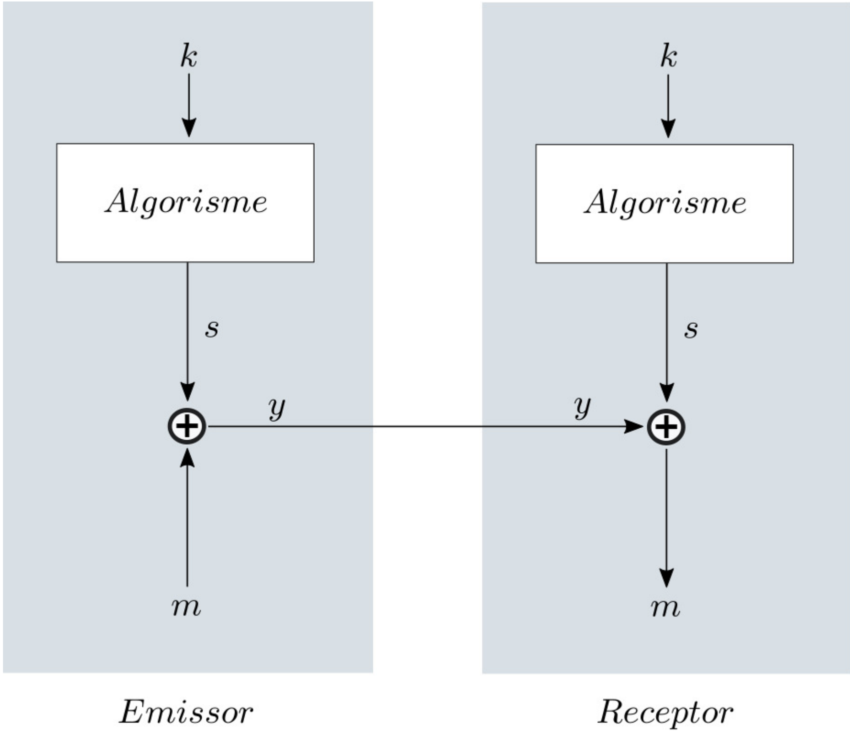


Figura 3.1: Esquema general de flux

completament aleatòria. En el nostre esquema no es pot donar aquesta condició ja que el generador que utilitzem ha de ser determinista per tal que emissor i receptor obtinguin la mateixa seqüència quan donen com a entrada la mateixa clau secreta. Així doncs, la seqüència xifrant tindrà propietats molt properes a les que té una seqüència completament aleatòria i per tant s'anomenarà **seqüència pseudoaleatòria**.

Concretament, si una seqüència no és aleatòria, vol dir que a partir d'un cert moment es repeteix. Aquesta subseqüència que es va repetint és el que s'anomena període. El que és important, doncs, és que aquesta subseqüència, el període, sigui indistingible d'una seqüència completament aleatòria d'igual longitud.

Per això aquesta seqüència ha de complir certes propietats que veurem en els propers apartats.

No hem d'oblidar que els criptosistemes de clau compartida basen la seva seguretat en el fet que la clau utilitzada per xifrar i desxifrar només és coneguda per l'emissor i el receptor. En el xifratge en flux, si bé la clau no s'utilitza directament per xifrar, cal igualment que no es faci pública ja que l'algorisme determinista es conegut i per tant es podria obtenir la seqüència xifrant a partir d'ell i la clau.

Si ens fixem en l'esquema de xifratge en flux de la Figura 3.1 veiem que per tal d'obtenir el text xifrat que enviem al receptor hem d'anar sumant el text en clar amb la seqüència xifrant que resulta del generador pseudoaleatori. Això vol dir que la velocitat de transmissió de les dades entre l'emissor i el receptor ve determinada pel mínim entre la velocitat de generació del missatge, m, i la velocitat de generació de la seqüència xifrant, s. Així doncs, cal tenir en compte aquest fet quan estudiem els possibles generadors pseudoaleatoris ja que en funció de la seva implementació (ja sigui en hardware o en software), obtindrem una velocitat o una altra. Cal que l'algorisme que ens generi la seqüència sigui de fàcil implementació, tant des del punt de vista de complexitat com pel que fa al vessant econòmic.

No oblidem el món real

Els telèfons mòbils amb tecnologia GSM incorporen un xifrador de flux. Seria impensable que el cost econòmic del xifrador incrementés el preu del telèfon mòbil. Tampoc no seria admissible que la velocitat de la comunicació es veiés afectada per la velocitat de l'esmentat xifrador.

3.2.1 Període

Hem vist que per tal d'implementar un criptosistema de xifratge de flux necessitem un algorisme que ens doni com a sortida la seqüència xifrant. Com ja hem dit anteriorment, el fet que aquest algorisme sigui determinista fa que la seqüència que en resulta no sigui completament aleatòria i per tant implica que a partir d'un cert moment es repeteix. Aquesta subseqüència que es va repetint és el que s'anomena **període**. Formalment, sigui $\{s_i\}_{i\geq 0}$ una seqüència periòdica, el període p és l'enter més petit tal que $s_{i+p} = s_i$ per a tot $i \geq 0$.

Atès que el període es repeteix, una vegada es coneix ja es pot determinar exactament tota la seqüència xifrant i per tant el criptosistema es pot trencar. Per això les seqüències que s'utilitzen per al xifratge en flux cal que tinguin un període molt gran, ja que d'aquesta manera triguen molt a repetir-se i, per tant, és més difícil predir-ne la sortida.

Període gran

El concepte de període gran és relatiu al xifrador i a l'aplicació. Un període de 2^{32} pot no ser prou llarg per a un xifrador que xifri a 1 Mbyte/seg. ja que a aquesta velocitat el període es repeteix només cada 8.5 minuts.

3.2.2 Aleatorietat

Com hem comentat, les xifres de flux fan servir generadors pseudoaleatoris.

Un **generador pseudoaleatori** (o PRNG, de l'anglès, *Pseudo Random Number Generator*) és un algorisme **determinista** que genera una seqüència a partir d'una entrada que anomenem **llavor**. La seqüència generada per un PRNG intenta reproduir les propietats que tindria una seqüència generada de manera aleatòria.

Determinisme en els PRNG

Noteu que els PRNG són algorismes deterministes, és a dir, donat un PRNG i una llavor, la seqüència generada serà sempre la mateixa.

Els **generadors pseudoaleatoris criptogràficament segurs** (o CSPRNG, de l'anglès, *Cryptographycally Secure Pseudo Random Number Generator*) són un tipus especial de PRNG que generen seqüències no predictibles. En concret, per a que un PRNG sigui considerat un CSPRNG, cal que les seqüències que genera tinguin dues propietats. A partir de k bits de la seqüència generada, $s_{i+1}, s_{i+2}, \cdots, s_{i+k}$:

1. No existeix un algorisme en temps polinomial que pugui predir el següent bit de la seqüència, s_{i+k+1}, amb probabilitat major al 50% i
2. No és computacionalment possible predir el bit anterior de la seqüència, s_i.

PRNG i CSPRNG

Tots els CSPRNG són PRNGs però l'afirmació contrària no és certa, és a dir, un generador pseudoaleatori no té perquè ser criptogràficament segur.

El concepte de complexitat lineal ens mesura el grau d'impredictibilitat d'una seqüència. En concret, la complexitat lineal ens diu quina part de la seqüència ens cal conèixer per tal de poder-la predir tota. Per tal de calcular la complexitat lineal utilitzarem l'algorisme de Massey. La definició formal de la complexitat lineal va lligada al concepte d'LFSR, que presentarem més envadant. Així doncs, detallarem la definició formal de complexitat lineal una vegada haguem introduït l'arquitectura dels LFSR.

Una configuració bastant habitual en criptografia és fer servir CSPRNG amb valors veritablement aleatoris com a llavors. Aconseguir nombres (veritablement) aleatoris no és una tasca senzilla. Per tal de generar-los cal disposar d'una font d'aleatorietat natural. Addicionalment, si aquesta font d'aleatorietat es vol fer servir en criptografia, caldrà assegurar també que un adversari no és capaç de manipular-la ni observar-la. Existeixen, principalment, dues maneres d'obtenir valors realment aleatoris: a través de hardware, explotant l'aleatorietat que es produeix en fenòmens físics, o a través de software, a partir d'observacions afectades pel comportament de l'usuari. Així, per exemple, es pot fer servir el so capturat per un micròfon, el soroll tèrmic[2] d'una resistència o d'un diode, les turbolències creades per l'aire en segons quins dispositius o el moviment del ratolí.

Tests d'aleatorietat del NIST

El NIST disposa d'un banc de proves estadístiques per avaluar l'aleatorietat de seqüències binàries generades per PRNG. El banc consta de 15 testos. En aquest apartat, descriurem els testos més senzills, amb l'objectiu de donar una idea del que es busca en l'avaluació de l'aleatorietat de seqüències. Per tal de descriure els testos, suposarem que s'avalua la seqüència binària $S = \{s_1, s_2, \cdots, s_n\}$ de mida n bits.

El test de **freqüència de bits individuals** comprova que la proporció d'uns i zeros de la seqüència proporcionada s'aproxima a la que observaríem en una seqüència veritablement aleatòria, és a dir, que la proporció d'uns i zeros és similar i s'aproxima, per tant, a 0.5. Per fer-ho, en primer lloc es transforma la seqüència binària d'entrada a una seqüència de -1 i 1:

$$X = \{x_i \mid x_i = 2 \cdot s_i - 1, \forall i \in [1, n]\}$$

és a dir, els zeros es converteixen en -1 i els uns segueixen representant-se amb 1.

Després, es calcula s_{obs}:

$$s_{obs} = \frac{\mid \sum_{i=1}^{n} x_i \mid}{\sqrt{n}}$$

Si la seqüència és aleatòria s_{obs} tendirà cap a 0, mentre que si hi ha massa zeros o massa uns en la seqüència, aleshores s_{obs} tendirà a ser major a zero.

Superació dels testos

A partir del valor s_{obs}, els testos del NIST calculen el nivell de significació observat per decidir si la seqüència supera o no la comprovació. En concret, es considera que la seqüència supera la prova si el valor p és major o igual a 0.01. El lector interessat en els detalls sobre els testos estadístics pot consultar la publicació original del NIST: A. Rukhin, J. Soto, J. Nechvatal, et al. (2010). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*.

[2] S'anomena soroll tèrmic o soroll de Johnson-Nyquist a les fluctuacions elèctriques generades per l'agitació tèrmica dels electrons.

Exemple 3.1 Càlcul de la prova de freqüència de bits individuals

Donada la seqüència:
$S = \{0,1,0,1,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,1,0,1,0,1,0,0,0\}$
amb $n = 28$, procedim a calcular s_{obs}.

En primer lloc, transformem la seqüència de zeros i uns en una seqüència de -1 i 1:

$$X = \{-1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,1,1,-1,-1,1,1,1,1,-1,1,-1,1,-1,1,-1,-1,-1\}$$

Seguidament, calculem el valor s_{obs}:

$$s_{obs} = \frac{|\sum_{i=1}^{28} x_i|}{\sqrt{28}} = \frac{|-6|}{\sqrt{28}} \approx 1.1339$$

Superació del test	En aquest cas, el nivell de significació per a $s_{obs} = 1.1339$ és de 0.256, de manera que el test es considera superat ja que $0.256 \geq 0.01$.

El test de **freqüència en un bloc** comprova que el número de zeros i uns en un bloc de m bits sigui aproximadament $m/2$. Per fer-ho, es particiona la seqüència a avaluar en $b = \lfloor n/m \rfloor$ blocs de m bits, descartant els bits sobrants.

$$\underbrace{\overbrace{s_1, s_2, \ldots, s_m}^{k=1}}_{m \text{ bits}} \quad \underbrace{\overbrace{s_{m+1}, s_{m+2}, \ldots, s_{m+m}}^{k=2}}_{m \text{ bits}} \quad \cdots \quad \underbrace{\overbrace{s_{(b-1)m+1}, \ldots, s_{(b-1)m+m}}^{k=b}}_{m \text{ bits}}$$

Aleshores, per cada bloc k (amb $k = 1, \cdots, b$), es calcula:

$$\pi_k = \frac{\sum_{j=1}^{m} s_{(k-1)m+j}}{m}$$

és a dir, es calcula la proporció d'uns que hi ha a cada bloc.

Finalment, es calcula:

$$\chi^2_{obs} = 4m \sum_{k=1}^{b} (\pi_k - 1/2)^2$$

Exemple 3.2 Càlcul de la prova de freqüència en un bloc

Donada la mateixa seqüència que en l'exemple anterior:
$S = \{0,1,0,1,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,1,0,1,0,1,0,0,0\}$
amb $n = 28$, procedim a calcular π_k per a cada bloc amb $m = 6$.

En primer lloc, dividim la seqüència en $b = \lfloor n/m \rfloor = \lfloor 28/6 \rfloor = 4$ blocs de $m = 6$ bits

$$\overbrace{0,1,0,1,0,0}^{k=1}\ \overbrace{0,0,0,0,0,1}^{k=2}\ \overbrace{1,0,0,1,1,1}^{k=3}\ \overbrace{1,0,1,0,1,0}^{k=4}\ 1,0,0,0$$
$$\underbrace{}_{6\text{ bits}}\ \underbrace{}_{6\text{ bits}}\ \underbrace{}_{6\text{ bits}}\ \underbrace{}_{6\text{ bits}}$$

Els últims 4 bits de la seqüència es descarten i no s'utilizen en el test.

Aleshores, per cada bloc k (amb $k = 1, \cdots, 4$), calculem π_k:

$$\begin{aligned}
\pi_1 &= \frac{\sum_{j=1}^{m} s_{(k-1)m+j}}{m} = \frac{\sum_{j=1}^{6} s_j}{6} = \frac{2}{6} = 1/3 \\
\pi_2 &= \frac{\sum_{j=1}^{6} s_{6+j}}{6} = 1/6 \\
\pi_3 &= \frac{\sum_{j=1}^{6} s_{2\cdot 6+j}}{6} = \frac{4}{6} = 2/3 \\
\pi_4 &= \frac{\sum_{j=1}^{6} s_{3\cdot 6+j}}{6} = \frac{3}{6} = 1/2
\end{aligned}$$

I finalment estem en disposició de calcular χ^2_{obs}:

$$\begin{aligned}
\chi^2_{obs} &= 4m \sum_{k=1}^{b} (\pi_k - 1/2)^2 \\
&= 4 \cdot 6 \sum_{k=1}^{4} (\pi_k - 1/2)^2 \\
&= 24 \cdot \left((1/3 - 1/2)^2 + (1/6 - 1/2)^2 + (2/3 - 1/2)^2 + (1/2 - 1/2)^2\right) \\
&= 24 \cdot (1/36 + 1/9 + 1/36 + 0) = 4
\end{aligned}$$

Superació del test	En aquest cas, el nivell de significació per a $\chi^2_{obs} = 4$ (tenint en compte que tenim $b = 4$ blocs) és de 0.4060, fent que el test es consideri superat ja que $0.4060 \geq 0.01$.

El test de **ràfegues** comprova si el número de ràfegues tant d'uns com de zeros de la seqüència s'assembla al que trobaríem en una seqüència aleatòria.

Definirem una **ràfega** com un conjunt de bits consecutius iguals, és a dir una ràfega de longitud k consta dels elements $s_t, \cdots, s_{t+k-1}$, tals que $s_{t-1} \neq s_t = s_{t+1} = \cdots = s_{t+k-1} \neq s_{t+k}$.

Per avaluar la prova de ràfegues, es calcula:

$$V_n(obs) = \left(\sum_{i=1}^{n-1} r(i) \right) + 1$$

on $r(i)$ és la funció:

$$r(i) = \begin{cases} 0, & \text{si } s_i = s_{i+1} \\ 1, & \text{altrament} \end{cases}$$

Oscil·lacions

La seqüència 10101010 oscil·la molt ràpidament, ja que cada bit canvia el valor respecte al bit anterior. En canvi, la seqüència 11111100 oscil·la molt lentament, ja que només es produeix un canvi de valor en tota la seqüència.

Valors grans de V_{obs} indiquen que les oscil·lacions de valors en la seqüència avaluada (és a dir, els canvis de u a zero o de zero a u) succeeixen ràpidament, mentre que valors petits indiquen que les oscil·lacions són lentes.

El NIST recomana que les seqüències avaluades amb aquest test tinguin com a mínim 100 bits (és a dir, $n \geq 100$).

Addicionalment, aquest test té com a prerequisit que la seqüència passi el test de freqüència de bits individuals que hem descrit anteriorment. És a dir, si una seqüència no supera el test de bits individuals, aleshores ja no es realitza el test de ràfegues.

Exemple 3.3 Càlcul de la prova de ràfegues

Seguint amb l'avaluació de la mateixa seqüència que en els exemples anteriors:
$S = \{0,1,0,1,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,1,0,1,0,1,0,0,0\}$
amb $n = 28$. Noteu que si seguíssim les recomanacions del NIST, no efectuaríem el test de ràfegues sobre aquesta seqüència, ja que aquesta no té una longitud mínima de 100 bits. A tall d'exemple, però, realitzarem els càlculs per a aquesta seqüència.

En primer lloc comprovem que la seqüència superi el test de freqüència de bits individuals. Com hem vist al primer exemple, la seqüència supera aquest test, així que procedim a calcular $V_{28}(obs)$.

$$\begin{aligned} V_n(obs) &= \left(\sum_{i=1}^{n-1} r(i) \right) + 1 \\ &= (1+1+1+1+0+0+0+0+0+0+1+0+1+0+1+0+ \\ &\quad +0+0+1+1+1+1+1+1+1+0+0)+1 = 15 \end{aligned}$$

Superació del test

En aquest cas, el nivell de significació per a $V_n(obs) = 15$ (i tenint en compte que la seqüència té 28 bits i una proporció de $11/28$ d'uns) és de 0.5151, fent que el test es consideri superat ja que $0.5151 \geq 0.01$.

Com s'ha comentat, el banc de proves del NIST recull 15 testos diferents. A més dels tres comentats, els altres testos comproven l'aparició de les ràfegues d'uns més llargues, la repetició de subseqüències concretes dins la seqüència, la facilitat de comprimir-la, la seva complexitat lineal o les seves propietats espectrals, entre d'altres.

3.3 Generadors lineals de seqüència xifrant

En l'apartat anterior, hem estudiat les propietats que han de tenir les seqüències xifrants per tal de poder-les utilitzar en criptosistemes de xifratge de flux. Tractem ara com han de ser els algorismes deterministes que generen aquests tipus de seqüències.

Des d'un punt de vista general tenim dos tipus de generadors: els lineals i els no lineals.

Els **generadors lineals** són aquells que només realitzen operacions lineals sobre els elements d'entrada per obtenir la seqüència de sortida. Contràriament, els **generadors no lineals** són els que realitzen a més a més operacions no lineals, com podrien ser permutacions.

3.3.1 Generadors congruencials

Els **generadors congruencials** es basen en equacions modulars recurrents del tipus:

$$x_n = (ax_{n-1} + b) \bmod m$$

En aquest cas el valor x_0 seria la llavor de la seqüència xifrant. Un criptosistema que utilitzi un generador d'aquest tipus tindrà com a clau secreta els valors $\{x_0, a, b, m\}$, i per tal que el període sigui màxim caldrà que compleixi que $\gcd(a,m) = 1$.

Cal dir però, que aquests tipus de generadors pseudoaleatoris no són segurs des d'un punt de vista criptogràfic ja que s'ha pogut demostrar que amb pocs valors x_i coneguts ja es poden esbrinar els paràmetres secrets $\{x_0, a, b, m\}$. Fins hi tot, només amb una part dels bits que formen els x_i, però això sí, coneixent els paràmetres $\{a, b, m\}$, es pot arribar a determinar el valor de la llavor x_0.

Tot i això, aquests tipus de generadors són molt utilitzats en sistemes informàtics per a aplicacions no criptogràfiques.

Exemple 3.4 La funció *rand*() del sistema UNIX BSD utilitza el següent generador congruencial afí:

$$x_n = (1103515245x_{n-1} + 12345) \bmod 2^{31}$$

on la llavor especifica el valor inicial.

3.3.2 Registres de desplaçament realimentats linealment (LFSR)

Un registre de desplaçament realimentat linealment (o LFSR, de l'anglès, *Linear Feedback Shift Register*) de longitud n és un dispositiu físic o lògic format per n cel·les de memòria i n portes lògiques que té una estructura com es mostra a la Figura 3.2.

Inicialment, les cel·les contenen els valors d'entrada, i a cada impuls de rellotge el contingut de la cel·la s_i es desplaça a la cel·la s_{i-1} realitzant les operacions associades. D'aquesta manera es genera un nou element, s_{n+1} que és determinat per l'expressió:

$$s_{n+1} = c_1 s_n + \cdots + c_n s_1 \tag{3.1}$$

on els $c_i \in \{0, 1\}$ corresponen als valors de les portes lògiques de l'esquema. És a dir, els coeficients

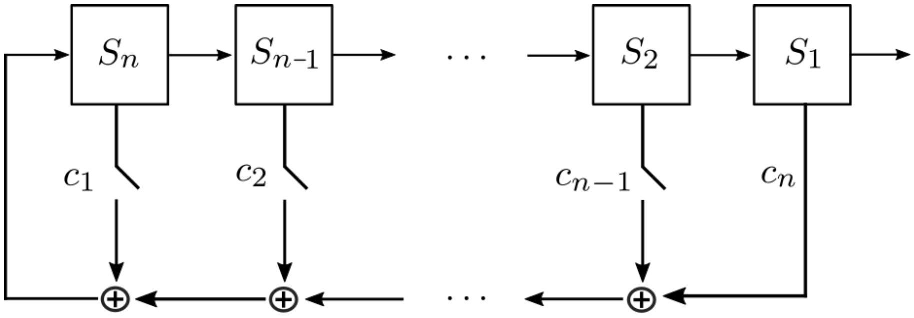


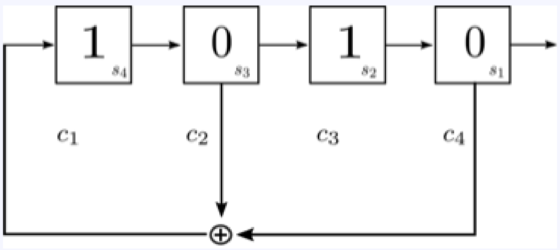
Figura 3.2: Esquema general d'un LFSR

seran 1 si hi ha una connexió i 0 si no n'hi ha. Aquest nou element, s_{n+1}, se situa a la cel·la s_n que ha quedat buida a causa del desplaçament.

El conjunt de valors continguts en cada cel·la en un instant de temps s'anomena **estat**. L'**estat inicial** és l'estat en què es troba l'LFRS en el moment de començar el procés.

Exemple 3.5 Funcionament d'un LFSR

Aquest exemple segueix el funcionament de l'LFSR de 4 cel·les representat en la següent figura:



Com es pot veure, l'estat inicial és 1010, que correspon a l'impuls de rellotge $t = 0$. La taula següent mostra l'evolució de l'LFSR en els diferents instants de temps.

Impuls de rellotge (t)	s_4	s_3	s_2	s_1	**Sortida**
0	1	0	1	0	0
1	0	1	0	1	1
2	0	0	1	0	0
3	0	0	0	1	1
4	1	0	0	0	0
5	0	1	0	0	0
6	1	0	1	0	0
7	0	1	0	1	1
⋮		⋮			⋮

En $t = 0$ les cel·les s_1 i s_3 contenen un 0 i, per tant, el bit s_4 serà 0 en el següent impuls de rellotge

$(t = 1)$. Noteu com la resta de valors es desplacen: a $t = 1$ la cel·la s_1 conté el valor que hi havia en $t = 0$ a la cel·la s_2, la cel·la s_2 conté el valor que hi havia a s_3, etc.

En general, per $i \geq 1$ tenim:

$$
\begin{aligned}
s_1(t=i) &= s_2(t=i-1) \\
s_2(t=i) &= s_3(t=i-1) \\
s_3(t=i) &= s_4(t=i-1) \\
s_4(t=i) &= s_1(t=i-1) \oplus s_3(t=i-1)
\end{aligned}
$$

Si ens fixem, en l'impuls de rellotge $t = 6$ tornem a tenir l'estat inicial i, per tant, a partir d'aquí la seqüència es torna a repetir. Aquesta seqüència, doncs, té període 6.

Un cop definit el que és un LFSR podem passar a fer un estudi una mica més exhaustiu per tal de determinar-ne les seves característiques més importants. L'avantatge principal dels LFSR és que tenen una formulació matemàtica molt simple, com veurem a continuació i, per tant, es poden estudiar de manera força clara i completa. A més, com que es defineixen per mitjà de cel·les i portes lògiques, s'implementen fàcilment en el maquinari, fet que permet obtenir generadors de gran velocitat.

Primerament cal fer notar que l'estat inicial d'un LFSR no pot ser tot zeros. Si fos així, la seqüència que produiria seria també tota de zeros, ja que totes les operacions són lineals. Es diu que l'estat que tan sols té zeros és un **estat absorbent**. També convé destacar que el període màxim d'un LFSR és $2^n - 1$. Aquest valor s'obté de considerar tots els estats possibles 2^n i eliminar-ne l'estat absorvent.

Si ens fixem en l'expressió 3.1 ens adonarem que tota la seqüència de sortida d'un LFSR queda determinada per l'estat inicial $\{s_1, \cdots, s_n\}$ i per la relació

$$s_{n+k} = \sum_{i=1}^{n} c_i s_{n+k-i} \quad \text{per } k \geq 0 \tag{3.2}$$

on $c_i \in \{0,1\}$ per $1 \leq i \leq n$.

El **polinomi de connexions** d'un LFSR de longitud n és el polinomi de grau n

$$C(x) = 1 + c_1 x^1 + c_2 x^2 + \cdots + x^n$$

on els $c_i \in \{0,1\}$ corresponen als valors de les portes lògiques de la figura de l'esquema general d'un LFSR.

Exemple 3.6 Polinomi de connexions d'un LFSR

El polinomi de connexions corresponent a l'LFSR de l'exemple 3.5 és:

$$C(x) = 1 + 0 \cdot x^1 + 1 \cdot x^2 + 0 \cdot x^3 + 1 \cdot x^4 = 1 + x^2 + x^4$$

Un LFSR queda determinat pel seu polinomi de connexions. Una **seqüència** queda determinada pel polinomi de connexions i pel seu estat inicial.

Definit el polinomi de connexions, ja podem determinar les característiques de l'LFSR d'acord amb les del seu polinomi de connexions:

1. Polinomi de connexions **factoritzable**: els LFSR's que tenen polinomis de connexions factoritzables generen seqüències que depenen de l'estat inicial. A més, el període de les esmentades seqüències és sempre més petit que el període màxim que pot tenir un LFSR, que és $2^n - 1$.
2. Polinomi de connexions **irreductible**: igual que en el cas anterior, un LFSR amb un polinomi de connexions irreductible (però que no sigui primitiu) genera seqüències que depenen de l'estat inicial de l'LFSR i, en aquest cas, el seu període és un divisor de $2^n - 1$.
3. Polinomi de connexions **primitiu**: un LFSR amb polinomi de connexions primitiu té la seqüència de sortida de període màxim, $2^n - 1$. Aquesta seqüència de període màxim s'obté per a qualsevol estat inicial, llevat de l'estat absorbent.

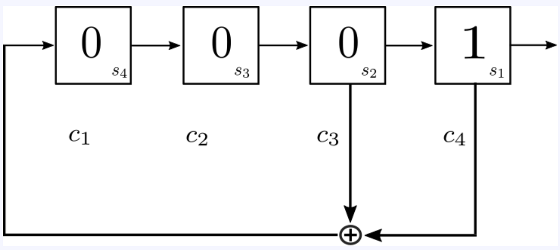
Polinomi primitiu

Un polinomi primitiu és també irreductible. Per tant, les seqüències generades per LFSR amb polinomis de connexions primitius no dependran de l'estat inicial.

Considerant les propietats de les seqüències segons els seus polinomis de connexions, veiem que per a esquemes de xifratge de flux és aconsellable fer servir la que determina el període màxim.

Exemple 3.7 LFSR amb polinomi de connexions primitiu

El polinomi $1 + x^3 + x^4$ (amb coeficients a $\mathbb{Z}_2$) és primitiu. Construïm un LFSR amb aquest polinomi de connexions, i observem la seqüència de sortida de l'LFSR al fer servir els valors 0001 com a estat inicial.



La seqüència generada serà:

$L_1 = \underline{100010011010111}1000\ldots$

Efectivament, el període de la seqüència generada és $2^n - 1 = 2^4 - 1 = 15$: a partir del bit 16, la seqüència torna a repetir-se.

Si generem una segona seqüència amb el mateix LFSR però fent servir els valors 1010 com a estat inicial, la seqüència que s'obté és:

$L_2 = \underline{010111100010011}0101\ldots$

De nou, el període de la seqüència generada és 15.

El polinomi de connexions que hem fet servir és també irreductible. Per tant, les seqüències generades amb diferents estats inicials són les mateixes, però amb un desplaçament. En efecte, podem veure com la seqüència L_2 és la seqüència L_1 desplaçada 9 posicions a l'esquerra:

$$\begin{array}{lll} L_1 = & 100010011 & 010111100010011010 1 \\ L_2 = & & 010111100010011010 1 \end{array}$$

Per últim, aprofitarem l'exemple per mostrar perquè no podem generar una seqüència de període superior amb un LFSR de 4 cel·les. La taula següent mostra tots els estats per els quals passa l'LFSR per generar la seqüència L_1:

t	Estat	Sortida	t	Estat	Sortida
0	0001	1	8	0101	1
1	1000	0	9	1010	0
2	0100	0	10	1101	1
3	0010	0	11	1110	0
4	1001	1	12	1111	1
5	1100	0	13	0111	1
6	0110	0	14	0011	1
7	1011	1	15	0001	1

Fixeu-vos que l'estat a $t = 15$ correspon a l'estat inicial ($t = 0$), motiu per el qual la seqüència comença a repetir-se. Noteu també com l'LFSR passa per 15 estats diferents, que són tots els possibles estats que es poden generar amb 4 bits, exceptuant l'estat absorvent (0000). El període és doncs màxim, i no hi ha manera de generar un període superior amb l'estructura d'un LFSR, ja que no hi ha més estats possibles.

Addicionalment, fixeu-vos que l'estat en $t = 9$ correspon a l'estat inicial amb el que generem la seqüència L_2.

Exercici 3.1 Calculeu els primers 15 bits de la seqüència de sortida d'un LFSR de 5 cel·les que té com a polinomi de connexions $1 + x^2 + x^5$ i que s'inicialitza amb l'estat $0, 0, 0, 1, 1$.

3.3.3 Limitacions dels generadors lineals

Número de polinomis primitius	No hem d'oblidar que el nombre de polinomis primitius de grau n ve donat per l'expressió $\phi(2^n - 1)/n$ on ϕ és la funció totient d'Euler.

Ja hem posat en relleu que els LFSR es comporten molt bé en termes de facilitat d'anàlisi, d'implementació i de velocitat. Un dels punts negatius d'aquests generadors és que per tal que el període $2^n - 1$ sigui gran cal que la longitud de l'LFSR, també ho sigui. Això pot representar un problema ja que el cost de trobar polinomis primitius amb grau gran és força elevat.

Malgrat els avantatges i inconvenients presentats, la raó principal per la qual els LFSR no serveixen per si sols per a sistemes de xifratge en flux és que són fàcilment predictibles.

En efecte, suposem que coneixem $2n$ bits consecutius, $s_{k+1}, s_{k+2}, \cdots, s_{k+2n}$ aleshores podem determinar els coeficients del polinomi de realimentació, c_i, i per tant tota la seqüència. Per fer-ho només cal basar-se en l'expressió 3.2 per plantejar el següent sistema d'equacions:

$$\begin{pmatrix} s_{k+1} & s_{k+2} & \cdots & s_{k+n} \\ s_{k+2} & s_{k+3} & \cdots & s_{k+n+1} \\ \vdots & \vdots & \cdots & \vdots \\ s_{k+n} & s_{k+n+1} & \cdots & s_{k+2n-1} \end{pmatrix} \begin{pmatrix} c_n \\ c_{n-1} \\ \vdots \\ c_1 \end{pmatrix} = \begin{pmatrix} s_{k+n+1} \\ s_{k+n+2} \\ \vdots \\ s_{k+2n} \end{pmatrix}$$

Per tant, tenim un sistema d'n equacions amb n incògnites, c_i per $1 \leq i \leq n$, amb la qual podem determinar tots els coeficients.[3]

Així doncs, a l'hora d'utilitzar un generador per a un procés de xifratge en flux, cal fixar-se també (com ja hem esmentat anteriorment) en la seva predictibilitat, és a dir, el que s'anomena la complexitat lineal.

Atès que qualsevol seqüència periòdica es pot generar amb un LFSR no singular, J.L. Massey[4] va definir la complexitat lineal d'una seqüència de la següent manera:

La **complexitat lineal** d'una seqüència és el nombre de cel·les de l'LFSR més curt capaç de generar-la.

Per tant, una seqüència generada per un LFSR de longitud n té òbviament com a molt complexitat lineal n, molt baixa comparada amb el període, $2^n - 1$. El mateix Massey va proposar un algorisme que, a partir d'una seqüència, determina l'LFSR mínim que la genera amb l'estat inicial corresponent.

Exercici 3.2 Quin és el període i la complexitat lineal màxima de les seqüències generades per l'LFSR amb polinomi de connexions $1 + x^2 + x^5$?

Exercici 3.3 Donada la seqüència $s = 0001001101011110001 0$, sintetitzeu l'LFSR que l'ha generada, sabent que el polinomi de connexions té grau 4.

Per a disminuir la predictibilitat de la seqüència de xifratge cal, doncs, augmentar la complexitat lineal de la seqüència de xifratge, que convindria que fos de llargada propera a la del període. Una manera de fer-ho és basant-se en operacions no lineals, tal com veurem més endavant.

3.4 Generadors no lineals

A continuació analitzarem alguns del generadors no lineals destinats a augmentar la complexitat lineal de les seqüències de flux que es fan servir en l'actualitat. En concret, descriurem l'A5 i el Trivium.

[3] Noteu que aquest sistema d'equacions es pot resoldre fàcilment amb qualsevol mètode de resolució de sistemes d'equacions lineals, per exemple, el mètode de Gauss. Per a una introducció als sistemes d'equacions lineals, podeu consultar el primer capítol del llibre *Elementary linear algebra*, d'H. Anton.

[4] Massey va proposar un algorisme per sintetitzar l'LFSR més curt capaç de generar una seqüència l'any 1969 a l'article *Shift-Register Synthesis and BCH decoding*.

3.4.1 A5

L'A5 és un algorisme de xifrat en flux que s'utilitza per al xifrat de dades en les transmissions de la xarxa GSM[5].

L'A5 disposa de quatre variants, denotades amb els noms A5/0, A5/1, A5/2 i A5/3. L'A5/0 no fa servir xifrat (retorna el propi text en clar), l'A5/1 correspon a la versió original de l'algorisme que es fa servir a Europa, l'A5/2 és un algorisme de xifrat més dèbil creada per poder complir amb les regulacions per exportar criptografia (i feta servir als Estats Units) i l'A5/3 és un algorisme de xifratge totalment diferent (afegida amb posterioritat). En aquest apartat, descriurem el funcionament de l'algorisme A5/1.

L'A5 va començar a utilitzar-se a la xarxa GSM sense fer pública la seva especificació, seguint el principi de seguretat per obscuritat (en anglès, *security through obscurity*). L'ús d'aquest paradigma està totalment desconsellat pels experts ja que viola el principi de Kerckhoffs.[6] Tot i no fer-se públic oficialment, un primer esborrany de l'algorisme va ser publicat al 1994 i l'especificació completa va ser finalment obtinguda a través d'un procés d'enginyeria inversa del firmware d'un telèfon mòbil i donada a conèixer al públic el 1999.

El criptosistema A5/1 és un criptosistema de flux que utilitza una combinació no lineal de la sortida de tres LFSR. Si pensem que l'A5/1 xifra cadenes de text en clar de 228 bits (el que en el llenguatge de telefonia mòbil es coneix com trames de 228 bits), podem dividir el funcionament de l'A5/1 en tres etapes:

1. la inicialització dels LFSR,
2. l'obtenció dels 228 bits de la seqüència de xifrat a partir del moviment dels LFSR,
3. el xifrat del text en clar pròpiament dit, que segueix el procediment habitual dels xifrats de flux, realitzant una XOR de la seqüència de xifrat amb el text en clar.

Per xifrar 228 bits més caldrà tornar a reinicialitzar els LFSR, obtenir els nous 228 bits de la seqüència xifrant i fer l'XOR amb els nous bits de text en clar.

La inicialització dels tres LFSR que formen l'A5/1 no es limita a donar-ne els seus valors inicials, sinó que el contingut inicial de les cel·les dels LFSR es calcula a partir d'unes claus d'entrada i d'unes transformacions que descriurem més endavant. Com que la inicialització dels LFSR es fa a partir de propi funcionament del sistema, passem primer a descriure com s'obtenen els bits de la seqüència de xifrat.

L'A5/1 té una estructura formada per 3 LFSR tal com es mostra en la Figura 3.3.

La Taula 3.1 detalla les longituds de cadascun dels LFSR de l'A5/1 així com els seus polinomis de connexions.

La no linealitat del sistema ve donada perquè a cada impuls de rellotge no tots els LFSR avancen. Només ho fan aquells LFSR els bits dels quals són majoria en les cel·les anomenades clocking bit (en el cas de l'esquema, els clocking bits són les cel·les marcades amb sombrejat, és a dir la cel·la 9 per al primer LFSR i les cel·les 11 per al segon i tercer). Aquest esquema es coneix com a clocking

[5]GSM són les sigles de *Global System for Mobile Communcation*, la xarxa que englobava més del 80% de les connexions mòbils el 2010. L'ús de la xarxa ha anat minvant amb l'aparició de xarxes amb més ample de banda com ara el 3G o 4G.

[6]Recordem que el principi de Kerckhoffs postula que un criptosistema ha de ser segur encara que tota la informació sobre el criptosistema sigui pública, exeptuant la clau que ha de romandre privada. És a dir, la seguretat d'un criptosistema ha de recaure únicament en el secret de la clau.

irregular segons la funció majoritària.

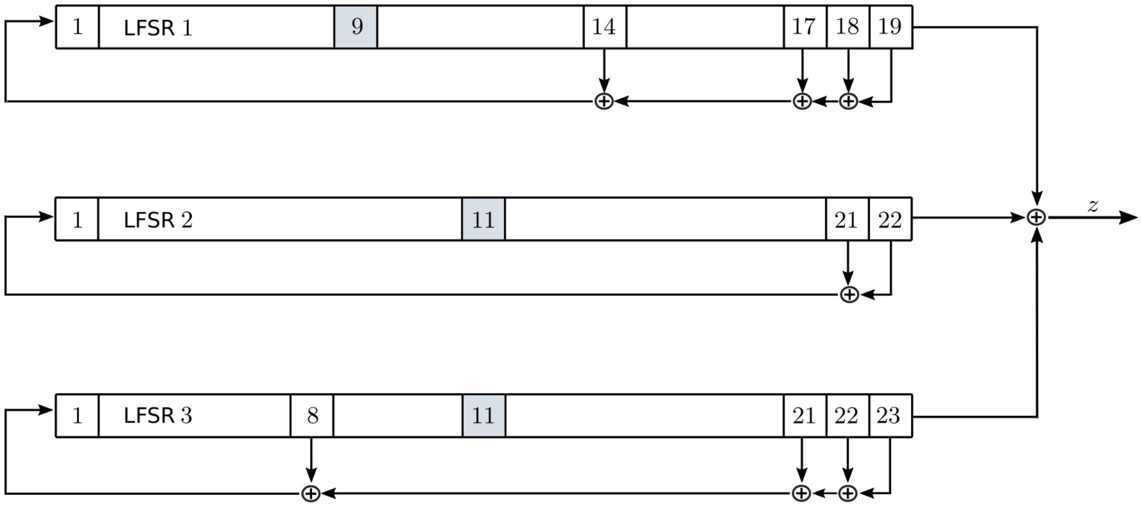


Figura 3.3: Esquema de l'A5.

LFSR	Longitud	Polinomi de connexions	Clocking bit
1	19	$x^{19}+x^{18}+x^{17}+x^{14}+1$	9
2	22	$x^{22}+x^{21}+1$	11
3	23	$x^{23}+x^{22}+x^{21}+x^{8}+1$	11

Taula 3.1: Descripció dels LFSR de l'A5.

Així, per exemple, si en la cel·la 9 del primer LFSR hi ha un 1, i en les cel·les 11 del segon i tercer LFSR hi ha un 0, només avançaran el segon i el tercer LFSR, que tenen un 0. Si els tres són iguals, aleshores avancen tots. D'aquesta manera es van obtenint les sortides de cada un dels LFSR que formen l'XOR que acabarà proporcionant cada bit de la seqüència de xifratge.

Exemple 3.8 Iteració de l'A5

Si en l'instant t tenim els següents estats interns[a] en els LFSR:
LFSR1: $\mathtt{1011100011\ 01100001\underline{0}}$
LFSR2: $\mathtt{1011011011\ 1101000010\ 0\underline{1}}$
LFSR3: $\mathtt{1110111110\ 0111001000\ 00\underline{1}}$

La sortida en aquest mateix instant de temps t serà doncs:
$z = 0 \oplus 1 \oplus 1 = 0$ i es calcula a partir de la sortida dels tres LFSR (els bits subratllats en els estats interns).

Per tal de calcular l'estat dels LFSR en el següent instant de temps $t+1$, observarem el bit de clocking de cada LFSR (indicat amb negreta). En aquest cas, els bits de clocking són 1, 1 i 0 per a l'LFSR 1, 2 i 3, respectivament. Per tant, el bit majoritari és 1, i avançaran doncs els LFSR 1 i 2. Així, l'estat intern dels LFSR en $t+1$ és:
LFSR1: $\mathtt{1101110001\ 10110000\underline{1}}$

LFSR2: `1101101101 1110100001 00`
LFSR3: `1110111110 0111001000 001`

[a] L'agrupació de bits de 10 en 10 respon únicament a qüestions estètiques: s'ha triat aquesta representació per tal que sigui més fàcil de llegir.

Passem ara a descriure el procés d'inicialització de l'A5. La inicialització requereix dos valors, una clau de sessió de 64 bits i un número de trama de 22 bits, i consta de quatre passos:

1. En primer lloc, s'omplen tots els registres dels tres LFSR amb el valor 0.
2. Seguidament, s'executen 64 impulsos de rellotge dels tres LFSR sense fer servir clocking irregular. És a dir, a cada impuls de rellotge, els tres LFSR avancen. La particularitat d'aquest pas és que el bit de retroalimentació de l'LFSR fa una XOR amb un bit de la clau de sessió abans de ser inserit a la primera cel·la de l'LFSR. Cadascuna de les 64 polsacions fa servir un dels bits de la clau de sessió diferent, de manera seqüencial.
3. De manera similar al pas anterior, s'executen 22 impulsos de rellotge dels tres LFSR sense fer servir clocking irregular. Aquest cop, però, el bit de retroalimentació fa una XOR amb els bits del número de trama abans d'inserir-se de nou a la cel·la corresponent.
4. Finalment, es realitza una fase d'escalfament, on s'executen 100 impulsos de rellotge amb clocking irregular.

La Figura 3.4 esquematitza el procés utilitzat per a realitzar els passos 2 i 3 de l'algorisme d'inicialització. Noteu que els passos 2 i 3 poden unir-se també amb un sol pas, on s'executen $64+22=86$ polsacions de rellotge fent una xor amb cadascun dels bits de la clau de sessió seguida del número de trama.

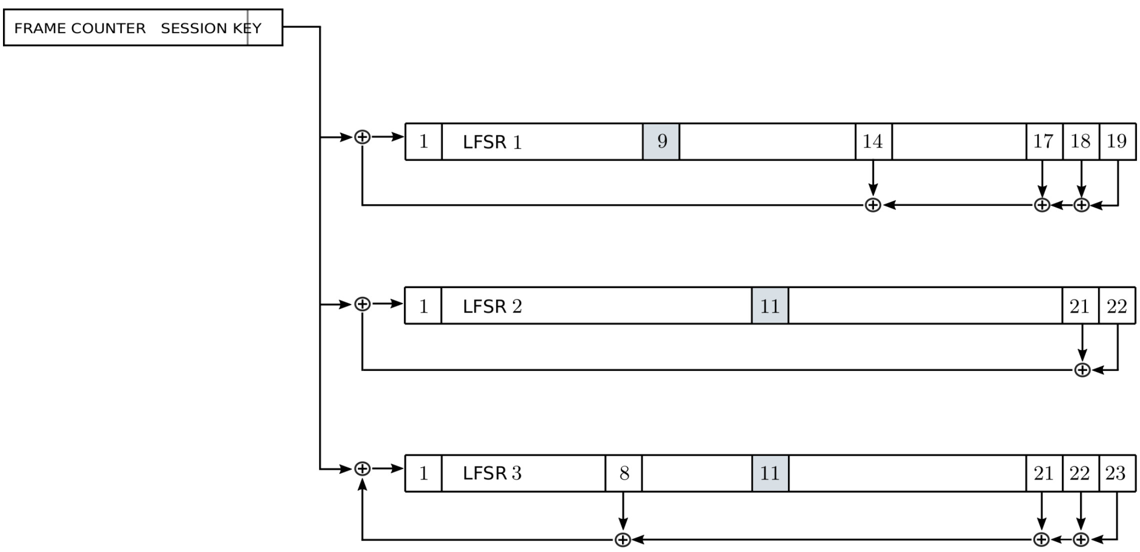


Figura 3.4: Esquema dels passos 2 i 3 de la inicialització de l'A5.

És important remarcar que en aquests passos d'inicialització el que interessa és el contingut que acabaran tenint les cel·les dels LFSR i per tant, els bits de sortida dels LFSR en tots aquests passos es descarten. Un cop inicialitzats els LFSR es procedeix a obtenir els 228 bits de la seqüència de xifratge. Finalment, per tal de xifrar una trama, es farà una xor amb els 228 bits obtinguts de la sortida de l'A5/1 i els 228 bits que representen el text en clar de la trama.

Per tal de xifrar la següent trama de 228 bits, procedirem a incrementar el comptador de trama i tornarem a realitzar el procés d'inicialització amb la mateixa clau de sessió i el nou valor de comptador de trama.

Noteu que la clau de sessió no es canvia per cada nova trama a xifrar, sinó que, en el context de telefonia mòbil en el que s'utilitza aquest sistema, la clau de sessió s'actualitza quan la xarxa decideix tornar a autenticar el dispostiu mòbil.

3.4.2 Trivium

El Trivium és un generador pseudoaleatori dissenyat pels criptògrafs Christophe De Cannière i Bart Preneel que aprofita una implementació hardware molt simple amb una velocitat elevada de generació de la seqüència, fet que el fa interessant en dispositius amb unes capacitats limitades de processat, com ara etiquetes RFID. El seu funcionament està descrit en l'estàndard ISO/IEC 29192-3.

El Trivium utilitza una clau de 80 bits i un vector d'inicialització també de 80 bits i permet generar seqüències de fins a 2^{64} bits.

A diferència de l'A5, el Trivium no es basa en LFSR, però sí que està format per 3 registres de desplaçament, tot i que la seva realimentació no és lineal. És a dir, les cel·les que contenen els registres es desplacen a la dreta com en un LFSR però la seva retroalimentació no està definida per una funció lineal. En la Figura 3.5 podem veure l'esquema del Trivium.

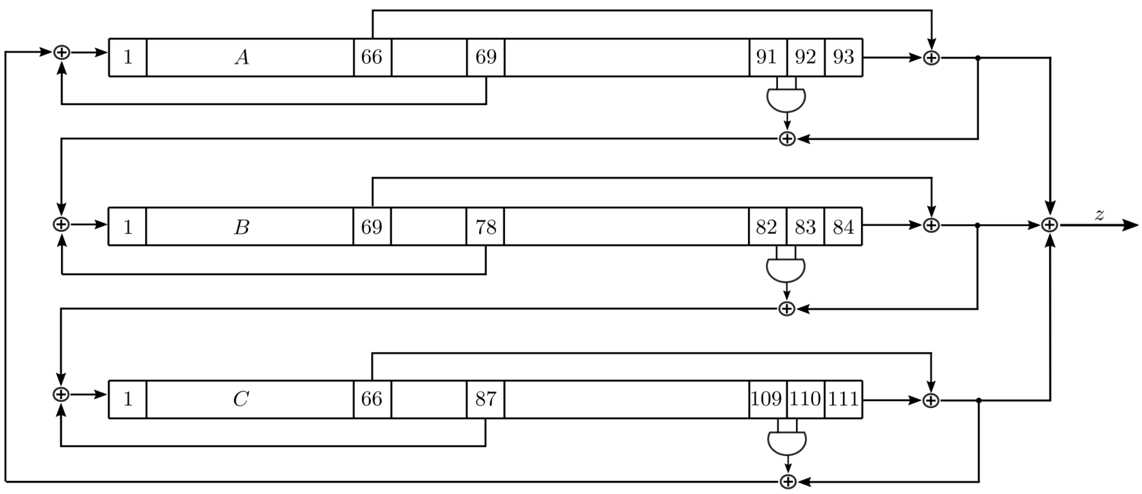


Figura 3.5: Esquema del Trivium.

Com es pot veure, el Trivium està format per tres registres de desplaçament, A, B i C, de 93, 84 i 111 cel·les, respectivament. La retroalimentació de cada registre no és lineal i, a més, la sortida de cada un dels registres retroalimenta un altre dels registres.

D'una banda, la sortida del Trivium (z) ve determinada en cada instant per les sortides dels tres registres (t^a, t^b, t^c):

$$z = t^a + t^b + t^c$$

on cada un dels elements són bits i, per tant, la suma es realitza mòdul 2.

Cada una de les sortides t queden determinades per l'estat dels registres de la següent manera:

$$t^a = s^a_{93} + s^a_{66}$$
$$t^b = s^b_{84} + s^b_{69}$$
$$t^c = s^c_{111} + s^c_{66}$$

Per tal de calcular el valor de la cel·la en la retroalimentació, es fan servir les sortides t, de manera que la sortida del registre a, t^a, s'utilitza en el càlcul de la retroalimentació del registre b; la sortida del registre b, t^b, es fa servir en la retroalimentació de c; i finalment la sortida del registre c, t^c, es fa servir en la retroalimentació del registre a. En concret, la retroalimentació de cada registre ve donada per les expressions:

$$s^a_{new} = t^c + (s^c_{109} \cdot s^c_{110}) + s^a_{69}$$
$$s^b_{new} = t^a + (s^a_{91} \cdot s^a_{92}) + s^b_{78}$$
$$s^c_{new} = t^b + (s^b_{82} \cdot s^b_{83}) + s^c_{87}$$

on, de nou, tots els operands són bits i tant la suma com el producte[7] d'aquesta expressió es realitzen mòdul 2.

La taula següent resumeix les accions que realitza cada posició específica de cada un dels registres:

	Feedback bit	**Feedforward bit**	**AND inputs**
A	69	66	91,92
B	78	69	82,83
C	87	66	109,110

Taula 3.2: Taula 3.2. Posicions destacades dels registres del Trivium.

Exemple 3.9 Iteració del Trivium

Si en l'instant t tenim els següents estats interns[a] en els registres:

A: 0111111100 1101111010 1111100101 1101010001 0111100010 0110110001
11001$\underline{1}$0111 1111100110 0101011100 01$\underline{1}$

B: 0100001010 1111011011 0110101000 1100010001 1111011000 1011110001
11011101$\underline{0}$0 0111100001 101$\underline{1}$

C: 1111110100 0111011101 0101111100 1010111100 0100011100 0001111011
10000$\underline{1}$1010 0111000011 1101010011 0101001000 0100000011 $\underline{0}$

[7] De forma equivalent, també podem pensar el producte mòdul 2 com un AND.

Les sortides dels registres t corresponen als valors:

$$t^a = s^a_{93} + s^a_{66} = 1 + 1 = 0$$
$$t^b = s^b_{84} + s^b_{69} = 1 + 0 = 1$$
$$t^c = s^c_{111} + s^c_{66} = 0 + 1 = 1$$

Noteu que els bits involucrats en els càlculs dels valors t es troben subratllats en l'estat dels registres per facilitar la lectura.

Així, la sortida del Trivium en l'instant t correspon a:

$$z = t^a + t^b + t^c = 0 + 1 + 1 = 0$$

Podem calcular també els bits que es faran servir en la retroalimentació dels registres, per tal d'actualitzar-ne el seu estat:

$$s^a_{new} = t^c + (s^c_{109} \cdot s^c_{110}) + s^a_{69} = 1 + (1 \cdot 1) + 1 = 1 + 1 + 1 = 1$$
$$s^b_{new} = t^a + (s^a_{91} \cdot s^a_{92}) + s^b_{78} = 0 + (0 \cdot 1) + 0 = 0 + 0 + 0 = 0$$
$$s^c_{new} = t^b + (s^b_{82} \cdot s^b_{83}) + s^c_{87} = 1 + (0 \cdot 1) + 0 = 1 + 0 + 0 = 1$$

Noteu que els bits involucrats en els càlculs dels valors s_{news} es troben indicats en negreta en l'estat dels registres per facilitar la lectura.

Els bits s_{new} serviran per actualitzar l'estat intern de cadascun dels registres. A tall d'exemple, veiem quin seria l'estat del registre A en l'instant $t+1$:

A: `1011111110 0110111101 0111110010 1110101000 1011110001 0011011000`
`1110011011 1111110011 0010101110 001`

[a] De nou, l'agrupació de bits de 10 en 10 respon únicament a qüestions estètiques: s'ha triat aquesta representació per tal que sigui més fàcil de llegir. Noteu, però, que els 80 bits corresponen a l'estat del registre, sense cap mena de separació entre ells.

Inicialització

A l'hora de xifrar un missatge, en primer lloc caldrà realitzar la **fase d'inicialització** del Trivium. Aquesta fase fa servir el vector inicial, VI, i la clau, k, ambdós valors de 80 bits. Aleshores, es prenen els 80 bits del vector inicial i es posen en les cel·les de més a l'esquerra del registre B. Seguidament, es prenen els 80 bits de la clau i es posen en les cel·les de més a l'esquerra del registre A. La resta de cel·les, de qualsevol dels tres registres, que no han quedat plenes s'omplen amb zeros, llevat de les 3 cel·les de més a la dreta del registre C, en les que s'hi inclou un 1 en cada un d'elles. La Figura 3.6 mostra gràficament la inicialització del Trivium.

Una vegada s'han situat aquests valors en els estats dels 3 registres, s'executen 1152 cicles de rellotge descartant els bits de sortida d'aquestes 1152 iteracions.

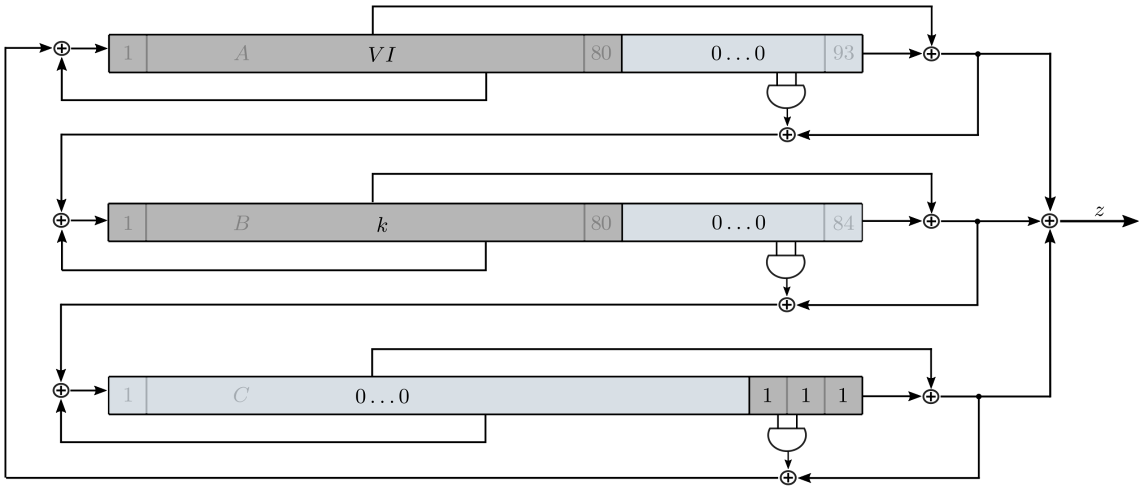


Figura 3.6: Fase d'inicialització del Trivium.

Finalment, una vegada s'ha inicialitzat el generador ja es pot utilitzar la seqüència de sortida per xifrar el missatge en clar. Així, cada bit de sortida del generador a partir de la iteració 1153 (un cop s'ha inicialitzat el generador) es combinarà amb una XOR amb el bit de text en clar a xifrar.

Per desxifrar un missatge utilitzant el Trivium, caldrà realitzar exactament el mateix procés aquesta vegada sobre el missatge xifrat, procés que es pot dur a terme perquè emissor i receptor comparteixen tant el vector inicial com la clau, ja que estem davant d'un criptosistema de clau simètrica.

3.5 Resum

En aquest capítol hem descrit el funcionament i les característiques principals dels esquemes de xifratge de flux. Hem estudiat les propietats que ha de tenir una seqüència aleatòria perquè es pugui utilitzar com a seqüència de xifratge. Hem presentat igualment diferents tipus de generadors per a obtenir seqüències pseudoaleatòries. Hem assenyalat que els registres de desplaçament realimentats linealment (LFSR) eren els més interessants perquè són fàcils d'estudiar, tot i que, com ja hem apuntat, no n'aconsellem l'aplicació en criptografia perquè la seva criptoanàlisi és força senzilla. Finalment, hem estudiat dos generadors que es fan servir avui en dia en productes habituals, l'A5 i el Trivium.

3.6 Solucions dels exercicis

Exercici 3.1:

Tenint en compte el polinomi de connexions de l'LFSR, el nou bit es calcula fent una XOR entre els bits de les cel·les s_4 i s_1 (que es troben subratllats a la taula) en l'instant de temps anterior:

Impuls de rellotge (t)	Estat	Sortida
0	0 $\underline{0}$ 0 1 $\underline{1}$	1
1	1 $\underline{0}$ 0 0 $\underline{1}$	1
2	1 $\underline{1}$ 0 0 $\underline{0}$	0
3	1 $\underline{1}$ 1 0 $\underline{0}$	0
4	1 $\underline{1}$ 1 1 $\underline{0}$	0
5	1 $\underline{1}$ 1 1 $\underline{1}$	1
6	0 $\underline{1}$ 1 1 $\underline{1}$	1
7	0 $\underline{0}$ 1 1 $\underline{1}$	1
8	1 $\underline{0}$ 0 1 $\underline{1}$	1
9	1 $\underline{1}$ 0 0 $\underline{1}$	1
10	0 $\underline{1}$ 1 0 $\underline{0}$	0
11	1 $\underline{0}$ 1 1 $\underline{0}$	0
12	0 $\underline{1}$ 0 1 $\underline{1}$	1
13	0 $\underline{0}$ 1 0 $\underline{1}$	1
14	1 $\underline{0}$ 0 1 $\underline{0}$	0

Així doncs, els 15 primers bits de la seqüència de sortida són: 110001111100110

Exercici 3.2:

El polinomi $1+x^2+x^5$ té grau $n=5$ i és un polinomi primitiu. Per tant, la complexitat lineal màxima de les seqüències que genera és $n=5$ i el període serà $2^n-1=2^5-1=31$.:

Exercici 3.3:

Per trobar el polinomi de connexions necessitem únicament $2n=8$ bits consecutius de la seqüència de sortida. Si agafem, per exemple, els 8 primers bits, podem plantejar el següent sistema d'equacions:

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_4 \\ c_3 \\ c_2 \\ c_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

La solució del sistema és $c_4=1, c_3=1, c_2=0, c_1=0$ i, per tant, el polinomi de connexions és x^4+x^3+1.

3.7 Bibliografia

Christophe De Cannière and Bart Preneel (2005). *Trivium - Specifications*. Technical report.

Joan Daemen and Vincent Rijmen (2002). *The Design of Rijndael, AES - The Advanced Encryption Standard*. Springer-Verlag (238 pp.)

GSMA (2017). *GSMA The Mobile Economy 2017*.
https://www.gsmaintelligence.com/research/

James L. Massey (1969). *Shift-register synthesis and BCH decoding*. IEEE transactions on Information Theory, 15(1), 122-127.

Alfred Menezes, Paul van Oorschot, and Scott Vanstone (1996). *Handbook of Applied Cryptography*. CRC Press.
http://cacr.uwaterloo.ca/hac/

NIST (2001). *Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197.
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

Christof Paar and Jan Pelzl (2009). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer.

Andrew Rukhin,Juan Soto, James Nechvatal, et al. (2010). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* . NIST Special Publication.
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

Thomas Stockinger (2005). *GSM network and its privacy - the A5 stream cipher*.

4. Les xifres de bloc

Una alternativa a les xifres de flux són les **xifres de bloc**. Aquestes xifres s'inclouen també dins dels criptosistemes de clau compartida ja que la clau que s'utilitza per a xifrar i desxifrar és la mateixa i la comparteixen emissor i receptor. La diferència bàsica entre el xifratge en flux i el xifratge en bloc és la utilització de la memòria en els algorismes de xifratge.

Ja hem vist en el capítol anterior que el xifrat de flux utilitza una clau diferent per cada bit d'informació. Aquesta clau depèn de l'estat inicial del generador, però també de l'estat del generador en el moment de xifrar un bit concret. Per tant, dos bits iguals es poden xifrar de maneres diferents depenent de l'estat en què es trobi el generador. En el xifratge en bloc això no passa ja que les xifres en bloc actuen sense memòria, i per tant el text xifrat només depèn del text en clar i de la clau. D'aquesta manera, dos blocs de text en clar iguals es xifren sempre de la mateixa manera quan s'utilitza la mateixa clau. Caldrà estudiar aquest fet en detall ja que si no es corregeix, els sistemes de xifrat que en resulten són força vulnerables, ja que es poden inserir o esborrar blocs de text xifrat sense que es pugui detectar. A més, el fet que dos blocs de text en clar quedin xifrats d'una mateixa manera, pot donar pistes per a una possible criptoanàlisi de tipus estadístic.

Pel que fa a la seva utilització, les xifres de bloc són força utilitzades ja que aconsegueixen una velocitat acceptable de xifratge. En concret, el xifrador en bloc més utilitzat és l'AES (Advanced Encryption Standard) ja que està establert com a estàndard per el NIST des de l'any 2002.

4.1 Definició de les xifres de bloc

Les xifres de bloc són un dels elements més importants en criptografia i es fan servir en diferents contextos. D'una banda, es poden fer servir directament en esquemes de xifrat per tal de proporcionar confidencialitat. D'altra banda però, també es fan servir com a primitives bàsiques en altres esquemes criptogràfics, com ara els generadors pseudoaleatoris, les funcions hash o els codis d'autenticació de missatges (coneguts per les seves sigles en anglès, MAC de *Message Authentication Codes*).

Una **xifra de bloc** és una funció que rep un bloc b d'n bits de text en clar i retorna un text xifrat c també d'n bits:

$$c = E_k(b)$$

Diem que n és, aleshores, la **mida de bloc** del criptosistema.

Noteu que la funció rep com a paràmetre el valor k, que representa la clau. La mida de la clau és la longitud en bits de k.

Per tal d'assegurar que al desxifrar un text xifrat amb E (amb una mateixa clau k) obtenim el text original, la funció E ha de ser invertible. Així doncs, les xifres de bloc diposen també d'una funció de desxifrat, que realitza el procés invers de la de xifrat:

$$b = D_k(c)$$

La majoria de vegades que fem servir un criptosistema de bloc voldrem xifrar contingut que supera la mida del bloc del criptosistema utilitzat. En aquests casos, el que es fa és partir el text que cal xifrar, m, en diversos blocs, $m_1, m_2, \ldots$, cada un dels quals té la llargada corresponent al bloc per a xifrar (n bits), i xifrar cadascun dels fragments. El procediment a seguir per xifrar cadascun dels fragments queda determinat per el **mode d'operació**.

4.1.1 Modes d'operació

El mode d'operació més senzill és coneix com a **ECB** (de l'anglès, *Electronic Code Book*) i consisteix a xifrar cada un dels blocs del missatge en clar, m_i, de manera individual, fent servir la mateixa clau. Així, s'obtenen els blocs xifrats c_i, que es concatenen per formar el text xifrat c. La Figura 4.1 esquematitza el procés de xifrat en mode ECB.

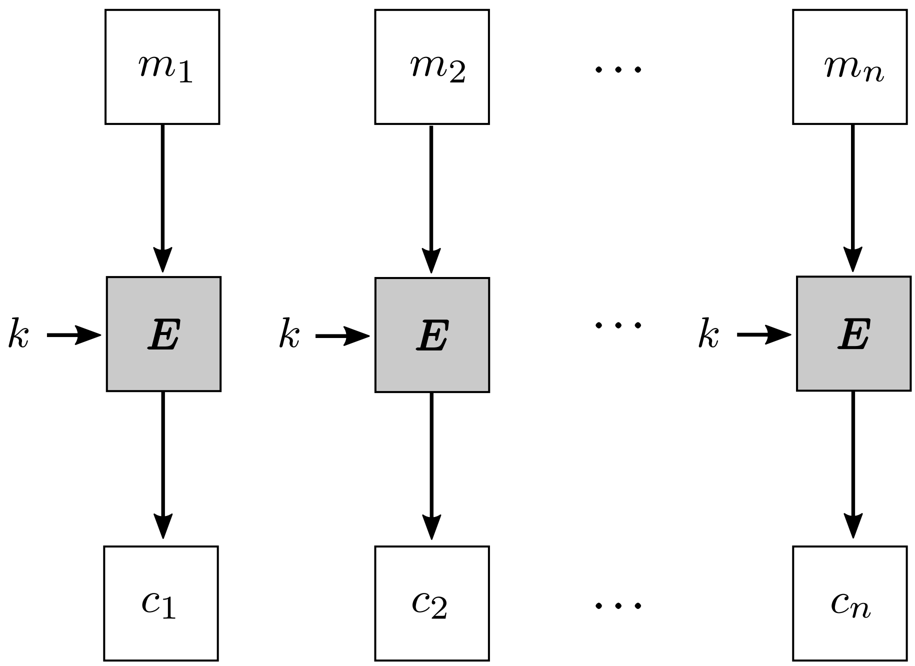


Figura 4.1: Esquema de xifrat amb el mode ECB.

Les propietats principals que ens ofereix el mode ECB són:

1. Els blocs de text en clar idèntics resulten en blocks xifrats també idèntics (si es fa servir la mateixa clau).
2. Cada bloc es xifra de manera independent als altres blocs.

3. Permet accés aleatori al contingut, és a dir, és possible desxifrar un bloc i sense haver de desxifrar els anteriors.
4. Els errors no es propaguen: un error en un bloc afecta només a aquell bloc.

Com a conseqüència immediata d'aquestes propietats, el mode ECB és vulnerable a certs atacs. D'una banda, per la propietat 1) un atacant que observi el text xifrat pot aprendre directament si el text original conté blocs iguals. A més, aquesta propietat també pot facilitar els atacs de tipus estadístic per a obtenir la clau k. Així mateix, el mode ECB no és capaç d'amagar els patrons en les dades. D'altra banda, per la propietat 2), un atacant pot reordenar el text xifrat, fent que al desxifrar-se s'obtingui el text en clar reordenat, sense que el receptor pugui detectar el canvi. Addicionalment, un atacant també pot inserir blocs de text xifrat o eliminar-ne, sense que el desxifrat posterior falli.

Per tal d'exemplificar les conseqüències de fer servir el mode ECB per a xifrar dades de mida superior al bloc, procedim a xifrar una imatge amb aquest mode, i a visualitzar el text xifrat resultant també en forma d'imatge (veure Figura 4.2.

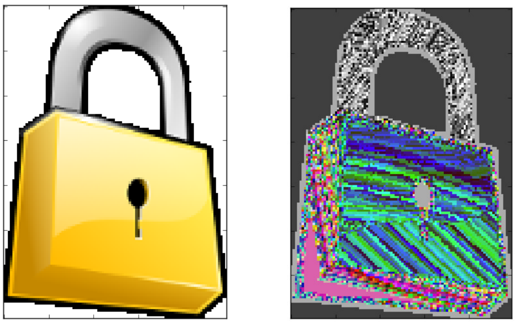


Figura 4.2: Exemple de xifrat d'una imatge amb ECB.

La imatge de l'esquerra correspon a la imatge en clar i la de la dreta és el resultat de xifrar la primera imatge fent servir el mode d'operació ECB. Com es pot apreciar, tot i que detalls concrets de la imatge original no es revel·len en la versió xifrada (per exemple, el color), la silueta de la imatge queda perfectement reconeixible.

Exercici 4.1 Suposem un esquema de xifrat de bloc amb mida de bloc de 2 bits i mida de clau també de 2 bits que implementa la següent funció de xifrat E:

Entrada	k	Sortida	Entrada	k	Sortida
00	00	11	00	01	00
01	00	10	01	01	01
10	00	01	10	01	10
11	00	00	11	01	11
00	10	01	00	11	10
01	10	11	01	11	00
10	10	00	10	11	11
11	10	10	11	11	01

Xifreu el missatge $m = 1001100100110000$ amb $k = 10$ fent servir la funció de xifrat E i el mode d'operació ECB.

El mode **CBC** (de l'anglès, *Cipher Block Chaining*) consisteix en l'encadenament dels blocs per al xifratge, de manera que es crea una dependència del xifratge de cada bloc amb l'immediatament anterior. De nou, cada bloc es xifra amb la mateixa clau k, però el text que es xifra no és directament el bloc en clar, sinó el resultat d'una XOR entre el bloc en clar i el bloc xifrat anterior. La Figura 4.3 esquematitza el funcionament del mode CBC.

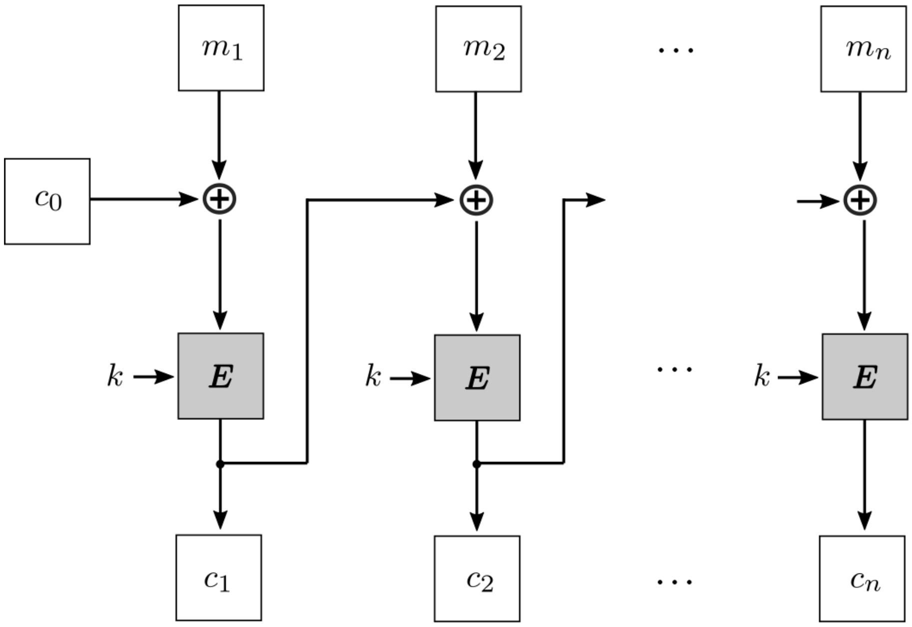


Figura 4.3: Esquema de xifrat amb el mode CBC.

Suposem un xifratge de bloc amb una clau k, una funció de xifratge E i una de desxifratge D. Si $m_1,\dots,m_m$ són els blocs de text en clar que cal xifrar, mitjançant el sistema CBC el xifratge del bloc m_i es porta a terme de la manera següent:

$$c_i = E_k(m_i \oplus c_{i-1}).$$

Per a fer-ne el desxifratge també ens cal partir del text xifrat anterior, i aleshores hem d'executar l'operació següent:

$$D_k(c_i) \oplus c_{i-1} = D_k(E_k(m_i \oplus c_{i-1})) \oplus c_{i-1}) = (m_i \oplus c_{i-1}) \oplus c_{i-1})m_i.$$

Per a xifrar el primer bloc necessitarem un bloc inicial aleatori, c_0, que no cal que sigui secret. Aleshores, incloent aquest nou vector inicial en el xifratge podrem obtenir dos textos en clar iguals però xifrats de manera diferent; així, encara que emprem la mateixa clau, k, només ens caldrà canviar el vector inicial, c_0, que, a més, pot incorporar una marca temporal.

En contraposició amb el mode ECB, si un atacant canvia l'ordre dels blocs xifrats amb CBC, aleshores el procés de desxifrat no es realitza correctament. Addicionalment, un error en un bloc xifrat afecta el desxifrat d'aquell bloc, però també del següent. Noteu que els blocs successius es desxifren ja correctament.

Amb aquesta estructura, el mode CBC aconsegueix ocultar els patrons del text en clar molt millor que el mode ECB. Si repetim el procediment de xifrar la imatge del cadenat fent servir ara el mode CBC, podem observar com ara en la Figura 4.4 no podem intuir el perfil de la imatge a partir de la imatge xifrada.

Exercici 4.2 Xifreu el mateix missatge m amb la funció E definida en l'exercici 4.1 i la clau $k = 10$, fent servir ara el mode d'operació CBC amb el vector inicial 10.

El mode de xifratge **CFB** (de l'anglès, *Cipher Feedback*) utilitza indirectament el xifrador de bloc, com veurem a continuació. Per això, la llargada dels blocs que s'han de xifrar no cal que sigui la mateixa que la

Figura 4.4: Exemple de xifrat d'una imatge amb CBC.

dels blocs del criptosistema amb què actua, sinó que pot ser més petita. L'esquema general de funcionament d'aquest mètode es mostra a la Figura 4.5.

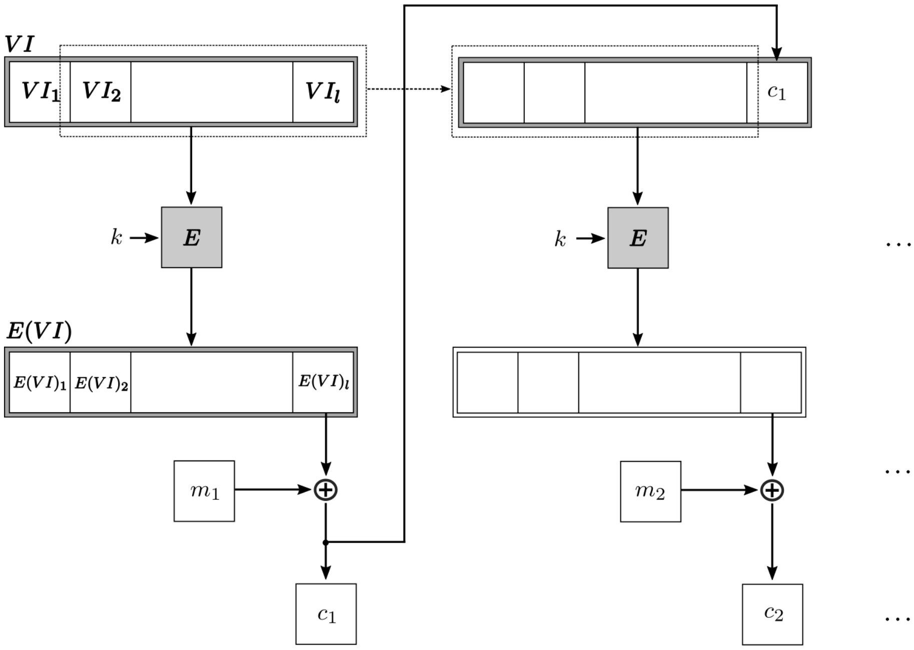


Figura 4.5: Esquema de xifrat amb el mode CFB.

Donat $m = m_1m_2\ldots$, en què m és el missatge de text en clar, i $m_1, m_2, \ldots$ representen els blocs de longitud n que formen el missatge, si considerem el vector inicial VI com una concatenació d'l blocs de longitud n, és a dir, $VI = VI_1VI_2\ldots VI_l$, on VI_i i té n bits de llargada, podrem calcular el xifratge del vector VI, $E(VI)$, mitjançant el criptosistema de bloc.

El resultat tindrà la mateixa llargada que VI i, per tant, el podrem descompondre de la mateixa manera que aquell:

$$E(VI) = E(VI)_1E(VI)_2\ldots E(VI)_l$$

Finalment, ja podrem xifrar el primer bloc de text en clar, m_1, fent la suma bit a bit amb el darrer bloc, $E(VI)_l$:

$$c_1 = m_1 \oplus E(VI)_l;$$

obtenim així el primer bloc xifrat de longitud n, c_1.

Per a xifrar el segon bloc, m_2 , tornarem a fer el mateix procés, però aquesta vegada prendrem com a vector inicial el vector format pels fragments següents:

$$VI = VI_2 VI_3 \dots VI_l c_1,$$

és a dir, hem desplaçat els blocs d'n bits cap a l'esquerra per afegir-hi el bloc c_1 i descartar-ne el VI_1. D'aquesta manera, el segon bloc de text xifrat l'obtenim fent l'operació següent:

$$c_2 = E(VI_b)_l \oplus m_2.$$

El procés es repeteix al llarg dels blocs de text que es vol xifrar: per al bloc següent es desplacen els blocs del vector inicial anterior, $VI_b, \dots$ a l'esquerra per afegir-hi el darrer bloc de text xifrat obtingut i anar aplicant el que ja hem descrit anteriorment.

Exercici 4.3 Xifreu el mateix missatge m amb la funció E definida en l'exercici 4.1 i la clau $k = 10$, fent servir ara el mode d'operació CFB amb el vector inicial 10.

El mode de xifratge **OFB** (de l'anglès, *Output Feedback*) utilitza el criptosistema de bloc com a generador pseudoaleatori. És un sistema molt semblant a l'anterior; l'única diferència que presenta és que el vector inicial es realimenta directament amb el resultat del xifratge de bloc abans de fer la suma bit a bit amb el bloc de text en clar, com es pot veure a la Figura 4.6.

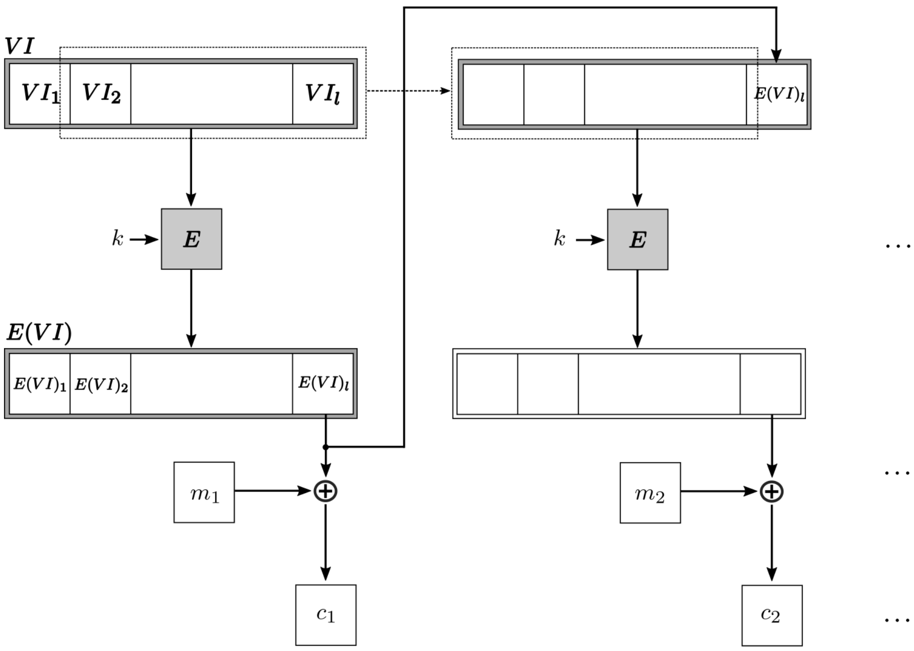


Figura 4.6: Esquema de xifrat amb el mode OFB.

Com que el xifrador de bloc actua com un generador pseudoaleatori, cal que els criptosistemes de bloc que emprem amb el mode OFB compleixin les característiques requerides per als generadors pseudoaleatoris, tant pel que fa a la impredictibilitat de la seqüència resultant com a la complexitat lineal.

Exercici 4.4 Xifreu el mateix missatge m amb la funció E definida en l'exercici 4.1 i la clau $k = 10$, fent servir ara el mode d'operació OFB amb el vector inicial 10.

El mode **CTR** (de l'anglès, *counter*) és similar a l'OFB, convertint també el criptosistema de bloc amb un xifrador de flux. La seqüència de xifrat es genera xifrant succesius valors d'un comptador (d'aquí en sorgeix el seu nom), que pot ser qualsevol funció que tingui un període gran (veure Figura 4.7).

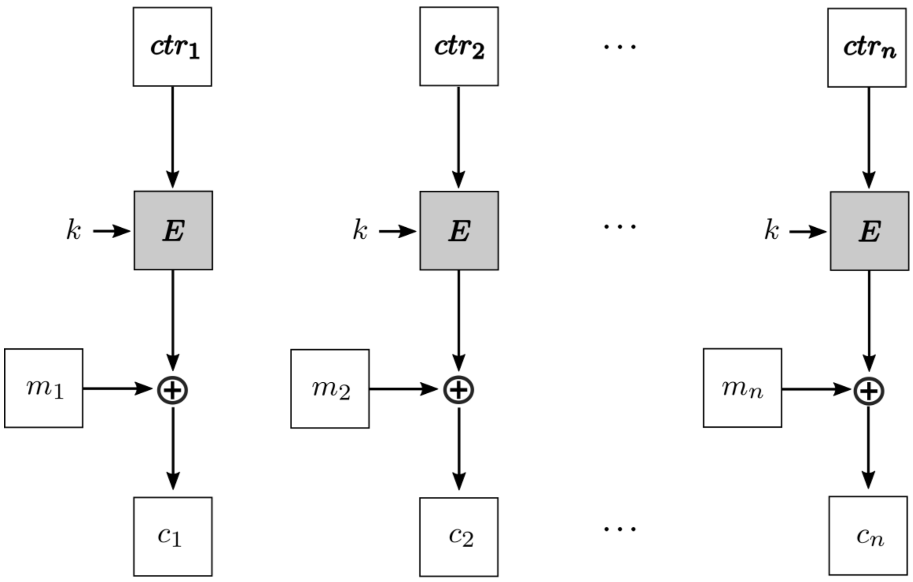


Figura 4.7: Esquema de xifrat amb el mode CTR.

Un ús habitual és fer servir un valor de nonce aleatori concatenat amb un comptador que s'incrementi d'un en un. Així, per exemple, si la mida de bloc del xifrador a utilitzar és de 128 bits, se selecciona una nonce de 64 bits i un comptador de 64 bits. Per a xifrar el primer bloc, es concatena la nonce amb el comptador inicialitzat a 0. Per a cada nou bloc, el comptador s'incrementa en 1. D'aquesta manera, es poden xifrar 2^{64} blocs amb la mateixa nonce.

El principal avantatge d'aquest mode d'operació és que permet paral·lelitzar tant el procés de xifrat com el de desxifrat, el que el fa addient per funcionar en dispositius amb més d'un processador. A més, permet accés aleatori (com el mode ECB).

Exercici 4.5 Xifreu el mateix missatge m amb la funció E definida en l'exercici 4.1 i la clau $k = 10$, fent servir ara el mode d'operació CTR amb el vector inicial 10.

4.2 El criptosistema AES

L'any 1998, els criptògrafs belgues Vincent Rijmen i Joan Daemen van desenvolupar l'algorisme anomenat (en reconeixement dels autors) criptosistema de Rijndael. Aquest criptosistema va ser triat pel NIST com a AES (de l'anglès, *Advanced Encryption Standard*) l'any 2000, reemplaçant el DES.

De fet, el Rijndael és una família d'algorismes de xifrat amb diferents mides de clau i de bloc. En concret, el Rijndael defineix blocs i claus de mida mínima 128 i màxima de 256, acceptant múltiples de 32 bits. L'AES n'és només un subconjunt, amb mida de bloc fixada a 128 bits.

El **criptosistema AES** xifra blocs de text en clar de **128 bits** de longitud. La longitud de les claus de xifratge que aquest criptosistema empra pot variar entre **128, 192 o 256** bits. Les operacions criptogràfiques es basen en un grup finit d'ordre 2^8.

4.2.1 Descripció del funcionament

El funcionament de l'AES es mostra en la Figura 4.8. Es basa en una transformació inicial seguida d'un nombre d'iteracions que varien entre 10 i 14, segons la longitud de la clau.

El nombre d'iteracions El nombre d'iteracions que es mostren en el gràfic és $n-1$ perquè la iteració final, tot i que es considera iteració, no conté la funció mixColumn.

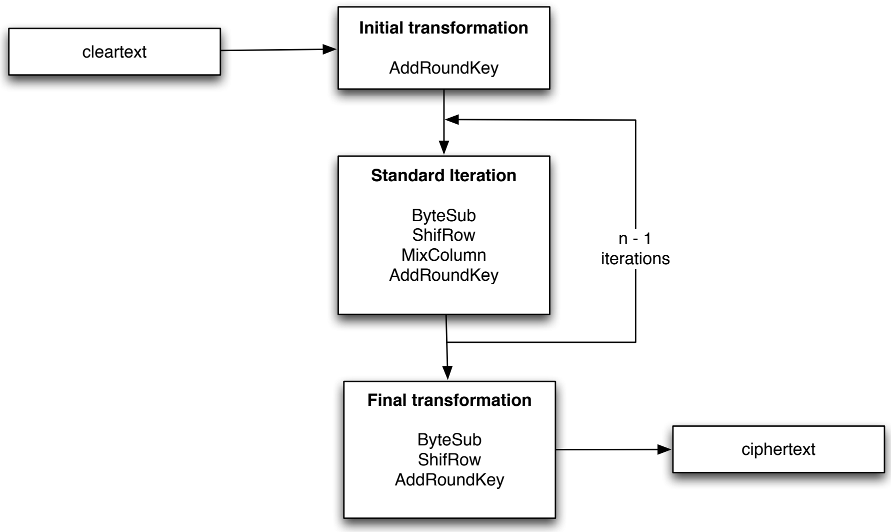


Figura 4.8: Estructura de l'AES.

La taula següent mostra el nombre exacte d'iteracions Nr en funció del nombre de paraules de 32 bits que té la clau que s'utilitza per xifrar (Nk):

$Nk=4$	$Nk=6$	$Nk=8$
10	12	14

La unitat bàsica d'informació amb què treballa l'AES és el byte. Totes les cadenes de bits (textos en clar i claus) es representen amb matrius de bytes. Per exemple, una cadena de 128 bits de text en clar:
$m = m_1 m_2 \cdots m_{127} m_{128}$

es representarà amb 16 bytes de la següent manera:
$a_{0,0} = m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8$
$a_{1,0} = m_9 m_{10} m_{11} m_{12} m_{13} m_{14} m_{15} m_{16}$
$\ldots$

$a_{3,3} = m_{121}m_{122}m_{123}m_{124}m_{125}m_{126}m_{127}m_{128}$

i aquests bytes es poden expressar de forma matricial:

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Les diferents funcions que executa l'AES (per exemple, AddRoundKey, ByteSub, etc.) tenen com a entrada i com a sortida una matriu de bytes com l'anterior.

Les matrius intermèdies amb què treballa el criptosistema AES s'anomenen **matrius d'estat**. Les matrius d'estat són matrius 4×4 i cada element de la matriu és un byte. Els elements de cada estat es denoten per s_{ij}, on i determina la fila i j la columna.

Les operacions "suma" i "producte" de bytes que executa l'AES no són les operacions convencionals que coneixem. En concret, l'AES considera els bytes en una representació de polinomi. Cada byte b es pot representar amb 8 bits:

$b = [b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0]$, on $b_i \in \{0, 1\}$

Aquest conjunt de bits es pot expressar com els coeficients d'un polinomi de grau 7:

$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i$. Per exemple, el byte 01100011 té com a representació el polinomi $x^6 + x^5 + x + 1$

Per tal de simplificar la notació, representarem els bytes en notació hexadecimal. Així, l'element 01100011 en base binària es representarà per un 63 en base hexadecimal, ja que $01100011_{(2} = 99_{(10} = 63_{(16}$.

Donades aquestes representacions, considerem que la "suma" i el "producte" es defineixen de la manera següent.

Siguin les representacions binàries dels bytes $x = (x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$ i $y = (y_7, y_6, y_5, y_4, y_3, y_2, y_1, y_0)$.

Definim l'operació suma:
$x \oplus y = (x_7 \oplus y_7, x_6 \oplus y_6, x_5 \oplus y_5, x_4 \oplus y_4, x_3 \oplus y_3, x_2 \oplus y_2, x_1 \oplus y_1, x_0 \oplus y_0)$
on $\oplus$ denota l'operació XOR bit a bit.[1]

D'altra banda, definim l'operació producte:
$x \otimes y = (x_7x^7 + x_6x^6 + x_5x^5 + x_4x^4 + x_3x^3 + x_2x^2 + x_1x + x_0)(y_7x^7 + y_6x^6 + y_5x^5 + y_4x^4 + y_3x^3 + y_2x^2 + y_1x + b_0) \pmod{x^8 + x^4 + x^3 + x + 1}$.

Exemple 4.1 Càlcul de "suma" i "producte":

Donats els bytes x i y:

$x = 57_{(16} = 01010111_{(2} = x^6 + x^4 + x^2 + x + 1$

$y = 83_{(16} = 10000011_{(2} = x^7 + x + 1$,

calculem la suma i el producte de bytes:

[1] Recordeu que l'operació XOR queda definida per: $1 \oplus 0 = 0 \oplus 1 = 1, 1 \oplus 1 = 0 \oplus 0 = 0$.

$x \oplus y = 57_{(16} \oplus 83_{(16} = D4_{(16}$,

ja que: $01010111_{(2} \oplus 10000011_{(2} = 11010100_{(2} = D4_{(16}$.

D'altra banda, pel "producte" tenim:
$x \otimes y = 57_{(16} \otimes 83_{(16} = C1_{(16}$,

ja que:

$(x^6+x^4+x^2+x+1)\cdot(x^7+x+1) \pmod{x^8+x^4+x^3+x+1} =$

$= (x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1) \pmod{x^8+x^4+x^3+x+1} =$

$= x^7+x^6+1 = 11000001_{(2} = C1_{(16}$.

Un cop vistes aquestes representacions, ja podem passar a veure el funcionament de l'algorisme.

4.2.2 Detall d'una iteració

En el gràfic del funcionament general de l'algorisme es mostra com l'AES realitza, primer, una transformació inicial del text d'entrada, aplicant la funció AddRoundKey. Després, s'executen $n-1$ iteracions, cadascuna de les quals aplica les funcions ByteSub, ShiftRow, MixColumn i AddRoundKey. Finalment, es realitza una transformació final que executa tres de les quatre funcions anteriors, deixant d'aplicar la funció MixColumn.

A més d'aquestes operacions, en la transformació inicial el text en clar s'ha de convertir en una matriu d'estat, que serà utilitzada per la funció AddRoundKey. De manera similar, la transformació final transforma la sortida de la funció AddRoundKey (que és una matriu d'estat) en el text xifrat final.

Passem a descriure cada una de les funcions que s'executen en cada iteració.

4.2.3 Funció AddRoundKey

La funció AddRoundKey s'utilitza tant en les transformacions inicial i final com en les iteracions estàndard.

La funció **AddRoundKey** fa una suma XOR de la matriu d'estat amb cada byte de la subclau $K(i)$ corresponent. En el cas de la transformació inicial tenim, $i=0$; per tant, utilitzem la primera subclau $K(0)$.

Les subclaus

L'índex i denota la subclau de 128 bits que es fa servir en la i-èsima iteració tenint en compte que $K(0)$ serà la subclau que es farà servir per a la transformació inicial. Podeu trobar la descripció de com s'obtenen les subclaus a partir de la clau inicial de xifratge en el subapartat 4.2.7 d'aquest capítol.

Exemple 4.2 Càlcul de la funció AddRoundKey

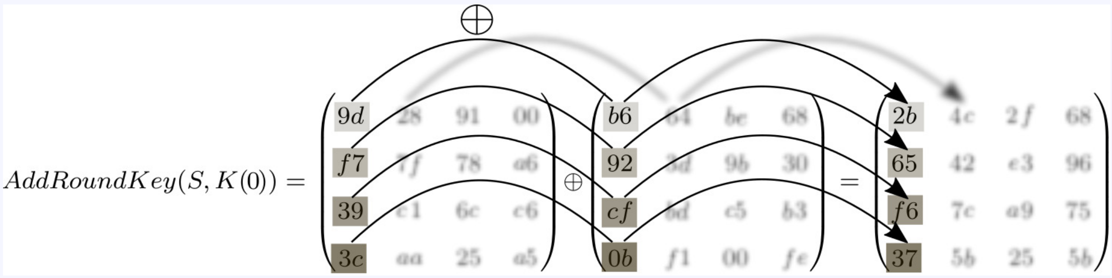
Considerem la subclau: $K(0) = b692cf0b643dbdf1be9bc5006830b3fe$

i la matriu d'estat $S = \begin{pmatrix} 9d & 28 & 91 & 00 \\ f7 & 7f & 78 & a6 \\ 39 & c1 & 6c & c6 \\ 3c & aa & 25 & a5 \end{pmatrix}$

El resultat d'aplicar la funció AddRoundKey serà:

$$AddRoundKey(S,K(0)) = \begin{pmatrix} 9d & 28 & 91 & 00 \\ f7 & 7f & 78 & a6 \\ 39 & c1 & 6c & c6 \\ 3c & aa & 25 & a5 \end{pmatrix} \oplus \begin{pmatrix} b6 & 64 & be & 68 \\ 92 & 3d & 9b & 30 \\ cf & bd & c5 & b3 \\ 0b & f1 & 00 & fe \end{pmatrix} = \begin{pmatrix} 2b & 4c & 2f & 68 \\ 65 & 42 & e3 & 96 \\ f6 & 7c & a9 & 75 \\ 37 & 5b & 25 & 5b \end{pmatrix}$$

Fixeu-vos que la suma XOR de les matrius correspon a la suma XOR de cada una de les seves entrades. Així, per exemple, la primera posició de la transformació val $2B$, ja que $9D \oplus B6 = 10011101 \oplus 10110110 = 2B$.



4.2.4 Funció ByteSub

La funció **ByteSub** aplica una substitució no lineal dels bytes de la matriu d'estat.

La funció ByteSub[2] rep com a entrada una matriu d'estat A, hi aplica una transformació S i obté una altra matriu d'estat B, de manera que $b_{ij} = S(a_{ij})$. La transformació de cada byte de la matriu es realitza de manera independent.

Les caixes S de l'AES

Les caixes S de l'AES són una matriu de 256 elements que s'utilitza com una taula de consulta. Normalment es representa com una matriu de 16 files i 16 columnes. Si representem cada byte a processar amb dos caracters hexadecimals xy, aleshores el valor x indica la fila i el valor y la columna de la posició on es troba el byte resultant.

Taula de consulta

Una taula de consulta (en anglès, *lookup table*) és una estructura de dades que substitueix una execució algorísmica per una operació d'indexació. Normalment l'objectiu d'utilitzar taules de consulta és reduir el temps d'obtenció del resultat esperat.

[2] La funció **ByteSub** apareix amb aquesta denominació a la proposta inicial del criptosistema de Rijndael. A la publicació de l'AES en l'estàndard FIP-197, la funció s'anomena SubBytes. Sigui quin sigui el nom que se li doni, en els dos casos és la mateixa funció.

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0y	63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
1y	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2y	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3y	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
4y	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5y	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6y	d0	ef	aa	fb	43	4d	33	85	45	f9	2	7f	50	3c	9f	a8
7y	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8y	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9y	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ay	e0	32	3a	0a	49	6	24	5c	c2	d3	ac	62	91	95	e4	79
by	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	8
cy	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dy	70	3e	b5	66	48	3	f6	0e	61	35	57	b9	86	c1	1d	9e
ey	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fy	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Exemple 4.3 Càlcul de la funció ByteSub

$$S = \begin{pmatrix} b5 & b1 & b9 & b5 \\ c9 & cc & c5 & c8 \\ 17 & 11 & 1b & 15 \\ 9e & 99 & 92 & 9d \end{pmatrix}$$

Calculem la transformació de la primera entrada de la matriu, $S_{00} = b5$. Busquem el valor de primera component, b a les files de la taula de les caixes S, i el valor de la segona component 5 a les columnes. Això ens indica que el valor que hi ha a la intersecció serà el valor resultant, en aquest cas el $d5$. Si fem el mateix procés amb tots els elements de la matriu tenim com a resultat:

$$ByteSub(S) = \begin{pmatrix} d5 & c8 & 56 & d5 \\ dd & 4b & a6 & e8 \\ f0 & 82 & af & 59 \\ 0b & ee & 4f & 5e \end{pmatrix}$$

4.2.5 Funció ShiftRow

La funció **ShiftRow** desplaça les files de la matriu d'estat de manera que la fila zero es deixa igual, la fila 1 es desplaça una posició a l'esquerra, la fila 2 es desplaça dues posicions a l'esquerra i la fila 3, tres posicions a l'esquerra.

Exemple 4.4 Càlcul de la funció ShiftRow

Si suposem la matriu d'estat:

$$S = \begin{pmatrix} d5 & c8 & 56 & d5 \\ dd & 4b & a6 & e8 \\ f0 & 82 & af & 59 \\ 0b & ee & 4f & 5e \end{pmatrix}$$

Podem realitzar el càlcul de la funció ShiftRow tal com es mostra a la figura següent, deixant la fila zero de la matriu sense modificar i desplaçant les files 1, 2 i 3, una, dues i tres posicions, respectivament:

$$ShiftRow(S) = ShiftRow\left(\begin{pmatrix} d5 & e8 & 56 & d5 \\ dd & 4b & a6 & e8 \\ f0 & 82 & af & 59 \\ 0b & ee & 4f & 5e \end{pmatrix}\right) = \begin{pmatrix} d5 & c8 & 56 & d5 \\ 4b & a6 & e8 & dd \\ af & 59 & f0 & 82 \\ 5e & 0b & ee & 4f \end{pmatrix}$$

La matriu d'estat resultant de la transformació serà doncs:

$$ShiftRow(S) = \begin{pmatrix} d5 & c8 & 56 & d5 \\ 4b & a6 & e8 & dd \\ af & 59 & f0 & 82 \\ 5e & 0b & ee & 4f \end{pmatrix}$$

4.2.6 Funció MixColumns

La funció **MixColumns** barreja les columnes de la matriu d'estat a partir d'operacions polinomials.

Concretament, aquesta funció considera les columnes de la matriu d'estat com polinomis de grau 3. Cada columna es multiplica pel polinomi $c(x) = \text{“03”}x^3 + \text{“01”}x^2 + \text{“01”}x + \text{“02”}$ i el resultat es redueix mòdul $x^4 + 1$. Aquest producte dels polinomis es pot escriure com un producte de matrius:

$$\begin{pmatrix} s'_{0j} \\ s'_{1j} \\ s'_{2j} \\ s'_{3j} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_{0j} \\ s_{1j} \\ s_{2j} \\ s_{3j} \end{pmatrix}$$

Tingueu en compte que les operacions “suma” i “producte” entre els elements de la matriu i els del vector columna són les operacions $\oplus$ i $\otimes$ definides en el subapartat anterior.

El polinomi $c(x)$ és coprimer amb $x^4 + 1$ i, per tant, invertible. D'aquesta manera, l'operació MixColumns es pot desfer multiplicant cada columna per el polinomi $d(x)$ tal que:

$$c(x) \otimes d(x) = \text{“01''}$$

El polinomi $d(x)$ és doncs “$0B$”x^3 + “$0D$”x^2 + “09”x + “$0E$”.

Exemple 4.5 Càlcul de la funció MixColumns

Suposem una matriu d'estat: $S = \begin{pmatrix} d5 & c8 & 56 & d5 \\ 4b & a6 & e8 & dd \\ af & 59 & f0 & 82 \\ 5e & 0b & ee & 4f \end{pmatrix}$

Per a obtenir la transformació de la primera columna calcularem:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} d5 \\ 4b \\ af \\ 5e \end{pmatrix}$$

Això ens donarà un vector columna de quatre bytes determinats pels valors següents:

$$\begin{pmatrix} (02 \otimes d5) \oplus (03 \otimes 4b) \oplus (01 \otimes af) \oplus (01 \otimes 5e) \\ (01 \otimes d5) \oplus (02 \otimes 4b) \oplus (03 \otimes af) \oplus (01 \otimes 5e) \\ (01 \otimes d5) \oplus (01 \otimes 4b) \oplus (02 \otimes af) \oplus (03 \otimes 5e) \\ (03 \otimes d5) \oplus (01 \otimes 4b) \oplus (01 \otimes af) \oplus (02 \otimes 5e) \end{pmatrix}$$

Per exemple, vegem quant val la segona posició del vector columna:

$(01 \otimes d5) \oplus (02 \otimes 4b) \oplus (03 \otimes af) \oplus (01 \otimes 5e)$

Si passem els valors hexadecimals a representació polinòmica (passant per la seva representació binària) tenim:

Hexadecimal	Binari	Polinomi
01	00000001	1
d5	11010101	$x^7+x^6+x^4+x^2+1$
02	00000010	x
4b	01001011	x^6+x^3+x+1
03	00000011	$x+1$
af	10101111	$x^7+x^5+x^3+x^2+x+1$
5e	01011110	$x^6+x^4+x^3+x^2+x$

Si ara fem els càlculs, resulta:

$(\text{“01”} \otimes \text{“}D5\text{”}) = (1)(x^7+x^6+x^4+x^2+1) \pmod{x^8+x^4+x^3+x+1} = x^7+x^6+x^4+x^2+1 \rightarrow 11010101$

$(\text{“02”} \otimes \text{“}4B\text{”}) = (x)(x^6+x^3+x+1) \pmod{x^8+x^4+x^3+x+1} = x^7+x^4+x^2+x \rightarrow 10010110$

$(\text{“03”} \otimes \text{“}AF\text{”}) = (x+1)(x^7+x^5+x^3+x^2+x+1) \pmod{x^8+x^4+x^3+x+1} = x^7+x^6+x^5+x^3+x \rightarrow 11101010$

$(\text{“01”} \otimes \text{“}5E\text{”}) = (1)(x^6+x^4+x^3+x^2+x) \pmod{x^8+x^4+x^3+x+1} = x^6+x^4+x^3+x^2+x \rightarrow 01011110$

Finalment, fem la XOR:

$11010101 \oplus 10010110 \oplus 11101010 \oplus 01011110 \oplus 11110111 \rightarrow f7$

Concretament, el resultat de tots els elements de la primera columna és:

$$\begin{pmatrix} (02 \otimes d5) \oplus (03 \otimes 4b) \oplus (01 \otimes af) \oplus (01 \otimes 5e) \\ (01 \otimes d5) \oplus (02 \otimes 4b) \oplus (03 \otimes af) \oplus (01 \otimes 5e) \\ (01 \otimes d5) \oplus (01 \otimes 4b) \oplus (02 \otimes af) \oplus (03 \otimes 5e) \\ (03 \otimes d5) \oplus (01 \otimes 4b) \oplus (01 \otimes af) \oplus (02 \otimes 5e) \end{pmatrix} = \begin{pmatrix} 9d \\ f7 \\ 39 \\ 3c \end{pmatrix}$$

I el resultat de la funció MixColumns sobre tota la matriu d'estat és:

$$MixColumns(S) = \begin{pmatrix} 9d & 28 & 91 & 00 \\ f7 & 7f & 78 & a6 \\ 39 & c1 & 6c & c6 \\ 3c & aa & 25 & a5 \end{pmatrix}$$

4.2.7 Generació de subclaus

A l'igual que la majoria de criptosistemes en bloc, l'algorisme de Rijndael treballa amb diferents subclaus en cada iteració. Aquestes subclaus s'obtenen per l'aplicació d'una funció d'ampliació a la clau de xifratge inicial.

La funció d'expansió genera, a partir de les Nk paraules de 32 bits de clau de xifratge, $K = (K_0, K_1, \ldots, K_{Nk-1})$, una clau estesa $W = (W_0, W_1, \ldots, W_{4(Nr+1)-1})$ que conté $4(Nr+1)$ paraules de 32 bits. Cada iteració de l'algorisme de xifrat farà servir 4 paraules de 32 bits i caldran 4 paraules addicionals per a la inicialització. Si denotem per $K(i)$ cada una de les subcadenes de W de 4 paraules de 32 bits tindrem que $K(i)$ és la subclau que s'utilitza en la i-èssima iteració. Gràficament les subclaus de cada iteració en relació amb la clau estesa es poden expressar com:

$$W = (\underbrace{W_0, W_1, W_2, W_3,}_{K(0)} \underbrace{W_4, W_5, W_6, W_7,}_{K(1)} \ldots \underbrace{W_{4Nr}, \ldots, W_{4(Nr+1)-1}}_{K(Nr)})$$

Els paràmetres Nk **i** Nr

Recordem que els paràmetres (Nk, Nr), que representen respectivament la mida de la clau en paraules de 32 bits i el número d'iteracions, poden prendre els valors $(4,10)$, $(6,12)$ i $(8,14)$.

Així, la transformació inicial utilitza la subclau $K(0)$ formada per les primeres 4 paraules de W i en cada una de les Nr iteracions s'utilitzen 4 paraules. D'aquesta manera, per valors d'Nk de 4, 6 i 8 es generaran, respectivament, claus exteses W de 44, 52 i 60 paraules de 32 bits (que corresponen a 1408, 1664 i 1920 bits).

L'algorisme d'expansió de clau consta de dues fases:

- Fase d'inicialització, on la clau de xifratge és copia integrament a les primeres posicions de la clau extesa. És a dir:
$$W_i = K_i, \forall i = 0, \ldots, Nk-1$$
- Fase d'expansió, on s'agafa l'última paraula calculada i s'extén. L'algorisme que implementa aquesta fase queda descrit pel següent pseudocodi:

```
for (i = Nk ; i < 4(Nr + 1); i++)
        temp = W_{i-1}
        if  i = 0  mod Nk  then
                temp = SubWord(RotWord(temp)) ⊕ Rcon[i/Nk]
        else if ((Nk > 6) and (i mod Nk = 4)) then
                temp = SubWord(temp)
        endif
        W_i = W_{i-Nk} ⊕ temp
```

La fase d'expansió fa servir dues funcions: SubWord i RotWord. La funció SubWord és la mateixa funció que ByteSub (definida anteriorment). La funció RotWord simplement fa una permutació cíclica a la paraula de 4 bytes, és a dir, si tenim $[a0, a1, a2, a3]$ com a entrada, la sortida serà $[a1, a2, a3, a0]$. D'altra banda també es fa servir la constant $Rcon[i]$ que val $Rcon[i] = [x^{i-1}, \text{“00”}, \text{“00”}, \text{“00”}]$. Recordeu que x en hexadecimal val “02” ja que correspon a la representació en binari de 00000010.

L'esquema següent resumeix el procés d'expansió de claus per al cas $Nk = 4$, és a dir, per a claus de 128 bits. En aquest cas, si considerem la clau $K = (K_0K_1K_2K_3)$, aleshores els valors $W_0 \ldots W_3$ contindrien la clau inicial K, i la resta de valors (fins a W_{43}) es calcularien en funció d'aquestes quatre paraules inicials.

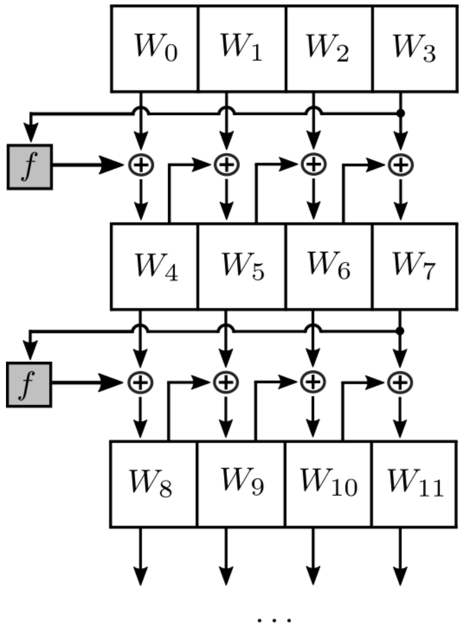


Figura 4.9: Esquema d'expansió de claus de l'AES per a $Nk = 4$

Noteu que l'esquema inclou la funció f, que correspon a aplicar $SubWord(RotWord(temp)) \oplus Rcon[i/Nk]$ sobre el valor que es rep a l'entrada.

Exemple 4.6 Càlcul de l'expansió de claus

Suposem que la longitud de la clau és de 128 bits, és a dir, $Nk = 4$ paraules de 32 bits i que la clau de xifrat (representada en hexadecimal[a]) correspon a:

$$K = \underbrace{\texttt{00 01 02 03}}_{K_0}\ \underbrace{\texttt{04 05 06 07}}_{K_1}\ \underbrace{\texttt{08 09 0A 0B}}_{K_2}\ \underbrace{\texttt{0C 0D 0E 0F}}_{K_3}$$

Amb aquests paràmetres tenim que el nombre d'iteracions és $Nr = 10$. Això vol dir que la clau extesa W tindrà $4 \cdot (10 + 1) = 44$ paraules de 32 bits.

Denotant per $K(i)$ la clau que es fa servir a l'i-èssima iteració. Els primers bytes de la clau extesa són els mateixos que els de la clau de xifratge:
$W_0 = \texttt{00 01 02 03}$
$W_1 = \texttt{04 05 06 07}$
$W_2 = \texttt{08 09 0A 0B}$
$W_3 = \texttt{0C 0D 0E 0F}$
Per tant:
$K(0) = W_0W_1W_2W_3 = \texttt{00010203 04050607 08090A0B 0C0D0E0F} = K$ Aquestes quatre paraules són les que es fan servir en la transformació inicial de l'algorisme.

La segona subclau serà:

$$
\begin{aligned}
W_4 =& W_0 \oplus SubWord(RotWord(W_3)) \oplus Rcon[1] \\
& SubWord(RotWord(W_3)) = RotWord(\texttt{0C 0D 0E 0F}) = \texttt{0D 0E 0F 0C} \\
& SubWord(\texttt{0D 0E 0F 0C}) = (\texttt{D7 AB 76 FE}) \\
& W_4 = \texttt{00 01 02 03} \oplus \texttt{D7 AB 76 FE} \oplus \texttt{01 00 00 00} = \texttt{D6 AA 74 FD} \\
W_5 =& W_1 \oplus W_4 = \texttt{04 05 06 07} \oplus \texttt{D6 AA 74 FD} = \texttt{D2 AF 72 FA} \\
W_6 =& W_2 \oplus W_5 = \texttt{08 09 0A 0B} \oplus \texttt{D2 AF 72 FA} = \texttt{DA A6 78 F1} \\
W_7 =& W_3 \oplus W_6 = \texttt{0C 0D 0E 0F} \oplus \texttt{DA A6 78 F1} = \texttt{D6 AB 76 FE}
\end{aligned}
$$

Per tant, la subclau $K(1) =$ `D6 AA 74 FD D2 AF 72 FA DA A6 78 F1 D6 AB 76 FE`.
La resta de la clau ampliada es calcula de la mateixa manera.

[a] Recordeu que cada caràcter hexadecimal permet representar 4 bits (és a dir, valors des de 0 fins a 15).

Exercici 4.6 Suposem que la clau de xifratge de 192 bits d'un xifrador AES expressada en hexadecimal és la següent:
`8E 73 B0 F7 DA 0E 64 52 C8 10 F3 2B 80 90 79 E5 62 F8 EA D2 52 2C 6B 7B`. Doneu-ne les dues primeres subclaus, és a dir, $K(0)$ i $K(1)$.

Exercici 4.7 Donat un xifrador Rijndael amb clau de xifratge K i un bloc de text per xifrar B:

```
K = 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C
B = 32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34
```

Quantes iteracions cal fer per xifrar aquest bloc de text en clar amb aquesta clau? Quina és la matriu d'estat a l'inici de la segona iteració?

4.2.8 Desxifrat

En el subapartats anteriors hem definit amb tot detall les operacions de xifratge de l'AES. Totes les funcions que s'utilitzen en el procés de xifratge (ByteSub, ShiftRow, MixColumn i AddRoundKey) són invertibles i, per tant, se'n pot definir la corresponent funció inversa.

Si les funcions definides en el xifratge s'apliquen en l'ordre oposat al que s'executen en el procés de xifratge, obtenim el procés de desxifratge del criptosistema.

4.3 Resum

En aquest capítol hem descrit el funcionament i les característiques principals dels esquemes de xifratge de bloc. En primer lloc n'hem descrit la seva estructura general. Després, hem passat a detallar com es poden fer servir les xifres de bloc per a xifrar textos de mida superior al bloc, descrivint diferents modes d'operació: ECB, CBC, CFB, OFC i CTR. Finalment, hem presentat el criptosistema de bloc més utilitzat avui en dia, l'AES, tot detallant-ne tant l'arquitectura com les funcions internes que fa servir.

4.4 Solucions dels exercicis

Exercici 4.1:

En primer lloc, procedim a separar el missatge en blocs de 2 bits, la mida de bloc de la funció de xifrat:

$m = 10\ 01\ 10\ 01\ 00\ 11\ 00\ 00$

Després procedim a aplicar la funció de xifrat a cada bloc individual, i concatenem els resultats:

$c = 00\ 11\ 00\ 11\ 01\ 10\ 01\ 01$

Exercici 4.2:

En primer lloc, procedim a separar el missatge en blocs de 2 bits. Després, per cada bloc, realitzem una xor amb el bloc xifrat anterior (fent servir el vector inicial com a bloc xifrat anterior per al primer bloc, M_1). Finalment, apliquem el xifrador de bloc sobre la sortida de la xor. El procés a seguir és doncs:

Bloc	$M_i \oplus C_{i-1}$	$C_i = E(M_i \oplus C_{i-1})$
$M_1 = 10$	$M_1 \oplus C_0 = 10 \oplus 10 = 00$	$E(00) = 01$
$M_2 = 01$	$M_2 \oplus C_1 = 01 \oplus 01 = 00$	$E(00) = 01$
$M_3 = 10$	$M_3 \oplus C_2 = 10 \oplus 01 = 11$	$E(11) = 10$
$M_4 = 01$	$M_4 \oplus C_3 = 01 \oplus 10 = 11$	$E(11) = 10$
$M_5 = 00$	$M_5 \oplus C_4 = 00 \oplus 10 = 10$	$E(10) = 00$
$M_6 = 11$	$M_6 \oplus C_5 = 11 \oplus 00 = 11$	$E(11) = 10$
$M_7 = 00$	$M_7 \oplus C_6 = 00 \oplus 10 = 10$	$E(10) = 00$
$M_8 = 00$	$M_8 \oplus C_7 = 00 \oplus 00 = 00$	$E(00) = 01$

El text xifrat correspon a la concatenació dels blocs xifrats: 0101101000100001.

Exercici 4.3:

En aquest cas, la mida de bloc del criptosistema és de 2 bits, pel que els blocs de text a xifrar poden ser com a molt de 2 bits. Agafem doncs blocs de text a xifrar de 2 bits i procedim a realitzar el procés de xifrat. Particionem el missatge M en blocs de 2 bits, i fem una xor de cada bloc amb el resultat de xifrar el bloc anterior, utilitzant el vector inicial com a bloc anterior per a la primera iteració:

Bloc	$E(C_{i-1})$	$C_i = E(C_{i-1}) \oplus M_i$
$M_1 = 10$	$E(C_0) = E(10) = 00$	$M_1 \oplus E(C_0) = 10 \oplus 00 = 10$
$M_2 = 01$	$E(C_1) = E(10) = 00$	$M_2 \oplus E(C_1) = 01 \oplus 00 = 01$
$M_3 = 10$	$E(C_2) = E(01) = 11$	$M_3 \oplus E(C_2) = 10 \oplus 11 = 01$
$M_4 = 01$	$E(C_3) = E(01) = 11$	$M_4 \oplus E(C_3) = 01 \oplus 11 = 10$
$M_5 = 00$	$E(C_4) = E(10) = 00$	$M_5 \oplus E(C_4) = 00 \oplus 00 = 00$
$M_6 = 11$	$E(C_5) = E(00) = 01$	$M_6 \oplus E(C_5) = 11 \oplus 01 = 10$
$M_7 = 00$	$E(C_6) = E(10) = 00$	$M_7 \oplus E(C_6) = 00 \oplus 00 = 00$
$M_8 = 00$	$E(C_7) = E(00) = 01$	$M_8 \oplus E(C_7) = 00 \oplus 01 = 01$

El text xifrat correspon a la concatenació dels blocs xifrats: 1001011000100001.

Exercici 4.4:

En aquest cas, la mida de bloc del criptosistema és de 2 bits, pel que els blocs de text a xifrar poden ser com a molt de 2 bits. Agafem doncs blocs de text a xifrar de 2 bits i procedim a realitzar el procés de xifrat. Particionem el missatge M en blocs de 2 bits, i fem una xor de cada bloc M_i amb el resultat de xifrar v_i, on $v_i = E(v_{i-1})$, amb $v_0 = VI$:

Bloc	$v_i = E(v_{i-1})$	$C_i = v_i \oplus M_i$
$M_1 = 10$	$v_1 = E(v_0) = E(10) = 00$	$v_1 \oplus M_1 = 00 \oplus 10 = 10$
$M_2 = 01$	$v_2 = E(v_1) = E(00) = 01$	$v_2 \oplus M_2 = 01 \oplus 01 = 00$
$M_3 = 10$	$v_3 = E(v_2) = E(01) = 11$	$v_3 \oplus M_3 = 11 \oplus 10 = 01$
$M_4 = 01$	$v_4 = E(v_3) = E(11) = 10$	$v_4 \oplus M_4 = 10 \oplus 01 = 11$
$M_5 = 00$	$v_5 = E(v_4) = E(10) = 00$	$v_5 \oplus M_5 = 00 \oplus 00 = 00$
$M_6 = 11$	$v_6 = E(v_5) = E(00) = 01$	$v_6 \oplus M_6 = 01 \oplus 11 = 10$
$M_7 = 00$	$v_7 = E(v_6) = E(01) = 11$	$v_7 \oplus M_7 = 11 \oplus 00 = 11$
$M_8 = 00$	$v_8 = E(v_7) = E(11) = 10$	$v_8 \oplus M_8 = 10 \oplus 00 = 10$

El text xifrat correspon a la concatenació dels blocs xifrats: 1000011100101110.

Exercici 4.5:

En aquest cas, com que la mida de bloc és molt petita, farem servir directament un comptador que s'incrementa d'un en un, sense incorporar cap nonce. Noteu que el comptador només té 4 valors, pel que la seqüència és repeteix. En una situació real, cal evitar aquest fet ja que compromet la seguretat del sistema.

Procedim doncs a particionar el missatge M en blocs de 2 bits, i fem una xor de cada bloc M_i amb el resultat de xifrar v_i, on v_i és un comptador cíclic que s'inicia amb el valor 00 i s'incrementa per cada nou bloc a xifrar:

Bloc	$v_i = E(i-1 \mod 4)$	$C_i = v_i \oplus M_i$
$M_1 = 10$	$v_1 = E(00) = 01$	$v_1 \oplus M_1 = 01 \oplus 10 = 11$
$M_2 = 01$	$v_2 = E(01) = 11$	$v_2 \oplus M_2 = 11 \oplus 01 = 10$
$M_3 = 10$	$v_3 = E(10) = 00$	$v_3 \oplus M_3 = 00 \oplus 10 = 10$
$M_4 = 01$	$v_4 = E(11) = 10$	$v_4 \oplus M_4 = 10 \oplus 01 = 11$
$M_5 = 00$	$v_5 = E(00) = 01$	$v_5 \oplus M_5 = 01 \oplus 00 = 01$
$M_6 = 11$	$v_6 = E(01) = 11$	$v_6 \oplus M_6 = 11 \oplus 11 = 00$
$M_7 = 00$	$v_7 = E(10) = 00$	$v_7 \oplus M_7 = 00 \oplus 00 = 00$
$M_8 = 00$	$v_8 = E(11) = 10$	$v_8 \oplus M_8 = 10 \oplus 00 = 10$

El text xifrat correspon a la concatenació dels blocs xifrats: 1110101101000010.

Exercici 4.6:

Atès que la clau de xifratge és de 192 bits, el nombre de paraules de 32 bits de la clau val $Nk = 6$; per tant, haurem d'aplicar l'algorisme per al cas $Nk \leq 6$.

Els primers bits de la clau estesa són exactament els mateixos bits de la clau de xifratge:

$$
\begin{aligned}
W_0 &= \texttt{8E 73 B0 F7}\\
W_1 &= \texttt{DA 0E 64 52}\\
W_2 &= \texttt{C8 10 F3 2B}\\
W_3 &= \texttt{80 90 79 E5}\\
W_4 &= \texttt{62 F8 EA D2}\\
W_5 &= \texttt{52 2C 6B 7B}
\end{aligned}
$$

Per tant:

$$
\begin{aligned}
K(0) = W_0W_1W_2W_3W_4W_5 &=\\
&= \texttt{8E73B0F7 DA 0E 64 52 C810F32B 809079E5 62F8EAD2 522C6B7B} =\\
&= K
\end{aligned}
$$

Si apliquem l'algorisme per al cas $Nk \leq 6$ amb els valors W_i anteriors obtenim:

$$\begin{aligned}
W_6 =& W_0 \oplus SubWord(RotWord(W_5)) \oplus Rcon[1] \\
& RotWord(W_5) = RotWord(\texttt{52 2C 6B 7B}) = \texttt{2C 6B 7B 52} \\
& SubWord(\texttt{2C 6B 7B 52}) = (\texttt{71 7F 21 00}) \\
W_6 =& \texttt{8E 73 B0 F7} \oplus \texttt{71 7F 21 00} \oplus \texttt{01 00 00 00} = \\
=& \texttt{8E 73 B0 F7} \oplus \texttt{70 7F 21 00} = \texttt{FE 0C 91 F7} \\
W_7 =& W_1 \oplus W_6 = \texttt{DA 0E 64 52} \oplus \texttt{FE 0C 91 F7} = \texttt{24 02 F5 A5} \\
W_8 =& W_2 \oplus W_7 = \texttt{C8 10 F3 2B} \oplus \texttt{24 02 F5 A5} = \texttt{EC 12 06 8E} \\
W_9 =& W_3 \oplus W_8 = \texttt{80 90 79 E5} \oplus \texttt{EC 12 06 8E} = \texttt{6C 82 7F 6B} \\
W_{10} =& W_4 \oplus W_9 = \texttt{62 F8 EA D2} \oplus \texttt{6C 82 7F 6B} = \texttt{0E 7A 95 B9} \\
W_{11} =& W_5 \oplus W_{10} = \texttt{52 2C 6B 7B} \oplus \texttt{0E 7A 95 B9} = \texttt{5C 56 FE C2}
\end{aligned}$$

Per tant, la subclau:
$K(1) = \texttt{FE 0C 91 F7 24 02 F5 A5 EC 12 06 8E 6C 82 7F 6B 0E 7A 95 B9 5C 56 FE C2}$.

Exercici 4.7:

Caldrà fer deu iteracions per a xifrar aquest bloc de text en clar, ja que tant la longitud de la clau és de 16 bytes; per tant, $Nk = 4$.

En la transformació inicial s'aplica la transformació addRoundKey. En el nostre cas:

$$AddRoundKey(S, K(0)) = \begin{pmatrix} 32 & 88 & 31 & e0 \\ 43 & 5a & 31 & 37 \\ f6 & 30 & 98 & 07 \\ a8 & 8d & a2 & 34 \end{pmatrix} \oplus \begin{pmatrix} 2b & 28 & ab & 09 \\ 7e & ae & f7 & cf \\ 15 & d2 & 15 & 4f \\ 16 & a6 & 88 & 3c \end{pmatrix} = \begin{pmatrix} 19 & a0 & 9a & e9 \\ 3d & f4 & c6 & f8 \\ e3 & e2 & 8d & 48 \\ be & 2b & 2a & 08 \end{pmatrix} = S_1$$

El resultat de la primera iteració correspondrà a executar les funcions ByteSub, ShiftRow, MixColumns i AddRoundKey. El resultat de la funció ByteSub sobre la matriu d'estat S_1 és:

$$ByteSub\left(\begin{pmatrix} 19 & a0 & 9a & e9 \\ 3d & f4 & c6 & f8 \\ e3 & e2 & 8d & 48 \\ b3 & 2b & 2a & 08 \end{pmatrix}\right) = \begin{pmatrix} d4 & e0 & b8 & 1e \\ 27 & bf & b4 & 41 \\ 11 & 98 & 5d & 52 \\ ae & f1 & e5 & 30 \end{pmatrix} = S_2$$

El resultat de la funció ShiftRow sobre la matriu d'estat S_2 és:

$$ShiftRow\left(\begin{pmatrix} d4 & e0 & b8 & 1e \\ 27 & bf & b4 & 41 \\ 11 & 98 & 5d & 52 \\ ae & f1 & e5 & 30 \end{pmatrix}\right) = \begin{pmatrix} d4 & e0 & b8 & 1e \\ bf & b4 & 41 & 27 \\ 5d & 52 & 11 & 98 \\ 30 & ae & f1 & e5 \end{pmatrix} = S_3$$

El resultat de la funció MixColumns sobre la matriu d'estat S_3 és:

$$MixColumns\left(\begin{pmatrix} d4 & e0 & b8 & 1e \\ bf & b4 & 41 & 27 \\ 5d & 52 & 11 & 98 \\ 30 & ae & f1 & e5 \end{pmatrix}\right) = \begin{pmatrix} 04 & e0 & 48 & 28 \\ 66 & cb & f8 & 06 \\ 81 & 19 & d3 & 26 \\ e5 & 9a & 7a & 4c \end{pmatrix} = S_4$$

Ara ens cal calcular el valor de la clau de la segona iteració, és a dir, el valor $K(1)$. Per fer-ho, aplicarem l'algorisme d'expansió de claus descrit en l'apartat 4.2.7, que ens permetrà obtenir la matriu:

$$\begin{pmatrix} a0 & 88 & 23 & 2a \\ fa & 54 & a3 & 6c \\ fe & 2c & 39 & 76 \\ 17 & b1 & 39 & 05 \end{pmatrix}$$

Una descripció gràfica per a la generació d'aquesta subclau la podeu trobar en aquest enllaç.

Finalment, el resultat de la funció AddRoundKey sobre la matriu d'estat S_4 resulta:

$$\begin{pmatrix} 04 & e0 & 48 & 28 \\ 66 & cb & f8 & 06 \\ 81 & 19 & d3 & 26 \\ e5 & 9a & 7a & 4c \end{pmatrix} \oplus \begin{pmatrix} a0 & 88 & 23 & 2a \\ fa & 54 & a3 & 6c \\ fe & 2c & 39 & 76 \\ 17 & b1 & 39 & 05 \end{pmatrix} = \begin{pmatrix} a4 & 68 & 6b & 02 \\ 9c & 9f & 5b & 6a \\ 7f & 35 & ea & 50 \\ f2 & 2b & 43 & 49 \end{pmatrix}$$

Així, el valor de la matriu d'estat a l'inici de la segona iteració valdrà:

$$S = \begin{pmatrix} a4 & 68 & 6b & 02 \\ 9c & 9f & 5b & 6a \\ 7f & 35 & ea & 50 \\ f2 & 2b & 43 & 49 \end{pmatrix}$$

4.5 Bibliografia

Christophe De Cannière and Bart Preneel (2005). *Trivium - Specifications*. Technical report.

Joan Daemen and Vincent Rijmen (2002). *The Design of Rijndael, AES - The Advanced Encryption Standard*. Springer-Verlag (238 pp.)

GSMA (2017). *GSMA The Mobile Economy 2017*.
https://www.gsmaintelligence.com/research/

James L. Massey (1969). *Shift-register synthesis and BCH decoding*. IEEE transactions on Information Theory, 15(1), 122-127.

Alfred Menezes, Paul van Oorschot, and Scott Vanstone (1996). *Handbook of Applied Cryptography*. CRC Press.
http://cacr.uwaterloo.ca/hac/

NIST (2001). *Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197.
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

Christof Paar and Jan Pelzl (2009). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer.

Andrew Rukhin,Juan Soto, James Nechvatal, et al. (2010). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* . NIST Special Publication.
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

Thomas Stockinger (2005). *GSM network and its privacy - the A5 stream cipher*.

5. Funcions hash

Les funcions hash són una primitiva criptogràfica cada vegada més important en diferents protocols i aplicacions criptogràfiques. Com veurem, una funció hash és una funció que permet obtenir un valor fixat de mida reduïda a partir d'una entrada arbitràriament gran. Gràcies a les propietats que ofereixen a aquest valor de sortida, els usos de les funcions hash són múltiples, des de la seva utilització per a l'autentificació d'informació sense l'ús de signatures digitals (fent servir criptografia simètrica) fins a la verificació de proves de treball en criptomonedes, passant per la generació de contrasenyes o la reducció de la complexitat de càlcul en un procés de signatura digital.

L'ús de les funcions hash cada vegada en més contextos implica que la seva importància també hagi anat augmentant. Com és sabut, la seguretat que ofereix un sistema criptogràfic és equivalent a la seguretat que ofereix el seu component més feble o insegur. Per tant, a mida que les funcions hash han anat incloent-se en nous sistemes, la robustesa de les funcions hash afecta de ple en la seguretat d'aquests sistemes. Aquest punt és molt rellevant perquè una vulnerabilitat en una funció hash implicaria una vulnerabilitat en tots els sistemes criptogràfics que l'utilitzen. Per exemple, si un atacant pogués predir la sortida d'una funció hash donada una entrada fixada, podria arribar a trencar la seguretat d'algunes criptomonedes.

En aquest capítol definirem què són les funcions hash i quines propietats presenten. Posteriorment, veurem com es poden construir utilitzant com a base un criptosistema de bloc. Repassarem també quines són les funcions hash més utilitzades, funcions hash construïdes específicament per a aquest propòsit i que no es basen en cap criptosistema de bloc. En concret, veurem en detall el funcionament de la funció hash SHA256. Finalment, enumerarem algunes de les múltiples aplicacions que tenen les funcions hash i també algunes propietats addicionals que es poden demanar a les funcions hash que són útils en algunes de les aplicacions esmentades.

5.1 Les funcions hash

Com ja hem avançat, les funcions hash s'utilitzen en múltiples aplicacions i la raó d'aquest fet recau en les seves propietats. En aquest apartat definirem acuradament què són les funcions hash i quina diferència hi ha entre una funció hash i una funció hash criptogràfica.

També descriurem el concepte d'atac a una funció hash i explicarem com n'indiquem el nivell de seguretat.

5.1.1 Definicions

Una **funció hash** de mida n és una funció que pren com a entrada un missatge (o cadena) d'una mida arbitràriament gran i en retorna una cadena de mida fixa n. A més, una funció hash és eficientment calculable i determinista, és a dir, donades dues entrades iguals sempre ens proporcionarà la mateixa sortida.

Mida d'una funció hash La mida de les funcions hash es determina en bits.

L'eficiència de les funcions hash és un element molt important ja que el seu ús està especialment indicat per a reduir missatges de mida molt gran. Per aquest motiu, la facilitat per tractar aquest tipus de missatges tan grans ha d'estar garantida per tal que la seva utilització no faci augmentar la complexitat del sistema que les utilitza. D'altra banda, malgrat sembli innecessari indicar el caràcter determinista de les funcions hash, és important ressaltar-lo perquè, com veurem més endavant, les funcions hash s'utilitzen de forma similar a un oracle aleatori i això pot induir a pensar que el seu funcionament no és determinista.

Exemple 5.1 Exemple de funció hash

Un exemple de funció hash de mida 3 dígits decimals seria la següent: $h(x) = x \pmod{1000}$

Aquesta funció hash retorna sempre, per a qualsevol mida de l'entrada, un valor fixat de 3 dígits, considerant que representem el nombre amb tres dígits incloent els zeros que calgui davant. Per exemple, $h(8472937003) = 8472937003 \pmod{1000} = 003$.

De la mateixa manera, la funció $h(x) = x \pmod{2^{256}}$ també seria una funció hash, en aquest cas de mida 256 bits.

Si bé les funcions hash tal com les acabem de definir tenen algunes aplicacions, la seva potència s'incrementa quan se li afegeixen un seguit de propietats que conformen el que es coneix com a funció hash criptogràfica.

Una **funció hash criptogràfica** és una funció hash, $h(x)$, amb les següents propietats:

1. Resistent a preimatge (o unidireccional): donat un valor y no és possible calcular una x tal que $h(x) = y$.
2. Resistent a segones preimatges (o resistent a col·lisions febles): donat un valor x tal que $y = h(x)$, no és possible trobar un valor x' tal que $x' \neq x$ i que a més $y = h(x')$.
3. Resistent a col·lisions (o resistent a col·lisions fortes): no és possible trobar dos valors x_1 i x_2 diferents ($x_1 \neq x_2$) tals que $h(x_1) = h(x_2)$.

Un punt important a destacar sobre les funcions hash criptogràfiques és que, tal i com hem vist en la seva definició, no incorporen cap tipus de clau ni d'informació secreta. Donada una entrada, si coneixem de quina funció hash es tracta, en podrem calcular la sortida sense cap problema. És important destacar aquest fet perquè es pot pensar que, tractant-se d'una funció criptogràfica, cal que involucri una clau i en el cas de les funcions hash no és així.

Funcions hash i claus Tot i que les funcions hash, per definició, no utilitzen cap clau, es poden utilitzar en esquemes en les que se'ls associï una clau, tal i com veurem en l'apartat d'aplicacions d'aquest mateix capítol.

Exemple 5.2 Contraexemple de funció hash criptogràfica

Si ens fixem en les funcions hash que hem definit en l'exemple anterior, veurem que tot i ser funcions hash, no són funcions hash criptogràfiques.

Si analitzem la funció $h(x) = x \pmod{1000}$ veiem que no compleix cap de les tres propietats que hem enumerat. Per exemple, si prenem $y = 345$, és molt simple trobar una imatge x que retorni aquest valor hash, en concret, qualsevol cadena que acabi en 345, com per exemple $x = 642345$. Per tant, la primera propietat ja no es compleix. De fet, és trivial observar que la segona i la tercera tampoc es compleixen, simplement per la simplicitat amb la que s'ha definit la funció. Per exemple, donat $x = 3456$ sabem que $y = h(3456) = 456$ i és trivial trobar un valor $x' \neq x$ tal que $h(x') = h(x)$, per exemple, $x' = 958456$ (o qualsevol cadena acabada en aquests tres nombres).

De fet, malgrat que definir les propietats d'una funció hash criptogràfica és força simple, no és gens fàcil construir una funció que les complexi, com veurem més endavant quan analitzem com es construeixen les funcions hash que s'utilitzen en l'actualitat.

De cara a simplificar tant la redacció com la lectura de la resta del capítol, abusarem del llenguatge i assumirem que totes les funcions hash a les que fem referència a partir d'aquest punt són funcions hash criptogràfiques, excepte quan diguem explícitament el contrari.

5.1.2 Propietats

És important aturar-se a mirar amb deteniment les tres propietats de les funcions hash criptogràfiques ja que l'anàlisi del seu detall permet veure que són més diferents del que aparenten.

En primer lloc, és important remarcar que una funció hash no pot ser una funció bijectiva sinó que únicament és una funció exhaustiva. És a dir, tot element té una imatge però no és cert que donada una imatge només hi hagi una sola antiimatge. Aquest fet és obvi si pensem que el conjunt de sortida pot ser de mida arbitrària (és a dir, tan gran com es vulgui) i el d'arribada té mida fixada n, més petita que la del conjunt de sortida. Per tant, si hem de poder calcular el hash de qualsevol dels elements de sortida, donat que hi ha menys elements al conjunt d'arribada, forçosament se'n repetiran, tal i com es mostra en la Figura 5.1:

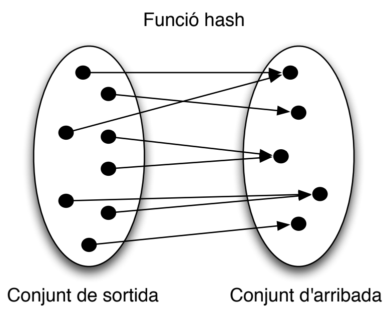


Figura 5.1: L'exhaustivitat de les funcions hash

Un altre punt a analitzar és la diferència entre la segona i la tercera propietat de les funcions hash criptogràfiques, és a dir, la diferència entre col·lisions febles i fortes. Aparentment, les dues propietats poden semblar la mateixa però una anàlisi més acurada ens mostra que ni de bon tros són iguals. La diferència entre aquestes dues propietats s'explica amb el que es coneix com la paradoxa de l'aniversari.

La paradoxa de l'aniversari ens diu que si volem que, amb probabilitat del 50%, almenys dues persones d'un grup tinguin l'aniversari el mateix dia, només cal que el grup tingui 23 persones (suposant uniforme la distribució dels naixements al llarg dels dies de l'any). Aquests valors contradiuen la nostra intuïció que semblaria que el nombre de persones hagués de ser molt més gran, per exemple, proper o més gran a 183 que és la meitat de dies que té l'any. De fet, la contradicció ve de pensar que aquest problema pot ser equivalent a trobar dues persones que tinguin l'aniversari en un dia concret de l'any. Si fem l'analogia amb les propietats de les funcions hash criptogràfiques, el primer cas correspondria a la tercera propietat (colisions fortes) i el segon cas a la segona (col·lisions febles). Ara bé, si calculem detingudament les dues probabilitats veurem que no s'assemblen gens.

Exemple 5.3 Càlcul de les probabilitats en la paradoxa de l'aniversari.

Donat un grup de $n = 23$ persones, si triem una d'elles a l'atzar, quina és la probabilitat que una de les altres persones del grup tingui l'aniversari el mateix dia (fixeu-vos que això és el cas de les col·lisions febles).

La probabilitat que una persona tingui l'aniversari aquell dia fixat és fàcil de calcular, ja que és $\frac{1}{365}$ si suposem naixements uniformes i anys de no traspàs. Per tant, la probabilitat que l'aniversari d'aquesta persona no sigui el dia triat serà el complementari, és a dir, $1 - \frac{1}{365}$. Si ara mirem per a una altra persona del grup, com que el naixement de les dues és independent, veiem que les probabilitats valen el mateix i, per tant, la probabilitat que l'aniversari de dues persones sigui diferent del dia fixat serà $(1 - \frac{1}{365})^2$. Si repetim l'argument, les 22 persones restants tindran l'aniversari en un dia diferent al fixat amb probabilitat $(1 - \frac{1}{365})^{22} = 0,94$. Així doncs, alguna persona tindrà l'aniversari al dia fixat amb probabilitat $(1 - 0,94) = 0,06$, és a dir, hi ha un 6% de probabilitat que un d'ells tingui l'aniversari en el mateix dia d'un dels altres membres del grup, un cop el membre ja s'ha fixat prèviament.

Ara bé, quina és la probabilitat que donat un grup de $n = 23$ persones, com a mínim dues d'elles tinguin l'aniversari el mateix dia. Aquest seria el cas de les col·lisions fortes.

Si prenem dues persones, la probabilitat que tinguin l'aniversari en el mateix dia és $\frac{1}{365}$ i per tant, la probabilitat que el tinguin en un dia diferent és $1 - \frac{1}{365}$. Ara bé, si afegim una tercera persona, la probabilitat que aquesta nova tingui l'aniversari en un dia diferent de les dues serà de $\frac{2}{365}$, però com que les dues primeres també han de tenir l'aniversari en un dia diferent, ens queda que per a que les tres persones tinguin l'aniversari en un dia diferent la probabilitat és $\left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right)$. Si ho generalitzem a les 23 persones, ens queda que la probabilitat que totes tinguin l'aniversari en un dia diferent és de $\left(1 - \frac{1}{365}\right) \cdot \ldots \cdot \left(1 - \frac{23-1}{365}\right) = 0,493$. Per tant, la probabilitat que almenys dues tinguin l'aniversari en el mateix dia és de $(1 - 0,493) = 0,507$.

Així, en aquest cas, amb 23 persones hi ha un 50% de probabilitat que dos d'elles tinguin l'aniversari el mateix dia. Fixeu-vos que això és molt més del que teníem en el primer cas.

Exercici 5.1 Calculeu la probabilitat que en un grup de 50 persones triades a l'atzar, dues d'elles tinguin l'aniversari el mateix dia. Quina és la probabilitat que almenys una d'elles hagi nascut el dia 1 de gener?

5.1.3 Seguretat de les funcions hash

Per parlar de seguretat d'una funció hash ens cal primer definir què s'entén per atac a una funció hash.

Un **atac a una funció hash criptogràfica** és aquell que intenta trencar alguna de les seves propietats: unidireccionalitat o no-existència de colisions (febles o fortes).

Com ja hem comentat en l'apartat anterior, és molt més probable trobar dos elements diferents que proporcionin la mateixa imatge que no pas fixar-ne un i trobar un altre element que retorni la mateixa imatge que l'element fixat. Per tant, la manera més fàcil d'atacar un funció hash (des del punt de vista probabilístic) és per mitjà de la cerca de col·lisions fortes. Per tant, com a conseqüència de la paradoxa de l'aniversari, podem obtenir la següent fórmula:

$$t \approx 2^{\frac{n+1}{2}} \sqrt{\ln\left(\frac{1}{1-\lambda}\right)}$$

que ens proporciona el nombre de missatges t als quals hem de calcular-los el hash per trobar una col·lisió amb una probabilitat λ, on n és la mida en bits de la funció hash.

En la Taula 5.1 veiem les dades per a diferents mides de funció hash.

Taula 5.1: Nombre de missatges per aconseguir col·lisions.

	Mida de la funció hash (n)				
λ	128 bits	160 bit	256 bits	384 bits	512 bits
$0,5$	2^{65}	2^{81}	2^{129}	2^{193}	2^{257}
$0,9$	2^{67}	2^{82}	2^{130}	2^{194}	2^{258}

Així, per exemple, si tenim una funció hash de mida 160 bits, ens caldrà calcular 2^{81} missatges per trobar una col·lisió amb una probabilitat de 0,5. Però només 2^{82} perquè la probabilitat de trobar-la sigui de 0,9. Per tant, com a conclusió, veiem que per a que una funció hash tingui un nivell de seguretat d'x bits necessitarem que la seva mida sigui, com a mínim, de $2x$.

Finalment, és important indicar que malgrat que trobar una única col·lisió en una funció hash és un fet que en posa en entredit la seva seguretat, al tractar-se d'un fet probabilístic, cal analitzar com s'ha trobat la col·lisió ja que una funció hash es considera trencada només quan es pot reduir la complexitat de l'atac a valors més petits dels que determina la Taula 5.1.

5.2 Construcció de funcions hash

Les propietats que es demanen a una funció hash criptogràfica, en particular les propietats que fan referència a col·lisions, ja donen una idea de la complexitat que poden arribar a tenir aquestes funcions. Cal recordar que una funció hash no incorpora cap clau de manera que qualsevol usuari coneix el funcionament exacte i complert de la funció (no hi ha cap paràmetre desconegut) i per tant un atacant pot estudiar la construcció i funcionament per atacar-la. És per aquest motiu, que la complexitat en la definició d'aquest tipus de funcions és molt elevada, com podrem veure al llarg d'aquest capítol.

Tot i la seva complexitat, les funcions hash tenen una estructura general estàndard que s'esquematitza en la Figura 5.2.

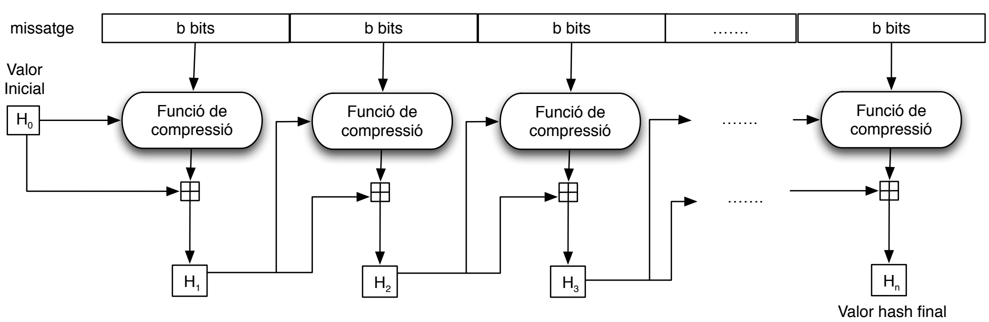


Figura 5.2: Estructura general d'una funció hash

Com es pot veure en la figura, les funcions hash processen els missatges partint-los en blocs (de forma similar a com fan els criptosistemes de bloc), tractant cada bloc de forma específica i combinant les sortides que proporciona la funció per cada bloc amb la resta de sortides dels altres blocs (també de forma semblant als modes de xifrat de bloc).

La base de les funcions hash és una funció interna que s'identifica com a funció de compressió. Aquesta funció processa cada bloc del missatge a tractar proporcionant-ne una sortida de mida igual o més petita que el propi bloc, d'aquí la seva denominació de compressió. La mida de la sortida d'aquesta funció de compressió serà la mida de la pròpia funció hash.

Depenent de com es dissenyi aquesta funció de compressió, les funcions hash es poden dividir en dos grups: funcions hash basades en criptosistemes de bloc i funcions hash de disseny específic.

5.2.1 Funcions hash basades en criptosistemes de bloc

Una manera de construir una funció hash és partint d'un criptosistema de bloc. Per a fer-ho, trobem diferents tècniques que poden combinar de diferent manera les sortides del criptosistema i el mateix bloc que s'està tractant.

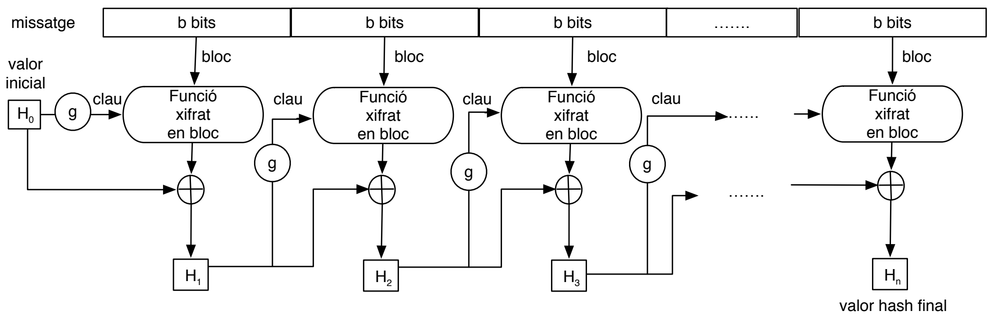


Figura 5.3: Funció hash a partir de criptosistema de bloc

En la Figura 5.3 es pot veure l'esquema d'una funció hash a partir d'un criptosistema de bloc. En la figura, el valor b indica la mida dels blocs amb el que es partirà el missatge a tractar. Per tant, com que cada bloc serà l'entrada del criptosistema de bloc, el criptosistema de bloc ha de poder treballar amb blocs de mida b. D'altra banda, el criptosistema de bloc també treballarà amb una clau. Aquesta clau té una mida l que pot coincidir, o no, amb la mida b del bloc. En el cas que les dues mides no coincideixin, com que s'utilitza la sortida d'un bloc com a clau del següent bloc ens caldrà una funció g que converteixi cadenes de b-bits a cadenes d'l-bits. En el cas que la mida del bloc sigui igual a la mida de la clau, podem prescindir de la funció g, simplement suposant que és la funció identitat. Finalment, el signe $\oplus$ del gràfic representa una operació XOR, fet que no representa un problema perquè la mida dels dos blocs que arriben a cada XOR sempre és la mateixa. Per últim, el gràfic també mostra que la mida de la funció hash és justament b, que és la mida del valor final de la sortida de l'esquema i que també coincideix amb la mida del criptosistema de bloc que estem fent servir.

Padding	En el cas que la mida del missatge no sigui múltiple del bloc, caldrà fer el *padding* del missatge per forçar-ho.

De forma més analítica, podem expressar l'esquema de la Figura 5.3 de la següent manera. Partint d'un missatge d'entrada m, el dividim en blocs de b bits obtenint $m_1, m_2, \cdots, m_n$ i per a cada bloc m_i, per a $i = 1, \cdots, n$, apliquem la següent funció definida de forma recursiva com:

$$h_i = E_{g(h_{i-1})}(m_i) \oplus h_{i-1}$$

on $E_k(\cdot)$ és la funció de xifrat en bloc amb la clau k i $h_0 = VI$, on VI és un vector inicial públicament especificat per a la funció hash en qüestió.

Exemple 5.4 Exemple de funció hash basada en un criptosistema de bloc

Definició del criptosistema de bloc:
Definim un criptosistema en bloc que treballa sobre blocs de 4 bits i denotem per $m_0m_1m_2m_3$ un bloc de text en clar. La mida de la clau d'aquest criptosistema serà de 2 bits, que denotarem per $k = k_0k_1$. La nostra funció de xifrat serà una XOR del text en clar i la clau de la següent manera:

$$c = E_k(m) = c_0c_1c_2c_3 = (m_0m_1m_2m_3) \oplus k_0k_1k_1k_0$$

D'aquesta manera, per exemple, si tenim $m = 0111$ i $k = 01$ el valor xifrat correspondrà a $c = E_k(m) = 0111 \oplus 0110 = 0001$.

Definició de la funció hash:
Definirem la nostra funció hash, $h(\cdot)$ de mida $n = 4$ bits, utilitzant el criptosistema de bloc definit anteriorment i el vector inicial $VI = 0111$. La funció $g(\cdot)$ rebrà 4 bits d'entrada i en retornarà 2 de la següent manera $g(x_0x_1x_2x_3) = (x_0 \oplus x_1)(x_2 \oplus x_3)$.

En base a aquests paràmetres, veiem com es calcularia el valor hash del missatge $m = 11001110$, és a dir $h(11001110)$.

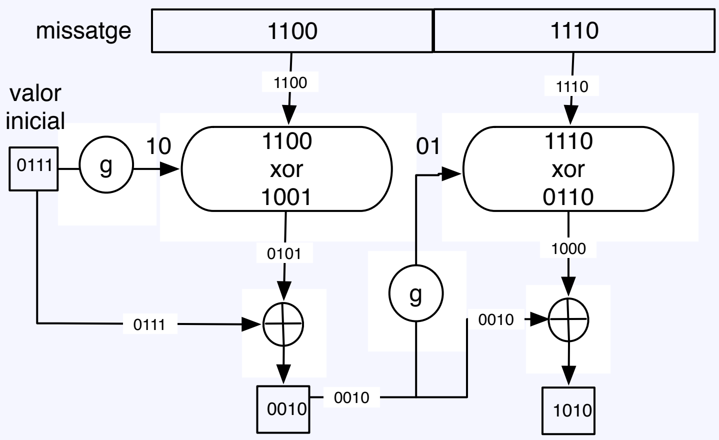
En primer lloc, partirem el missatge en blocs de 4 bits. En aquest cas tenim dos blocs $m_1 = 1100$ i $m_2 = 1110$.

Apliquem la funció de xifrat sobre m_1 amb la clau $g(VI)$. En aquest cas, $g(VI) = g(0111) = (0 \oplus 1)(1 \oplus 1) = 10$ per tant la clau que utilitzarem per al primer bloc serà $k = 10$ i el resultat del xifrat del primer bloc serà $c = E_k(m_1) = E_{10}(1100) = (1 \oplus 1)(1 \oplus 0)(0 \oplus 0)(0 \oplus 1) = 0101$. Si ara fem la XOR amb h_0 tenim $h_1 = 0101 \oplus 0111 = 0010$.

Un cop processat el primer bloc podem processar el següent utilitzant, en aquest cas, l'expressió $E_{g(h_1)}(m_2) \oplus h_1 = E_{g(0010)}(1110) \oplus 0010 = E_{01}(1110) \oplus 0010 = 1000 \oplus 0010 = 1010$.

Com que ja hem processat tots els blocs, ja hem obtingut el resultat final: $h(11001110) = 1010$.

En el gràfic de la figura següent es pot veure la versió gràfica dels càlculs:



Amb aquests tipus de construccions i utilitzant un criptosistema en bloc prou robust, podem crear funcions hash. Per exemple, utilitzant un AES amb una mida de block de 256 bits podem obtenir una funció hash amb una seguretat de 128 bits. De tota manera, a la pràctica i de forma general, s'utilitzen funcions hash de disseny específic. com les que passem a descriure en el següent apartat.

Exercici 5.2 Tenim un criptosistema de bloc que actua sobre blocs de 4 bits de longitud amb una mida de clau, k, de 3 bits, és a dir $k = k_1k_2k_3$ on $k_i \in \{0,1\}$. La funció de xifrat queda definida per la següent expressió:

$$E_k(m) = m \oplus k_{ext}$$

on k_{ext} s'obté a partir de la clau k amb la següent expressió:

$$k_{ext} = k_1k_2k_3(k_1 \oplus k_3)$$

Construïu una funció hash amb aquest criptosistema de bloc utilitzant la construcció de la Figura 5.3 prenent com a $IV = 1111$ i com a funció $g(x_1x_2x_3x_4) = x_2x_3(x_1 \oplus x_4)$.

Dibuixeu-ne l'esquema i calculeu el resultat $h(01010101)$.

5.2.2 Funcions hash de disseny específic

Més enllà de poder construir una funció hash a partir d'un criptosistema de bloc, hi ha moltes funcions hash que s'han definit específicament per a aquest propòsit. A continuació, llistem les més rellevants, donant una breu informació sobre cada una d'elles.

Les funcions MD4 i MD5 (acrònim de *Message Digest*) són funcions criptogràfiques creades per Ronald Rivest els anys 1990 i 1992, respectivament. Ambdues tenen una mida de 128 bits i processen blocs de dades de 512 bits. Les primeres vulnerabilitats de l'MD4, definida en l'RFC 1320, van ser provades ja el 1991 i en el 1995 ja es podien realitzar atacs de col·lisions en pocs segons fet que posteriorment va propiciar la retirada de la funció, explicitada en l'RFC 6150. La funció MD5, definida en l'RFC 1321, va ser desenvolupada per pal·liar les vulnerabilitats de l'MD4. De tota manera, en l'actualitat, l'MD5 es considera també insegura i el seu ús està totalment desaconsellat ja que és fàcil trobar-ne col·lisions i, fins hi tot, generar certificats digitals amb claus públiques diferents que tinguin el mateix valor hash MD5.

Certificats digitals.	La descripció i ús dels certificats digitals s'inclou en el capítol: "Infraestructures de clau pública".

RIPEMD, acrònim de *RACE Integrity Primitives Evaluation Message Digest*, és una família de funcions hash creades pels criptògrafs belgues Hans Dobbertin, Antoon Bosselaers i Bart Preneel l'any 1996 basades en la funció MD4, incorporant un seguit de millores en base a les anàlisis de seguretat i atacs realitzats sobre l'MD4. De les funcions de la família, la més coneguda i utilitzada és la RIPEMD-160, una funció hash de mida 160 bits, tot i que el conjunt de la família inclou funcions de mida 128, 256 i 320 bits. Totes elles processen el missatge amb blocs de 512 bits. L'ús d'aquesta funció, malgrat no conèixer-se'n cap atac, està poc estès ja que té una mida igual que altres funcions estandarditzades, com ara el SHA-1.

WHIRLPOOL és una altra funció hash criptogràfica creada pels criptògrafs Vincent Rijmen i Paulo S. L. M. Barreto l'any 2000. La mida d'aquesta funció és de 512 bits, la mateixa mida dels blocs que processa i la seva estructura està basada en un criptosistema semblant a l'AES. Aquesta funció ha estat estandarditzada per la International Organization for Standardization (ISO) i la International Electrotechnical Commission (IEC) sota l'estàndard ISO/IEC 10118-3.

La família de funcions SHA

Els *Secure Hash Algorithms* són un conjunt de funcions hash que estan estandarditzades pel *National Institute of Standards and Technology* (NIST) dels Estats Units. Aquestes funcions s'agrupen bàsicament en tres grans grups: SHA-1, SHA-2 i SHA-3.

En el grup SHA-1 s'hi inclou una única funció hash de mida 160 bits. Aquesta funció va ser dissenyada per la *National Security Agency* (NSA) per a la seva utilització en signatures digitals amb l'estàndard DSA. Des del 2010, però, a causa de les seves debilitats, no se'n recomana el seu ús.

El grup SHA-2, també dissenyat per la NSA, el formen essencialment dues funcions: la SHA-256 i la SHA-512. A partir de la definició d'aquestes dues funcions, que tenen una mida de 256 i 512 bits respectivament, l'estàndard de NIST també defineix un seguit de variants de diferents mides (SHA-224, SHA-384, SHA-512/224, SHA-512/256) que s'aconsegueixen truncant els resultats del SHA-256 o del SHA-512 a més d'utilitzar uns vectors inicials diferents.

Aquests dos primers grups, SHA-1 i SHA-2, estan detalladament descrits en l'estàndard FIPS 180-4: *Secure Hash Standard* publicat pel NIST al març del 2012.

L'últim grup de funcions hash, el SHA-3, és el més nou i és un conjunt de funcions hash definides en l'estàndard FIPS 202, publicat a l'agost del 2015. Està format per quatre funcions hash, SHA3-224, SHA3-256, SHA3-384 i SHA3-512, on la numeració indica la mida en bits de cada funció. A diferència de les funcions dels dos grups anteriors, la tria del SHA-3 es va realitzar a través d'una selecció pública i oberta en la que van participar investigadors de tot el món, de forma semblant a la que es va realitzar per a la tria de l'AES. En aquest cas, l'algorisme seleccionat va ser el KECCAK, proposat per Guido Bertoni, Joan Daemen, Michaël Peeters, i Gilles Van Assche que és el que s'ha estandarditzat sota les sigles SHA-3.

5.3 L'estàndard SHA-256

En aquest apartat estudiarem en detall una de les funcions hash més utilitzades en l'actualitat: el SHA-256. Veurem quines són les seves característiques i descriurem amb detall tot el seu funcionament.

Com ja hem comentat, el SHA-256 és una de les funcions hash definides en l'estàndard FIPS-180-4 publicat pel NIST i que va ser desenvolupat al 2001 per la NSA. Com a característiques generals, el SHA-256 és una funció hash de mida 256 bits que processa els missatges d'entrada en blocs de 512 bits. Pot processar missatges de fins a 2^{64} bits i utilitza un sistema de càlcul iteratiu amb un total de 64 iteracions.

L'estructura del SHA-256 segueix l'esquema mostrat en la Figura 5.2 amb una funció de compressió que s'executa sobre cada bloc del missatge d'entrada, el resultat de la qual es combina amb el resultat de la mateixa funció del bloc anterior.

Prèviament al processat de cada un dels blocs del missatge, el SHA-256 processa el missatge a tractar per tal d'assegurar-se que la mida del missatge coincideix amb un nombre enter de blocs. Aquest procés és coneix com a *padding* i es descriu en el següent apartat.

5.3.1 Padding del missatge

Quan s'utilitzen funcions que processen els missatges en blocs, pot succeir que la mida del missatge a tractar no sigui un múltiple de la mida del bloc, és a dir, que quan dividim el missatge en blocs ens quedi un últim bloc més petit que la mida del bloc amb el que treballa la funció. En aquests casos el que es fa és un procés de farcit (en anglès *padding*). L'estàndard SHA-256 defineix, de la següent manera, com s'ha de realitzar aquest procés.

Padding

Tingueu en compte que tot i que el padding és una tècnica molt utilitzada per a funcions que tracten els missatges en blocs, la manera com es fa aquest padding pot diferir en cada cas. Per exemple, el padding que utilitza el SHA-256 és diferent del que utilitza el SHA-512 i també diferent del que utilitza el criptosistema de bloc AES.

Suposem un missatge M de mida l bits, on $l \neq 0 \mod 512$. En aquest cas procedirem a:

1. afegir el bit 1 al final del missatge,
2. seguit per k bits a zero, on $k = 448 - (l+1) \mod 512$, prenent la solució més petita i no negativa,
3. afegir un bloc de 64 bits que sigui igual al nombre l expressat en binari.

Exemple 5.5 Exemple de càlcul de padding en el SHA-256

Suposem qeu volem processar el missatge `abc` amb la funció hash SHA-256.

Prenem el missatge `abc`, que expressat en codi ASCII de 8 bits és:

$$\underbrace{01100001}_{\texttt{a}} \quad \underbrace{01100010}_{\texttt{b}} \quad \underbrace{01100011}_{\texttt{c}}$$

Per tant, tenim que la mida del missatge l val $l = 3 \cdot 8 = 24$ i com que no és 512 ens cal fer padding.

En primer lloc afegim el bit 1:

$$\underbrace{01100001}_{\texttt{a}} \quad \underbrace{01100010}_{\texttt{b}} \quad \underbrace{01100011}_{\texttt{c}} \quad 1$$

Ara, calculem el valor k com $k = 448 - (24+1) \bmod 512 = 423$. És a dir, caldrà afegir 423 zeros al missatge:

$$\underbrace{01100001}_{\texttt{a}} \quad \underbrace{01100010}_{\texttt{b}} \quad \underbrace{01100011}_{\texttt{c}} \quad 1 \quad \underbrace{00\cdots 0}_{423 \text{ zeros}}$$

i finalment, caldrà afegir els 64 bits que falten fins a completar els 512 que necessitem. Aquests 64 bits seran el valor de l en binari, és a dir $l = 24_{(10} = 00\cdots011000_{(2}$, de manera que el missatge final amb el padding serà:

$$\underbrace{01100001}_{\texttt{a}} \quad \underbrace{01100010}_{\texttt{b}} \quad \underbrace{01100011}_{\texttt{c}} \quad 1 \quad \underbrace{00\cdots 0}_{423 \text{ zeros}} \quad \underbrace{00\cdots011000}_{64 \text{ bits}}$$

Mida màxima dels missatges

Fixeu-vos que aquest mecanisme de *padding* implica que la mida màxima dels missatges que pot tractar el SHA-256 és de 2^{64} bits, ja que és el màxim valor que es pot representar en la última part de la cadena de *padding*.

Exercici 5.3 Calculeu la cadena de bits que processarà la funció SHA256 una vegada s'ha realitzat el padding al missatge d'entrada $m = \texttt{SALA}$, on els caràcters s'han codificat en ASCII amb 8 bits.

5.3.2 Funció de compressió del SHA-256

Tal com ja hem comentat, el SHA-256 treballa amb blocs de 512 bits els quals processa a través de la funció de compressió. En aquesta funció de compressió podem identificar tres fases:

1. Expansió del bloc (*block schedule*).
2. Inicialització del buffer.
3. Procés de compressió.

En l'expansió del bloc és processen els 512 bits del bloc per tal d'obtenir-ne una cadena molt més llarga de 2048 bits (64 paraules de 32 bits). En la inicialització del buffer es carreguen en memòria els valors d'inicialització de la funció recurrent, valors definits en l'estàndard. Posteriorment, s'aplica el procés de compressió a la cadena de 2048 bits.

Prèviament a detallar cada un d'aquests passos, definirem algunes funcions internes que s'utilitzen en cada un dels processos que acabem d'enumerar.

Definició de les funcions internes del SHA-256

En aquest apartat definirem un seguit de funcions que s'utilitzaran tant en la part d'expansió del bloc com en el procés de compressió del SHA-256.

La funció $ROTR^n(x)$ efectua una rotació circular a la dreta d'n bits.
La funció $SHR^n(x)$ és un operador lògic de desplaçament a la dreta: $SHR^n(x) = x \ggg n$, és a dir, mou n bits a la dreta omplint els nous bits amb zeros.

Evidentment, ambdues funcions treballen a nivell de bit.

Exemple 5.6 Exemple de càlcul de la funció $ROTR^n(x)$

Sigui $x = abcdefgh$ una cadena de 8 bits i $n = 3$, aleshores
$ROTR^3(abcdefgh) = fghabcde$

Exemple de càlcul de la funció $SHR^n(x)$

Sigui $x = abcdefgh$ una cadena de 8 bits i $n = 3$, aleshores
$SHR^3(abcdefgh) = 000abcde$

Expansió del bloc

Per a processar un bloc de 512 bits, en primer lloc, la funció SHA-256 l'expandeix a un total de 2048 bits. Per fer-ho, parteix el bloc de 512 bits en 16 paraules de 32 bits. Sigui M el bloc de dades de 512 bits, el podem expressar com $M = M_0||M_1||\cdots||M_{15}$ on cada M_i té una mida de 32 bits. A partir d'aquests blocs, generarem 64 blocs de 32 bits, denotats per $W_0, W_1, \cdots, W_{63}$ que formaran el total de 2048 bits que necessitarem. Per fer-ho utilitzarem la següent expressió:

$$W_t = \begin{cases} M_t & 0 \le t \le 15 \\ \sigma_1(W_{t-2}) \boxplus W_{t-7} \boxplus \sigma_0(W_{t-15}) \boxplus W_{t-16} & 16 \le t \le 63 \end{cases}$$

on les funcions σ_0 i σ_1 estan definides de la següent manera:

- $\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$
- $\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

i l'operació $\boxplus$ és una suma mòdul 2^{32}.

Exemple 5.7 Exemple d'expansió d'un bloc

Suposem que volem fer l'expansió del bloc que hem obtingut en l'exemple del padding del missatge `abc` expressant en codi ASCII de 8 bits. Hem vist que el bloc de 512 bits, expressat en hexadecimal amb paraules de 32 bits és:

$M = M_0||M_1||\cdots||M_{15} =$ 0x61626380 0x00000000 0x00000000 0x00000000

0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000018

Els primers 16 valors $W_0, \cdots, W_{15}$ de la cadena expandida seran aquests mateixos valors del bloc, és a dir:

$W = W_0||W_1||\cdots||W_{15} = $ 0x61626380 0x00000000 0x00000000 0x00000000

0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000018

Passem ara a calcular la següent paraula de 32 bit, W_{16}.

$$\begin{aligned}
W_{16} &= \sigma_1(W_{14}) \boxplus W_9 \boxplus \sigma_0(W_1) \boxplus W_0 = \\
&= \sigma_1(0x00000000) \boxplus 0x00000000 \boxplus \sigma_0(0x00000000) \boxplus 0x61626380 = \\
&= (ROTR^{17}(0x00000000) \oplus ROTR^{19}(0x00000000) \oplus SHR^{10}(0x00000000)) \boxplus \\
&\boxplus 0x00000000 \boxplus \\
&\boxplus (ROTR^{7}(0x00000000) \oplus ROTR^{18}(0x00000000) \oplus SHR^{3}(0x00000000)) \boxplus \\
&\boxplus 0x61626380 = \\
&= 0x00000000 \boxplus 0x00000000 \boxplus 0x00000000 \boxplus 0x61626380 = \\
&= 0x61626380
\end{aligned}$$

Per tant, $W_{16} = 0x61626380$.

De la mateixa manera podem calcular el següent element W_{17}:

$$\begin{aligned}
W_{17} &= \sigma_1(W_{15}) \boxplus W_{10} \boxplus \sigma_0(W_2) \boxplus W_1 = \\
&= \sigma_1(0x00000018) \boxplus 0x00000000 \boxplus \sigma_0(0x00000000) \boxplus 0x00000000 = \\
&= (ROTR^{17}(0x00000018) \oplus ROTR^{19}(0x00000018) \oplus SHR^{10}(0x00000018)) \boxplus \\
&\quad \boxplus 0x00000000 \boxplus \\
&\quad \boxplus (ROTR^{7}(0x00000000) \oplus ROTR^{18}(0x00000000) \oplus SHR^{3}(0x00000000)) \boxplus \\
&\quad \boxplus 0x00000000 = \\
&= 0x000f0000 \boxplus 0x00000000 \boxplus 0x00000000 \boxplus 0x00000000 = \\
&= 0x000f0000
\end{aligned}$$

Així, $W_{17} = 0x000f0000$.

De la mateixa manera es calculen la resta de paraules fins a completar els 2048 bits.

Inicialització del buffer

Tal com veurem en el següent apartat, el procés de compressió és un procés recursiu. Per aquest motiu, ens caldrà definir uns valors als quals s'inicialitzaran les variables de la funció hash. Aquests valors, que es detallen a continuació, estan definits en el propi estàndard:

$$\begin{aligned}
H_0^{(0)} &= \texttt{0x6a09e667} \\
H_1^{(0)} &= \texttt{0xbb67ae85} \\
H_2^{(0)} &= \texttt{0x3c6ef372} \\
H_3^{(0)} &= \texttt{0xa54ff53a} \\
H_4^{(0)} &= \texttt{0x510e527f} \\
H_5^{(0)} &= \texttt{0x9b05688c}
\end{aligned}$$

$$H_6^{(0)} = \texttt{0x1f83d9ab}$$
$$H_7^{(0)} = \texttt{0x5be0cd19}$$

A més d'aquests valors d'inicialització, l'estàndard també defineix 64 constants que s'utilitzen en cada una de les iteracions de la funció de compressió. Aquests constants són les següents:

```
K = [ 0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1,
      0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
      0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174, 0xe49b69c1, 0xefbe4786,
      0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
      0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
      0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
      0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b,
      0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
      0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a,
      0x5b9cca4f, 0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
      0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2 ]
```

Funció de compressió

La funció de compressió és l'encarregada de prendre la cadena estesa de 64 paraules de 32 bits (és a dir, 2048 bits) i reduir-la a una cadena de 256 bits, que és justament la mida de la funció hash. Aquesta funció de compressió és un procés iteratiu en el qual s'executen 64 rondes. En la Figura 5.4 es pot veure el diagrama de la funció de compressió del SHA-256 aplicada a un únic bloc.

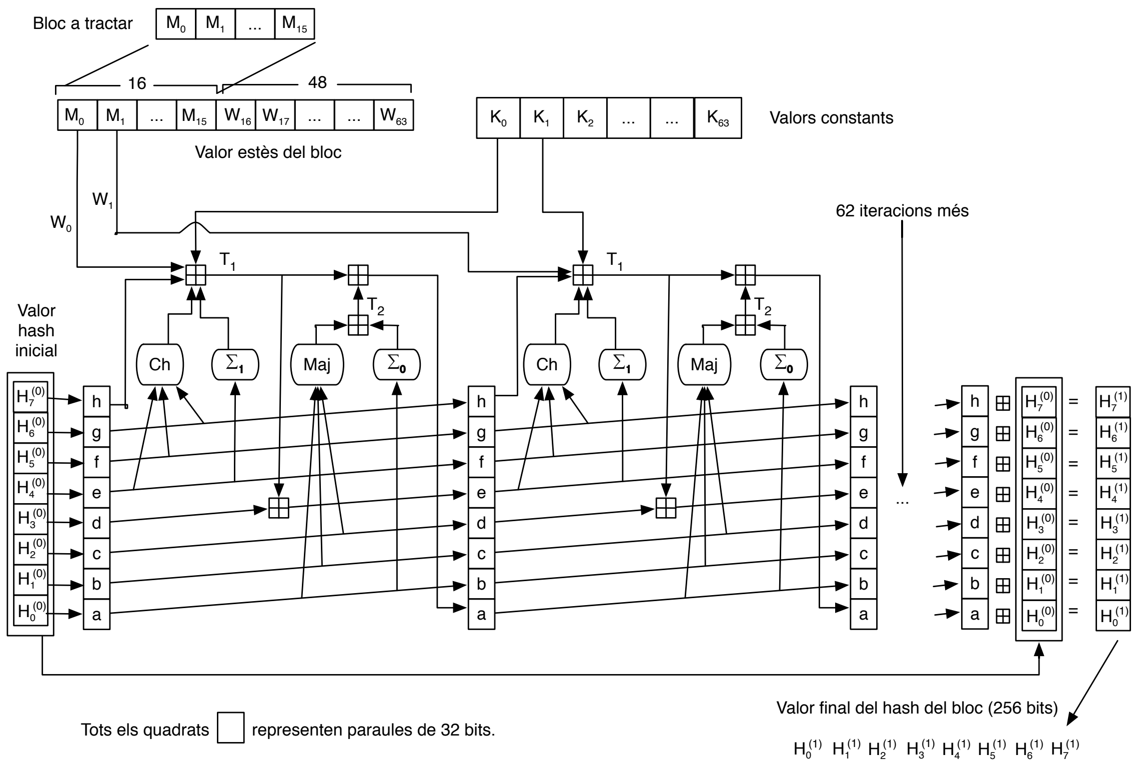


Figura 5.4: Esquema de compressió d'un bloc de la funció SHA-256

En l'esquema de la Figura 5.4 es pot veure com en cada una de les 64 iteracions es fa servir tant una de les paraules de 32 bits, W_i, com una de les constants k_i del vector K, també de 32 bits, definides en l'estàndard.

També veiem que els valors del bloc a comprimir es combinen amb els valors inicials del hash, definits també en l'estàndard com $H^{(0)}$. El gràfic també mostra com aquestes combinacions estan formades per quatre funcions, Ch, Maj, $\sum_0$ i $\sum_1$, que es descriuen a continuació:

- $\mathrm{Ch}(e,f,g) = (e \wedge f) \oplus (\neg e \wedge g)$
- $\mathrm{Maj}(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$
- $\sum_0(a) = ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$
- $\sum_1(e) = ROTR^6(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e)$

Noteu que les funcions $\mathrm{Ch}(e,f,g)$ i $\mathrm{Maj}(a,b,c)$ malgrat la complicació de la seva formulació tenen una interpretació força simple. La funció $\mathrm{Ch}(e,f,g)$ és una funció de tria (Ch- *choose*). Si el bit del valor e és un 1, la sortida de la funció és el bit del valor f i si el bit del valor e és un 0, la sortida és el bit del valor g. La funció $\mathrm{Maj}(a,b,c)$ és una funció de majoria. El bit de sortida de la funció és el bit que, en cada posició, apareix més vegades quan comparem les tres cadenes a,b i c.

Recordeu

Una XOR es pot pensar com una suma mòdul 2 (component a compoment) $000101 \oplus 000111 = 000010$ i un AND com un producte mòdul 2 (component a compoment) $000101 \wedge 000111 = 000101$. Recordeu també l'operant de negació $\neg 0100 = 1011$. A més, en la nostra notació, l'operació $\boxplus$ és una suma mòdul 2^{32}.

La funció Ch actua sobre tres paraules de 32 bits amb operacions lògiques bàsiques.

Exemple 5.8 Exemple de càlcul de la funció Ch.

Càlcul de la funció $\mathrm{Ch}(e,f,g)$ per als valors:
$e = 0\mathrm{x}510e527f, f = 0\mathrm{x}9b05688c, g = 0\mathrm{x}1f83d9ab$.

e	01010001000011100101001001111111
f	10011011000001010110100010001100
$e \wedge f$	00010001000001000100000000001100

$\neg e$	10101110111100011010110110000000
g	00011111100000111101100110101011
$\neg e \wedge g$	00001110100000001100010011000000

$e \wedge f$	00010001000001000100000000001100
$\neg e \wedge g$	00001110100000001100010011000000
$(e \wedge f) \oplus (\neg e \wedge g)$	00011111100001011100100110001100

Per tant, el resultat en hexadecimal serà $\mathrm{Ch}(e,f,g) = 0\mathrm{x}1f85c98c$

La funció Maj també actua sobre 3 paraules de 32 bits i, al igual que la funció Ch, també opera utilitzant operacions lògiques bàsiques.

Exemple 5.9 Exemple de càlcul de la funció Maj.

Càlcul de la funció $\mathrm{Maj}(a,b,c)$ per als valors:
$a = 0\mathrm{x}6a09e667, b = 0\mathrm{x}bb67ae85, c = 0\mathrm{x}3c6ef372$.

a	01101010000010011110011001100111
b	10111011011001111010111010000101
$a \wedge b$	00101010000000001101001100000101

a	01101010000010011110011001100111
c	00111100011011101111001101110010
$a \wedge c$	00101000000010001110001001100010

b	10111011011001111010111010000101
c	00111100011011101111001101110010
$b \wedge c$	00111000011001101010001000000000

$a \wedge b$	00101010000000011010011000000101
$a \wedge c$	00101000000010001110001001100010
$b \wedge c$	00111000011001101010001000000000
$(a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$	00111010011011111110011001100111

Per tant, el resultat en hexadecimal serà $\mathrm{Maj}(a,b,c) = 0\text{x}3a6fe667$

La funció $\sum_0$ actua únicament sobre una sola paraula de 32 bits generant tres paraules a partir de diferents rotacions dels seus bits i realitzant una XOR d’aquestes tres paraules.

Exemple 5.10 Exemple de càlcul de la funció $\sum_0$.

Càlcul de la funció $\sum_0(a)$ per al valor: $a = 0\text{x}6a09e667$

a	01101010000010011110011001100111
$ROTR^2(a)$	11011010100000100111100110011001

a	01101010000010011110011001100111
$ROTR^{13}(a)$	00110011001110110101000001001111

a	01101010000010011110011001100111
$ROTR^{22}(a)$	00100111100110011001110110101000

$ROTR^2(a)$	11011010100000100111100110011001
$ROTR^{13}(a)$	00110011001110110101000001001111
$ROTR^{22}(a)$	00100111100110011001110110101000
$ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$	11001110001000001011010001111110

Per tant, el resultat en hexadecimal serà $\sum_0(a) = 0\text{x}ce20b47e$.

La funció $\sum_1$ és molt similar a la funció $\sum_0$ i únicament es diferencia en el número de bits que es roten per derivar les tres paraules.

Exemple 5.11 Exemple de càlcul de la funció $\sum_1$.

Càlcul de la funció $\sum_1(e)$ per al valor: $e = 0\text{x}510e527f$

e	01010001000011100101001001111111
$ROTR^6(e)$	11111101010001000011100101001001

e	01010001000011100101001001111111
$ROTR^{11}(e)$	01001111111010100010000111001010

e	01010001000011100101001001111111
$ROTR^{25}(e)$	10000111001010010011111110101000

$ROTR^{6}(e)$	11111101010001000011100101001001
$ROTR^{11}(e)$	01001111111010100010000111001010
$ROTR^{25}(e)$	10000111001010010011111110101000
$ROTR^{6}(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e)$	00110101100001110010011100101011

Per tant, el resultat en hexadecimal serà $\sum_1(e) = 0\text{x}3587272b$

Exercici 5.4 Realitzeu els següents càlculs de les funcions internes del SHA256 tenint en compte els valors de les cadenes:
$m_1 = 00000000000000001111111111111111$
$m_2 = 11110000000000001111111111110000$
$m_3 = 11111111000000001111111100000000$

1. Calculeu el resultat de la funció $ROTR^7(m_1)$.
2. Calculeu el resultat de la funció $SHR^{10}(m_1)$.
3. Calculeu el resultat de la funció $\sigma_0(m_1)$.
4. Calculeu el resultat de la funció $\sigma_1(m_1)$.
5. Calculeu el resultat de la funció $\sum_0(m_1)$.
6. Calculeu el resultat de la funció $\sum_1(m_1)$.
7. Calculeu el resultat de la funció $\mathrm{Ch}(m_1, m_2, m_3)$.
8. Calculeu el resultat de la funció $\mathrm{Maj}(m_1, m_2, m_3)$.

5.3.3 SHA-256 sobre múltiples blocs

En els apartat anteriors hem proporcionat el detall de la funció SHA-256 quan aquesta s'aplica a un únic bloc de dades de 512 bits, que és la mida del bloc amb el que treballa la funció. Ara bé, si el missatge del qual volem calcular el hash conté més d'un bloc, aleshores cal aplicar la funció de compressió de forma recursiva sobre cada bloc, encadenant la sortida de cada bloc amb l'entrada del següent. En la Figura 5.5 es pot veure l'esquema complert per al càlcul d'un hash sobre un missatge amb tres blocs, és a dir un missatge de 1536 bits de longitud.

Com es pot apreciar en la figura, per a cada bloc es realitza l'expansió per obtenir la cadena W de 2048 bits. Les paraules de 32 bits que formen aquesta cadena són utilitzades en cada una de les 64 iteracions de la funció de compressió juntament amb els valors constants K definits en l'estàndard. Fixeu-vos que en cada bloc s'utilitzen els corresponents valors W_i obtinguts del propi bloc però en canvi, els valors K_i utilitzats en el processat de cada bloc són sempre els mateixos. Per últim, cal notar que el resultat del hash de cada bloc s'utilitza com a valor inicial per al càlcul del hash del següent bloc.

5.4 Aplicacions de les funcions hash

Un cop estudiades les característiques i propietats de les funcions hash i després de veure com es poden construir passem al darrer punt d'aquest capítol en el que veurem les múltiples aplicacions on s'utilitzen les funcions hash.

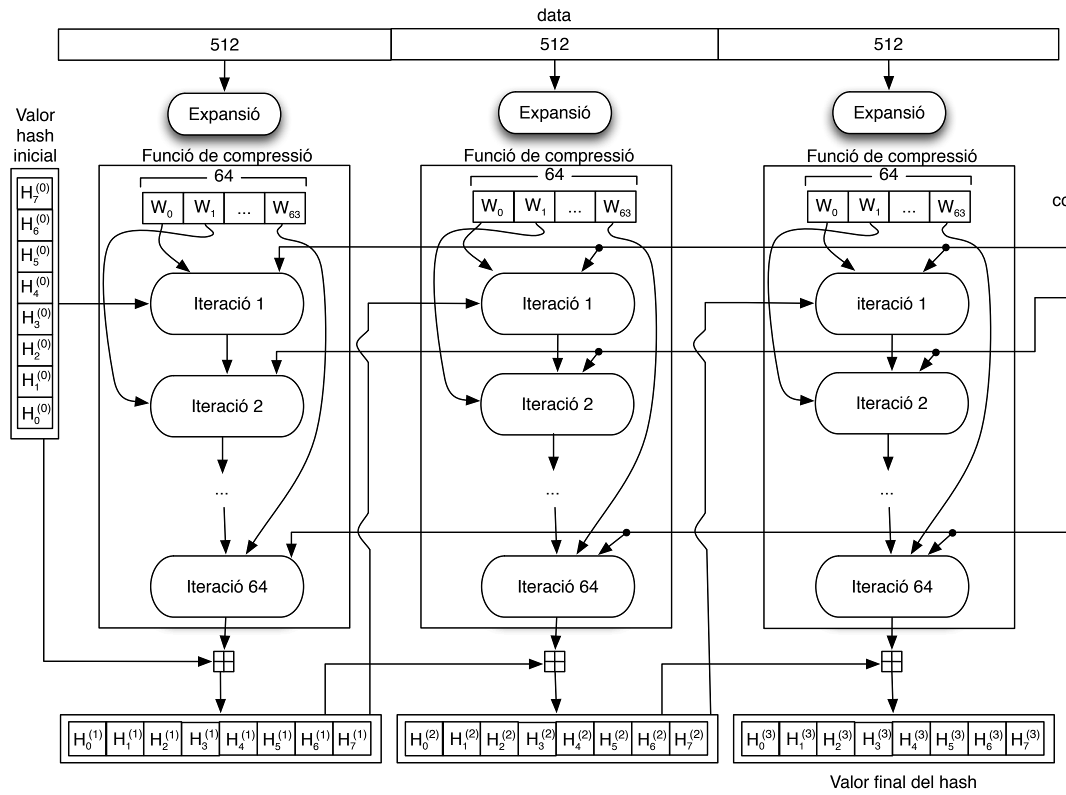


Figura 5.5: SHA-256 mostrant el processat de 3 blocs.

5.4.1 Codis d'autenticació de missatges

Un **codi d'autentificació de missatge**, en anglès *Message Authentication Code* (MAC), és una cadena curta d'informació relacionada amb el propi missatge a través d'una clau de manera que permet la seva autenticació.

Altres denominacions — Els *message authentication codes* també s'acostumen a conèixer com a *cryptographic checksums* o *keyed hash functions*.

Donat que els codis d'autentificació permeten autentificar missatges, comparteixen algunes propietats amb les signatures digitals com ara la pròpia autentificació així com la integritat del missatge. Tot i això, els codis d'autenticació no ofereixen la propietat de no repudi, propietat que sí que ofereixen les signatures digitals. Ara bé, els codis d'autenticació són molt més ràpids i eficients de calcular i és per aquest motiu que s'utilitzen en entorns on la propietat de no repudi no és essencial.

Signatures digitals — La definició i funcionament de les signatures digitals la trobareu en el capítol "Criptografia de clau pública".

Com veurem a continuació, els MAC es poden implementar de forma molt simple utilitzant conjuntament funcions hash i una clau. Aquest tipus de funcions MAC s'acostumen a denominar HMAC, justament per la utilització de la funció hash. Aquesta idea d'utilitzar una clau pot semblar contradictòria amb el que hem comentat anteriorment, indicant que les funcions hash no incorporen cap clau ni cap element secret. La manera, però, com s'utilitza la clau és simplement per variar d'alguna manera la forma del missatge que es vol autenticar. Per exemple, donada una funció hash $h(\cdot)$ podem derivar-ne dos MACs de la següent manera:

$$HMAC1_k(m) = h(k \parallel m)$$
$$HMAC2_k(m) = h(m \parallel k)$$

on el símbol $\parallel$ representa la concatenació de cadenes. La primera expressió es coneix com a *secret prefix HMAC* i la segona com a *secret suffix HMAC*.

Exemple 5.12 Exemple de càlcul d'un HMAC

Veiem un exemple de com utilitzar la funció hash definida en l'Exemple 5.4 per a calcular un *secret prefix* HMAC.

El missatge sobre el que calcularem l'HMAC serà el següent $m = 11101010$ i utilitzarem la clau $k = 1100$.

Per tant,

$$HMAC_k(m) = h(k \parallel m) = HMAC_{1100}(11101010) = h(1100 \parallel 11101010) = h(110011101010)$$

En aquest cas tenim tres blocs: $m_1 = 1100$, $m_2 = 1110$ i $m_3 = 1010$. Si ens hi fixem, els dos primers blocs són els mateixos que els de l'exemple de la funció hash, per tant tenim que $h_2 = 1010$. Per tant, el valor final de sortida de la funció dels tres blocs serà $h = h_3 = E_{g(h_2)}(m_3) \oplus h_2 = E_{11}(1010) \oplus 1010 = 0101 \oplus 1010 = 1111$

Per tant, $HMAC_k(m) = HMAC_{1100}(11101010) = 1111$.

Des del punt de vista pràctic, la manera com els MAC s'utilitzen per autentificar missatges es troba esquematitzat en la Figura 5.6.

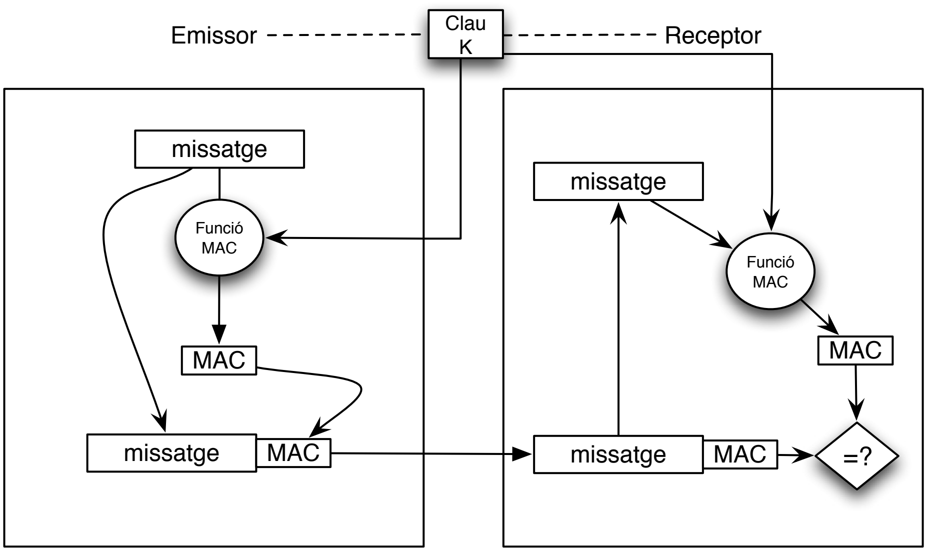


Figura 5.6: Utilització d'un HMAC per autentificar missatges.

Com es pot veure en la figura, emissor i receptor comparteixen una clau secreta. Cada vegada que l'emissor vol enviar un missatge al receptor, en calcula el seu valor HMAC utilitzant la clau secreta que comparteix amb el receptor i annexa al missatge el valor resultant. Quan el receptor rep el missatge pot utilitzar la clau i la funció hash establerta per tornar a calcular-ne el valor HMAC i comprovar que efectivament coincideix. Fixeu-vos que un atacant que canviï el missatge que emissor i receptor s'intercanvien, ha de canviar també el valor HMAC del missatge ja que si no ho fa, la comprovació del receptor no serà correcta. Ara bé, l'atacant no coneix el valor de la clau que intercanvien i per tant no pot calcular el valor correcte de l'HMAC per al missatge modificat.

Autenticació vs confidencialitat

Fixeu-vos que els codis HMAC s'utilitzen per assegurar-se que ningú no pot canviar el missatge sense que el receptor se n'adoni. Ara bé, donat que no xifrem el missatge, la comunicació no oferirà confidencialitat i un atacant pot conèixer-ne el contingut.

Exercici 5.5 Els usuaris A i B s'intercanvien missatges. Com que A i B ja comparteixen una clau simètrica, han decidit que calcularan un HMAC dels missatges per assegurar-ne la seva integritat, és a dir, per assegurar-se que ningú que intercepti la informació pugui modificar-la. Calculant un HMAC del missatge a partir de la clau simètrica que comparteixen volen evitar que si algú modifica el missatge no pugui modificar l'HMAC de forma correcta, ja que l'atacant desconeix la clau. Fan servir una funció HMAC basada en la funció hash $h(\cdot)$ definida en l'Exemple 5.4, concretament utilitzant la clau k com un *secret prefix*, és a dir $\mathrm{HMAC}_k(m) = h(k \parallel m)$. D'aquesta manera, A envia el missatge $m = 0111$ a B seguit de l'$\mathrm{HMAC}_k = 0111$, on k és la clau simètrica que fan servir i només ells dos coneixen.

Malauradament, no saben que la tècnica del *secret prefix* no és segura i nosaltres, com a atacants, podem afegir la cadena que vulguem al missatge original i calcular l'HMAC sense conèixer k. Podríeu calcular l'HMAC corresponent al missatge $m' = 01111111$?

5.4.2 Resum de missatges

Una altra de les aplicacions en les que s'utilitzen les funcions hash és per obtenir una representació compacta d'un missatge més gran. Gràcies a que el valor hash d'un missatge pot permetre identificar-lo de forma pràcticament unívoca, aquest resum es pot utilitzar en diferents contexts. Per exemple, quan es volen emmagatzemar fitxers molt grans, sovint en format multimèdia, en una base de dades, s'acostuma a guardar-ne només el seu valor hash en la pròpia base de dades i una localització externa. D'aquesta manera, es pot referenciar el contingut i fer-ne cerques fins hi tot partint del propi contingut, també utilitzant-ne el valor hash, per tal d'obtenir-ne informació associada.

Tenir un resum d'un missatge també és molt rellevant quan les operacions que s'han de realitzar sobre el missatge són molt costoses i ens és suficient realitzar-les sobre un resum. Aquest és el cas de les signatures digitals. Tal com veurem més endavant, les signatures digitals són computacionalment poc eficients i per aquest motiu, en comptes de realitzar-les sobre el missatge sencer s'apliquen sobre un resum d'aquest. La mida reduïda i fixa que s'obté amb una funció hash permet augmentar molt l'eficiència de les signatures digitals.

5.4.3 Emmagatzematge de contrasenyes

Una altra de les aplicacions de les funcions hash és la seva utilització en l'emmagatzematge d'algunes dades sensibles, com ara les contrasenyes d'accés a un sistema informàtic. La protecció de les contrasenyes és altament necessària per assegurar que cap usuari maliciós se'n pugui apoderar i pugui accedir al sistema suplantant altres usuaris. Per aquest motiu les contrasenyes mai es guarden en clar.

L'emmagatzematge de les contrasenyes serveix per poder-les comparar amb les que els usuaris introdueixen en el procés d'autenticació. Si la contrasenya proporcionada per l'usuari coincideix amb la que el sistema emmagatzema, l'autenticació es considera vàlida. Ara bé, com ja hem dit, les contrasenyes no s'emmagatzemen en clar en el sistema sinó que s'emmagatzema la imatge de la contrasenya per una funció hash.

Contrasenyes no-xifrades

Tot i que col·loquialment sovint es parla que les contrasenyes en els sistemes es guarden xifrades, aquesta denominació no és correcta ja que una informació xifrada s'ha de poder desxifrar i la imatge d'una funció hash no permet "desxifrar-ne" el seu valor, perquè voldria dir invertir la funció hash, cosa que no és possible.

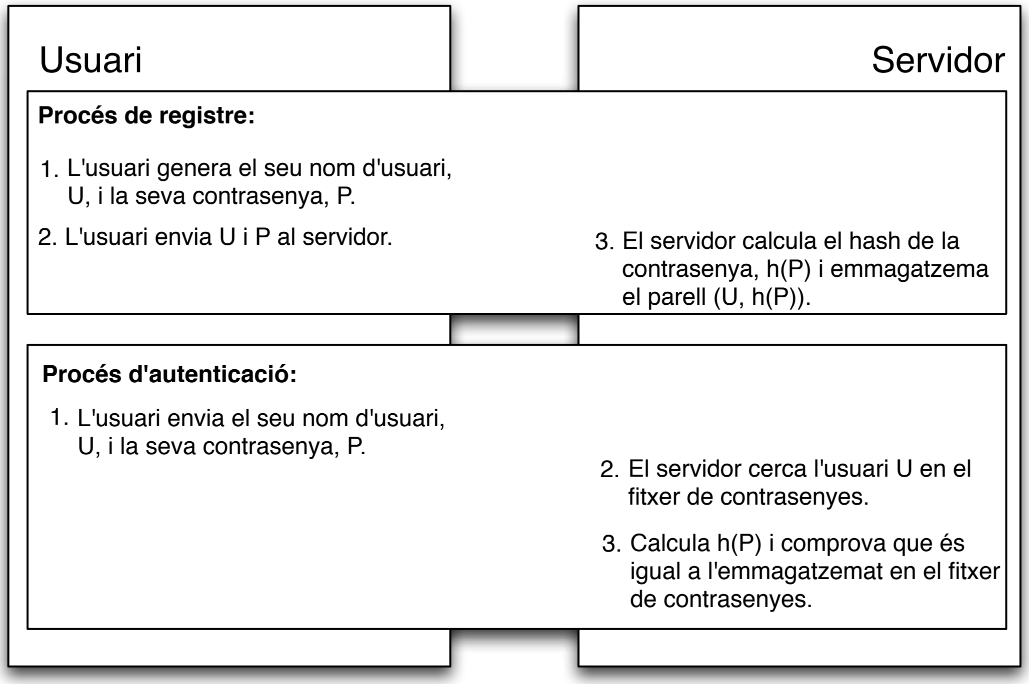


Figura 5.7: Esquema d'autenticació amb contrasenya

El procés d'autenticació amb contrasenyes guardades com a imatge d'una funció hash es mostra en la Figura 5.7. Quan un usuari vol accedir al sistema, proporciona el seu usuari i la seva contrasenya. El sistema, a partir de la contrasenya que li ha fet arribar l'usuari, en calcula el seu hash i el compara amb el valor que té emmagatzemat. En el cas que els dos valors coincideixin, l'usuari queda autenticat correctament.

Recuperació de contrasenya

En un sistema d'accés amb contrasenya ben implementat, ni tan sols l'administrador del sistema us pot dir la vostra contrasenya en cas que l'hàgiu oblidat, perquè ell no la coneix i només en té la imatge per una funció hash. Per aquest motiu, quan oblidem la contrasenya el sistema ens demana que en generem una de nova.

En realitat, el sistema descrit anteriorment és una simplificació del sistema que realment es fa servir per emmagatzemar contrasenyes, ja que les contrasenyes emmagatzemades únicament amb el seu hash permeten atacs eficients com ara el següent. Suposem un sistema que té les contrasenyes emmagatzemades utilitzant el hash SHA256 de la contrasenya. En aquest cas, el fitxer de contrasenyes tindrà un seguit de valors de 256 bits cada un d'ells vinculat a un usuari. En el cas que un atacant pogués aconseguir aquest fitxer, podria realitzar el següent atac: Prenent un diccionari de contrasenyes habituals, pot anar calculant el valor SHA256 d'aquestes contrasenyes i anar-lo comparant amb cada un dels valors del fitxer. Fixeu-vos que només que algun dels usuaris del sistema tingui una de les contrasenyes del diccionari, a l'atacant només li caldrà calcular un sol hash i comparar-lo amb cada un dels valors del fitxer fins a trobar el correcte. A més, en el cas que diferents sistemes utilitzessin la mateixa funció hash, l'atacant també podria tenir un diccionari dels hashos en comptes del diccionari de les contrasenyes.

Per evitar aquests tipus d'atacs, abans de calcular el hash de la contrasenya per a emmagatzemar-la, el que es fa és afegir a la contrasenya un valor fixat, que s'anomena *salt*, i que és diferent per cada usuari, de manera que encara que dos usuaris tinguin la mateixa contrasenya el hash que s'emmagatzemi sigui diferent. Evidentment, aquest valor també s'haurà d'afegir en el procés d'autenticació quan s'està validant la correcció de la contrasenya proporcionada per l'usuari. Fixeu-vos que un atacant que s'enfronta a un fitxer de contrasenyes amb *salt*, tot i conèixer el *salt* de cada usuari, ha de calcular el hash de cada un dels valors del diccionari de contrasenyes per cada un dels usuaris del sistema al que està atacant.

Rainbow tables

Les *rainbow tables* són unes taules construïdes per optimitzar la informació que s'emmagatzema i la que calcula un atacant que fa un atac sobre un fitxer de contrasenyes a les quals no se'ls ha afegit un *salt*. Tot i això, les *rainbow tables* no són útils amb contrasenyes desades amb *salt*.

Les *salts* acostumen a emmagatzemar-se juntament amb el hash de la contrasenya, ja que la seva funció no és pas impedir el càlcul del hash sinó evitar que l'atacant pugui reaprofitar càlculs. Quan es vol afegir un nivell més de protecció, es poden fer servir *salts* secretes (també conegudes com a *pepper*) que es desen en un altre dispositiu, diferent del que emmagatzema les contrasenyes. Així, un atacant que només té accés al fitxer de contrasenyes no pot fer l'atac, ja que no coneix les *salts* que s'han fet servir per a calcular cadascun dels hashos.

5.4.4 Derivació de claus

En criptografia és habitual l'ús de claus criptogràfiques en diferents contextos, com per exemple per a xifrar informació. Ara bé, la capacitat de les persones per a generar i recordar cadenes de zeros i uns és més aviat limitada, sobretot si aquestes cadenes són molt llargues, com podria ser una simple clau de l'AES de 128 bits. Per aconseguir que les persones puguin generar i recordar claus de forma simple, es fan servir les contrasenyes de sempre, que els usuaris estan acostumats a utilitzar, combinades amb funcions hash. Així, donada una contrasenya se li aplica una funció hash per derivar-ne una clau. Si sempre s'utilitza la mateixa funció hash, donat que aquesta és determinista, per a la mateixa contrasenya d'entrada generarà la mateixa clau. Aquesta idea simple presenta algunes debilitats de seguretat i per aquest motiu s'han dissenyat funcions específiques de derivació de claus que, això sí, es basen en una funció hash.

La funció PBKDF2

La funció ***Password-Based Key Derivation Function*** **(PBKDF2)** és una funció definida en l'RFC2898 que proporciona un mecanisme segur per obtenir una clau a partir d'una contrasenya.

Aquesta funció és una funció força utilitzada en diferents aplicacions, com ara per a les claus dels accessos a les xarxes WIFI (amb els protocols WPA i WPA2), en el xifrat amb AES en el WinZip i en múltiples aplicacions de programari que permeten xifrar el disc dur de l'ordinador.

La funció PBKDF2 rep com a entrada cinc paràmetres i retorna la clau que n'ha derivat. En la següent expressió s'inclouen els paràmetres que requereix la funció:

$$K = \texttt{PBKDF2(PRF, Contrasenya, Salt, c, dkLen)}$$

D'aquests paràmetres, el més evident és la contrasenya (codificada en UTF-8), que serà el valor que l'usuari proporcionarà per tal d'obtenir-ne la clau. La resta de valors són interns de cada implementació i estaran fixats per tal que cada contrasenya només pugui derivar una única clau. El valor `PRF` indica una funció pseudoaleatòria que utilitza dos paràmetres, una clau i un valor. Aquesta funció proporcionarà una sortida de mida `hLen`. Si ens hi fixem, aquesta definició de funció coincideix amb el d'una funció HMAC com la que hem definit en l'Apartat 5.4.1 i és en aquest punt on les funcions hash queden lligades a la funció de derivació de claus. D'altra banda, el valor `Salt` és una seqüència de bits utilitzada per afegir aleatorietat al procés, com s'acostuma a fer amb els valors de salt en altres processos de seguretat. El valor `c` és un valor que determina el nombre d'iteracions que realitzarà la funció de derivació abans de proporcionar la clau. Com més gran sigui aquest valor més robusta serà la clau generada però també més trigarà la funció a calcular-la. Es recomana que aquest valor sigui, com a mínim, 1000. Finalment, el valor `dkLen` és la mida de la clau K que es vol generar.

Una vegada definits cada un dels paràmetres que utilitza la funció de generació de claus, passem a detallar-ne el seu funcionament. La següent expressió proporciona el sistema per a calcular la clau utilitzant la funció PBKDF2:

$$K = \mathtt{T}_1 \parallel \mathtt{T}_2 \parallel \cdots \parallel \mathtt{T}_{\lceil \text{dkLen/hLen} \rceil}$$

on el símbol $\parallel$ indica la concatenació i els valors $\mathtt{T}_i$ es descriuen a continuació.

Cada valor $\mathtt{T}_i$ el definim com $\mathtt{T}_i = \mathtt{F(Contrasenya,\ Salt,\ c,}\ i)$ on la funció `F` queda explicitada en la següent expressió:

$$\mathtt{F(Password,\ Salt,\ c,}i) = \mathtt{U}_1 \oplus \mathtt{U}_2 \oplus \cdots \oplus \mathtt{U}_\mathtt{C}$$

on els valors $\mathtt{U}_i$ són els següents:

$\mathtt{U}_1 = \mathtt{PRF(Contrasenya,\ Salt} \parallel \mathtt{INT\_32\_BE(i))}$

$\mathtt{U}_2 = \mathtt{PRF(Contrasenya,\ U}_1\mathtt{)}$

$\vdots$

$\mathtt{U}_\mathtt{C} = \mathtt{PRF(Contrasenya,\ U}_{\mathtt{C}-1}\mathtt{)}$

on `INT_32_BE(i)` és l'índex i codificat com un enter de 32 bits en notació *big-endian*.

Com hem comentat anteriorment, el valor `c` és el que determina el nombre d'iteracions que es realitzaran. Fixeu-vos a més, que es pot donar el cas en que el valor `dkLen/hLen` no sigui un enter, i per tant la part entera superior de la divisió, és a dir $\lceil \mathtt{dkLen/hLen} \rceil$, proporcionarà un nombre total de bits de la clau superior al que s'havia indicat en el valor `dkLen`. En aquest cas, la última paraula de la clau, $\mathtt{T}_{\lceil \mathtt{dkLen/hLen} \rceil}$ es truncarà per la dreta per tal que la clau tingui exactament `dkLen` bits.

5.4.5 Pseudonimització de dades

Actualment les dades tenen un paper clau en molts dels sectors de la societat. L'ús de dades massives ha permès progressar en àmbits tant diversos com la medicina, les telecomunicacions o les finances, però l'ús indiscriminat de dades personals comporta problemes de privadesa que cal adreçar.

Les funcions hash s'utilitzen sovint per a pseudonimitzar identificadors en conjunts de dades, tot i que, com veurem a continuació, aquesta no sempre és una bona alternativa.

Informalment, la pseudonimització permet dissassociar la identitat d'un subjecte de les dades d'aquest. Normalment aquest procés es duu a terme substituint un o diversos identificadors per un pseudònim, per exemple, una cadena generada pseudoaleatòriament. Així, les dades queden associades a aquest pseudònim, i desvinculades de la identitat del seu propietari.

Una primera aproximació a la pseudonimització amb funcions hash consistirà a substituir els identificadors d'un conjunt de dades pel resultat d'avaluar una funció hash sobre aquests.

Exemple 5.13 **Exemple de pseudonimització trivial amb funcions hash** Suposem que disposem d'un conjunt de dades amb notes d'estudiants que conté els atributs DNI (sense lletra) i nota de l'estudiant en una assignatura.

La pseudonimització trivial d'aquest conjunt de dades substituiria el DNI dels estudiants pel resultat d'aplicar una funció hash al DNI. A priori, això desvincularia la identitat de l'estudiant de la seva nota.

A continuació veurem dos dels problemes d'aquesta tècnica. D'una banda, un atacant que coneix el

DNI d'un estudiant de l'assignatura, podria reidentificar el registre i aconseguir saber la nota d'aquest estudiant. Per fer-ho, simplement hauria de calcular el hash del DNI, i consultar la nota al conjunt de dades pseudonimitzat. D'altra banda, si l'atacant no coneix el DNI de cap estudiant, podria llançar un atac de força bruta per trobar els DNIs dels estudiants, aprofitant el fet que els DNIs tenen un format concret per a reduir l'espai de cerca. Els DNIs estan formats per 8 dígits, de manera que caldria calcular 10^8 hashos per a comprovar tots els DNIs possibles. Si assumim que l'atacant pot calcular un 10^6 hashos per segon (cosa que es podria fer amb qualsevol ordinador sense hardware especialitzat), l'atac tardaria menys de dos segons.

Dues alternatives més robustes per a pseudonimitzar identificadors consisteixen en l'ús de MACs (*Message Authentication Code*) o la incorporació de *salts* secretes. En ambdues alternatives, a cada identificador li corresponen diversos pseudònims, depenent de la clau o la *salt* utilitzada. La clau o la *salt* secreta es mantenen separats de les dades, de manera que un atacant que només diposa de les dades no pot reidentificar-ne els registres reproduint el procés que s'ha dut a terme per calcular els pseudònims.

Exemple 5.14 Exemple de pseudonimització amb *salt* secreta o *pepper*

Suposem que disposem del mateix conjunt de dades que a l'exemple anterior, i que s'aplica un procés de pseudonimització substituint el DNI pel hash d'una *salt* secreta (generada pseudoaleatòriament) concatenada al DNI.

La Figura següent mostra un exemple del procediment, fent servir SHA-256 com a funció hash. Les dades originals són dividides en dos conjunts de dades diferents, que seran desades separadament. D'una banda, es desarà la *salt* de cada registre, que s'haurà generat pseudoaleatòriament (Taula 1.3). D'altra banda, es desaran les notes, associades al pseudònim, que s'haurà calculat com el SHA-256 de la concatenació de la *salt* i el DNI.

Table 1.2: Conjunt de dades original

DNI	Nota
50705923	5
88046921	9.8
20091322	7
64802452	2

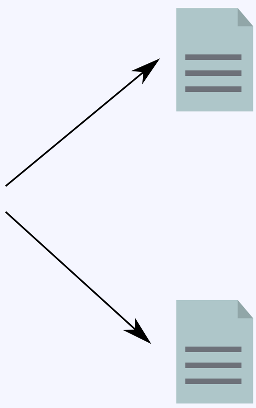


Table 1.4: Conjunt de dades pseudonimitzat

Pseudoidentificador	Nota
`0x340fff ...`	5
`0x1648da ...`	9.8
`0x40e616 ...`	7
`0x9d480f ...`	2

Table 1.3: Salts secretes

Salt
56714
78910
28285
92670

Aquesta tècnica permet adreçar les limitacions de la tècnica trivial. Ara, si un atacant accedeix al conjunt de dades pseudonimitzades, i coneix un dels DNIs dels estudiants de l'assignatura, no podrà reidentificar el registre que correspon a aquest estudiant, ja que no podrà recrear el hash. Anàlogament, un atacant tampoc podrà calcular els pseudònims de tots els DNIs existents, encara que sàpiga que aquests estan formats per 8 dígits.

Ara bé, cal anar molt en compte a l'aplicar aquest tipus de tècniques per pseudonimitzar dades. De fet, en podríem fer una assignatura sencera només explicant com fer-ho! Per exemple, suposem que el conjunt de dades conté les notes de diverses activitats:

Pseudoidentificador	Act. 1	Act. 2	Act. 3
`0x340fff...`	5	6	4
`0x1648da...`	10	9	7
`0x40e616...`	7	7	3
`0x9d480f...`	2	5	3

i que l'atacant sap quin estudiant és l'únic que ha aprovat la tercera activitat de l'assignatura (per exemple, pot ser que sigui l'únic que va sortir content de classe després de saber-se les notes d'aquesta activitat). En aquest cas, l'atacant serà capaç de reindentificar el registre de l'estudiant, i obtenir així informació addicional sobre aquest (en particular, les notes exactes de totes les activitats).

5.4.6 Generació de cadenes de bits pseudoaleatòries

La criptografia requereix sovint de l'ús de cadenes de bits generades aleatòriament, per exemple, en la generació de claus, de vectors d'inicialització o de valors per a reptes en protocols interactius. Per tal de generar cadenes de bits aleatòries, es disposa principalment de dues estratègies: o bé es fa servir una font d'aleatorietat a partir d'algun procés físic que no sigui predictible, o bé es calculen els bits de manera determinista amb un algorisme a partir d'una llavor. Les tècniques que fan servir aquesta segona estratègia s'engloben sota el nom de generadors de nombres pseudoaleatoris o PRNG (per les seves inicials en anglès, *PseudoRandom Number Generator*) i, de les cadenes de bits que generen, en direm que són pseudoaleatòries.

Els PRNG són algorismes deterministes que produeixen una seqüència de bits a partir d'una llavor, que s'ha d'obtenir d'una font aleatòria. Quan la llavor és secreta, els bits que genera un PRNG no són predictibles.

Es poden construir PRNG a partir de funcions hash. Així, per exemple, el NIST defineix un PRNG basat en funcions hash anomenat Hash_DRBG (DRBG són les inicials de *Deterministic Random Bit Generator*, un altre terme per referir-se als PRNG). L'algorisme Hash_DRBG emmagatzema un estat format per una variable i una constant (V i C, respectivament) i un comptador. Inicialment, V i C es deriven de la llavor aleatòria. Després, el valor de la variable V es fa servir per derivar els bits pseudoaleatoris, i s'actualitza el valor d'aquesta variable (en aquesta actualització es fa servir la constant C). Cada vegada que es generen nous bits, s'incrementa el comptador de l'estat intern. Quan aquest comptador arriba a un llindar preestablert, cal tornar a introduir aleatorietat a l'algorisme per tal de seguir generant bits pseudoaleatoris, procés que es coneix com a ressembrat (de l'anglès, *reseeding*).

5.4.7 Compromís de bit

Hi ha situacions quotidianes en les que estem acostumats a fer servir alguns mecanismes molt simples que funcionen sense cap dificultat d'execució. Un d'aquests casos és el de 'tirar una moneda a l'aire' per, per exemple, decidir quin dels dos jugadors d'una partida d'escacs tindrà les fitxes blanques. Ara bé, quan les dues parts que duen a terme aquest petit protocol no es troben físicament al mateix lloc, la simplicitat de tirar una moneda a l'aire no ens serveix en el cas que hi hagi certa desconfiança entre els dos participants.

Si analitzem el procés de tirar una moneda a l'aire veiem que, normalment, un dels dos usuaris tria cara o creu i l'altre, una vegada s'ha decidit qui guanyarà segons el revers de la moneda, tira la moneda a l'aire. En aquest simple esquema, l'usuari que tria cara o creu ho fa de forma pública, de manera que després (quan cau la moneda) no pot dir que ha triat una altra cosa. I l'usuari que tira la moneda no pot fer trampa (assumint que la moneda no està trucada!) perquè tira la moneda davant de l'altre usuari i els dos veuen el resultat que en surt, de manera que qui tira la moneda no pot canviar-ne el resultat.

Per emular aquest protocol de forma remota (o digital) es fa servir un esquema de compromís de bit.

Un **esquema de compromís de bit** (en anglès, *bit commitment*) és una tècnica per la qual un usuari A es compromet, davant d'un usuari B, a un valor m per mitjà d'un valor $C(m)$, que serà el compromís. Aquest compromís ha de tenir les següents propietats:

1. Donat el compromís $C(m)$, B no pot obtenir informació del valor compromès m.
2. A ha de poder obrir el compromís $C(m)$ mostrant el valor compromès m.
3. A no pot obrir el compromís $C(m)$ mostrant un valor diferent al valor m compromès inicialment.

Amb un esquema de compromís de bit com el que acabem de descriure, el protocol de tirar una moneda a l'aire es pot definir amb els següents passos.

1. L'usuari A tria cara o creu i codifica la seva tria en el missatge m. Posteriorment, calcula el compromís d'm, $C(m)$, i l'envia a B.
2. B genera aleatòriament un bit, on 1 correspondrà al valor cara i 0 correspondrà a creu. B enviarà a A el valor aleatori generat.
3. A obrirà el compromís $C(m)$ mostrant a B quin valor (cara o creu) havia triat, de manera que es veurà qui ha guanyat en el protocol de tirar una moneda a l'aire.

Fixeu-vos que en el pas 2 del protocol, l'usuari A ja ha triat cara o creu però l'usuari B, tot i tenir el compromís $C(m)$, no pot saber quin valor ha triat (gràcies a la primera propietat de l'esquema de compromís de bit). En el pas 2, tot i que l'usuari B no generés el bit de forma aleatòria (per intentar alterar el protocol) el fet que no coneix si A ha triat cara o creu fa que la tria d'aquest valor aleatori sigui intrascendent. D'altra banda, en el pas 3, A ja sap quin valor ha obtingut B i per tant B no pot desdir-se'n. A més, A obre el seu compromís i, tot i conèixer el valor obtingut per B, no pot obrir-lo mostrant un altre valor diferent al que s'ha compromès, gràcies a la tercera propietat de l'esquema de compromís de bit.

Els protocols de compromís de bit es descriuen per mitjà de dues fases: fase de generació del compromís i fase d'obertura del compromís i en els següents apartats veurem dues tècniques diferents que implementen un esquema de compromís de bit.

Compromís de bit utilitzant funcions hash

Una de les tècniques més utilitzades per implementar un esquema de compromís de bit és mitjançant una funció hash.

Sigui m el missatge al qual l'usuari es vol comprometre, en la **fase de generació del compromís** l'usuari A selecciona un valor aleatori r i calcula $C(m) = h(r \parallel m)$ on h és una funció hash criptogràfica.

En la **fase d'obertura del compromís** $C(m)$, l'usuari A revela els valors r i m. A partir d'aquests valors, l'usuari B pot calcular $h(r \parallel m)$ i comprovar que efectivament coincideix amb el valor $C(m)$ al qual A s'havia compromès.

Comprovem que aquest esquema compleix amb les tres propietats d'un esquema de compromís de bit.

1. B no pot obtenir el valor compromès m a partir el compromís $C(m)$ ja que $h(\cdot)$ és una funció hash criptogràfica i per tant no es pot invertir. Fixeu-vos que el valor aleatori r s'utilitza en cas que el missatge m se seleccioni d'un conjunt petit de missatges, per tal d'evitar que B pugui calcular totes les imatges de la funció hash per a tots els possibles valors diferents d'm i descobrir-ne el valor compromès.
2. A pot obrir el compromís $C(m)$ fent públics els valors r i m.
3. A no pot obrir el compromís, $C(m)$, obtenint un valor $m' \neq m$ perquè això voldria dir que A pot trobar $(r \parallel m) \neq (r' \parallel m')$ tal que $h(r \parallel m) = h(r' \parallel m')$ i això no és possible per les propietats que hem enumerat de la funció hash criptogràfica que s'utilitza.

5.4.8 Prova de treball

En l'execució d'alguns protocols criptogràfics, en ocasions, és necessari assegurar que un participant realitza un cert esforç de càlcul abans de poder realitzar una operació per tal que l'operació en qüestió no sigui fàcil de realitzar de forma automàtica i repetitiva. Aquests tipus de mecanismes s'anomenen prova de treball.

Una **prova de treball** (en anglès *proof-of-work*), és un mecanisme que permet a l'usuari d'un sistema demostrar a la resta d'usuaris de forma fidedigna que ha realitzat una certa quantitat de feina, normalment, una certa quantitat de càlculs.

El concepte de prova de treball el van proposar Cyntia Dwrok i Moni Naor en un article publicat al congrés *Crypto* l'any 1992 però no va ser fins més tard, l'any 1999, que M. Jakobsson i A. Juels van formalitza-lo i van proposar-ne el terme *proof-of-work*.

Les aplicacions de les proves de treball són variades i van des de la prevenció de correu brossa fins al manteniment d'integritat en els sistemes de criptomonedes.

La propietat més important d'una prova de treball és la seva asimetria, en el sentit que el cost de realització de la prova de treball s'ha de poder prefixar de forma arbitrària, però la verificació de la prova de treball, independentment de la dificultat fixada en el cost, ha de ser extremadament eficient i, per tant, no ha de requerir tornar a realitzar els càlculs que s'han de realitzar per produir-la. És per aquest motiu que les funcions unidireccionals utilitzades en criptografia, com ara les funcions hash, són una bona base per a la creació de proves de treball.

Una de les proves de treball més utilitzades en l'actualitat, ja que moltes de les criptomonedes existents la fan servir, és el Hashcash, una prova de treball proposada per A. Back l'any 1997 per tal de limitar el correu brossa i, en general, altres atacs de denegació de servei. Aquesta prova de treball consisteix a calcular el valor hash d'una certa informació i aconseguir que la imatge resultant sigui un valor inferior a un cert llindar. Per a fer-ho, cal habilitar un camp aleatori en la informació en qüestió per tal de poder-lo variar per obtenir-ne diferent valors hash.

Per exemple, una simplificació del sistema anti-correu brossa basat en aquesta prova de treball seria el següent. Quan l'usuari A vol enviar un correu a l'usuari B, un cop generat tot el missatge, inclosa l'adreça del destinatari, l'usuari A afegeix a la capçalera un nou camp, que podrà contenir qualsevol valor aleatori. Amb tota aquesta informació, A en calcularà la imatge per una funció hash determinada, que haurà consensuat amb B. Prèviament, A i B hauran també fixat quin és l'esforç (en la prova de treball) que A ha de fer per enviar-li un correu a B. Aquest esforç s'explicitarà triant un **valor objectiu** concret d'entre totes les imatges possibles de la funció hash. Abans de processar el correu, l'usuari B comprovarà si el hash del missatge que ha rebut d'A és inferior al valor objectiu. En cas afirmatiu, processarà el correu, en cas negatiu el descartarà. Fixeu-vos que una vegada A ha redactat el correu, si al realitzar el càlcul del hash n'obté un valor superior al valor objectiu, no pot enviar el missatge (ja que B el descartaria). Abans de fer-ho ha de modificar el nou camp que ha afegit a la capçalera amb un valor aleatori i tornar a calcular-ne el hash. Si és menor al valor objectiu, ja podrà enviar-lo, però si no ho és haurà de tornar a modificar el valor del camp, tornar a calcular el hash i anar repetint aquesta operació fins que el hash del correu sigui més petit que el valor objectiu. Fixeu-vos que la mida del valor objectiu fixarà la dificultat de la prova de treball, com més petit sigui el valor objectiu, més feina haurà de fer A per enviar el missatge a B.

Fixeu-vos que la necessitat que té l'emissor del missatge per enviar-lo fa que si aquest emissor és un generador de correu brossa, per enviar cada correu brossa li sigui necessari realitzar un cert volum de càlcul per a cada correu (ja que el destinatari del correu forma part de la informació que s'inclou en el hash i per tant no pot reaprofitar els càlculs d'un altre correu) i per tant es desincentiva aquest pràctica.

Exemple 5.15 La dificultat de la prova de treball i les probabilitats

Les propietats estadístiques de les funcions hash criptogràfiques fan que la seva sortida es pugui considerar un generador pseudoaleatori en el sentit que donada una entrada no se'n pot predir la sortida i una mínima modificació de l'entrada provoca una modificació significativa del valor de la sortida. Amb aquesta premissa, suposem una funció hash de mida 3 dígits, és a dir, el resultat d'aplicar aquesta funció hash a un missatge ens pot donar un valor entre el 0 i el 999, és a dir 1000 valors. Així, la probabilitat que donada una entrada m el seu valor hash siguin un nombre menor que 1000 serà 1, ja que qualsevol sortida ens donarà un d'aquests valors. Ara bé, si fixem el valor objectiu de la nostra prova de treball en 500, la probabilitat que l'entrada d'aquesta funció sigui menor que 500 és de $\frac{1}{2}$. I si el valor objectiu és 100, la probabilitat és de $\frac{1}{10}$. En aquest últim cas, fixeu-vos que un emissor del sistema Hashcash que vulgui enviar un correu, haurà de regenerar el valor aleatori i recalcular el hash 10 vegades, en mitjana, fins a obtenir una sortida inferior al valor objectiu i per tant un correu que sigui acceptat pel receptor. Per tant, com més petit és el valor objectiu, més dura és la prova de treball.

Aquest mateix mecanisme de prova de treball es fa servir en moltes criptomonedes, com ara el Bitcoin, per assegurar que un usuari no pot gastar de nou uns diners que ja havia gastat prèviament.

Exercici 5.6 Tenim un sistema que utilitza una prova de treball a través d'una funció hash. Aquesta funció hash té una mida de 64 bits i la potència de càlcul de la xarxa que l'utilitza està fixada en 100.000 hashos per segon. Fixeu un valor objectiu de la funció hash per tal que, amb la potència de càlcul que s'indica, es trobi una imatge del hash menor que valor objectiu, en mitjana, cada 10 minuts.

5.4.9 Taules hash

Les funcions hash també s'utilitzen molt sovint com a primitives en la creació d'estructures de dades. Aquestes aplicacions es fan servir a vegades en el context de la seguretat de la informació, però també trascendeixen a altres contextos de les ciències de la computació. A continuació presentarem tres estructures de dades basades en funcions hash: les taules hash, els arbres de Merkle, i els filtres de Bloom.

Les taules hash són una estructura de dades utilitzada per a implementar diccionaris (també coneguts com a arrays associatius), és a dir, estructures que emmagatzemen parells no ordenats de clau-valor, on les claus són úniques. Les taules hash permeten inserir, buscar i eliminar elements de manera eficient.

Una **taula hash** és una estructura de dades que implementa un diccionari o array associatiu.

Exemple 5.16 Exemple de diccionari o array associatiu

Els diccionaris són estructures que es fan servir sovint en programació. A continuació es llisten un parell d'exemples de dades que es poden desar en un diccionari:

- Un diccionari pot utilitzar-se per a desar el hash de la contrasenya d'un conjunt d'usuaris d'un sistema. Les claus del diccionari contindran els identificadors dels usuaris (de manera que no hi ha claus repetides) i el valor associat a cada clau serà el hash de la seva contrasenya.
- Un diccionari pot emmagatzemar els prefixos telefònics de cada província. Les claus del diccionari contindran el nom de la província (que és únic) i el valor associat a cada clau serà el prefix telefònic d'aquella província.

Les taules hash fan servir una funció hash per calcular l'índex d'un element a partir de la seva clau. Aquest índex indica a on està desat l'element.

Així, els procesos d'**afegir**, **eliminar** o **buscar** un element a la taula hash consten d'una primera fase comuna, que consisteix en calcular l'índex de l'element. La segona fase és específica per a cada procés i consisteix a fer l'acció especificada, és a dir, escriure un nou element a la taula, eliminar-ne un d'existent, o bé comprovar si un element hi és.

Exemple 5.17 Exemple de taula hash

Seguint amb l'exemple de les contrasenyes, suposem que disposem del següent diccionari:

```
{
  "morpheus":  0x4c9a82ce72ca2519f38d0af0abbb4cecb9fceca9,
  "neo":       0x356a192b7913b04c54574d18c28d46e6395428ab,
  "trinity":   0x7110eda4d09e062aa5e4a390b0a572ac0d2c0220
}
```

i que el volem implementar amb una taula hash que fa servir com a índex els 4 primers bits del SHA-256 de la clau de cada element. Procedim a calcular l'índex de cada element:

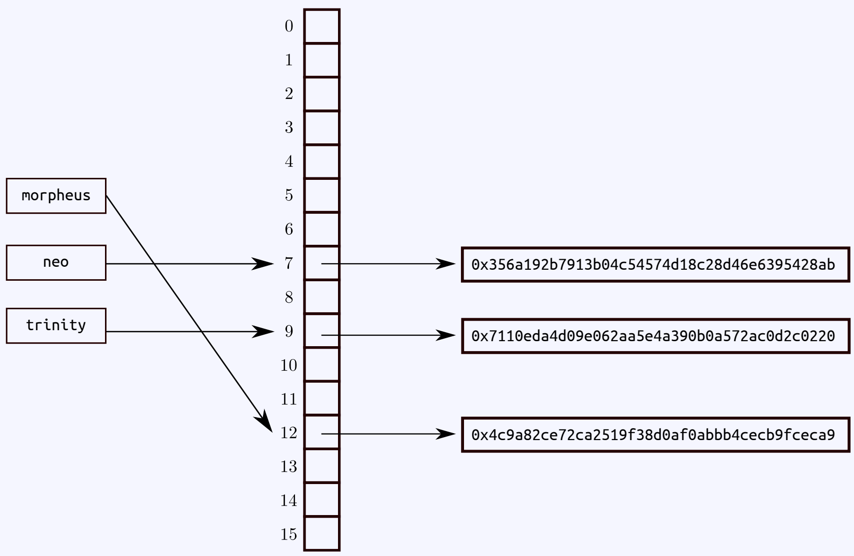
$$\begin{aligned} SHA256(\texttt{morpheus}) &= \texttt{0xc1a1e4aa ...} \\ i(\texttt{morpheus}) &= SHA256(\texttt{morpheus})_{0..3} = \texttt{0xc} = 12 \end{aligned}$$

$$\begin{aligned} SHA256(\texttt{neo}) &= \texttt{73ef176d ...} \\ i(\texttt{neo}) &= SHA256(\texttt{neo})_{0..3} = \texttt{0x7} = 7 \end{aligned}$$

$$\begin{aligned} SHA256(\texttt{trinity}) &= \texttt{0x934a11e6 ...} \\ i(\texttt{trinity}) &= SHA256(\texttt{trinity})_{0..3} = \texttt{0x9} = 9 \end{aligned}$$

on l'expressió $X_{i..j}$ denota els bits des de la posició i a la posició j del valor X.

Aleshores, la taula hash quedaria de la manera següent:



La versió de la taula hash que acabem d'explicar té però un problema evident: no és capaç de gestionar les col·lisions d'índexs. És a dir, si dos elements tenen el mateix índex, no es podran emmagatzemar tots dos, ja que en cada posició només s'hi desa un element. Això és un problema important, que es pot adreçar de diverses maneres.

Una alternativa per a adreçar les col·lisions en taules hash és l'ús de llistes enllaçades. Així, cada posició de la taula hash apunta al primer element d'una llista enllaçada, que contindrà tots els elements que comparteixin el mateix índex. En aquesta variant, els processos de cerca, inserció i eliminació fan ús, per tant, de dues estructures de dades. D'una banda, calculen l'índex de l'element a la taula hash i, d'altra banda, operen sobre la llista enllaçada per tal de cercar, inserir o eliminar elements.

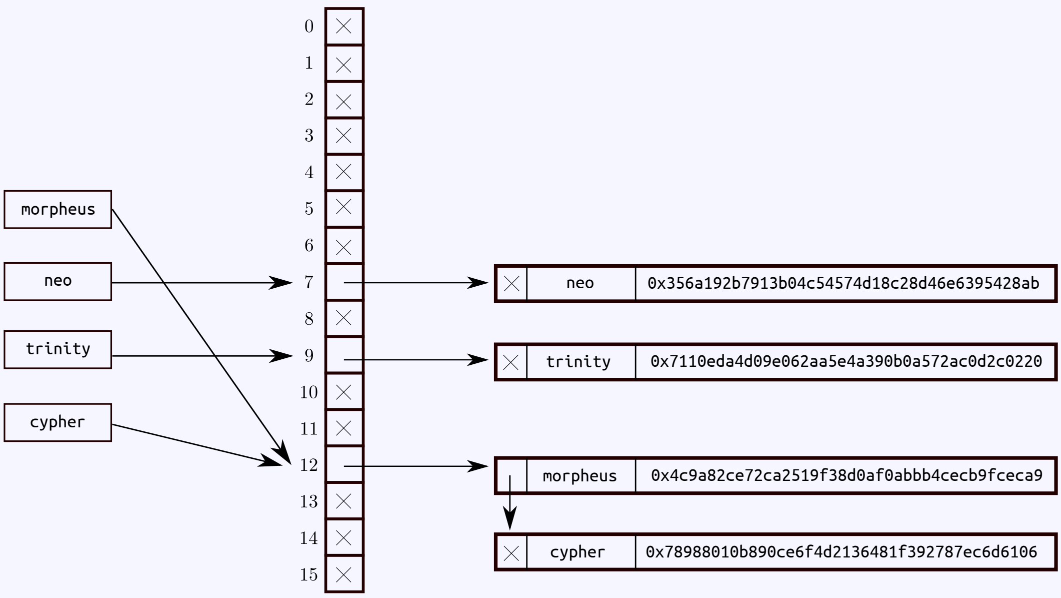
Exemple 5.18 Exemple de taula hash amb llista enllaçada

Suposem que fem servir una taula hash amb llista enllaçada per emmagatzemar els mateixos elements que a l'exemple anterior, més la contrasenya d'un usuari nou, en *cypher*.

En primer lloc, calculem l'índex del nou element:

$$\begin{aligned} SHA256(\texttt{cypher}) &= \texttt{0xc9d22bd2 ...} \\ i(\texttt{cypher}) &= SHA256(\texttt{cypher})_{0..3} = \texttt{0xc} = 12 \end{aligned}$$

Per tant, la taula hash quedaria ara:



Fixeu-vos que ara hem de desar tant la clau com el valor de cada element, ja que hem de poder distingir les contrasenyes de diferents usuaris que comparteixen índexs.

Triar la funció hash adequada per a implementar una taula hash és una tasca complicada i, alhora, crítica. Cal tenir en compte tant la distribució de valors com el rendiment de l'estructura de dades. Sovint es fan servir funcions hash no criptogràfiques per a implementar taules hash, ja que la seva avaluació és molt més ràpida. Així, la funció hash a utilitzar dependrà dels requeriments de l'escenari en què es desplegui la taula hash.

5.4.10 Arbres de Merkle

Els arbres de Merkle van ser proposats l'any 1979 per Ralph Merkle, de qui en deuen el nom, i permeten desar un resum d'un conjunt de dades, de tal manera que es pugui demostrar que una dada pertany a aquest conjunt eficientment.

Una **arbre de Merkle** és un arbre en el qual cada fulla conté el hash d'un bloc de dades, i els nodes interns contenen el hash de la concatenació dels valors dels seus fills.

Habitualment, els arbres de Merkle són binaris, és a dir, cada node intern té com a molt dos fills.

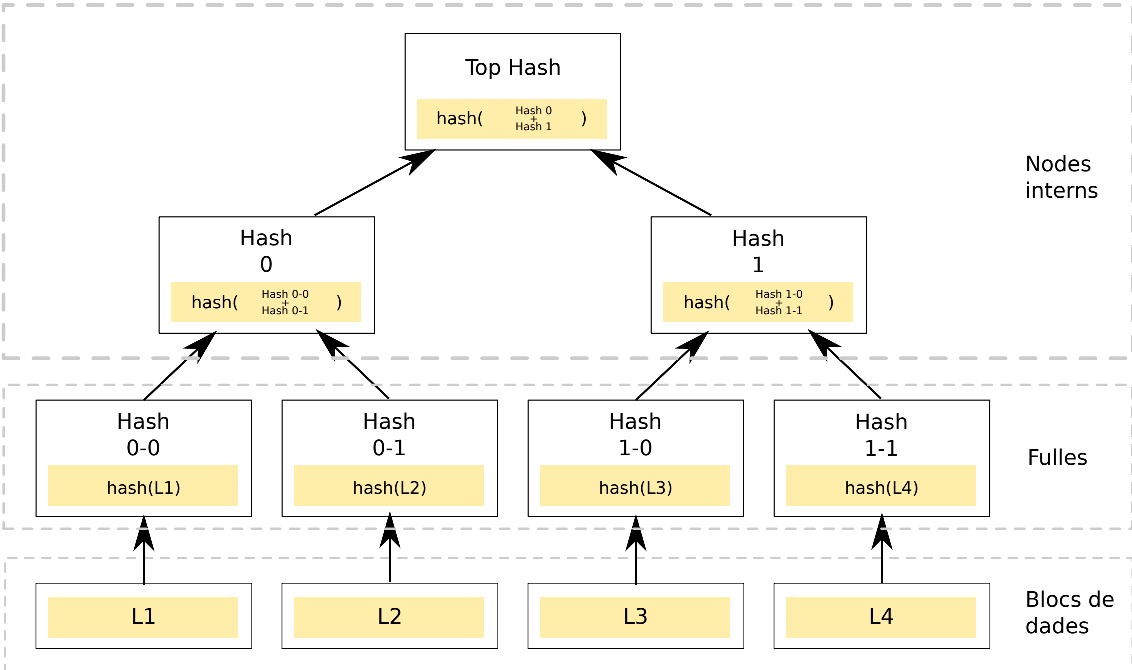


Figura 5.8: Exemple d'un arbre de Merkle. Il·lustració original de David Göthberg, sota llicència CC0. 1.0.

Exemple 5.19 Exemple d'arbre de Merkle

La Figura 5.8 mostra un arbre de Merkle per a quatre blocs de dades $(L_1, \cdots, L_4)$.

L'arbre té quatre fulles (els nodes 0-0, 0-1, 1-0 i 1-1), que contenen el hash de cadascun dels blocs de dades, és a dir, $h_{00} = H(L_1)$, $h_{01} = H(L_2)$, $h_{10} = H(L_3)$ i $h_{11} = H(L_4)$.

El segon nivell de l'arbre té dos nodes: el node 0 conté el hash de la concatenació dels nodes 0-0 i 0-1 $(h_0 = H(h_{00}||h_{01}))$ i el node 1 conté el hash de la contatenació dels nodes 1-0 i 1-1 $(h_1 = H(h_{10}||h_{11}))$.

El primer nivell conté l'arrel de l'arbre, un únic node que desa el hash de la concatenació dels nodes 0 i 1 $(h_r = H(h_0||h_1))$.

Exercici 5.7 Calculeu el hash de l'arrel de l'arbre de Merkle per al següent conjunt de blocs de dades fent servir SHA-256 com a funció hash.

$L_1 =$ `GREAT PYRAMID OF GIZA`
$L_2 =$ `COLOSSUS OF RHODES`
$L_3 =$ `HANGING GARDENS OF BABYLON`
$L_4 =$ `LIGHTHOUSE OF ALEXANDRIA`
$L_5 =$ `MAUSOLEUM AT HALICARNASSUS`
$L_6 =$ `STATUE OF ZEUS AT OLYMPIA`
$L_7 =$ `TEMPLE OF ARTEMIS AT EPHESUS`
$L_8 =$ `MILFORD SOUND`

Els arbres de Merkle són utilitzats per realitzar **proves de pertinença** a un conjunt de manera eficient. Suposem que tenim un conjunt d'n blocs de dades, $\mathscr{L} = \{L_1, \ldots, L_n\}$, i que volem generar-ne un resum, de manera que posteriorment puguem demostrar que els elements L_i (amb $i = 1, \ldots, n$) pertanyen al conjunt $\mathscr{L}$ eficientment. En primer lloc, calcularíem el resum h_r, que correspondria al hash de l'arrel de l'arbre de Merkle amb els blocs de dades d'$\mathscr{L}$ a les fulles. Aquest resum h_r seria l'únic valor que caldria que el verificador desés per tal de poder comprovar, posteriorment, que qualsevol dels blocs L_i pertany a $\mathscr{L}$. Per tal de demostrar que un bloc L_i pertany al conjunt $\mathscr{L}$, el provador genera una prova Π que conté el bloc L_i i els hashos de tots els nodes germans que hi ha en el camí des del node L_i a l'arrel de l'arbre h_r. El verificador pot comprovar que la prova és correcta calculant el hash del bloc L_i i reconstruint l'arbre de Merkle amb els hashos dels germans proporcionats a la prova. Si l'arrel de l'arbre de Merkle calculat és igual a l'arrel h_r que havia emmagatzemat, la prova és correcta, i el verificador queda convençut que $L_i \in \mathscr{L}$.

Exemple 5.20 Exemple de prova de pertinença amb arbre de Merkle

Seguint amb l'exemple de la figura 5.8, suposem que h_r és el valor del hash de l'arrel de l'arbre, i que volem crear una prova de pertinença per al bloc L_3. La prova de pertinença per a L_3 seria $\Pi = (L_3, h_{11}, h_0)$.

Per tal de verificar la prova de pertinença, el verificador procediria a calcular:

$$h_{10} = H(L3)$$

$$h_1 = H(h_{10}||h_{11})$$
$$h_{r'} = H(h_0||h_1)$$

Si $h_{r'} = h_r$, aleshores la prova de pertinença seria satisfactòria. En cas contrari, es rebutjaria la prova.

Un detall a notar és que per tal de verificar la prova de pertinença, el verificador ha de saber en quin ordre concatenar els hashos a cada nivell. Aquesta informació es pot incloure a la prova de pertinença, afegint un únic bit per a cada element que indiqui si és el fill dret o l'esquerra, o bé indicant explícitament la posició del node dins de l'abre.

Exercici 5.8 Genereu una prova de pertinença del bloc de dades `MILFORD SOUND` per a l'arbre de Merkle de l'Exercici 5.7. Valideu la prova generada.

D'una banda, la prova de pertinença que acabem de descriure és eficient i concisa. Independentment del número de blocs de dades n i de la seva mida, el resum h_r a guardar per tal de poder verificar les proves de pertinença és petit, i té una mida constant (que correspondrà a la mida de sortida de la funció hash que es faci servir). Addicionalment, la prova de pertinença conté únicament $\log_2 n$ elements de mida constant (la mida de sortida del hash), més el bloc de dades a verificar. Finalment, el còmput a realitzar per fer la verificació és també de $\log_2 n + 1$ hashos (el hash del bloc de dades i un hash per cada element de la prova).

D'altra banda, si la funció hash que es fa servir en la construcció de l'arbre de Merkle és una funció hash criptogràfica, aleshores un atacant no podrà construir una prova de pertinença falsa, és a dir, no podrà convèncer al verificador que un bloc $L_j \notin \mathscr{L}$ sí que pertany a $\mathscr{L}$. Fixeu-vos que, per a crear una prova de pertinença falsa, l'atacant hauria de ser capaç de crear un nou bloc $L_j \notin \mathscr{L}$ que tingués el mateix hash que algun dels blocs $L_i \in \mathscr{L}$, és a dir, trobar un $L_j \notin \mathscr{L}$ tal que $H(L_j) = H(L_i)$ per a algun $i \in 1, \ldots, n$. Això no és possible ja que una funció hash criptogràfica és resistent a segones preimatges. Una altra estratègia que podria seguir l'atacant és modificar els valors dels hashos germans que conformen la prova de pertinença, per tal d'intentar que el hash de l'arrel de l'arbre de Merkle calculat coincideixi amb l'emmagatzemat, h_r. De nou, això no és possible si la funció hash és criptogràfica, ja que suposaria crear segones preimatges (amb restriccions addicionals sobre el seu contingut).

Les proves de pertinença en arbres de Merkle que hem presentat permeten a un provador demostrar a un verificador que un determinat bloc de dades pertany a un conjunt. Ara bé, tal com les hem presentat, aquestes proves no serveixen per a demostrar el contrari, és a dir, que un bloc de dades no pertany al conjunt. Fixeu-vos que si la prova de pertinença falla, el verificador no pot assegurar que el bloc no es troba present (el bloc pot ser-hi però en una altra posició, o bé els hashos germans presentats poden ser erronis). Una petita variant dels arbres de Merkle permet provar que un element no està en un conjunt.

Una **arbre de Merkle ordenat** és un arbre de Merkle en el qual els blocs de dades de les fulles es troben ordenats, de manera que $L_1 < L_2 < \cdots < L_n$.

Els arbres de Merkle ordenats es poden fer servir per a fer proves de no pertinença. Com en el cas de les proves de pertinença, calcularem el hash de l'arrel de l'arbre, h_r, que serà l'únic valor que el verificador haurà de desar per poder verificar les proves. Ara, per tal de crear una prova que demostri que un bloc de dades L_j no pertany a $\mathscr{L}$, en primer lloc cal localitzar els blocs L_i i L_{i+1} tals que $L_i < L_j < L_{i+1}$. La prova de no pertinença $\bar{\Pi}$ consistirà en els dos blocs de dades, L_i i L_{i+1}, juntament amb les proves de pertinença de cadascun d'ells.

A partir d'aquesta prova $\bar{\Pi}$, es pot verificar que un bloc L_j no pertany a $\mathscr{L}$ de la següent manera. En primer lloc, es comprova que efectivament $L_i < L_j < L_{i+1}$. A continuació, es calculen els hashos dels blocs de dades L_i i L_{i+1}, i es validen les proves de pertinença de casdascun d'aquests blocs. Finalment, es comprova que els blocs de dades L_i i L_{i+1} són blocs consecutius, és a dir, que es troben un immediatament a continuació de l'altre en les fulles de l'arbre de Merkle. Aquesta última comprovació es fa validant la posició que ocupen els blocs de dades en l'arbre, que es pot derivar de l'ordre en què cal concatenar els hashos per tal d'aconseguir obtenir el hash de l'arrel esperat.

També es poden generar proves de no pertinença per a valors L_j inferior a L_1 o superiors a L_n, amb una petita variant del protocol que acabem de presentar.

Exemple 5.21 Exemple de prova de no pertinença amb arbre de Merkle

Seguint amb l'exemple de la Figura 5.8, suposem que h_r és el valor del hash de l'arrel de l'arbre i que els blocs de dades L_i emmagatzemen els següents enters: $L_1 = 31, L_2 = 37, L_3 = 41, L_4 = 43$. L'arbre de Merkle és un arbre ordenat, ja que $L_1 < L_2 < L_3 < L_4$. En aquest exemple crearem una prova de no pertinença per al bloc 42.

En primer lloc, es localitzen els dos blocs de dades consecutius entre els quals es trobaria el bloc de dades 42, que són L_3 i L_4 (ja que $L_3 < 42 < L_4$ i 3 i 4 són consecutius).

La prova de no pertinença seria $\bar{\Pi} = (42, (L_3, h_{11}, h_0), (L_4, h_{10}, h_0))$.

Per tal de verificar la prova de no pertinença, en primer lloc el verificador comprovaria que $41 < 42 < 43$.

Després, procediria a calcular:

$$h_{10} = H(L3)$$
$$h_1 = H(h_{10}||h_{11})$$
$$h_{r'} = H(h_0||h_1)$$

$$h_{11} = H(L4)$$
$$h_1 = H(h_{10}||h_{11})$$
$$h_{r'} = H(h_0||h_1)$$

i validaria que els valors h'_r obtinguts coincideixen amb l'h_r que té emmagatzemat.

Finalment, comprovaria que les fulles en posicions L_3 i L_4 són consecutives.

Si les tres verificacions són satisfactòries, aleshores el provador pot estar segur que 42 no pertany a $\mathscr{L}$.

Els arbres de Merkle es fan servir per a fer proves de pertinença o no pertinença en diversos contextos. Per exemple, es fan servir en la criptomoneda Bitcoin per a que clients lleugers (com podrien ser els que s'executen en un dispositiu mòbil) puguin validar la inclusió de transaccions en els blocs que formen la cadena de blocs (la *blockchain*), sense haver d'emmagatzemar la cadena de blocs sencera (que ocupa diversos centenars de gigabytes). Cada bloc de la cadena conté l'arrel de l'arbre de Merkle de totes les transaccions que s'hi emmagatzemen. Quan un client lleuger necessita comprovar si una transacció s'ha inclòs en un bloc (per exemple, per saber si ha rebut un pagament), el client demana una prova de pertinença de la transacció a la cadena de blocs. Aleshores, un servidor que sí que disposa de totes les dades, genera la prova de pertinença per a la transacció i l'envia al client, que la valida reconstruint l'arbre de Merkle. D'aquesta manera, el client pot estar segur que la transacció s'ha inclòs a la cadena de blocs, ja que el servidor no pot falsificar la prova.

5.4.11 Filtres de Bloom

Els filtres de Bloom van ser proposats l'any 1970 per Burton Howard Bloom. Són estructures de dades que permeten fer testos de pertinença fent servir molt poc espai d'emmagatzemament però, a diferència dels arbres de Merkle, són estructures probabilístiques, que poden retornar resultats erronis (amb una certa probabilitat).

Un **filtre de Bloom** és una estructura de dades probabilística que permet fer testos de pertinença aproximats fent un ús eficient de l'espai d'emmagatzemament.

Un filtre de Bloom pot retornar falsos positius però mai falsos negatius. És a dir, la resposta a una consulta de pertinença amb un filtre de Bloom serà o bé que l'element no es troba en el filtre o bé que probablement sí que hi és.

Un filtre de Bloom f està format per:

1. un vector binari V d'n bits,
2. i un conjunt de k funcions hash independents $h_1, h_2, \cdots, h_k$ que tenen rang $[0, n-1]$.

El procés de **creació del filtre** consisteix a seleccionar els paràmetres del filtre (la mida del vector n, el nombre de funcions hash k i les k funcions hash a utilitzar) i a inicialitzar el vector, assignant 0 a totes les posicions.

Per tal d'**afegir un element al filtre** es procedeix de la manera següent. Primer, s'apliquen les k funcions hash a l'element, obtenint k valors entre 0 i $n-1$ (un valor per a cada funció hash). A continuació, s'assignen a 1 les k posicions del vector indicades per les sortides de les funcions hash. Definirem doncs la funció d'afegir un element e al filtre de Bloom com:

$$V[h_i(e)] = 1 \quad \forall i \in [1,k]$$

on $V[j]$ és la posició j del vector V.

Aquest procediment es repeteix per a tots els elements a afegir al filtre, procés en el qual es poden generar col·lisions. És a dir, es pot haver d'assignar un 1 a una posició que ja havia estat fixada a 1 per un altre element. La freqüència de les col·lisions vindrà determinada per la mida del filtre i el nombre de funcions hash utilitzades. En aquest cas, si un dels bits a assignar a 1 ja és 1, no caldrà modificar-lo, i seguirà sent 1.

Per tal de **comprovar si un element** e **es troba en el filtre**, s'apliquen de nou les k funcions hash a l'element, i es comprova si totes les posicions del vector binari indicades per les sortides de les funcions hash són 1. Si alguna de les posicions indicades conté un 0, aleshores direm amb tota seguretat que l'element no pertany al filtre. En canvi, si totes les posicions contenen un 1, aleshores direm que l'element pertany al filtre, tot i que en aquest cas només podrem afirmar-ho amb certa probabilitat. Així, doncs, definim la funció p que retorna 1 si l'element e es troba en el filtre f i 0 en cas contrari com a:

$$p(e,f) = \prod_{i=1}^{k} V[h_i(e)]$$

És interessant notar perquè un filtre de Bloom mai no dóna falsos negatius. La funció p retornarà 0 si alguna de les posicions indicades per les funcions hash són 0. En aquest cas, tenint en compte que quan s'afegeixen els elements les posicions es marquen amb 1, podem estar segurs que l'element no hi és. En canvi, la funció p retornarà 1 si totes les posicions indicades per les funcions hash són 1. En aquest cas, podria ser que les posicions estessin a 1 perquè s'han modificat a l'afegir l'element, però també podria ser que s'haguessin marcat a 1 afegint d'altres elements, generant aleshores un fals positiu.

Exemple 5.22 Exemple de filtre de Bloom

En aquest exemple tenim un filtre de Bloom f que consta d'un vector binari de mida $n = 8$ bits i $k = 3$ funcions hash.

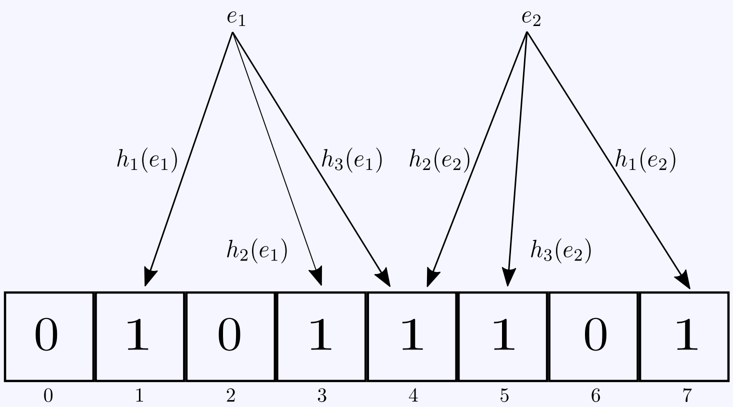
Inicialment, el filtre es troba buit i, per tant, totes les posicions del vector es troben a 0:

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7

A continuació s'afegeixen dos elements, e_1 i e_2, al filtre. Per fer-ho, s'apliquen les tres funcions hash a cadascun dels elements, i s'estableixen a 1 els bits indicats. Suposem que els resultats de les funcions hash són els següents:

$h_1(e_1) = 1$ $\qquad$ $h_1(e_2) = 7$
$h_2(e_1) = 3$ $\qquad$ $h_2(e_2) = 4$
$h_3(e_1) = 4$ $\qquad$ $h_3(e_2) = 5$

La següent figura mostra el filtre de Bloom després d'afegir els elements e_1 i e_2.



A partir del vector binari i les tres funcions hash, es poden fer testos de pertinença sobre el filtre. Així, per exemple, per a comprovar si l'element e_1 pertany al filtre, calcularíem:

$$p(e_1, f) = \prod_{i=1}^{k} V[h_i(e_1)] = V[1] \times V[3] \times V[4] = 1 \times 1 \times 1 = 1$$

i diríem, per tant, que l'element e_1 es troba en el filtre.

Suposem ara que disposem de dos elements addicionals, e_3 i e_4 per als quals també volem comprovar si es troben al filtre, i que els resultats d'aplicar les funcions hash a aquests elements són els següents:

$$h_1(e_3) = 1 \qquad h_1(e_4) = 7$$
$$h_2(e_3) = 0 \qquad h_2(e_4) = 3$$
$$h_3(e_3) = 7 \qquad h_3(e_4) = 1$$

Calculem doncs si els elements es troben al filtre:

$$p(e_3, f) = \prod_{i=1}^{k} V[h_i(e_3)] = V[1] \times V[0] \times V[7] = 1 \times 0 \times 1 = 0$$

$$p(e_4, f) = \prod_{i=1}^{k} V[h_i(e_4)] = V[7] \times V[3] \times V[1] = 1 \times 1 \times 1 = 1$$

Per a e_3 obtenim una resposta correcta, indicant que l'element no es troba al filtre quan, efectivament, no hi és. En canvi, per a e_4 obtenim una resposta errònia: el filtre ens indica que l'element hi és quan, en realitat, aquest no ha estat afegit. El filtre genera un fals positiu per a l'element e_4, produït per les col·lisions que es generen en afegir els elements e_1 i e_2.

Exercici 5.9 Sigui f un filtre de Bloom amb el vector binari següent:

1	1	1	1	0	1	1	1	1	1	1	1	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

i les 5 funcions h_i definides de la manera següent:

$$
\begin{aligned}
h_1(e) &= e \mod 16 \\
h_2(e) &= e+1 \mod 16 \\
h_3(e) &= e+2 \mod 16 \\
h_4(e) &= e+3 \mod 16 \\
h_5(e) &= e+4 \mod 16
\end{aligned}
$$

on els elements e a afegir al filtre sempre són enters.

1. Digueu si els elements següents es troben al filtre:
   $e_1 = 0$, $e_2 = 1429$, $e_3 = 117$ i $e_4 = 15839$.
2. Justifiqueu si l'elecció de les funcions h_i és adient per al seu ús en filtres de Bloom.

Exercici 5.10 Siguin f_1 i f_2 dos filtres de Bloom de la mateixa mida n i que fan servir les mateixes k funcions h_i. Expliqueu com construiríeu un únic filtre que contingui tots els elements que hi ha en els dos filtres.

Pel que fa a l'eficiència de l'estructura de dades, els filtres de Bloom permeten tant afegir elements com consultar si hi pertanyen amb complexitat temporal $\mathcal{O}(k)$, ja que les dues operacions impliquen calcular k hashos. És a dir, afegir elements i comprovar si hi són no depèn de la mida del filtre ni del nombre d'elements que hi pertanyen! Aquesta característica dels filtres de Bloom els fa adequats per a tractar certs tipus de problemes com els que presentem a continuació.

Exemple 5.23 L'ús de filtres de Bloom en aplicacions reals

Un dels usos més habituals dels filtres de Bloom és com a part d'un sistema de *cache*. Per exemple, a l'estudiar el problema del disseny de sistemes de *cache* per a pàgines web, es va observar que la gran majoria de pàgines només són descarregades una única vegada, mentre que un conjunt petit de pàgines es descarreguen molt sovint. A partir d'aquesta observació, els proveïdors intenten crear sistemes de *cache* que incloguin aquest conjunt petit de pàgines que es fan servir sovint, ja que això permet optimitzar la descàrrega sense consumir innecessàriament recursos de *cache* per a pàgines que no tornaran a descarregar-se més.

En aquest cas, es pot servir un filtre de Bloom per emmagatzemar les pàgines que han estat visitades alguna vegada. Quan un client fa una petició d'una pàgina, es consulta el filtre per saber si aquesta pàgina ja ha estat buscada anteriorment.

- Si la pàgina no es troba al filtre, vol dir que no ha estat buscada en el passat. Aleshores, s'afegeix la pàgina al filtre i es recupera de l'emmagatzemament principal. Com que la pàgina només ha estat buscada una vegada, aquesta no s'afegeix a la *cache*, ja que potencialment no és d'interés per a altres usuaris.
- Si la pàgina es troba ja al filtre, vol dir que aquesta ja havia estat consultada en el passat. Aleshores, s'intenta recuperar la pàgina de la *cache*. Si hi és, se serveix al client aquesta versió, guanyant velocitat de descàrrega. En canvi, si la pàgina no es troba a la *cache*, voldrà dir que és el segon cop que es busca, i aleshores s'afegirà a la *cache*.

D'aquesta manera, la cache contindrà totes les pàgines que s'han buscat com a mínim dues vegades.

Facebook i Akamai fan servir aquest tipus d'estratègies en les seves plataformes.

Un filtre de Bloom és una tècnica eficient per a un sistema de *cache* com el que acabem de presentar: el nombre de possibles pàgines a descarregar és immens (de manera que mantenir una llista completa amb el nombre de descàrregues de cada pàgina seria costós) i un fals positiu no provoca un error en el sistema (sinó que simplement implica afegir una pàgina addicional a la *cache*).

Probabilitat de generar falsos positius

El disseny del filtre de Bloom presenta un compromís entre l'espai que es vol destinar al filtre i la probabilitat de generar falsos positius que es vol acceptar.

La probabilitat de generar un fals positiu en un filtre de Bloom (FPP o *False Positive Probability*) ve determinada per la mida del vector binari (n), el nombre funcions hash (k) i el nombre d'elements que conté (m):

$$FPP(n,k,m) = \left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^{k}$$

Vegem pas per pas d'on sorgeix aquesta expressió. La probabilitat que un bit específic del vector segueixi a 0 després d'haver afegit un element al filtre és $(1-1/n)^k$, ja que amb probabilitat $1/n$ el bit es fixarà a 1 per cadascuna de les k funcions hash. Després d'haver afegit els m elements, la probabilitat que un bit segueixi a 0 és doncs $(1-1/n)^{km}$ (repetim m vegades el procés d'afegir un element). Finalment, la probabilitat de generar un fals positiu és la probabilitat que les k posicions consultades per a l'element siguin 1.

Així doncs, donat un filtre d'una mida i nombre de funcions hash determinats, la FPP augmenta conforme es van afegint elements al filtre. La Figura 5.9 mostra com varia la probabilitat d'un fals positiu per a un filtre de 64 bits que fes servir dues, tres o quatre funcions hash.

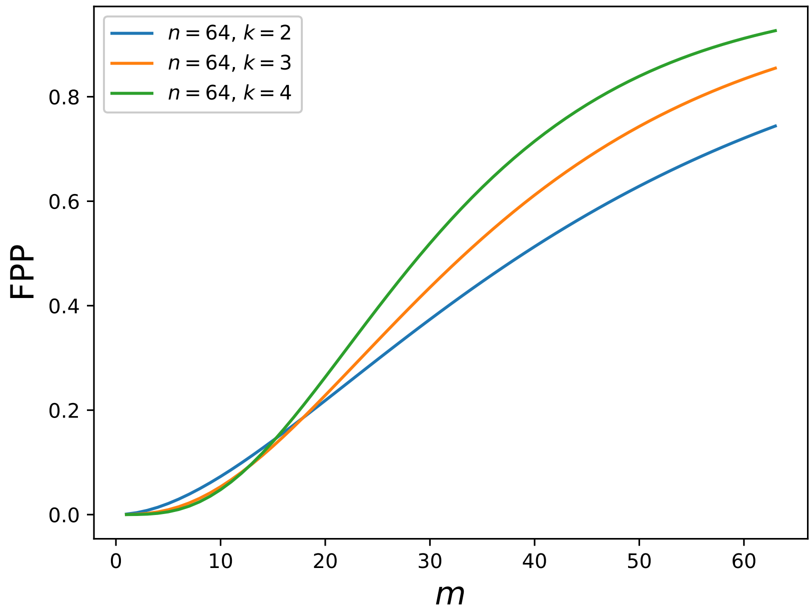


Figura 5.9: Probabilitat de fals positiu (FPP) segons el nombre d'elements del filtre (m), per a $k=2$ (blau), $k=3$ (taronja) i $k=4$ (verd) funcions hash.

Donat un filtre de mida n amb m elements, ens podem preguntar quin és el nombre de funcions hash k òptim per tal de minimitzar la FPP del filtre. La resposta no és immediata, doncs d'una banda, augmentar k permet comprovar més bits per cada element que es vulgui testejar, minimitzant així la FPP però, d'altra banda, disminuir k permet augmentar la probabilitat de trobar un bit a 0, que és el que ens permet evitar un fals positiu.

El valor de k òptim per minimitzar la FPP ve donat per l'expressió següent:

$$k_{opt} \approx \frac{n}{m} \ln 2$$

Nombre òptim de funcions hash	El lector interessat pot consultar el capítol 5 del llibre Mitzenmacher, Michael, and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis.* Cambridge university press, 2017, per aprendre com deduir l'expressió que permet calcular el nombre òptim de funcions hash a partir de l'expressió de la FPP.

Així, el nombre òptim de funcions hash ve determinat pel factor n/m, que representa el nombre de bits per element emmagatzemat al filtre. La Figura 5.10 mostra l'evolució de la FPP en base al nombre de funcions hash que s'utilitzen per a filtres amb diferents bits per element (n/m). També s'hi mostra el nombre òptim de funcions hash a fer servir en cadascun dels casos.

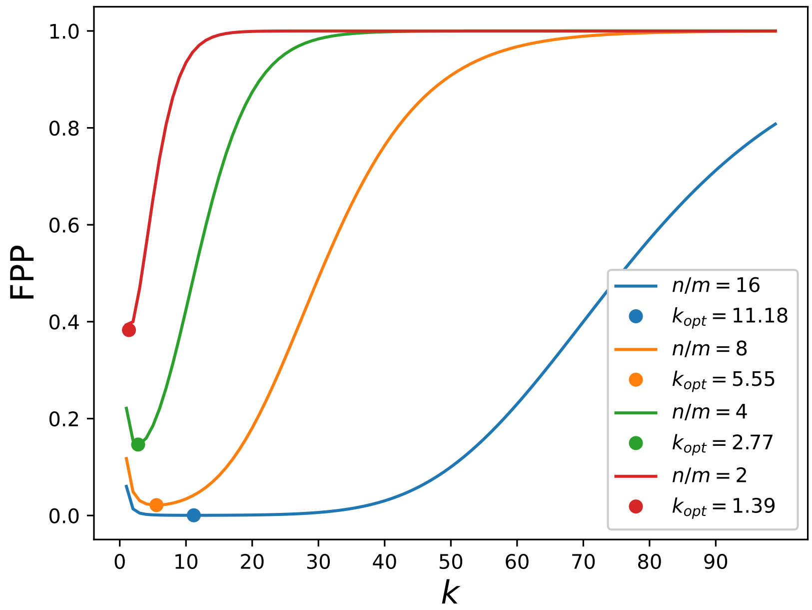


Figura 5.10: Probabilitat de fals positiu (FPP) segons el nombre de funcions hash (k).

Cal tenir en compte que l'expressió que permet calcular k_{opt} pot retornar un nombre real, però el nombre de funcions hash d'un filtre sempre serà un enter, que caldrà triar en el moment del disseny.

Les funcions hash dels filtres de Bloom

Els filtres de Bloom fan ús de diverses funcions hash, que han de ser independents entre elles i han de tenir una distribució de sortida uniforme.

Es fan servir diferents tècniques per tal de poder implementar aquestes funcions sense fer ús de funcions hash diferents, cosa que sovint seria molt costosa. Així, per exemple, es poden agafar diferents parts de la sortida d'una mateixa funció hash per a cadascuna de les h_i; es pot concatenar un valor inicial, diferent per a cada h_i, a l'entrada de la funció hash; o bé es poden combinar les sortides de dues funcions hash per a recrear-ne les k necessàries.

Exemple 5.24 Exemples de definicions per a les h_i

Suposem que construim un filtre per a una *cache* d'$n = 256$ bits i $k = 4$ funcions hash, i que hi volem afegir la pàgina `uoc.edu`. Per a indexar les 256 posicions del filtre necessitem 8 bits ($2^8 = 256$), de manera que cadascuna de les h_i haurà de tenir una sortida de 8 bits.

Particionar la sortida d'una funció hash

Una manera d'implementar les quatre h_i que necessitem és fer servir una única funció hash que tingui una sortida de 32 bits com a mínim, i prendre blocs de 8 bits d'aquesta sortida per a cadascuna de les k funcions hash.

Per exemple, prenem el SHA-1, que té una sortida de 160 bits, com a funció hash base, i calculem els h_i de la manera següent:

$$
\begin{aligned}
SHA1(\texttt{uoc.edu}) &= \texttt{0xe6a62a58a28f94d745d3ea9a47163c846a065a3c} \\
h_1(\texttt{uoc.edu}) &= SHA1(\texttt{uoc.edu})_{0..7} = \texttt{0xe6} = 230 \\
h_2(\texttt{uoc.edu}) &= SHA1(\texttt{uoc.edu})_{8..15} = \texttt{0xa6} = 166 \\
h_3(\texttt{uoc.edu}) &= SHA1(\texttt{uoc.edu})_{16..23} = \texttt{0x2a} = 42 \\
h_4(\texttt{uoc.edu}) &= SHA1(\texttt{uoc.edu})_{24..31} = \texttt{0x58} = 88
\end{aligned}
$$

on l'expressió $X_{i..j}$ denota els bits des de la posició i a la posició j del valor X.

L'avantatge d'aquest mètode és que només requereix el càlcul d'una única funció hash. En canvi, però, el nombre de bits que s'obtenen queda limitat per la mida de la sortida de la funció hash, de manera que no serveix per a filtres molt grans o que utilitzin moltes funcions hash.

Exercici 5.11 Sigui f un filtre de Bloom amb un vector $n = 65536$ posicions i $k = 10$. Justifiqueu quina de les següents tres funcions hash seria més adient per a utilitzar per definir les 10 funcions h_i amb la tècnica de particionar la sortida i proposeu una possible definició de les h_i.

1. MD5
2. SHA1
3. SHA256

Ús d'una llavor

Una alternativa és fer servir una llavor concatenada amb l'element com a entrada d'una única funció hash, i variar el valor de la llavor per a cada h_i. Per exemple, si prenem de nou el SHA-1 com a funció hash base i un comptador com a llavor, procediríem a calcular:

$$
\begin{aligned}
h_1(\texttt{uoc.edu}) &= SHA1(\texttt{1uoc.edu})_{0..7} = \texttt{0x81} = 129 \\
h_2(\texttt{uoc.edu}) &= SHA1(\texttt{2uoc.edu})_{0..7} = \texttt{0x87} = 135 \\
h_3(\texttt{uoc.edu}) &= SHA1(\texttt{3uoc.edu})_{0..7} = \texttt{0xe9} = 233 \\
h_4(\texttt{uoc.edu}) &= SHA1(\texttt{4uoc.edu})_{0..7} = \texttt{0x16} = 22
\end{aligned}
$$

En aquest cas, com que només necessitem 8 bits per cada h_i, hem conservat els primers 8 bits de la sortida i hem descartat la resta de bits.

L'avantatge d'aquest mètode respecte a l'anterior és que no té límit en relació al nombre de funcions hash k a implementar ni el nombre de bits de sortida de cada h_i individual (si se'n necessiten més dels que ofereix la sortida de la funció hash base, es poden fer servir diverses llavors per a cada h_i). No obstant això, aquest mètode és molt més costós computacionalment que el mètode de particionar la sortida, ja que per cada element a afegir o comprovar caldrà calcular diversos hashos.

Alternatives eficients per al càlcul de les funcions hash

Per a conèixer una alternativa eficient per a derivar les k funcions hash recomanem la lectura de l'article Kirsch, Adam, and Michael Mitzenmacher. *"Less hashing, same performance: building a better bloom filter."* European Symposium on Algorithms. Springer, Berlin, Heidelberg, 2006.

Més enllà de com aconseguir k funcions hash per a construir filtres de Bloom, ens podem preguntar també quin tipus de funció hash és adient com a funció hash base en aquestes construccions. Hem esmentat que és necessari que les diferents h_i siguin independents entre elles, i també que generin una sortida uniforme. Les funcions hash criptogràfiques poden complir aquests requisits. Ara bé, fer servir com a funció base una funció hash criptogràfica (com ara el SHA1) és costós computacionalment. Serien útils, per a aquesta aplicació, l'ús de funcions hash no criptogràfiques, que tot i que no compleixen certs requisits de seguretat, són molt més ràpides de calcular? La resposta no és absoluta, i dependrà de l'entorn en el qual preveiem desplegar el filtre de Bloom. Si l'entorn no té adversaris, potser podem fer servir funcions no criptogràfiques, sempre que es repecti la uniformitat de les sortides i la independència entre les h_i que en derivem. Alguns exemples de funcions hash no criptogràfiques que es fan servir en implementacions de filtres de Bloom són la funció hash Murmur3 o la funció Fowler-Noll-Vo (FNV). En canvi, si l'entorn en el qual despleguem el filtre pot tenir adversaris, que tinguin un interés en fer fallar els testos de pertinença, aleshores en general serà preferible l'ús de funcions criptogràfiques, ja que les seves propietats faran el filtre més robust a atacs. En qualsevol cas, cal estudiar amb detall l'escenari i els possibles adversaris que s'hi poden trobar, per tal de decidir quin tipus de funció hash cal implementar.

Murmur3

La funció hash no criptogràfica Murmur3 deu el seu nom a les operacions en què basa el seu funcionament: **mu**ltiplicar-**r**odar-**mu**ltiplicar-**r**odar. La primera versió d'aquesta funció hash (coneguda com a Murmur1) va fer-se pública el 2008, i la versió actual té dues variants: Murmur3A, que genera una sortida de 32 bits, i Murmur3F, que té una sortida de 128 bits.

Fowler-Noll-Vo (FNV)

La funció hash no criptogràfica Fowler-Noll-Vo deu el seu nom als autors que la van dissenyar. La primera versió es va començar a gestar al 1991. La versió actual d'aquesta funció ofereix variants amb sortides de 32, 64, 128, 256, 512 i 1024 bits.

Variants de filtres de Bloom

Els filtres de Bloom que acabem de descriure corresponen a la variant bàsica d'aquesta estructura de dades. Ara bé, existeixen una gran diversitat de variants dels filtres de Bloom, cadascuna de les quals aporta alguna nova característica en relació a la versió bàsica.

Així, per exemple, la variant bàsica del filtre de Bloom no permet eliminar elements del filtre. Una vegada s'ha afegit un element ja no es pot esborrar, ja que si fixéssim a 0 totes les posicions indicades per les funcions hash per a aquell element, podríem estar afectant altres elements que també haguessin modificat aquelles posicions. Els filtres de Bloom amb comptadors (en anglès, es coneixen com a *Counting Bloom filters*) són una variant que permet eliminar elements.

Els **filtres de Bloom amb comptadors** canvien el vector binari per un vector d'enters, que s'inicialitza també a 0. Per a afegir un element, s'incrementa el comptador de les posicions indicades per les funcions hash. D'aquesta manera, es pot definir una operació d'esborrat, que consisteix simplement en decrementar el comptador de les posicions afectades.

Exercici 5.12 Sigui f un filtre de Bloom amb comptadors d'$n = 16$ posicions i $k = 3$ funcions hash, on $h_i = SHA1_{4i..4i+3}$.

1. Mostreu el contingut del filtre després d'afegir els tres elements següents: `uoc.edu`, `cv.uoc.edu`, `biblioteca.uoc.edu`.

2. Elimineu l'element `uoc.edu` del filtre i mostreu com queda el vector després d'aquesta operació.

Un altre dels problemes que presenta la variant bàsica en el seu ús en aplicacions reals és que cal decidir la mida del filtre abans de començar a treballar-hi, en base al nombre d'elements que s'hi preveuen emmagatzemar i la probabilitat de falsos positius que l'aplicació pot tolerar. Ara bé, estimar el nombre d'elements que s'hi emmagatzemaran abans de desplegar l'aplicació pot no ser fàcil i les conseqüències d'una mala estimació afectaran al rendiment. D'una banda, si l'estimació és superior als elements que realment s'hi emmagatzemen, estarem desaprofitant espai de disc. D'altra banda, si l'estimació és inferior, la probabilitat de falsos positius augmentarà per sobre del llindar que l'aplicació pot tolerar. Els filtres de Bloom escalables permeten afrontar aquest problema, oferint la possibilitat d'augmentar la mida dels filtres a mesura que aquests es van omplint.

Els **filtres de Bloom escalables** (SBF) estan formats per un o més filtres de Bloom bàsics. Quan els filtres existents en un moment donat s'omplen, aleshores s'afegeix un nou filtre bàsic a l'SBF. Cada nou filtre es dissenya de manera que la probabilitat de fals positiu en l'estructura completa (l'SBF) sigui l'especificada en el moment del disseny. D'aquesta manera, es pot desplegar una aplicació amb un filtre de Bloom petit, i anar-lo ampliant conforme creixen les necessitats de l'aplicació sens que augmenti la probabilitat de falsos positius.

5.5 Funcions hash amb propietats addicionals

Algunes de les aplicacions que acabem de presentar es poden beneficiar de l'ús de funcions hash amb algunes propietats addicionals, més enllà de les necessàries per a funcions hash criptogràfiques que s'han presentat a l'inici del capítol. Una d'aquestes propietats és que siguin computacionalment costoses de calcular i/o difícilment optimitzables en hardware específic. Aquesta propietat pot ser d'interès en les funcions hash utilitzades per emmagatzemar contrasenyes, per derivar claus o en proves de treball.

En el cas de les contrasenyes i la derivació de claus, augmentar el temps de còmput de la funció hash té poc impacte en l'ús legítim de les aplicacions, doncs l'usuari legítim que s'ha d'autenticar només necessita calcular un únic resultat. En canvi, aquest augment dificulta els atacs contra aquests sistemes, ja que els atacants necessiten calcular moltes vegades la funció hash (per exemple, per fer atacs de diccionari o de força bruta).

En el cas de les proves de treball, la situació és similar en el seu ús com a protecció per a correu brossa. El cas de les criptomonedes és una mica diferent, i té a veure amb la descentralització del minat: certs tipus de funcions hash són fàcilment implementables en dispositius hardware específics (ASICs o FPGAs) per al càlcul de la funció hash, però la creació i adquisició d'aquests dispositius no es troba a l'abast de tothom. Això fa que hi hagi un interès en evitar utilitzar funcions hash optimitzables per hardware, ja que aquestes porten a dificultar l'accés al minat per al públic en general.

ASICs

Un ASIC (de l'anglès, *application-specific integrated circuit*) és un circuit integrat d'aplicació específica, és a dir, un circuit dissenyat i fabricat per a dur a terme una funció específica. Això es contraposa amb circuits d'ús genèric, com ara les CPUs, que estan pensades per a poder executar diverses aplicacions diferents. Els ASICs es dissenyen fent servir llenguatges de descripció de harware, que després se sintetizen per produir una descripció a nivell de portes lògiques de l'aplicació. El disseny i creació d'un ASIC té uns costos fixos molt elevats, però en canvi són circuits molt eficients (en quant a velocitat i consum energètic) per a fer la tasca per la qual estan dissenyats. A més, els costos variables són petits, de manera que són adients per a fer grans produccions.

FPGAs

Una FPGA (de l'anglès, *Field-programmable gate array*) és un circuit dissenyat per a ser configurat després de la seva producció. Les FPGAs contenen arrays de blocs lògics, que poden implementar portes lògiques o altres funcions més compexes, i permeten configurar les interconnexions entre aquests blocs. Com els ASICs, es configuren fent servir també llenguatges de descripció de hardware. Les FPGAs ofereixen un rendiment menor que els ASICs, però el cost d'una implementació és molt més barat que el d'un ASIC, ja que són configurables després d'haver sortit de la fàbrica de producció.

Així, hi ha un interès creixent en el disseny de funcions hash resistents a ASICs, és a dir, funcions hash que no donin un gran avanatge al ser implementades en ASICs. Una de les tècniques que s'utilitza és la creació de funcions amb un ús intensiu de memòria (en anglès, es coneixen com a *memory-hard functions*). Com que la memòria té un cost similar, tant si es fa servir en un ASIC com des d'un dispositiu de propòsit general, les funcions que requereixen d'un ús intensiu de la memòria no es beneficien molt de la seva implementació en ASICs.

Una funció hash amb ús intensiu de memòria (en anglès, en diem ***memory-hard hash function***) és una funció hash que requereix d'un ús intensiu de memòria per a avaluar-la de manera ràpida.

Noteu que l'ús intensiu de memòria no és un requisit indispensable per a poder avaluar la funció hash: la memòria és necessària per a avaluar-la de manera ràpida. Si no es disposa de la memòria, aleshores la funció es pot avaluar però aquesta avaluació és molt més lenta.

La funció hash **scrypt** és una funció amb ús intensiu de memòria feta presentada l'any 2009 i publicada com a RFC el 2016 (RFC 7914). Per aconseguir l'ús intensiu de memòria, la funció fa ús d'un vector pseudoaleatori molt gran. Durant l'execució de la funció, cal accedir a diversos elements d'aquest vector, en un ordre també pseudoaleatori, i recuperant una mateixa posició diverses vegades. La generació d'aquest vector és un procés computacionalment costós. Així, una implementació que emmagatzemi el vector sencer serà ràpida a executar-se, ja que només calcularà el vector una única vegada i recuperarà els elements que vagi necessitant del vector durant l'execució. En canvi, una implementació que no desi aquest vector, necessitarà calcular els elements cada vegada que els necessiti, un procés costós per disseny.

5.6 Resum

Com hem pogut veure en aquest capítol, les funcions hash són una eina criptogràfica extremadament versàtil que s'utilitza cada vegada més en diferents aplicacions i protocols criptogràfics. La seva principal característica és la impossibilitat de predir-ne la sortida, malgrat conèixer-ne l'entrada, tot i assumint que la definició de la funció hash queda totalment determinada de forma pública. A més, aquesta predicció no es pot realitzar tot i conèixer la sortida d'altres valors propers a l'entrada, ja que les propietats d'aquestes funcions impliquen que un petit canvi en l'entrada provoqui un canvi significativament gran en la sortida.

Per aconseguir aquestes característiques, hem vist que les funcions hash estan formades per un seguit de subfuncions que incorporen un alt grau de no-linealitat justament per tal de fer imprevisible la seva sortida. A més, les operacions internes d'una funció hash s'iteren diverses vegades perquè encara sigui més complicat realitzar-ne una anàlisi. Per aquest motiu, la simple definició d'una funció hash, com ara el SHA-256, ja implica una complexitat força elevada.

Gràcies a aquestes propietats, les funcions hash es poden utilitzar per realitzar autenticació de missatges amb criptografia de clau simètrica, per obtenir resums quasi unívocs de missatges, per a generar valors pseudoaleatoris, en protocols de compromís, per a l'emmagatzemament de contrasenyes o bé per a la derivació de claus. A més, les funcions hash són també un dels pilars de les noves criptomonedes, gràcies a la seva utilització en les proves de treball, els arbres de Merkle o els filtres de Bloom.

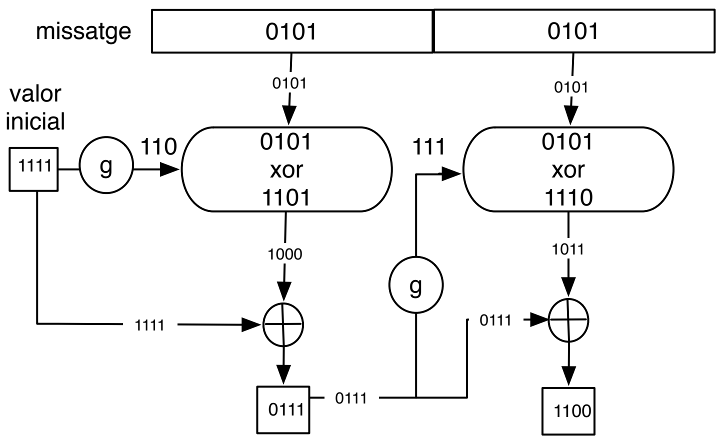
5.7 Solucions dels exercicis

Exercici 5.1:

Utilitzant les expressions de l'Apartat 1.2, la probabilitat que en un grup de 50 persones triades a l'atzar, dues d'elles tinguin l'aniversari el mateix dia és $1-[(1-\frac{1}{365})\cdot(1-\frac{2}{365})\cdots(1-\frac{49}{365})]=0,97$. D'altra banda, la probabilitat que almenys una d'elles hagi nascut el dia 1 de gener és de $1-(1-\frac{1}{365}^{49})=0,12$

Exercici 5.2:

El resultat de la funció hash és el valor 1100 i el procés de càlcul es mostra en el següent gràfic:



Exercici 5.3:

En primer lloc, cal passar el valor 'SALA' a una cadena de bits, obtenint:

01010011010000010100110001000001.

A continuació afegim un '1' a la cadena, obtenint 010100110100000101001100010000011.

Després, hi afegim $448-(32+1)=415$ zeros i els últims 64 bits són la representació en binari de la mida del missatge 'SALA' en bits. Com que la mida del missatge era de 32 bits, tenim que els 64 bits finals del missatge són: $000\cdots00100000$. Per tant els 512 bits de padding són:

$$010100110100000101001100010000011\ \underbrace{00\cdots0}_{415\text{ zeros}}\ \underbrace{00\cdots00100000}_{64\text{ bits}}$$

Exercici 5.4:

Els resultats de les funcions són els següents:

$ROTR^7(m_1)=11111110000000000000000111111111$
$SHR^{10}(m_1)=00000000000000000000000000111111$
$\sigma_0(m_1)=11000001111111111101111000000000$
$\sigma_1(m_1)=01100000000000000110000000111111$
$\sum_0(m_1)=00111100000001111100001111111000$
$\sum_1(m_1)=00000011100111111111110001100000$
$\mathrm{Ch}(m_1,m_2,m_3)=11111111000000001111111111110000$
$\mathrm{Maj}(m_1,m_2,m_3)=11110000000000001111111111110000$

Exercici 5.5:

La utilització d'una HMAC amb *secret prefix* no és segura perquè la informació secreta s'afegeix a l'inici i a continuació es calcula el hash. Això fa que en un disseny estàndard de funció hash, afegir un bloc al final d'un missatge únicament requereixi continuar iterant la funció hash amb el nou bloc sense necessitat de conèixer el valor de la clau secreta utilitzada per calcular l'HMAC. Així, en aquest exercici, com que volem calcular l'HMAC de la cadena 01111111 però coneixem l'HMAC de la cadena més curta 0111, només caldrà utilitzar la funció hash una iteració més amb el valor 0111 com a vector inicial i el fragment nou, 1111, com a missatge a calcular el hash. Si realitzem els càlculs trobarem que l'HMAC per al missatge 01111111 serà 0001.

Exercici 5.6:

Una funció hash de 64 bits té un nombre total de possibles imatges de 2^{64}. En 10 minuts, si podem provar 100.000 hashos per segon, haurem provat $100.000 \cdot 60 \cdot 10 = 60.000.000$ valors. Per tant, el valor objectiu per a la nostra prova de treball serà $\frac{2^{64}}{60.000.000} \approx 307445734561$ que si ho expressem com una cadena de 64 bits tenim

0000000000000000000000000100011110010101001100011001110010100001

Exercici 5.7:

En primer lloc es calculen els hashos de cadascun dels blocs de dades:

$$h_{000} = H(L_1) = \texttt{ea64f9d949b9508d847c85de6a03a0db71258ee7d7b01d135c5b7c794bbb9848}$$
$$h_{001} = H(L_2) = \texttt{9689fc250b6c60ec0c5c6b6f9bc7e621c69df7febb4ad1e641cc4e7792b26146}$$
$$h_{010} = H(L_3) = \texttt{876cb92390cf4b52cb3e58a48b7f221eccd99f01c1925eefc9aa1da5d4c88901}$$
$$h_{011} = H(L_4) = \texttt{bb063417e4eddab0426c3cdfef9da597af388a60690ceaa01fb15f49ce30c1e4}$$
$$h_{100} = H(L_5) = \texttt{ce9b1b67eb23b708147ac7a35c5a77ecdedf9ac991e0f084d1248521918795ab}$$
$$h_{101} = H(L_6) = \texttt{94335c418b09a0223a6f90757209ca69f96a26f5705d6c14d132b33cdea84c4e}$$
$$h_{110} = H(L_7) = \texttt{032cf476fae7facbb766e98e7dac817e75cd29434b15108e2b10b3d9b3a68967}$$
$$h_{111} = H(L_8) = \texttt{d7409ed2339ca2803e527d63ec1e4b08ae3705ebf97b8de4646911f91ee1294c}$$

Després, es procedeix a calcular els hashos del tercer nivell de l'arbre:

$$h_{00} = H(h_{000}||h_{001}) = H(\texttt{ea64f9d9}\cdots\texttt{9689fc25}\cdots) =$$
$$= \texttt{f4aceecdaec0b6e7702331a7d5c4d2dfd708dcda5becc9e6ae2ebcb55b891771}$$
$$h_{01} = H(h_{010}||h_{011}) = \texttt{344bdb1d3370f4e61c676421c048a4c9284ea5538dd678b4d2bb7ee9756c5337}$$
$$h_{10} = H(h_{100}||h_{101}) = \texttt{3dbbcce400af55fc8acdca4013550a0d76169ce584a82b802ddf2b4a2761c897}$$
$$h_{11} = H(h_{110}||h_{111}) = \texttt{2988ac56ddbfed1dfae8966ef1dfa1d8fe31bbd52f0002afd2d4cbbe1148f8e8}$$

A continuació es calculen els hashos del segon nivell:

$$h_0 = H(h_{00}||h_{01}) = H(\texttt{f4aceecd}\cdots\texttt{344bdb1d}\cdots) =$$
$$= \texttt{04d15a9750de7caa93411a12a1d53a9672cf1fd5213e31d4241650f4e7f34a59}$$
$$h_1 = H(h_{10}||h_{11}) = \texttt{e2f528e5024516fdaaeaeb2b92fcde9b8228908f780141e936a89b6cd6dc6c0d}$$

I finalment es calcula l'arrel de l'arbre:

$$h_r = H(h_0||h_1) = H(\texttt{04d15a97}\cdots\texttt{e2f528e5}\cdots) =$$
$$= \texttt{3cb579c97652a53b9997027665390bf927943a1cc09b69e5a5c2518862811d37}$$

Exercici 5.8:

La prova de pertinença per al bloc de dades `MILFORD SOUND` és:

$$\begin{aligned}\Pi &= (\texttt{MILFORD SOUND}, h_{110}, h_{10}, h_0) = \\ &= (\texttt{MILFORD SOUND}, \texttt{032cf476}\cdots, \texttt{3dbbcce4}\cdots, \texttt{04d15a97}\cdots)\end{aligned}$$

El verificador, que coneix $h_r = \texttt{3cb579c9}\cdots$ procedirà a validar la prova calculant:

$$\begin{aligned}h_{111} &= H(\texttt{MILFORD SOUND}) = \texttt{d7409ed2}\cdots \\ h_{11} &= H(h_{110}||h_{111}) = H(\texttt{032cf476}\cdots\texttt{d7409ed2}\cdots) = \texttt{2988ac56}\cdots \\ h_1 &= H(h_{10}||h_{11}) = H(\texttt{3dbbcce4}\cdots\texttt{2988ac56}\cdots) = \texttt{e2f528e5}\cdots \\ h_r' &= H(h_0||h_1) = H(\texttt{04d15a97}\cdots\texttt{e2f528e5}\cdots) = \texttt{3cb579c9}\cdots\end{aligned}$$

Com que h_r' és efectivament igual a h_r, el verificador donarà la prova per vàlida.

Exercici 5.9:

1. L'element $e_1 = 0$ no es troba al filtre, ja que el bit $h_5(0) = 4$ és 0. En canvi, els elements e_2, e_3 i e_4 sí que es troben al filtre, ja que totes les posicions indicades pels h_i són 1 al vector:

$$\begin{aligned}&h_1(e_2) = h_1(1429) = 5;\ h_2(e_2) = h_2(1429) = 6;\ h_3(e_2) = h_3(1429) = 7; \\ &\qquad h_4(e_2) = h_4(1429) = 8;\ h_5(e_2) = h_5(1429) = 9; \\ &h_1(e_3) = h_1(117) = 5;\ h_2(e_3) = h_2(117) = 6;\ h_3(e_3) = h_3(117) = 7; \\ &\qquad h_4(e_3) = h_4(117) = 8;\ h_5(e_3) = h_5(117) = 9; \\ &h_1(e_4) = h_1(15839) = 15;\ h_2(e_4) = h_2(15839) = 0;\ h_3(e_4) = h_3(15839) = 1; \\ &\qquad h_4(e_4) = h_4(15839) = 2;\ h_5(e_4) = h_5(15839) = 3;\end{aligned}$$

2. La tria de les funcions h_i és nesfasta ja que les funcions h_i no només no són independents entre elles, sinó que el resultat de qualsevol d'elles determina de forma única el resultat de la resta. Això fa que l'ús de diverses funcions h_i sigui contraproduent i augmenta els errors.

Exercici 5.10: Podem crear un filtre f_3 de la mateixa mida que f_1 i f_2 que contingui la unió dels elements que hi ha a f_1 i f_2 fent una OR lògica de cadascuna de les posicions dels dos filtres f_1 i f_2. Així, a la posició i del filtre f_3 hi posaríem el resultat d'una OR entre el valor de la posició i del filtre f_1 i el valor de la posició i del filtre f_2, per a $i = 1 \ldots n$.

Així, tot element que es troba a f_1 o a f_2 es trobaria també al filtre f_3, ja que les posicions que aquest element ha fixat a 1 seguirien sent 1 al nou filtre. No obtant això, la probabilitat de fals positiu seria superior a f_3 (ja que hi hauria més elements per a un filtre amb la mateixa mida i mateix nombre de funcions hash).

Exercici 5.11:

Per a indexar les $n = 65536$ posicions es necessiten 16 bits ($2^{16} = 65536$). Com que $k = 10$, la sortida de la funció hash haurà de tenir $10 \cdot 16 = 160$ bits com a mínim. Això descarta l'ús d'MD5, que té una sortida de 128 bits. Per tant, pel que fa a la mida de la sortida, tant SHA1 (160 bits) com SHA256 (256 bits) serien bones candidates.

Tot i que els resultats específics de velocitat de càlcul de SHA1 i SHA256 depenen del maquinari que es faci servir, en general SHA1 és més ràpid. Per tant, si volem prioritzar la velocitat d'afegir i consultar elements, preferirem utilitzar SHA1.

L'eina openssl permet executar testos de rendiment de les primitives criptogràfiques que implementa. Per comparar les tres funcions hash, podem executar:

```
openssl speed md5 sha1 sha256
```

El resultat d'executar la instrucció anterior en un Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz és:

```
The numbers are in 1000s of bytes per second processed.
type     16 bytes     64 bytes     256 bytes    1024 bytes   8192 bytes
md5      112326.65k   262123.46k   462351.10k   571945.30k   613750.10k
sha1     123068.22k   284170.87k   593228.37k   787015.58k   865129.81k
sha256   70635.03k    158946.75k   292461.14k   359245.23k   389903.70k
```

En aquest cas, SHA1 és prop del doble de ràpid que SHA256 a l'hora de calcular el hash.

Exercici 5.12:

Calculem h_1, h_2 i h_3 per a cada element a afegir al filtre:

$$\begin{aligned}
SHA1(\texttt{uoc.edu}) &= \texttt{0xe6a62a58a28f94d745d3ea9a47163c846a065a3c} \\
h_1(\texttt{uoc.edu}) &= SHA1(\texttt{uoc.edu})_{0..3} = \texttt{0xe} = 15 \\
h_2(\texttt{uoc.edu}) &= SHA1(\texttt{uoc.edu})_{4..7} = \texttt{0x6} = 6 \\
h_3(\texttt{uoc.edu}) &= SHA1(\texttt{uoc.edu})_{8..11} = \texttt{0xa} = 10
\end{aligned}$$

$$\begin{aligned}
SHA1(\texttt{cv.uoc.edu}) &= \texttt{0x061053c6a11e8cd254a35edca6d8ab0a29765bb2b} \\
h_1(\texttt{cv.uoc.edu}) &= SHA1(\texttt{cv.uoc.edu})_{0..3} = \texttt{0x0} = 0 \\
h_2(\texttt{cv.uoc.edu}) &= SHA1(\texttt{cv.uoc.edu})_{4..7} = \texttt{0x6} = 6 \\
h_3(\texttt{cv.uoc.edu}) &= SHA1(\texttt{cv.uoc.edu})_{8..11} = \texttt{0x1} = 1
\end{aligned}$$

$$\begin{aligned}
SHA1(\texttt{biblioteca.uoc.edu}) &= \texttt{0x37a0042194841562352317 9cc7d97877302d98d0} \\
h_1(\texttt{biblioteca.uoc.edu}) &= SHA1(\texttt{biblioteca.uoc.edu})_{0..3} = \texttt{0x3} = 3 \\
h_2(\texttt{biblioteca.uoc.edu}) &= SHA1(\texttt{biblioteca.uoc.edu})_{4..7} = \texttt{0x7} = 7 \\
h_3(\texttt{biblioteca.uoc.edu}) &= SHA1(\texttt{biblioteca.uoc.edu})_{8..11} = \texttt{0xa} = 10
\end{aligned}$$

Per tant, el contingut del filtre una vegada s'han afegit els elements `uoc.edu`, `cv.uoc.edu` i `biblioteca.uoc.edu` és:

1	1	0	1	0	0	2	1	0	0	2	0	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

El contingut del filtre després d'eliminar l'element `uoc.edu` és:

1	1	0	1	0	0	1	1	0	0	1	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

5.8 Bibliografia

Christof Paar, Jan Pelzl (2010). *Understanding Cryptography. Capítol 11*. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-642-04101-3

R. Rivest (1992). "The MD4 Message-Digest Algorithm" *Request for Comments (núm: 1320) RFC 1320*. Internet Engineering Task Force.

R. Rivest (2011). "MD4 to Historic Status" *Request for Comments (núm: 6150) RFC 6150*. Internet Engineering Task Force.

R. Rivest (1992). "The MD5 Message-Digest Algorithm" *Request for Comments (núm: 1321) RFC 1321*. Internet Engineering Task Force.

A. Lenstra, X. Wang, B. Weger (2005). *Colliding X.509 Certificates*. Cryptology ePrint Archive Report 2005/067.

H. Dobbertin, A. Bosselaers, B. Preneel (1996). *RIPEMD-160: A Strengthened Version of RIPEMD*. Proceedings of FSE, LNCS 1039, pp. 71-82, Springer-Verlag

V. Rijmen, P. Barreto (2000). *The WHIRLPOOL Hash Function*.

National Institute of Standard and Technology (NIST) (2015). "Secure Hash Standard (SHS)". *Federal information processing standards (num. 180-4) FIPS 180-4*. Washington: NIST. DOI: 10.6028/NIST.FIPS.180-4

National Institute of Standard and Technology (NIST) (2015). "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions". *Federal information processing standards (num. 202) FIPS 202*. Washington: NIST. DOI: 10.6028/NIST.FIPS.202

R. Rivest (1992). "PKCS #5: Password-Based Cryptography Specification Version 2.0" *Request for Comments (núm: 2898) RFC 2898*. Internet Engineering Task Force.

A. Narayanan et al. (2016). "Bitcoin and cryptocurrency technologies: a comprehensive introduction". *Princeton University Press*

D. Boneh, V. Shoup (2015). "A graduate course in applied cryptography".

E. Barker, L. Feldman, G. Witte (2015). "Recommendation for random number generation using deterministic random bit generators". *National Institute of Standards and Technology*

III Criptografia de clau pública

6. Criptografia de clau pública

En aquest capítol ens introduirem en el món de la criptografia de clau pública. En primer lloc, descriurem el concepte de criptografia de clau pública o asimètrica, així com les característiques més destacades d'aquest tipus d'algorismes. Seguidament, veurem dos dels algorismes de xifrat de clau pública més utilitzats avui en dia: l'RSA i l'ElGamal.

Seguidament presentarem el concepte de signatura digital i descriurem tres dels algorismes més populars de signatura digital: l'esquema de signatura RSA, el d'ElGamal, i el DSA.

A continuació, compararem els algorismes de clau pública presentats amb els algorismes de criptografia simètrica que havíem vist en capítols anteriors, destacant-ne les seves fortaleses i debilitats.

Després, descriurem alguns detalls a tenir en compte a l'hora d'implementar els algorismes de clau pública descrits.

Finalment, presentarem una pinzellada d'altres famílies de criptografia de clau pública que, a diferència de les presentades amb detall en aquest capítol, no estan basades en els problemes de factorització d'enters i càlcul del logaritme discret.

6.1 L'origen de la criptografia de clau pública

La criptografia de clau simètrica es caracteritza per fer servir una mateixa clau tant per xifrar com per a desxifrar. És a dir, en criptografia simètrica, tant l'emissor d'un missatge (que el xifrarà abans d'enviar-lo), com el receptor (que l'haurà de desxifrar per poder-lo interpretar), comparteixen una única clau.

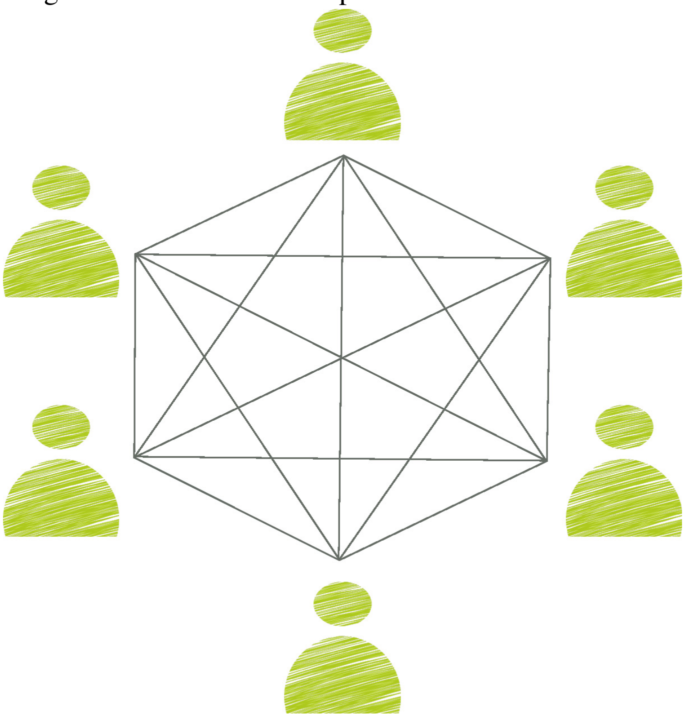
La criptografia de clau simètrica presenta algunes limitacions:

1. La **distribució de claus** s'ha de realitzar sobre un canal segur. Com que els dos usuaris comparteixen la mateixa clau i aquesta clau pot ser utilitzada directament per xifrar i desxifrar missatges, la clau no pot ser transmesa per un canal insegur, ja que aleshores un atacant que estigués escoltant el canal podria capturar-la i utilitzar-la. Per tant, per aconseguir que dos usuaris, l'Alice i en Bob, aconsegueixin tenir una clau simètrica amb què comunicar-se, caldrà que aquests dos usuaris hagin tingut un canal segur amb què transmetre-la amb anterioritat. Per exemple, caldrà que l'Alice i en

Bob s'hagin trobat presencialment o bé que ja disposin d'un canal segur per on transmetre la clau.

2. La **gestió de claus** es complica quan el número d'usuaris creix. Si hi ha n usuaris i cada parell d'usuaris necessita compartir una clau, caldrà gestionar $n(n-1)/2$ claus, és a dir, el número de claus d'un sistema creix quadràticament amb el número d'usuaris d'aquest. Així doncs, apareixen problemes d'escalabilitat a l'hora de gestionar totes aquestes claus. La Figura 6.1 mostra un exemple de 6 usuaris que comparteixen claus 2 a 2, necessitant per tant l'existència de 15 claus.

Figura 6.1: Sis usuaris comparteixen 15 claus diferents.



3. No es disposa de la propietat de **no-repudi**. Diem que un criptosistema ofereix la propietat de no-repudi si l'autor d'un missatge no en podrà negar posteriorment l'autoria. En criptografia simètrica, com que més d'un usuari comparteixen una mateixa clau, no es pot garantir que un usuari en concret ha realitzat una acció donada, ja que sempre hi haurà algun altre usuari que podria haver-la realitzat.

La criptografia de clau pública o asimètrica permet superar aquestes limitacions. D'una banda, la criptografia de clau pública proporciona mètodes d'establiment de clau, és a dir, mètodes per aconseguir que dos usuaris que es comuniquen per un canal insegur puguin crear claus que els permetin comunicar-se de manera segura. D'altra banda, la criptografia de clau pública permet que un conjunt d'usuaris es comuniquin dos a dos de manera segura fent servir únicament un parell de claus per cada usuari. Així, el nombre de claus d'un sistema de clau pública creix linealment amb el número d'usuaris d'aquest sistema (en contraposició al creixement quadràtic que presenten els esquemes de criptografia simètrica). Finalment, a través de les signatures digitals, la criptografia de clau pública ens ofereix la propietat de no-repudi.

L'origen de la criptografia de clau pública és una mica discutit. L'article *New directions in cryptography* (1979), de Whitfield Diffie i Martin Hellman, va donar a conèixer la criptografia de clau pública a la comunitat científica i va suposar l'inici d'un canvi de paradigma en la seguretat de la informació. Posteriorment però, amb la desclassificació de documents confidencials del govern Britànic l'any 1997, es va saber que James Ellis, Clifford Cocks i Graham Williamson havien descobert el principi de clau pública uns anys abans que Diffie i Hellman, però que el descobriment no s'havia fet públic ja que era confidencial. Tot i això, es creu que els criptògrafs del govern britànic no eren conscients en aquell moment del potencial del que havien descobert.

6.2 Intercanvi de claus de Diffie-Hellman

L'algorisme d'intercanvi de claus de Diffie-Hellman (DHKE, de l'anglès, *Diffie-Hellman Key Exchange*) és un algorisme d'intercanvi de claus basat en el problema del logaritme discret. L'algorisme va ser proposat per Whitfield Diffie i Martin Hellman al 1976 i està inspirat en la feina de Ralph Merkle. Va ser el primer algorisme de criptografia asimètrica.

L'ús del DHKE

L'algorisme d'intercanvi de claus de Diffie-Hellman es fa servir actualment a Internet en els protocols TLS, SSH o IPSec.

L'algorisme d'intercanvi de claus de Diffie-Hellman permet que dos usuaris que es comuniquen per un canal insegur puguin aconseguir derivar una clau compartida de manera segura. D'aquesta manera, encara que un atacant estigui escoltant el canal, l'atacant no pot aconseguir conèixer la clau derivada pels usuaris. Tot i així, l'esquema no és segur davant d'atacants que puguin modificar la informació que viatja pel canal.

L'algorisme requereix d'un procés d'inicialització on es trien dos valors, p i α, que es fan públics:

L'algorisme d'intercanvi de claus de Diffie-Hellman entre dos usuaris, A i B, consta dels següents passos:

1. Es tria un nombre primer aleatori p i un enter $\alpha \in [2,\ldots,p-2]$ primitiu. Es fan públics els valors p i α.
2. A tria un valor aleatori $a = k_{privA} \in [2,\ldots,p-2]$ i calcula $k_{pubA} = \alpha^a \mod p$.
3. B tria un valor aleatori $b = k_{privB} \in [2,\ldots,p-2]$ i calcula $k_{pubB} = \alpha^b \mod p$.
4. A i B intercanvien els seus valors k_{pub}, és a dir, A envia a B el valor k_{pubA} i B envia a A el valor k_{pubB}.
5. A deriva la clau compartida $k_{AB} = k_{pubB}^a \mod p$.
6. B deriva la clau compartida $k_{AB} = k_{pubA}^b \mod p$.

Així doncs, efectivament, el valor k_{AB} derivat per les dues parts participants en el protocol és el mateix, ja que, d'una banda, l'usuari A calcula

$$k_{AB} = k_{PubB}^a \mod p = (\alpha^b)^a \mod p$$

i, d'altra banda, l'usuari B calcula

$$k_{AB} = k_{PubA}^b \mod p = (\alpha^a)^b \mod p = (\alpha^b)^a \mod p$$

L'esquema descrit fa servir un element α primitiu a $\mathbb{Z}_p^*$ on p és un primer. Aquesta descripció correspon a la implementació original de l'algorisme. Tot i això, l'algorisme pot ser generalitzat per fer servir qualsevol grup cíclic finit G d'ordre n i un element α generador a G. En aquest cas, direm que el protocol es basa en el Problema de Diffie-Hellman generalitzat.

Logaritme discret en els enters mòdul p

Al 2005 es va aconseguir calcular el logaritme discret mòdul un primer fort de 431 bits. Al 2007 es va anunciar el càlcul d'un logaritme discret mòdul un primer segur de 530 bits. Al 2014 es va aconseguir per un primer segur de 596 bits i al 2016 d'un de 768 bits.

Pel que fa a la seguretat davant d'un atacant que escolti el canal, l'atacant coneixerà els dos valors intercanviats pel canal, k_{PubA} i k_{PubB}, a més dels paràmetres públics p i α, però calcular k_{AB} a partir d'aquests valors és un procés computacionalment difícil.

Exemple 6.1 **Exemple d'intercanvi de claus de Diffie-Hellman**

Els usuaris A i B disposen d'un canal insegur amb el qual comunicar-se, i volen aconseguir crear una clau compartida k_{AB}:

1. Se seleccionen els valors públics $p = 508107251$ i $\alpha = 5203822$.
2. A tria el valor aleatori $a = 385641303$ i calcula:
   $k_{pubA} = \alpha^a \mod p = 5203822^{385641303} \mod 508107251 = 421539355$.
3. B tria un valor aleatori $b = 467804164$ i calcula:
   $k_{pubB} = \alpha^b \mod p = 5203822^{467804164} \mod 508107251 = 230346411$.
4. A i B intercanvien els seus valors k_{pub}.
5. A deriva la clau compartida, calculant:
   $k_{AB} = (k_{pubB})^a \mod p = 230346411^{385641303} \mod 508107251 = 45453571$.
6. B deriva la clau compartida, calculant:
   $k_{AB} = (k_{pubA})^b \mod p = 421539355^{467804164} \mod 508107251 = 45453571$.

El protocol finalitza amb A i B compartint el mateix valor secret k_{AB}.

Exemple 6.2 **Exemple de MiM en l'intercanvi de claus de Diffie-Hellman**

Un dels problemes del protocol d'intercanvi de claus de Diffie-Hellman és que no és segur davant d'atacants que puguin situar-se al mig de la comunicació entre els dos usuaris i puguin modificar la informació que viatja entre ells. Suposem que els dos usuaris A i B de l'exemple anterior es comuniquen per un canal insegur, on l'usuari M pot interceptar les seves comunicacions i modificar els missatges que viatgen a través del canal. De nou, A i B intenten establir una clau compartida k_{AB}.

1. Els passos 1-3 es realitzen exactament igual que a l'Exemple 6.1. Per simplicitat, farem servir els mateixos valors, de manera que al finalitzar el pas 3, A ha calculat $k_{pubA} = 421539355$ i B ha calculat $k_{pubB} = 230346411$ ($p = 508107251$ i $\alpha = 5203822$).
2. A envia k_{pubA} a través del canal que el comunica amb B. Alhora, B envia k_{pubB} a través del mateix canal.
3. M intercepta els missatges k_{pubA} i k_{pubB}, i procedeix a:
   (a) Triar un valor aleatori $ma = 361369039$ i calcular:
   $k_{pubMA} = \alpha^{ma} \mod p = 5203822^{361369039} \mod 508107251 = 176105595$.
   (b) Triar un valor alearoti $mb = 504619741$ i calcular:
   $k_{pubMB} = \alpha^{mb} \mod p = 5203822^{504619741} \mod 508107251 = 342944530$.
   (c) Enviar el valor k_{pubMA} a A i el valor k_{pubMB} a B. Noteu que, a partir d'aquest moment, A espera rebre k_{pubB} però rebrà k_{pubMA} i B espera rebre k_{pubA} però rebrà k_{pubMB}.
4. A deriva la clau compartida, calculant:
   $k'_{AB} = (k_{pubMA})^a \mod p = 176105595^{385641303} \mod 508107251 = 36322887$.
   A creu que el valor k'_{AB} és una clau compartida amb B, però en realitat correspon a una clau compartida amb M.
5. B deriva la clau compartida, calculant:
   $k''_{AB} = (k_{pubMB})^b \mod p = 342944530^{467804164} \mod 508107251 = 461525945$.
   B creu que el valor k''_{AB} és una clau compartida amb A, però en realitat correspon a una clau compartida amb M. Noteu que els valors k'_{AB} i k''_{AB} difereixen.
6. M deriva les claus compartides amb A i B, calculant:
   $k_{MA} = (k_{pubA})^{ma} \mod p = 421539355^{361369039} \mod 508107251 = 36322887$
   $k_{MB} = (k_{pubB})^{mb} \mod p = 230346411^{504619741} \mod 508107251 = 461525945$.

   A partir d'aquest moment, M comparteix una clau amb A i una amb B, i és capaç de llegir i modificar tots els missatges que s'intercanvien pel canal.
   Noteu que aquest atac és possible ja que no hi ha autenticació de les parts que participen en el protocol.

6.3 Xifres de clau pública

Habitualment els algorismes de xifrat de clau pública comprenen tres funcions bàsiques: la generació de claus, el xifrat i el desxifrat. L'algorisme de generació de claus retorna un parell de claus de criptografia asimètrica, $[k_{pub}, k_{priv}]$; l'algorisme de xifrat rep un missatge m i una clau pública k_{pub} i genera el missatge xifrat c; i l'algorisme de desxifrat rep un missatge xifrat c i una clau privada k_{priv} i permet recuperar el missatge original m.

Les funcions bàsiques d'un esquema de xifrat amb clau pública són:
$[k_{pub}, k_{priv}]$ = `generació_claus()`
c = `E(` k_{pub}, m `)`
m = `D(` c, k_{priv} `)`

6.3.1 Xifratge basat en la factorització d'enters: RSA

L'RSA és un criptosistema de clau pública basat en el problema de la factorització d'enters. Va ser el primer criptosistema de clau pública proposat: presentat el 1977, només un any més tard que el concepte de criptografia de clau pública es fes públic. RSA són les sigles formades a partir de les inicials dels cognoms dels seus creadors, Ron Rivest, Adi Shamir i Leonard Adleman. Avui en dia, l'RSA és encara un dels criptosistemes de clau pública més utilitzats, tot i que els criptosistemes basats en corbes el·líptiques cada vegada van guanyant més terreny.

L'RSA es fa servir, principalment, en dos contextos: per xifrar dades de poca mida (normalment, per xifrar claus criptogràfiques) i en signatures digitals. En l'apartat 6.5 veurem una de les construccions més utilitzades per transmetre grans quantitats de dades fent servir l'RSA combinat amb un criptosistema de clau simètrica, obtenint així els beneficis dels dos criptosistemes.

L'algorisme de generació de claus de l'RSA consta dels següents passos:

1. Es trien dos primers aleatoris p i q.
2. Es calcula $n = p \cdot q$.
3. Es calcula $\phi(n) = (p-1)(q-1)$.
4. Se selecciona un exponent públic $e \in [1, \phi(n))$ tal que $gcd(e, \phi(n)) = 1$.
5. Es calcula l'exponent privat d tal que $d \cdot e = 1 \mod \phi(n)$. És a dir, es calcula $d = e^{-1} \mod \phi(n)$.
6. La clau pública k_{pub} és el parell (n, e), mentre que la clau privada k_{priv} és el valor (d). Els valors $\phi(n)$, p i q també són valors secrets que només coneix el propietari de la clau privada.

La funció $\phi(n)$ La funció totient d'Euler $\phi(n)$ (descrita al capítol de fonaments matemàtics) compta el número d'enters positius menors a n que són coprimers amb n.

La mida d'n Les recomanacions a data de Febrer del 2016 del NIST són de fer servir claus d'almenys 2048 bits i augmentar la mida a 3072 bits per algunes claus d'ús més crític.

Noteu que el valor d sempre existirà, ja que amb la condició $gcd(e, \phi(n)) = 1$ assegurem que e té invers a $\mathbb{Z}_n$.

Quan parlem de la longitud de la clau RSA, parlem de la mida del mòdul n (normalment expressada en bits).

Exemple 6.3 **Exemple de generació de claus RSA** Procedim a generar una clau RSA de 32 bits seguint els passos detallats a l'algorisme de generació de claus:

Seleccionem dos primers, per exemple, $p = 5879$ i $q = 484487$.
2. Calculem n:
$n = pq = 5879 \cdot 484487 = 2848299073$.
Podem comprovar com, efectivament, la clau generada és de 32 bits, ja que:
$2^{31} < 2848299073 < 2^{32}$.
3. Calculem $\phi(n)$
$\phi(n) = (p-1)(q-1) = (5879-1)(484487-1) = 2847808708$.
4. Seleccionem $e = 1535231195$.
Comprovem que, efectivament:
$gcd(e, \phi(n)) = gcd(1535231195, 2847808708) = 1$.
5. Calculem d:
$d = e^{-1} \mod \phi(n) = 1437751395$.
Podem comprovar que, efectivament:
$de = 1535231195 \cdot 1437751395 = 1 \mod 2847808708$.
6. Obtenim:
$k_{pub} = (n, e) = (2848299073, 1535231195)$
$k_{priv} = (d) = (1437751395)$.

Exercici 6.1 Indiqueu quins dels següents parells de claus RSA, $k_{pub} = (n, e)$ i $k_{priv} = (d)$ són vàlids. En cas que no ho siguin, detalleu-ne el motiu.

1. $k_{pub} = (3353361769, 1647529266), k_{priv} = (1853372443)$
2. $k_{pub} = (2660610913, 700422517), k_{priv} = (339543773)$
3. $k_{pub} = (111086984740301, 1890731431), k_{priv} = (66185553158551)$

Factorització d'enters

L'empresa RSA Laboratories va esponsoritzar durant uns anys una sèrie de reptes de factorització de mòduls RSA. Dins d'aquests reptes, un mòdul de 512 bits va ser factoritzat amb èxit al 1999, un de 704 bits al 2012, i un de 729 al maig de 2016.

Per tal de xifrar un missatge amb RSA, s'aplicarà l'algorisme de xifrat, que fa servir la clau pública del destinatari:

A partir d'un missatge en clar m i la clau pública del destinatari $k_{pub} = (n, e)$, es calcula el missatge xifrat c:

$$c = m^e \mod n$$

Quan el destinatari rebi el missatge xifrat c, podrà desxifrar-lo fent servir la seva clau privada (que només ell coneix):

A partir d'un missatge en xifrat c i la clau privada del destinatari $k_{priv} = (d)$, es recupera el text en clar m:

$$m = c^d \mod n$$

Noteu que, per tal de poder desxifrar correctament un missatge, caldrà que el missatge en clar original m sigui menor que el mòdul n.

Podem veure com, efectivament, desxifrar un missatge que ha estat prèviament xifrat resulta en l'obtenció del missatge original m:

$$c^d \mod n = (m^e)^d \mod n = m^{de} \mod n = m$$

Per a realitzar l'últim pas, recordem, d'una banda, que seguint l'algorisme de generació de claus assegurem que:

$$d \cdot e = 1 \mod \phi(n)$$

D'altra banda, el Teorema d'Euler estableix que si x i n són coprimers, aleshores:

$$x^{\phi(n)} = 1 \mod n$$

I, per tant:

$$x^{de} \mod n = x^{1+t\phi(n)} \mod n = x^1 \cdot x^{t\phi(n)} \mod n = x \cdot x^{\phi(n)^t} \mod n = x \cdot 1 = x \mod n$$

Exemple 6.4 Exemple de xifrat i desxifrat amb RSA

L'usuari Bob és el propietari del parell de claus RSA generats en l'Exemple 6.3. Si l'Alice vol enviar un missatge xifrat $m = 424242$ a en Bob, procedirà de la següent manera:

$$\begin{aligned} c &= m^e \mod n \\ &= 424242^{1535231195} \mod 2848299073 \\ &= 1914597261 \end{aligned}$$

En Bob, per desxifrar c i obtenir el missatge en clar original, procedirà a calcular:

$$\begin{aligned} m &= c^d \mod n \\ &= 1914597261^{1437751395} \mod 2848299073 \\ &= 424242 \end{aligned}$$

Exercici 6.2 L'Alice i en Bob són dos usuaris d'un sistema de clau pública RSA. Les seves respectives claus públiques i privades són:
$k_{pubA} = (3714176377, 1471178161), k_{privA} = (696390481)$

$k_{pubB} = (3720779831, 2037827401), k_{privB} = (2233915321)$
L'Alice vol enviar el missatge $m = 249$ xifrat a en Bob. Reproduïu el procés de xifrat realitzat per l'Alice i el procés de desxifrat que realitzarà en Bob al rebre el missatge, comprovant que efectivament el missatge rebut per en Bob coincideix amb el text en clar enviat per l'Alice.

Exercici 6.3 L'Alice i en Bob s'intercanvien missatges xifrats amb RSA fent servir la convenció de xifrar cada caràcter del missatge per separat, fent servir la codificació ASCII i utilitzant únicament lletres majúscules. Un atacant intercepta aquest missatge de l'Alice dirigit a en Bob:
$[2269693817, 1639486112, 1818812718, 2032163849, 3308958529, 251951562, 2224890518, 1639486112, 3489265165, 2032163849, 228316393, 1818812718]$
L'atacant coneix la clau pública d'en Bob (que l'Alice ha fet servir per xifrar el missatge), així com les convencions que fan servir l'Alice i en Bob en els intercanvis de missatges: $k_{pubB} = (3720779831, 2037827401)$
Quin és el missatge que l'Alice ha enviat a en Bob?
Nota: Trobeu el missatge sense calcular la clau privada d'en Bob, procés que no podríeu realitzar si les claus utilitzades en l'exercici fossin de mida real.

6.3.2 Xifratge basat en el logaritme discret: ElGamal

ElGamal és un criptosistema de clau pública basat en el problema del logaritme discret. En concret, ElGamal està basat en l'algorisme d'intercanvi de claus de Diffie-Hellman que s'ha presentat anteriorment a la Secció 6.2. El criptosistema deu el seu nom al seu creador, Taher ElGamal, que el va descriure el 1985.

L'algorisme de generació de claus del ElGamal consta dels següents passos:

1. Es tria un primer p i un element α d'ordre q.
2. Es tria un valor aleatori $d = k_{priv} \in [2, \ldots, p-2]$ i calcula $\beta = \alpha^d \mod p$.
3. La clau pública és $k_{Pub} = (p, \alpha, \beta)$ i la clau privada és el valor $k_{priv} = d$.

L'element α No és necessari que α sigui un element primitiu de $\mathbb{Z}_p^*$, pot ser-ho d'un subgrup de $\mathbb{Z}_p^*$ d'ordre q.

Per tal de xifrar un missatge amb ElGamal, es procedirà a aplicar l'algorisme de xifrat.

A partir d'un missatge en clar m i la clau pública del destinatari $k_{pub} = (p, \alpha, \beta)$, es calcula el missatge xifrat c:

1. Es tria un nombre aleatori h i es calcula $c_1 = \alpha^h \mod p$.
2. Es recupera la clau pública del receptor i es calcula $c_2 = m \cdot \beta^h \mod p$.
3. S'envia el missatge xifrat (c_1, c_2).

Noteu que el xifratge amb el criptosistema ElGamal és probabilístic, ja que per a una mateixa clau pública i un mateix missatge en clar, es poden generar múltiples textos xifrats, triant diferents valors aleatoris h durant el procés de xifrat.

Fixeu-vos, també que el xifrat amb ElGamal és expansiu, ja que per un missatge de mida m es generen textos xifrats de mida $2m$.

Quan un receptor rep el parell de valors que conformen un missatge xifrat, procedirà a desxifrar-lo amb l'algorisme de desxifrat.

A partir d'un missatge xifrat (c_1, c_2) i la clau privada del destinatari $k_{priv} = d$, es recupera el text en clar m:

$$m = \frac{c_2}{c_1^d} \mod p$$

Desxifrar amb ElGamal

Per tal de calcular $\frac{c_2}{c_1^d} \mod p$, es pot calcular l'invers modular de c_1^d i multiplicar el resultat per c_2.

Exemple 6.5 Exemple de generació de claus ElGamal Un usuari vol generar un parell de claus ElGamal i procedeix a executar l'algorisme de generació de claus:

1. Es tria un primer $p = 3725468627$ i un element $\alpha = 150083912$.
2. Es tria un valor aleatori $d = 807878087$ i es calcula:
   $\beta = \alpha^d \mod p = 150083912^{807878087} \mod 3725468627 = 3398986020$.
3. La clau pública és $k_{pub} = (3725468627, 150083912, 3398986020)$ i la clau privada és el valor $k_{priv} = 807878087$.

Exercici 6.4 Indiqueu quins dels següents parells de claus ElGamal, $k_{pub} = (p, \alpha, \beta)$ i $k_{priv} = d$ són vàlids. Per als que no ho són, detalleu-ne el motiu.

a) $k_{Pub} = (1474315399, 79643891, 269853666), k_{priv} = 84990634$
b) $k_{Pub} = (3383730189, 2011758775, 2122190089), k_{priv} = 2878050547$
c) $k_{Pub} = (337681733, 14736556, 93610277), k_{priv} = 144823569$
d) $k_{Pub} = (98011540216022814571886828168594180107,$
$73706495652936837455240336262679206568,$
$30382876101164794220971335754154344479),$
$k_{priv} = 61488904351572748379732502783030097644$

Exemple 6.6 Exemple de xifrat i desxifrat amb ElGamal

L'usuari Bob és el propietari del parell de claus ElGamal generats en l'Exemple 6.5. Si l'Alice vol enviar un missatge xifrat $m = 424242$ a en Bob, procedirà de la següent manera:

1. Alice tria un nombre aleatori $h = 1052400195$ i calcula:
   $c_1 = \alpha^h \mod p = 150083912^{1052400195} \mod 3725468627 = 434020969$.
2. Alice calcula:
   $c_2 = m \cdot \beta^h \mod p = 424242 \cdot 3398986020^{1052400195} \mod 3725468627 = 2787237740$.
3. Alice envia el missatge xifrat:
   $(c_1, c_2) = (434020969, 2787237740)$.

En Bob, per desxifrar (c_1, c_2) i obtenir el missatge en clar original, procedirà a calcular:

$$
\begin{aligned}
m = \frac{c_2}{c_1^d} \mod p &= \frac{2787237740}{434020969^{807878087}} \mod 3725468627 \\
&= \frac{2787237740}{1565777894} \mod 3725468627 \\
&= 2787237740 \cdot 1602291419 \mod 3725468627 \\
&= 424242
\end{aligned}
$$

Exercici 6.5 L'Alice i en Bob són dos usuaris d'un sistema de clau pública ElGamal. Les seves respectives claus públiques i privades són:
$k_{pubA} = (3346900289, 2210916257, 849352893), k_{privA} = (713795492)$
$k_{pubB} = (3575204279, 1113291034, 792418784), k_{privB} = (2036555019)$
L'Alice vol enviar el missatge $m = 424242$ xifrat a en Bob. Reproduïu el procés de xifrat realitzat per l'Alice i el procés de desxifrat que realitzarà en Bob al rebre el missatge, comprovant que efectivament el missatge rebut per en Bob coincideix amb el text en clar enviat per l'Alice.

6.4 Signatures digitals

Més enllà d'oferir una solució al problema de la distribució de claus, la criptografia de clau pública ens permet realitzar signatures digitals.

Tradicionalment, una signatura analògica, realitzada en bolígraf sobre un paper, permet demostrar que una persona concreta l'ha generada. Així, per exemple, les signatures permeten donar validesa a contractes legals, autoritzar compres amb targetes sense pin o emetre xecs. D'una manera anàloga, les signatures digitals ens permeten demostrar que el propietari d'una determinada clau privada ha realitzat una signatura sobre un document, de manera que només el propietari és capaç de generar una signatura vàlida per aquell document i que qualsevol que conegui la clau pública associada n'és capaç de validar-la.

Generalment, els algorismes de signatures digitals comprenen tres funcions bàsiques: la generació de claus, la generació de signatures i la validació de signatures. L'algorisme de generació de claus retorna un parell de claus de criptografia asimètrica, $[k_{pub}, k_{priv}]$; l'algorisme de signatura rep un missatge m i una clau privada k_{priv} i genera una signatura digital s del missatge m; i l'algorisme de verificació rep una signatura s, un missatge m i una clau pública k_{pub} i valida la correctesa de la signatura.

Les funcions bàsiques d'un esquema de signatura digital són:
$[k_{pub}, k_{priv}]$ = `generació_claus()`
s = `signatura(` k_{priv}, m `)`
v = `validació(` s, m, k_{pub} `)`

Atès que només el propietari de la clau privada és capaç de generar signatures amb aquella clau, diem que les signatures digitals ofereixen la propietat de no-repudi.

La propietat de **no-repudi** s'aconsegueix quan un usuari que realitza una acció (per exemple, signar un contracte) no pot negar posteriorment que l'acció ha estat realitzada per ell.

Així doncs, com que el propietari d'una clau privada n'és l'únic coneixedor, no podrà negar que ha signat algun document, ja que és l'únic capaç de realitzar aquella signatura.

D'altra banda, cal remarcar que per validar una signatura digital només cal conèixer la clau pública i el missatge signat, és a dir, la validació d'una signatura digital pot ser duta a terme per qualsevol part que conegui la clau pública.

Finalment, també cal destacar que les signatures digitals ofereixen integritat sobre els documents signats. En efecte, si un atacant modifica el contingut del document signat, la signatura s'invalida, i el receptor pot detectar aquesta modificació.

6.4.1 Signatures basades en la factorització d'enters: RSA

L'esquema de signatura RSA està basat en l'algorisme de xifrat RSA que s'ha descrit a la Secció 6.3.1 De la mateixa manera, la seguretat de l'algorisme de signatura RSA recau en la dificultat de factoritzar productes de dos primer grans.

A partir d'un missatge en clar m i la clau privada de l'emissor $k_{priv} = (d)$, es calcula la signatura digital del missatge s:

$$s = m^d \mod n$$

Quan el destinatari rebi el missatge m i la seva signatura s, podrà verificar la signatura fent servir la clau pública de l'emissor (que és de domini públic):

A partir d'un missatge m, la seva signatura s, i la clau pública de l'emissor $k_{pub} = (n, e)$, es valida la signatura calculant m':

$$m' = s^e \mod n$$

i validant que $m' \stackrel{?}{=} m$. Si $m' = m$, aleshores la signatura digital és vàlida, mentre que en cas contrari la signatura digital és invàlida.

Efectivament, si la signatura s no s'ha modificat, aleshores:

$$m' = s^e \mod n = (m^d)^e = m^{de} = m \mod n$$

Exemple 6.7 Exemple de signatura i validació amb RSA

L'usuari Bob és el propietari del parell de claus RSA que s'han fet servir en l'Exemple 6.3:
$k_{pub} = (n, e) = (2848299073, 1535231195)$
$k_{priv} = (d) = (1437751395)$.

En Bob vol enviar a l'Alice el missatge $m = 424242$ signat. Per fer-ho, procedirà de la següent manera:

$$
\begin{aligned}
s &= m^d \mod n \\
&= 424242^{1437751395} \mod 2848299073 \\
&= 2060449075
\end{aligned}
$$

Quan el destinatari, en aquest cas l'Alice, rebi el missatge i la signatura, podrà validar la correctesa de la signatura a partir de la clau pública de Bob, calculant:

$$
\begin{aligned}
m' &= s^e \mod n \\
&= 2060449075^{1535231195} \mod 2848299073 \\
&= 424242
\end{aligned}
$$

I comprovant que el valor m' obtingut és igual al missatge m.

6.4.2 Signatures basades en el logaritme discret: ElGamal

L'algorisme de signatura d'ElGamal va ser proposat al 1985 i es basa en la dificultat de calcular el logaritme discret.

Exercici 6.6 Calculeu el logaritme discret de 12483 en base 36848 a $\mathbb{Z}_{42841}$, és a dir, trobeu el valor x tal que $36848^x = 12483 \mod 42841$. Podríeu fer aquest mateix càlcul per a valors de 1024 bits?

A diferència de l'RSA, on els esquemes de xifrat i signatura són molt similars, l'algorisme de signatura d'ElGamal presenta diferències notables amb l'algorisme de xifratge.

A partir d'un missatge en clar m i la clau privada de l'emissor $k_{priv} = d$, es calcula la signatura digital del missatge s:

1. Es tria un valor aleatori $h \in [0, p-2]$ coprimer amb $p-1$ (és a dir, $gcd(h, p-1) = 1$).
2. Es calcula el valor $r = \alpha^h \mod p$.
3. Es troba el valor s tal que $m = dr + hs \mod p-1$. El valor s es pot trobar calculant: $s = (m - d \cdot r) \cdot h^{-1} \mod p-1$
4. La signatura correspon al parell de valors (r, s).

El càlcul del valor s

Noteu que sempre podem calcular $h^{-1} \mod p-1$ ja que al triar h hem assegurat que $gcd(h, p-1) = 1$.

Quan el destinatari rebi el missatge m i la seva signatura (r, s), podrà verificar la signatura fent servir la clau pública de l'emissor (que és de domini públic):

A partir d'un missatge m, la seva signatura s, i la clau pública del destinatari $k_{Pub} = (p, \alpha, \beta)$, es valida la signatura calculant:

$$t = \beta^r r^s \mod p$$

Es comprova si:

$$t == \alpha^m \mod p$$

Si la igualtat es compleix, la signatura és correcta. En cas contrari, la signatura és incorrecta.

Noteu com l'esquema de signatura de ElGamal, de la mateixa manera que amb l'esquema de xifratge, és probabilístic: per un mateix missatge i una mateixa clau privada, es poden generar múltiples signatures vàlides, variant el paràmetre h.

Noteu també que, a l'igual que en l'algorisme de xifrat, la mida de la signatura digital també és el doble que la del missatge.

Exemple 6.8 Exemple de signatura i validació amb ElGamal

L'usuari Bob és el propietari del parell de claus ElGamal que s'han fet servir en l'Exemple 6.5:
$k_{pub} = (p, \alpha, \beta) = (3725468627, 150083912, 3398986020)$
$k_{priv} = 807878087$.

En Bob vol enviar el missatge $m = 424242$ signat a l'Alice. Per fer-ho, procedirà de la següent manera:

1. Tria un valor aleatori $h = 249$ (comprovant que $gcd(249, 3725468627 - 1) = 1$).
2. Calcula el valor r:
   $r = \alpha^h \mod p = 150083912^{249} \mod 3725468627 = 1675101370$
3. Troba el valor s tal que $m = dr + hs \mod p - 1$.
   $$\begin{aligned} s &= (m - dr) \cdot h^{-1} \mod p - 1 = \\ &= (424242 - 807878087 \cdot 1675101370) \cdot 249^{-1} \mod 3725468626 = \\ &= 1431688902 \end{aligned}$$
4. La signatura correspon al parell de valors (r, s):
   $(r, s) = (1675101370, 1431688902)$

El càlcul d's	Noteu que el càlcul del valor s s'efectua mòdul $p - 1$ i no pas mòdul p.

Quan el destinatari, en aquest cas l'Alice, rebi el missatge i la signatura, podrà validar la correctesa de la signatura a partir de la clau pública de Bob, calculant:

$$\begin{aligned} t &= \beta^r r^s \mod p = \\ &= 3398986020^{1675101370} \cdot 1675101370^{1431688902} \mod 3725468627 = 1954079850 \end{aligned}$$

$$\alpha^m \mod p = 150083912^{424242} \mod 3725468627 = 1954079850$$

I comprovant com, efectivament, el resultat és igual al valor t, donant la signatura per vàlida.

Exercici 6.7 Genereu dues signatures diferents per al missatge $m = 45678$ fent servir l'esquema de signatura ElGamal i valideu, després, les signatures generades. Feu servir el parell de claus:
$k_{Pub} = (p, \alpha, \beta) = (797445667, 386331185, 505206688)$
$k_{priv} = (d) = (373845532)$

L'estàndard DSA

El DSA (per les seves sigles en anglès, *Digital Signature Algorithm*) és una variant molt popular de l'algorisme de signatura d'ElGamal que va ser proposada al 1991. Aquesta popularitat és deu, en part, a què des de 1994 és considerada un estàndard per a signatures digitals (DSS) del FIPS (de l'anglès, *Federal Information Processing Standards*), un conjunt d'estàndards públics que desenvolupa el govern federal dels Estats Units.

Els passos a seguir en l'algorisme de generació de claus DSA són els següents:

1. Es tria un primer p tal que $2^{L-1} < p < 2^L$.
2. Es busca un divisor q de $p-1$ tal que q sigui primer i $2^{N-1} < q < 2^N$.
3. Es busca un element g d'ordre q a $\mathbb{Z}_p$ $(1 < g < p)$, és a dir, g és un generador del subgrup de q elements.
4. Es tria un valor aleatori $x = k_{priv}$ tal que $0 < x < q$
5. Es calcula la clau pública $y = g^x \mod p$.
6. La clau pública és $k_{pub} = (p, q, g, y)$ i la clau privada és el valor $k_{priv} = (x)$.

L'estàndard especifica quatre possibles alternatives per a l'elecció de la mida (en bits) dels valors p i q (respectivament, els valors L i N).

L	N
1024	160
2048	224
2048	256
3072	256

Taula 6.1: Valors per a L i N detallats a l'estàndard.

A partir d'un missatge en clar m i la clau privada de l'emissor $k_{priv} = x$, es calcula la signatura digital del missatge s:

1. Es genera un valor aleatori k tal que $0 < k < q$.
2. Es calcula el valor $r = (g^k \mod p) \mod q$.
3. Es troba el valor s tal que $m = ks - xr \mod q$. El valor s es pot trobar calculant: $s = k^{-1}(m + x \cdot r) \mod q$
4. La signatura correspon al parell de valors (r, s).

Si durant el procés de generació de signatura es donés el cas que s o r fossin 0, aleshores es repeteix el procés triant un nou valor k. D'aquesta manera, s'assegura que la signatura generada mai tingui un 0 en cap de les dues parts que la formen.

A partir d'un missatge m, la seva signatura (r,s), i la clau pública de l'emissor $k_{pub} = (p,q,g,y)$, es valida la signatura calculant:

1. Es comprova que $s \neq 0$ i $r \neq 0$.
2. Es calcula:

$$
\begin{aligned}
w &= s^{-1} \mod q \\
u_1 &= mw \mod q \\
u_2 &= rw \mod q \\
v &= (g^{u_1} y^{u_2} \mod p) \mod q
\end{aligned}
$$

3. Es valida que $v == r \mod q$.

Si la igualtat es compleix, la signatura és correcta. En cas contrari, la signatura és incorrecta.

6.4.3 Atacs als esquemes de signatura digital

En el context de les signatures digitals, existeixen tres tipus d'atacs de falsificació:

Direm que un adversari realitza un atac de **falsificació existencial** quan aquest és capaç de crear almenys una signatura s corresponent a un missatge m.

En aquest tipus d'atacs, l'adversari no té cap control sobre el valor m, és a dir, m pot prendre qualsevol valor (el contingut del missatge no és important). L'atacant aconsegueix el seu objectiu només pel fet d'aconseguir una signatura vàlida s. Els atacs de falsificació existencial són els més senzills de realitzar.

Un atac de **falsificació selectiva** és un atac de falsificació en què l'adversari té com a objectiu crear una signatura vàlida per a un missatge m que ha triat el propi adversari amb anterioritat a l'inici de l'atac.

Així, si un atacant és capaç de dur a terme un atac de falsificació selectiva, això implica que és capaç també de dur a terme un atac de falsificació existencial.

Direm que un adversari pot dur a terme un atac de **falsificació universal** si és capaç de crear una signatura s vàlida per a qualsevol missatge donat m.

Un adversari capaç de realitzar falsificació universal pot, per tant, crear signatures vàlides per a missatges triats a l'atzar, seleccionats per ell mateix, seleccionats per una tercera part, etc.

Falsificació existencial de signatures RSA

L'algorisme de signatura RSA explicat anteriorment és susceptible a atacs de falsificació existencial de signatures.

En efecte, donada una clau pública $k_{pub} = (n,e)$, un atacant pot procedir de la següent manera:

1. L'atacant tria una signatura $s \in \mathbb{Z}_n$.
2. L'atacant calcula el missatge $m = s^e \mod n$.
3. L'atacant obté una signatura vàlida s per al missatge m.
4. La signatura és vàlida ja que $s^e = m \mod n$.

Noteu que, en aquest cas, l'atacant és capaç de construir signatures vàlides, però no té cap mena de control sobre els missatges que està signant.

Falsificació existencial de signatures ElGamal

L'algorisme de signatura d'ElGamal explicat anteriorment també és susceptible a atacs de falsificació existencial.

En efecte, donada una clau pública $k_{pub} = (p, \alpha, \beta)$, un atacant pot procedir de la següent manera:

1. Tria dos enters i i j tals que $gcd(j, p-1) = 1$.
2. Calcula la signatura:

$$r = \alpha^i \beta^j \mod p$$

$$s = -rj^{-1} \mod p-1$$

3. Calcula el missatge:

$$m = si \mod p-1$$

4. L'atacant obté una signatura vàlida (r, s) per al missatge m.
5. La signatura és vàlida ja que la igualtat $\beta^r r^s = \alpha^x$ és manté.

Vegem perquè, efectivament, la validació de la signatura és correcta:

$$\begin{aligned}
\beta^r r^s \mod p &= \alpha^{dr} r^s \mod p \\
&= \alpha^{dr} \alpha^{(i+dj)s} \mod p \\
&= \alpha^{dr} \alpha^{(i+dj)(-rj^{-1})} \mod p \\
&= \alpha^{dr-dr} \alpha^{(-rij^{-1})} \mod p \\
&= \alpha^{si} \mod p \\
&= \alpha^{x} \mod p
\end{aligned}$$

De la mateixa manera que amb els atacs de falsificació existencial de signatures RSA, en aquest cas l'atacat tampoc té cap control sobre els missatges signats.

Vulnerabilitat en la reutilització de valors ElGamal

L'algorisme de signatura de ElGamal fa servir un valor aleatori h a l'hora de signar els missatges. Aquest valor és el que permet que l'algorisme sigui probabilístic i que existeixin múltiples signatures vàlides per a un únic missatge i una clau determinada. L'aleatorietat en la selecció del paràmetre h és, però, crucial per a la seguretat del sistema. De fet, la seva reutilització permet a un atacant descobrir la clau privada feta servir per crear les signatures digitals.

En efecte, l'algorisme de signatura d'ElGamal (així com algunes de les seves variants) té una vulnerabilitat a través de la qual un atacant que obté dues signatures diferents, sig_1 i sig_2, realitzades amb la mateixa clau privada d i fent servir el mateix valor h, pot recuperar la clau privada feta servir per a signar:

$$sig_1 = (r, s_1)$$
$$sig_2 = (r, s_2)$$

$$s_1 \cdot h = (m_1 - d \cdot r) \mod p-1$$
$$s_2 \cdot h = (m_2 - d \cdot r) \mod p-1$$

$$(s_2 - s_1) \cdot h = m_2 - m_1 \mod p-1$$
$$h = \frac{m_2 - m_1}{s_2 - s_1} \mod p-1$$

$$d = (m_1 - hs_1) \cdot r^{-1} \mod p-1$$

Noteu que el càlcul descrit només pot realitzar-se si $(s_2 - s_1)$ és invertible, és a dir, si $gcd(s_2 - s_1, p-1) = 1$. En cas contrari, es pot realitzar el càlcul descomposant $p-1$ i tractant els casos concrets individualment.

Equacions modulars amb elements no invertibles

Malgrat que per resoldre equacions modulars en ocasions ens aniria bé calcular inversos, també les podem resoldre en cas que no poguem calcular algun invers explícitament. Per exemple, podem tenir una equació modular del tipus $10x = 4 \mod 26$ que clarament no podem resoldre directament perquè el 10 no té invers mòdul 26, perquè $gcd(10,26) \neq 1$. Ara bé, podem calcular el $gcd(10,26) = 2$ i dividir tota l'equació, incloent el mòdul, per aquest valor. Si ho fem tindrem $5x = 2 \mod 13$. Fixeu-vos que en aquest cas, 5 sempre tindrà invers amb el nou mòdul (aquest cas 13) perquè sempre serà coprimer amb aquest valor, justament perquè el nou mòdul és el mòdul anterior al qual li hem tret el propi factor 2. Si resolem l'equació $5x = 2 \mod 13$ obtindrem $x = 3 \mod 13$. Si us hi fixeu, el valor $x = 3$ és solució de la primera equació $10x = 4 \mod 26$, però a més també tenim una altra solució, que serà $x = 3 + 13 = 16$. De fet, tindrem tantes solucions com el valor del gcd(10,26), en aquest cas, aquestes dues indicades. Ara bé, podem tenir equacions de primer grau en mòduls no primers que no tinguin solució. Per exemple, si volem resoldre l'equació $10x = 5 \mod 26$, clarament aquesta equació no té solució, perquè és equivalent a $2x = 1 \mod 26$ i això és calcular l'invers de 2 mòdul 26 que ja sabem que no existeix.

Per tant, és important que el valor h sigui únic per a cada signatura.

Aconseguir una font d'aleatorietat prou bona com per garantir que el valor h serà únic per cada una de les signatures realitzades per un dispositiu pot ser problemàtic en segons quins entorns (per exemple, en telèfons mòbils). Per aquest motiu, existeix una variant del DSA determinista, on el valor h queda determinat de manera única per la clau privada i el missatge a signar. L'algorisme que genera el valor h actua de manera similar a un generador pseudoaleatori, fent servir la clau privada i el hash del missatge com a llavors del generador. Noteu que, amb aquesta versió del DSA, amb una mateixa clau només es podrà generar una única signatura per un missatge concret. Addicionalment, és important notar que la variant determinista del DSA no modifica l'algorisme de generació de claus ni la validació de signatures, de manera que és compatible amb sistemes que implementen el DSA probabilístic.

6.5 Criptografia simètrica i asimètrica

Més enllà de les propietats que ens ofereix la criptografia de clau pública, aquesta també difereix de la criptografia simètrica en la mida de les claus i les necessitats computacionals dels seus algorismes. Pel que fa a la mida de les claus, normalment es fa servir el concepte de nivell de seguretat per avaluar la fortalesa d'un algorisme criptogràfic amb una mida de claus concreta, i poder així comparar la seguretat que ofereixen els diferents algorismes:

El **nivell de seguretat** d'un algorisme criptogràfic és el número de passos que necessita el millor atac conegut per descobrir la clau. Direm que un algorisme proporciona una seguretat d'n bits quan el millor atac necessita 2^n passos.

És important notar que el nivell de seguretat d'un algorisme pot variar amb el descobriment de nous atacs que milloren l'eficiència dels ja coneguts.

Tenint en compte la definició de nivell de seguretat, és fàcil veure que un algorisme de criptografia simètrica amb mida de clau n bits ens proporciona una seguretat d'n bits. En canvi, calcular el nivell de seguretat d'un algorisme de clau pública no és tan directe. La Taula 6.2 ens descriu els nivells de seguretat que s'assumeixen avui en dia per als algorismes de clau simètrica i asimètrica més populars.

	Nivell de seguretat			
Algorisme	**80**	**128**	**192**	**256**
AES	80	128	192	256
RSA	1024	3072	7680	15360
ElGamal	1024	3072	7680	15360
DSA	1024	3072	7680	15360

Taula 6.2: Nivell de seguretat segons la mida de la clau.

Criptografia de corbes el·líptiques

Els criptosistemes de clau pública basats en corbes el·líptiques (que queden fora de l'abast d'aquest document) permeten obtenir el mateix nivell de seguretat que els algorismes de clau simètrica amb mides de clau superiors però molt més similars. Així, per exemple, caldrà una clau ECDSA de 160 bits per aconseguir 80 bits de seguretat.

Com es pot apreciar, d'una banda com més gran és la mida de la clau utilitzada major és la seguretat que ens ofereix. D'altra banda, per tal d'obtenir un mateix nivell de seguretat en un criptosistema de clau pública que en un de clau simètrica, caldrà que la clau del primer sigui molt més gran que la del segon. Això, unit al fet que els algorismes de clau pública requereixen càlculs computacionalment intensius, fa que en general els algorismes de clau pública siguin més lents que els de clau simètrica. Aquesta lentitud pot no ser problemàtica per als ordinadors actuals, però sí que pot ser-ho en dispositius amb capacitats més limitades com ara targetes intel·ligents. A continuació, es descriuen dues de les tècniques que s'utilitzen habitualment per accelerar el procés de xifrat i desxifrat amb RSA.

Per tal d'aprofitar els avantatges de la criptografia de clau pública pel que fa a la gestió de claus i a l'hora la rapidesa de la criptografia simètrica per xifrar, sovint es combinen els dos sistemes amb la tècnica coneguda com a sobre digital.

La tècnica del **sobre digital** consisteix a xifrar un missatge amb una clau simètrica aleatòria k i xifrar la clau simètrica k amb una clau pública.

Exercici 6.8 L'Alice i en Bob són dos usuaris d'un sistema de clau pública RSA. Les seves respectives claus públiques i privades són:
$k_{pubA} = (3714176377, 1471178161), k_{privA} = (696390481)$
$k_{pubB} = (3720779831, 2037827401), k_{privB} = (2233915321)$

L'Alice vol enviar el missatge:
$m = 011235813213455891442333776109871597258441816765$
xifrat a en Bob. Reproduïu el procés de xifrat realitzat per l'Alice.

A l'hora de realitzar signatures digitals, si volguéssim signar directament els missatges amb un sistema de clau pública ens trobaríem també amb la limitació de la mida de la clau, que reduiria en gran mesura el conjunt de missatges que seríem capaços de signar. En aquest cas, el que es fa és combinar les signatures digitals amb les funcions hash, descrites en el Capítol 5. És a dir, a l'hora de signar un missatge, l'usuari procedeix primer a calcular-ne un resum a través d'una funció hash, signant després aquest resum (en comptes del missatge original). Com que les funcions hash tenen una sortida de mida fixada, això ens permet assegurar que podem signar qualsevol missatge (de qualsevol mida) amb una clau d'una mida concreta (fent ús d'una funció hash amb una sortida de mida inferior a la clau).

El procediment a seguir per tal de signar un missatge m fent servir una funció hash H és:

1. Calcular el hash del missatge, $h = H(m)$.
2. Calcular la signatura del hash del missatge, s = `signatura(` k_{priv}, h `)`

El receptor del missatge m podrà validar la signatura s, procedint de la següent manera:

1. Calcular el hash del missatge, $h = H(m)$.
2. Valida la signatura s, v = `validació(` s, h, k_{pub} `)`

Noteu que en aquest cas emissor i receptor s'han de posar d'acord no només amb l'algorisme de signatura que utilitzaran sinó també amb la funció hash que faran servir.

6.6 Implementació dels algorismes de clau pública

Els algorismes de criptografia de clau pública descrits fins ara requereixen de l'ús d'operacions computacionalment costoses per funcionar. Per aquest motiu, a l'hora d'implementar aquests algorismes, es fan servir optimitzacions que permeten millorar-ne l'eficiència. En aquest capítol veurem algunes de les optimitzacions més populars que s'utilitzen a l'hora d'implementar l'RSA i l'ElGamal.

6.6.1 Optimització del xifrat RSA

L'algorisme de generació de claus RSA que hem vist a la Secció 6.3.1 tria un exponent públic e de manera aleatòria, amb les condicions que e estigui en l'interval $(1, phi(n))$ i que e sigui coprimer amb $phi(n)$. A la pràctica però, s'acostumen a triar valors d'e petits i amb pes de Hamming també petit, ja que això fa que el xifratge sigui més eficient.

Pes de Hamming

El **pes de Hamming** d'una seqüència binària és el número d'uns que conté. El pes de Hamming és equivalent a la distància de Hamming entre una seqüència donada i la seqüència de zeros de la mateixa longitud.

En concret, per xifrar (o per validar una signatura) amb RSA necessitem calcular $x^e \mod n$. Si fem servir l'algorisme d'exponenciació ràpida de multiplicar i elevar, caldran de l'ordre de $\log_2 e + pes(e)$ operacions per tal de realitzar el càlcul, amb $pes(e)$ representant el pes de Hamming de la representació binària d'e. Valors d'e petits minimitzen el terme $\log_2 e$ mentre que valors amb pes de Hamming petit minimitzen el terme $pes(e)$.

Així, alguns dels valors que es fan servir habitualment per a l'exponent públic e són 3 i 65,537 ($2^{16} + 1$), que requereixen 3 i 17 operacions per xifrar, respectivament. Aquest últim és el valor per omissió que fa servir la llibreria OpenSSL[1]. El valor 65,537 té alguns avantatges. En primer lloc és un primer de Fermat, el que fa que calguin poques multiplicacions per elevar a aquest valor i alhora simplifica la cerca dels primers p i q adequats. En segon lloc i a diferència de 3, $2^{16}+1$ és prou gran com per evitar certs atacs que es poden dur a terme amb valors d'e petits si no es fa servir *padding* de manera correcta.

Primers de Fermat

Els **primers de Fermat** són primers de la forma $F_n = 2^{(2^n)}+1$. Els únics primers de Fermats coneguts (a data de Febrer de 2016) són $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$ i $F_4 = 65{,}537$.

La selecció de l'exponent públic fent servir aquestes consideracions fa que xifrar amb RSA sigui molt més ràpid que desxifrar, i també redueix el temps de comprovació de les signatures RSA. Tot i així, aquest no és l'únic factor que afecta la velocitat de les implementacions d'RSA.

Tot i que l'algorisme d'exponenciació ràpida de multiplicar i elevar permet xifrar missatges amb RSA de manera molt ràpida, aquest no es fa servir per desxifrar (ni per signar) ja que és vulnerable a atacs de canal lateral.

Els atacs de **canal lateral** (en anglès, *side channel attacks*) són atacs que es basen en informació adquirida de la implementació física d'un criptosistema. Aquests tipus d'atacs poden fer servir, per exemple, informació sobre el temps d'execució, el consum energètic, els camps electromagnètics, el so, etc.

Un atacant que analitzi el consum energètic d'un dispositiu que implementa l'algorisme de multiplicar i elevar pot obtenir directament la clau. D'una banda, per cada bit de la clau, l'algorisme de multiplicar i elevar calcula una única potència si el bit és 0 o bé una potència i una multiplicació si el bit és 1. D'altra banda, el consum energètic d'un dispositiu implementant aquest algorisme és elevat mentre s'estan realitzant aquestes operacions (tant multiplicació com exponenciació), i baix quan no s'estan fent aquestes operacions. Així, veient la traça de consum energètic del dispositiu, es pot diferenciar clarament cadascuna de les iteracions de l'algorisme (moments de consum energètic elevat separats per instants de consum energètic baix). La durada dels moments de consum energètic elevat ens permet distingir quan s'estan processant bits de la clau a 1 o a 0. D'aquesta manera, només obtenint la traça de consum energètic d'una única operació, es pot obtenir la clau.

Existeixen altres algorismes d'exponenciació ràpida que ofereixen protecció contra atacs d'anàlisi del consum energètic, com ara l'algorisme d'exponenciació de Montgomery (en anglès, *Montgomery Powering Ladder*). Aquests algorismes fan que cada iteració, independentment de si aquesta tracta un bit de la clau fixat a 0 o a 1, tingui un còmput similar, de manera que el consum energètic de cada iteració és similar.

6.6.2 Optimització del desxifrat RSA

A més de triar valors d'e que permetin accelerar el xifratge i la validació de signatures digitals, algunes de les llibreries més populars emmagatzemen tres valors intermedis durant la generació de claus RSA, amb l'objectiu de reduir el temps de desxifrat i de signatura. Aquests valors són:

- exponent 1: $d_p = d \mod (p-1)$
- exponent 2: $d_q = d \mod (q-1)$
- coeficient: $q_{inv} = (1/q) \mod p$

Aquests valors permeten optimitzar el càlcul de $c^d \mod n$ a través del Teorema Xinès del Residu. Així, per

[1] `https://www.openssl.org/docs/manmaster/apps/genpkey.html`

a calcular $m = c^d \mod n$ es procedeix a calcular:

$$\begin{aligned} m_1 &= c^{d_p} \mod p \\ m_2 &= c^{d_q} \mod q \\ h &= q_{inv}(m_1 - m_2) \mod p \\ m &= m_2 + hq \end{aligned}$$

Exemple 6.9 Exemple de desxifrat RSA

Suposem que tenim la clau privada RSA de 64 bits formada pels valors:
$d = 7506774701030841737$
$n = 10639337161500789943$
i que en el moment de generar la clau hem desat també els valors:
$p = 696734729$
$q = 15270283967$
$d_p = d \mod (p-1) = 7506774701030841737 \mod (696734729-1) = 259426577$
$d_q = d \mod (q-1) = 7506774701030841737 \mod (15270283967-1) = 8567289973$
$q_{inv} = (1/q) \mod p = 1/15270283967 \mod 696734729 = 284277123$

Aleshores, per desxifrar el missatge $c = 3510853621447083634$, procediríem de la següent manera:
$m_1 = c^{d_p} \mod p = 3510853621447083634^{259426577} \mod 696734729 = 627709010$
$m_2 = c^{d_q} \mod q = 3510853621447083634^{8567289973} \mod 15270283967 = 11944757133$
$h = q_{inv}(m_1 - m_2) \mod p = 284277123(627709010 - 11944757133) \mod 696734729 = 27$
$m = m_2 + hq = 11944757133 + 27 \cdot 15270283967 = 424242424242$

Noteu que això és equivalent a fer directament:

$$\begin{aligned} m &= c^d \mod n = \\ &= 3510853621447083634^{7506774701030841737} \mod 10639337161500789943 = \\ &= 424242424242 \end{aligned}$$

però, en canvi, totes les operacions implicades tenen exponents i mòduls molt menors que els requerits per a aquest càlcul. D'aquesta manera, s'aconsegueix reduir el temps de càlcul.

Podem veure com s'emmagatzemen aquests valors en claus RSA de mida real fent servir l'eina OpenSSL:

```
openssl genrsa -out private.pem 1024
openssl rsa -noout -text -in private.pem
```

Hem vist com s'optimitza el xifratge i la validació de signatures a partir de la tria de l'exponent públic e i com s'optimitza el desxifrat i la realització de signatures emmagatzemant uns valors auxiliars relatius a la clau privada. Després de realitzar aquestes optimitzacions, ens podríem preguntar quin dels dos processos és doncs més ràpid. Per comprovar-ho a nivell pràctic podem recórrer de nou a la llibreria Openssl. En concret, podem analitzar les diferències de temps necessaris per signar i validar signatures digitals en RSA amb la sentència:

```
openssl speed rsa
```

Els resultats de l'execució de l'anterior sentència en un Intel Core i7-4770 CPU @ 3.40GH (Taula 6.3) indiquen clarament que és molt més ràpid validar signatures (i per tant xifrar) que no pas realitzar les signatures (o desxifrar).

Mida de la clau	Temps sign.	Temps val.	Sig. per segon	Val. per segon
512 bits	0.000041s	0.000003s	24317.5	328106.1
1024 bits	0.000118s	0.000008s	8485.8	131766.6
2048 bits	0.000548s	0.000024s	1825.2	41486.7
4096 bits	0.005865s	0.000088s	170.5	11299.6

Taula 6.3: Temps per signar / validar amb RSA.

6.6.3 Optimització del xifrat ElGamal

El procés de xifrat amb ElGamal consta de tres operacions: dues exponenciacions, que permeten calcular $c_1 = \alpha^v \mod p$ i un operand de c_2, $c_2^{aux} = \beta^v$; i una multiplicació que realitza el càlcul final de c_2, $c_2 = m \cdot c_2^{aux}$.

Com en el cas de l'RSA, les exponenciacions es poden calcular amb l'algorisme d'exponenciació ràpida de multiplicar i elevar, de manera que s'agilitza el càlcul. Ambdues exponenciacions fan servir com a exponent el paràmetre aleatori v, de manera que les exponenciacions poden optimitzar-se seleccionant valors v amb propietats especials, com ara valors amb pes de Hamming petit. Si es fa servir aquesta tècnica cal anar en compte, però, de tenir un número adequat d'exponents possibles.

Ara bé, en el cas del xifrat amb ElGamal, hi ha un altre detall important a tenir en compte: les dues exponenciacions a calcular són completament independents del missatge a xifrar. Això fa que en algunes aplicacions aquests valors puguin precalcular-se amb anterioritat al procés de xifrat, en moments quan la càrrega del sistema és baixa. A l'hora de xifrar, caldrà doncs recuperar els valors precalculats i calcular únicament la multiplicació, operació que sí que depèn del missatge a xifrar.

6.6.4 Optimització del desxifrat ElGamal

El procés de desxifrar amb el criptosistema ElGamal consta també de tres operacions: una exponenciació, que permet calcular c_1^d; una inversió, que calcula $(c_1^d)^{-1}$; i finalment una multiplicació, que recupera m calculant $c_2 \cdot (c_1^d)^{-1}$. El desxifrat es pot optimitzar unint les dues primeres operacions en una sola exponenciació, fent ús del Petit Teorema de Fermat (que és un cas concret del Teorema d'Euler). El Petit Teorema de Fermat afirma que, donat un p primer:

$$x^{p-1} = 1 \mod p, \forall x \text{ tal que } gcd(x,p) = 1$$

Aprofitant aquesta igualtat, podem reduir el càlcul de $(c_1^d)^{-1}$ a una única exponenciació: c_1^{p-d-1}, és a dir, el valor c_1 elevat a l'exponent $p-d-1$. Comprovem que realment les dues alternatives són equivalents:

$$(c_1^d)^{-1} \mod p = (c_1^d)^{-1} \cdot c_1^{p-1} \mod p = c_1^{p-d-1} \mod p$$

D'aquesta manera, desxifrar un missatge amb ElGamal se simplifica i passa a consistir en una exponenciació i una multiplicació.

6.7 Criptografia post-quàntica

Com hem vist, els algorismes de criptografia de clau pública més populars avui en dia es basen en la dificultat de resoldre tres problemes matemàtics: la factorització d'enters, el logaritme discret i el logaritme discret sobre corbes el·líptiques. Aquest problemes, però, deixarien de ser difícils si disposéssim d'un ordinador quàntic de prou capacitat.

L'algorisme de Shor és un algorisme quàntic que permet resoldre'ls en un temps polinomial respecte a la mida de l'entrada. L'algorisme de Shor va ser proposat per Peter Shor l'any 1994 a l'article *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*.

Hi ha tot un conjunt de famílies d'algorismes de criptografia de clau pública que es basen en altres problemes matemàtics, que es creuen difícils de resoldre tant en ordinadors clàssics com en ordinadors quàntics. A tots ells se'ls engloba sota el nom de criptografia post-quàntica. Exemples d'aquestes famílies en són la criptografia basada en hashos, en codis, en reticles o en equacions quadràtiques multivariades. Així, per exemple, el criptosistema de McEliece (desenvolupat per Robert McEliece el 1978) està basat en la dificultat de descodificar un codi lineal i l'esquema de signatura de Merkle (ideat per Ralph Merkle el 1979) es basa en la generació d'un arbre de hashos.

Atès que els algorismes de criptografia post-quàntica són segurs davant d'ordinadors quàntics, hom podria preguntar-se el motiu per el qual el seu ús no es troba molt més estès actualment que el dels algorismes que hem descrit en aquest mòdul, que no són segurs en aquesta situació. D'una banda, l'eficiència computacional dels algorismes de criptografia asimètrica post-quàntics és baixa, molt pitjor que la que ens ofereixen actualment l'RSA o ElGamal. D'altra banda, alguns d'aquests esquemes són molt nous, i la comunitat criptogràfica encara no hi ha depositat prou confiança. Per tal de generar aquesta confiança, és necessari que els criptoanalistes dediquin molt temps a analitzar-los. Finalment, els algorismes criptogràfics sovint necessiten d'un conjunt de protocols que n'estandarditzin el seu ús en diferents circumstàncies. Com veurem més endavant, aquest fet pot arribar a ser crític per la seguretat dels esquemes. La criptografia post-quàntica encara no ha arribat a la maduresa en aquest sentit.

6.8 Resum

En aquest capítol hem après el concepte de criptografia de clau pública, concepte que engloba tot un conjunt d'algorismes criptogràfics que fan servir parells de claus: una clau pública coneguda per tothom i una clau privada només coneguda pel seu propietari. En els esquemes de xifrat amb clau pública, els usuaris poden xifrar missatges per a un destinatari utilitzant la clau pública d'aquest destinatari. Alhora, només el destinatari del missatge, que coneix la clau privada, serà capaç de desxifrar els missatges dirigits a ell. Oposadament, els esquemes de signatura digital permetran a l'emissor d'un missatge signar-lo amb la seva clau privada (que només ell coneix), permetent que aquesta signatura sigui validada per qualsevol que en conegui la clau pública.

També s'ha explicat l'intercanvi de claus de Diffie-Hellman, un protocol que permet a dues parts establir un secret comú per mitjà d'un canal insegur. Així mateix, ens hem centrat a descriure els dos criptosistemes de clau pública més utilitzats avui en dia, l'RSA i ElGamal.

La tècnica del sobre digital ens permet disposar dels avantatges de la criptografia de clau pública sense trobar-nos amb les limitacions de mida i de capacitat computacional d'aquesta, combinant criptografia de clau pública amb algorismes de criptografia simètrica.

6.9 Solucions dels exercicis

Exercici 6.1:

La clau a no és vàlida ja que $e \cdot d \mod \phi(n) \neq 1$:

$$\begin{aligned}&1647529266 \cdot 1853372443 \mod \phi(3353361769) = \\ &= 1647529266 \cdot 1853372443 \mod 3353239120 = \\ &= 1499866678 \neq 1\end{aligned}$$

La clau c no és vàlida ja que el mòdul no és producte de dos primers:
$n = 111086984740301 = 45613 \cdot 41141 \cdot 59197$

Exercici 6.2:

Per tal de xifrar el missatge, l'Alice farà servir la clau pública d'en Bob i procedirà a calcular: $c = m^e \mod n = 249^{2037827401} \mod 3720779831 = 1920900242$.

Quan en Bob rebi el missatge xifrat c, procedirà a desxifrar-lo amb la seva clau privada, calculant: $m = c^d \mod n = 1920900242^{2233915321} \mod 3720779831 = 249$

Exercici 6.3:

L'atacant pot aprofitar el fet que coneix tots els possibles missatges que els usuaris s'intercanvien juntament amb el fet que l'RSA no és una xifra probabilística per generar tots els possibles textos xifrats i comparar el resultat amb el missatge intercanviat. Així, l'atacant procediria a calcular:

Lletra	m	c
A	65	$65^{2037827401} \mod 3720779831 = 3489265165$
B	66	$66^{2037827401} \mod 3720779831 = 3192216487$
C	67	$67^{2037827401} \mod 3720779831 = 2269693817$
D	68	$68^{2037827401} \mod 3720779831 = 3031724104$
E	68	$69^{2037827401} \mod 3720779831 = 1496836939$
	...	
G	71	$71^{2037827401} \mod 3720779831 = 2224890518$
H	72	$72^{2037827401} \mod 3720779831 = 228316393$
O	79	$79^{2037827401} \mod 3720779831 = 251951562$
P	80	$80^{2037827401} \mod 3720779831 = 2032163849$
R	82	$82^{2037827401} \mod 3720779831 = 1639486112$
T	84	$84^{2037827401} \mod 3720779831 = 3308958529$
Y	89	$89^{2037827401} \mod 3720779831 = 1818812718$

Taula 6.4: Càlculs realitzats per l'atacant.

Descobrint que el missatge xifrat correspon a la cadena:

$$[67, 82, 89, 80, 84, 79, 71, 82, 65, 80, 72, 89]$$

que alhora codifica el missatge `CRYPTOGRAPHY`.

Exercici 6.4:

Les claus a i d són vàlides;
La clau b no és vàlida ja que $p = 3383730189$ no és primer.
La clau c no és vàlida ja que $\beta \neq alpha^d \mod p$:

$$14736556^{144823569} \mod 337681733 = 93610276 \neq 93610277$$

Exercici 6.5:

Per tal de xifrar el missatge, l'Alice farà servir la clau pública d'en Bob i procedirà a seleccionar un valor aleatori v, per exemple, $v = 8341864$ i calcular:
$c_1 = \alpha^v \mod p = 1113291034^{8341864} \mod 3575204279 = 3323366879.$
$c_2 = m \cdot \beta^v \mod p = 424242 \cdot 792418784^{8341864} \mod 3575204279 = 3166979642.$
El text xifrat és el parell $(c_1, c_2) = (3323366879, 3166979642)$.

Quan en Bob rebi el missatge xifrat (c_1, c_2), procedirà a desxifrar-lo amb la seva clau privada, calculant:
$m = \frac{c_2}{c_1^d} \mod p = \frac{3166979642}{3323366879^{2036555019}} \mod 3575204279 = 3166979642 \cdot 663237018 = 424242$

Noteu que la solució de l'exercici no és única, ja que depèn de la selecció del paràmetre aleatori v.

Exercici 6.6:

Podem calcular el logaritme discret proposat per força bruta, és a dir, calculant $36848^x \mod 42841$ per a tots els valors possibles d'x (de 0 a 42840), fins a trobar el resultat que busquem, 12483. Així, trobarem que $36848^{4928} \mod 42841 = 12483$. No podríem seguir aquest mateix enfocament per a valors de 1024 bits, ja que el número de possibilitats a provar és massa gran.

Exercici 6.7:

Seleccionem un valor aleatori h, per exemple, $h = 55$. Calculem:
$r = \alpha^h \mod p = 386331185^{55} \mod 797445667 = 673983968$
$s = (m - d \cdot r) \cdot h^{-1} \mod p - 1 = (45678 - 373845532 \cdot 673983968) \cdot 55^{-1} \mod 797445666 = 1042804$
La signatura correspondria al parell $(r, s) = (673983968, 1042804)$.

Per a realitzar una segona signatura, seleccionaríem un segon valor aleatori h, per exemple, $h = 77$:
$r = \alpha^h \mod p = 386331185^{77} \mod 797445667 = 205790131$
$s = (m - d \cdot r) \cdot h^{-1} \mod p - 1 = (45678 - 373845532 \cdot 205790131) \cdot 77^{-1} \mod 797445666 = 284046946$
$(r, s) = (205790131, 284046946)$.

Per tal de validar les signatures, procedim a calcular:
$t = \beta^r r^s \mod p = 505206688^{673983968} \cdot 673983968^{1042804} \mod 797445667 = 137624270$
i comprovem que el valor sigui igual a $\alpha^m \mod p$:
$\alpha^m \mod p = 386331185^{45678} \mod 797445667 = 137624270$

Per a la segona signatura, realitzem el mateix procediment:
$t = \beta^r r^s \mod p = 505206688^{205790131} \cdot 205790131^{284046946} \mod 797445667 = 137624270$
i comprovem que el valor sigui igual a $\alpha^m \mod p$:

$$\alpha^m \mod p = 386331185^{45678} \mod 797445667 = 137624270$$

Noteu que la solució de l'exercici no és única, ja que depèn de les eleccions del paràmetre aleatori h.

Exercici 6.8:

El missatge m és major que el mòdul de la clau pública d'en Bob $n = 3720779831$. Per aquest motiu, el missatge no es pot xifrar directament amb RSA. Una alternativa es fer servir la tècnica del sobre digital: l'Alice genera una clau simètrica k aleatòria i xifra el missatge amb la clau k, $c = E_k(m)$, fent servir com a algorisme de xifrat E qualsevol esquema de clau simètrica. L'Alice xifra després la clau k amb la clau pública d'en Bob fent servir RSA, $c_k = E_{K_{pub}}(k)$. Finalment, l'Alice envia en Bob els dos valors, c i c_k.

6.10 Bibliografia

Bernstein, Daniel J. (2009). *Introduction to post-quantum cryptography*. Springer Berlin Heidelberg, 1-14.

Diffie, Whitfield, and Martin Hellman. (1976). *New directions in cryptography*. IEEE transactions on Information Theory, 22.6: 644-654.

ElGamal, Taher (1985). *A public key cryptosystem and a signature scheme based on discrete logarithms.*. IEEE transactions on information theory 31.4: 469-472.

Menezes, Alfred J., Paul C. Van Oorschot, and Scott A. Vanstone. (1996). *Handbook of applied cryptography*. CRC press.

NIST (2013). *FIPS PUB 186-4 Digital Signature Standard (DSS)*. National Institute of Standards and Technology. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`

Paar, Christof, and Jan Pelzl. (2009). *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media.

Pornin, T. (2013). *RFC6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).*
`https://tools.ietf.org/html/rfc6979`

Rescorla, E. (1999). *Diffie-Hellman Key Agreement Method.*
`https://tools.ietf.org/html/rfc2631`

Rivest, Ronald L., Shamir, Adi, and Adleman, Leonard. (1978). *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM 21.2: 120-126.

7. Infraestructura de clau pública

En el capítol anterior s'ha vist com l'aparició de la criptografia de clau pública permetia solucionar el problema de la distribució de claus. Ara bé, quan dues entitats volen iniciar una comunicació segura, com poden saber quina és la clau pública que correspon a cada una?

Per cobrir aquesta necessitat, és a dir, per vincular identitats amb les seves respectives claus públiques, apareixen els certificats digitals i, amb ells, la infraestructura de clau pública (en anglès *Public Key Infrastructure, PKI*), un conjunt de processos, rols i especificacions que permeten gestionar aquests certificats. Així, la infraestructura de clau pública permet transferir informació de manera segura, tot oferint serveis d'autenticació, integritat i confidencialitat.

En aquest capítol, es detallen quines són les principals entitats que formen part d'una infraestructura de clau pública i la seva funció, es descriu què són els certificats digitals i quin n'és el seu cicle de vida, s'exposen diferents estàndards que es fan servir en l'àmbit de les PKIs i, finalment, es discuteix sobre els problemes que pateixen els desplegaments de PKIs a nivell pràctic.

7.1 Entitats d'una PKI

Una PKI està formada per diverses entitats que interactuen entre elles. Algunes d'aquestes entitats existeixen en tots els desplegaments de PKI, mentre que d'altres en són opcionals i només existiran en segons quin tipus de desplegament. En aquesta secció, descriurem les diferents entitats que poden formar part d'una PKI, tot indicant si la seva existència és opcional si n'és el cas.

7.1.1 Autoritat de certificació

Una **autoritat de certificació** (CA per les seves sigles en anglès, *Certification Authority*) és una entitat que certifica el lligam entre un parell de claus i una identitat. Aquesta certificació es realitza mitjançant la signatura digital d'una estructura de dades que conté tant la identitat com la clau pública corresponent. L'autoritat de certificació també és l'encarregada de revocar aquest lligam si n'és el cas.

L'estructura de dades que signa una autoritat de certificació es coneix amb el nom de certificat de clau pública. Així:

Un **certificat digital** de clau pública és una estructura de dades que vincula una clau pública a una identitat.

Formats de certificats digitals

Existeixen diversos formats per a certificats digitals, per exemple, els certificats X.509 (que presentarem més endavant en aquest mateix capítol), els certificats PGP o els SPKI.

Aquesta vinculació es produeix fent que una autoritat de certificació de confiança signi el certificat digital, que conté tant la clau pública com la identitat. Addicionalment, un certificat digital acostuma a contenir altres camps com ara informació sobre l'emissor, la validesa, identificadors dels algorismes involucrats en la signatura del certificat, etc. A la Secció 7.3.1 es descriuen amb detall els certificats digitals X.509, un dels formats més utilitzats avui en dia.

Figura 7.1: Abstracció del contingut d'un certificat digital.

Les autoritats de certificació disposen d'un document anomenat Declaració de Pràctiques de Certificació (CPS, de l'anglès, *Certification Practice Statement*) que estableix les normes que regeixen l'emissió i gestió de certificats d'aquella CA. Alhora, la CPS deriva de la Política de Certificació (CP, per les seves sigles en anglès, *Certificate Policy*), que és un document més general que descriu l'arquitectura de la PKI en què s'engloba la CA i els actors que hi participen, els usos permesos per als certificats que emet la CA, la política de generació de claus, l'ús de CRLs i diversos processos que du a terme la CA.

7.1.2 Autoritat de registre

Tot i que les tasques de registre poden ser dutes a terme per les autoritats de certificació, en certs escenaris (com ara quan les entitats finals es troben dispersades geogràficament o bé quan el número de entitats a certificar és molt gran) pot ser d'interès que se separin i siguin realitzades per una altra entitat.

Una **autoritat de registre** (RA per les seves sigles en anglès, *Registration Authority*) és una autoritat que verifica la identitat de l'entitat que se certifica en un certificat digital.

Aquesta verificació de la identitat pot ser duta a terme, per exemple, a través de la personificació física i mostrant el document nacional d'identitat o el passaport.

Addicionalment, l'autoritat de registre pot dur a terme altres tasques. Per exemple, la generació de claus o bé la iniciació del procés de revocació del certificat, ambdues en nom de l'usuari final.

L'autoritat de registre és una entitat opcional en una PKI, ja que les tasques que realitza poden ser fetes per la pròpia CA.

7.1.3 Autoritat de validació

Amb la utilització pràctica de certificats digitals apareix la necessitat de poder-los revocar, és a dir, d'anul·lar-ne la seva validesa abans de la data de caducitat del propi certificat. Aquesta necessitat pot sorgir, per exemple, quan les claus privades corresponents s'han vist compromeses.

Una **llista de revocació de certificats** o CRL (de l'anglès, *Certificate Revocation List*) és una llista dels certificats que es troben revocats (però que no estan caducats) en un moment concret. Aquesta llista és signada per la CA (o bé per l'emissor de la CRL) i conté una marca de temps.

Així doncs, la publicació periòdica de llistes de certificats revocats és una manera de donar a conèixer els certificats que es troben revocats. A la Secció 7.3.2 es descriu amb detall les llistes de revocació de certificats definides per l'estàndard X.509 així com les diferents alternatives de publicació d'aquestes llistes.

Una alternativa (o un complement) a l'ús de CRLs és l'ús del protocol OSCP:

El protocol **OCSP** (de l'anglès, *Online Certificate Status Protocol*) permet obtenir l'estat d'un certificat digital identificat de manera interactiva.

En concret, el protocol OCSP permet a un client emetre una petició sobre l'estat d'un certificat digital a un servidor OSCP. El servidor OCSP respondrà amb una resposta signada indicant l'identificador del certificat, el seu estat de revocació, l'interval de temps en el qual es considera vàlida la resposta i informació addicional. L'estat del certificat pot ser `bo`, `revocat` (ja sigui permanentment o en suspensió) o `desconegut`.

Una **autoritat de validació** (VA per les seves sigles en anglès, *Validation Authority*) és una autoritat que ofereix un servei que permet verificar la validesa d'un certificat digital.

7.1.4 Autoritat de segellat de temps

Certes aplicacions requereixen de segells de temps segurs per operar. Dins d'una PKI, l'autoritat de segellat de temps és l'entitat encarregada de proporcionar aquests segells.

> Una **autoritat de segellat de temps** o TSA (per les seves sigles en anglès, *Time Stamping Authority*) és una entitat que crea segells de temps que permeten demostrar que una dada o document existia en un instant particular en el temps.

Un segell de temps és doncs una signatura digital realitzada per una TSA sobre una estructura de dades que conté, d'una banda, el hash d'un document i, d'altra banda, una representació d'un instant de temps (Figura 7.2).

És important notar que un segell de temps ens assegura que el document existia en la data esmentada en el segell, però no ens dóna informació sobre en quin moment va ser creat. Un ús habitual dels segells de temps es troba en la verificació que una signatura digital d'un missatge va ser creada abans que el corresponent certificat digital fos revocat. D'aquesta manera es permet que un certificat digital revocat pugui ser utilitzat per validar signatures digitals creades amb anterioritat a la revocació. Un altre dels usos habituals és en el lliurament de documents o sol·licituds que tenen una data límit: l'ús del segell de temps permet demostrar que el document o la sol·licitud existia en un instant de temps concret, anterior a la data límit.

Figura 7.2: Abstracció del contingut d'un segell de temps.

A la Secció 7.3.4 es descriu el protocol de timestamp definit per l'estàndard X.509.

7.1.5 Entitat final

> Una **entitat final** és una entitat que disposa d'un certificat en una PKI i que no és una autoritat de certificació. Les entitats finals poden ser individus, però també organitzacions, aplicacions o fins i tot dispositius.

Les entitats finals són titulars d'un certificat digital. S'anomenen entitats finals ja que apareixen al final de la cadena de certificació.

7.1.6 Repositori de certificats

L'existència d'un certificat digital emès per una entitat de confiança que vinculi una identitat amb un parell de claus és de poca utilitat si hom no es troba en disposició del certificat digital en qüestió.

La manera més senzilla de solucionar aquest problema consisteix a deixar que siguin els propis usuaris els que es facin arribar els certificats els uns als altres, per exemple, enviant-los per correu electrònic o trobant-se en persona i fent l'intercanvi en un suport físic. Aquest mètode es coneix com a **disseminació privada** i és viable quan el nombre d'usuaris és reduït i la majoria d'usuaris es coneixen, però no escala bé quan el nombre d'usuaris creix. Una de les alternatives consisteix a publicar els certificats digitals, de manera que els usuaris en tinguin accés.

Els **repositoris de certificats** emmagatzemen i proporcionen accés als certificats digitals.

El terme repositori és un terme genèric que es fa servir per referir-se a qualsevol base de dades centralitzada (almenys a nivell lògic) capaç d'emmagatzemar informació i servir-la quan aquesta és sol·licitada.

Pel que fa a la seguretat d'aquests repositoris, la integritat dels certificats queda garantida per la pròpia signatura que inclouen, de manera que un atacant no pot modificar amb èxit els certificats digitals del repositori sense ser detectat. Tot i això, els repositoris poden ser susceptibles a altres atacs, per exemple, atacs de denegació de servei que inhabilitin l'accés al repositori als usuaris legítims.

7.1.7 Repositori de llistes de revocació de certificats

A part dels repositoris de certificats, les PKI també poden disposar de repositoris de llistes de revocació de certificats, on les aplicacions que necessitin validar un certificat digital puguin descarregar-se la CRL corresponent. De la mateixa manera que amb els repositoris de certificats digitals, les CRLs garanteixen la integritat del seu contingut a través de signatures digitals, però els repositoris poden ser susceptibles a altres atacs.

Els **repositoris de llistes de revocació de certificats** emmagatzemen i proporcionen accés a les CRLs.

A la Secció 7.3.2 es descriuen les característiques i format de les CRLs segons l'estàndard X.509.

7.2 Cicle de vida d'un certificat digital

En aquesta secció es descriu el cicle de vida d'un certificat digital, és a dir, les etapes o processos per els quals passa un certificat, des dels preparatius que cal fer per poder-lo crear fins al processos que es duen a terme una vegada el certificat ja no està en ús.

Alguns d'aquests processos són intrínsecs a la vida d'un certificat digital i, per tant, sempre es duran a terme. Per exemple, sempre caldrà generar les claus que quedaran vinculades a un certificat generat, ja que les claus són essencials en la utilitat del certificat. En canvi, altres processos són opcionals, i només es duran a terme en circumstàncies concretes. N'és el cas, per exemple, de la revocació d'un certificat.

7.2.1 Generació del parell de claus

Existeixen diverses maneres de crear un parell de claus per a un usuari, totes elles amb els seus inconvenients i els seus avantatges. Tot i que alguns mètodes són preferits a altres, no hi ha una opinió unànime sobre quin és el millor mètode a utilitzar de manera universal:

- L'usuari crea el seu propi parell de claus. Aquest mètode té com a principal avantatge que l'usuari és l'únic que coneix la seva clau privada, ja que aquesta ha estat generada pels seus propis mitjans en el seu propi equip. Aquest mètode és especialment indicat quan les claus generades es volen utilitzar per realitzar signatures digitals amb no-repudi, ja que en aquest cas és convenient que cap altra entitat conegui la clau privada. Com a inconvenient, però, l'usuari ha de ser prou competent a nivell tècnic per poder generar el parell de claus de manera segura i per gestionar-ne les còpies de seguretat de manera adequada. Així, per exemple, un dels problemes que es poden presentar en aquest cas és la generació de claus en sistemes informàtics domèstics infectats de malware.
- El parell de claus és generat per la CA (o l'autoritat de registre). Tenint en compte que la CA ja és un element de confiança dins la PKI, hi ha qui opina que s'hi pot confiar també fins al punt de deixar que sigui aquesta qui generi les claus. El principal avantatge d'aquest mètode és que les CAs acostumen a tenir personal especialitzat, que és capaç de garantir la seguretat dels equips que generen les claus. Com a inconvenients, a més del fet de tenir una segona entitat que coneix la clau privada de l'usuari, apareix la centralització de la generació de claus.
- El parell de claus és generat per una tercera part. La tercera part genera aleshores les claus i comunica de manera físicament segura la clau privada a l'usuari. Després, la tercera part destrueix tota la informació relativa a la creació de les claus.
- El parell de claus és generat en un dispositiu hardware especialitzat. En aquest cas, sovint les claus privades queden emmagatzemades en una zona de seguretat del propi hardware, de manera que no poden ser extretes.

Altres consideracions que poden influir sobre la decisió de qui ha de generar el parell de claus són les implicacions legals que comporten o la capacitat de còmput necessària per a realitzar el procés. Aquest últim punt no és gaire problemàtic avui en dia, ja que els dispositius informàtics actuals tenen recursos suficients per generar claus de les mides que s'utilitzen en aquests moments.

Estàndards de CSR

Una de les sintaxis més utilitzades per a peticions de certificat és la que es descriu al PKCS#10.

Quan el parell de claus no és generat per la CA, serà necessari fer-li arribar la clau pública per tal que pugui ser inclosa en el certificat digital. La Petició de Signatura de Certificat (o CSR, de l'anglès, Certificate Signing Request) és un missatge amb una estructura de dades coneguda que envia el sol·licitant a la CA per tal d'informar-la de la clau pública que demana certificar. A més de la clau pública, la CSR inclou informació addicional, com ara informació d'identificació del sol·licitant.

Exemple 7.1 La generació de claus per a l'idCAT

La política de l'idCAT detalla que el ciutadà ha de generar les seves pròpies claus per a la identificació i la signatura electrònica en el seu ordinador personal. Així, el ciutadà serà l'encarregat de generar les claus i enviarà a la CA la clau pública a incloure en el certificat digital.

Per tal de simplificar aquest procés i fer-lo accessible a usuaris sense coneixements tècnics, el procés de generació de claus es pot realitzar a través del navegador, de manera gairebé transparent per a l'usuari.

Exemple 7.2 La generació de claus del DNI electrònic

El DNIe 3.0 conté un xip amb informació sobre el ciutadà que n'és propietari. Entre la informació que incorpora aquest xip, s'hi troben dos parells de claus RSA, un parell d'autenticació i un parell de signatura. Aquestes claus són generades dins del propi xip, amb una llibreria criptogràfica que porta incorporada.

Exercici 7.1 Quin procediment s'ha de seguir alhora de generar les claus vinculades a un certificat digital per tal de poder garantir que les signatures realitzades amb la clau privada corresponent tenen la propietat de no-repudi?

1. El subscriptor ha de crear el seu propi parell de claus
2. El parell de claus ha de ser generat per l'autoritat de certificació
3. El parell de claus ha de ser generat per una tercera part
4. El parell de claus és generat en un dispositiu hardware especialitzat

7.2.2 Registre

El **registre** és el procés per el qual una entitat final, ja sigui un individu o una organització, és verificada.

El nivell de verificació necessària dependrà de la Política de Certificació o de la Declaració De Pràctiques de Certificació. Així, per exemple, per a certificats amb polítiques de verificació laxes, l'usuari pot simplement omplir un formulari per tal de sol·licitar el registre. En canvi, per a polítiques de certificat més estrictes, serà necessari que l'usuari es personi físicament davant de l'autoritat de registre amb algun document d'identitat reconegut que incorpori una fotografia.

Exemple 7.3 El registre per a l'idCAT

L'Agència Catalana de Certificació ofereix els certificats digitals idCAT a la ciutadania. Per tal de realitzar el procés de registre, el ciutadà ha de personar-se a qualsevol de les Entitats de Registre idCAT (per exemple, als ajuntaments) i mostrar un document identificatiu (DNI, NIE o passaport). Prèviament, el ciutadà pot omplir un formulari amb les seves dades personals, de manera que el tramit s'agilitza.

Exemple 7.4 El registre per als certificats de persona física de la FNMT

La Fábrica Nacional de la Moneda y Timbre ofereix diferents tipus de certificats digitals, entre els quals hi ha els certificats de persona física. Per realitzar el registre d'un certificat de persona física de la FNMT, cal omplir un formulari per Internet i personar-se en alguna de les oficines de registre amb el codi de sol·licitud que s'obté al complimentar el formulari i un document d'identitat (DNI, passaport, carnet de conduir o NIE). Entre les oficines de registre disponibles s'hi troben les oficines de la Seguretat Social i les delegacions de l'Agència Tributària.

7.2.3 Creació del certificat

Una vegada s'ha generat el parell de claus i s'ha verificat la identitat de l'entitat, es crea el certificat digital. El certificat digital contindrà, principalment, la clau pública del titular (que haurà de ser enviada a la CA si el parell de claus no ha estat generat directament per la CA), la identitat verificada durant el registre i la signatura digital de la CA.

Exemple 7.5 La creació del certificat idCAT

Després de comprovar la identitat i les dades incloses en el certificat, i una vegada rebuda la clau pública, l'Agència Catalana de Certificació pot generar el certificat digital del sol·licitant. El titular del certificat el podrà obtenir mitjançant el procediment telemàtic d'obtenció del certificat, que també es realitza a través del navegador.

7.2.4 Disseminació i recuperació del certificat

El procés de disseminació del certificat dependrà del cas d'ús concret i de la Política de Certificació. En alguns casos, el certificat es lliura directament al seu titular, que serà l'encarregat de distribuir-lo a terceres parts sota la seva discreció. En altres casos, el certificat es publica en algun repositori públic, de manera que tothom en té accés lliure.

Addicionalment, si la generació de claus no ha estat realitzada per l'entitat final del certificat, caldrà que la clau privada sigui enviada també a l'usuari.

D'altra banda, quan parlem de recuperació del certificat ens referim a l'habilitat d'obtenir un certificat d'entitat final quan aquest és necessari. Els casos d'ús més habituals són quan es necessita enviar informació xifrada a un destinatari o bé quan és necessari verificar una signatura digital rebuda d'una altra entitat.

Exemple 7.6 El repositori de claus públiques del MIT

El MIT PGP Public Key Server (`http://pgp.mit.edu/`) és un dels repositoris de claus públiques més popular. Actualment, el repositori forma part de la xarxa SKS de servidors de claus públiques, de manera que totes les claus que s'hi publiquen es disseminen cap al centenar de servidors que formen part de la xarxa.

7.2.5 Validació del certificat

La validació d'un certificat digital és, en realitat, un procés constituït per un conjunt de validacions, que caldrà realitzar abans de permetre fer cap operació criptogràfica amb la clau que aquest certifica. Aquestes validacions passen per comprovar que:

- La **signatura digital** del certificat és vàlida, és a dir, la signatura ha estat realitzada amb la clau de l'emissor del certificat i és correcta per al contingut del certificat. Noteu que aquesta validació ens garanteix la integritat de les dades del certificat.
- La data actual es troba dins del **període de validesa** del certificat digital, és a dir, el certificat no ha expirat.
- El certificat no ha estat **revocat**.
- L'**ús** que s'està donant al certificat digital és correcte (tenint en compte les restriccions d'utilització, de nom, de política, les extensions d'utilització de la clau, etc.).
- El certificat ha estat emès per una **entitat de confiança**.

Per tal de validar un certificat digital, caldrà construir la cadena de certificats entre el certificat que es vol validar i l'entitat de confiança.

Una **cadena de certificats** és una llista ordenada de certificats de clau pública començant per un certificat signat per una entitat de confiança i acabant amb el certificat que es vol validar. Tots els certificats intermedis són certificats de CA en els quals el titular d'un certificat correspon amb l'emissor del següent.

Per tal de validar doncs un certificat, caldrà comprovar tots els certificats de la cadena.

7.2.6 Expiració del certificat

Els certificats digitals tenen un període de validesa en el qual es consideren vàlids per a la seva funció. Quan aquest període de validesa acaba, diem que el certificat digital ha **expirat**.

Abans que finalitzi el període de validesa d'un certificat digital, es pot **actualitzar** el certificat digital, de manera que l'entitat final certificada pugui seguir gaudint dels serveis de la PKI ininterrompudament. El procés d'actualització passa per la creació d'un nou parell de claus i d'un nou certificat associat a les noves claus, amb un nou període de validesa. Aquest procés s'acostuma a dur a terme quan s'aproxima la data d'expiració del certificat original. Com que el titular del certificat ja ha passat pel procés de registre i disposa d'un certificat vàlid en el moment d'actualitzar-lo, el procés d'actualització no requereix que el titular del certificat torni a passar pel procés de registre.

Una alternativa és la **renovació** del certificat digital. En aquest cas, la mateixa clau pública que hi ha al certificat que està a punt d'expirar s'inclou en un nou certificat, amb un nou període de validesa. Cal anar en compte, però, a l'hora de renovar certificats digitals, ja que pot suposar problemes de seguretat en certes circumstàncies.

Exemple 7.7 L'expiració dels certificats del DNI electrònic

Els certificats del DNIe 3.0 tenen una validesa de 60 mesos des de la data d'emissió (o inferior si la data de caducitat del dni és anterior a aquests 60 mesos). Els certificats del DNIe es poden actualitzar personant-se en un *Punt d'Actualització del DNIe* dins d'una oficina d'expedició. El procés d'actualització és un procés automatitzat on el ciutadà s'autentica amb dades biomètriques i introdueix el seu pin, i noves claus i certificats són creats.

7.2.7 Revocació del certificat

Els certificats digitals tenen un període de validesa indicat en el propi certificat. A vegades però, és necessari poder invalidar un certificat abans que finalitzi aquest període. N'és el cas, per exemple, quan la clau privada corresponent és compromesa, quan es produeix un canvi de nom o quan canvia l'associació entre un titular i la CA (en particular, quan un treballador es desvincula d'una empresa). En aquest casos, serà necessari revocar el certificat digital.

Un certificat digital **revocat** és aquell que ha estat cancel·lat abans de la seva data d'expiració.

Exemple 7.8 La revocació certificats del DNI electrònic

El certificat de signatura digital del DNIe pot ser revocat personificant-se físicament en qualsevol de les oficines d'expedició del DNIe.

7.2.8 Història i arxivament de claus

Tot i que els certificats digitals tenen una data d'expiració, això no implica que totes les dades xifrades amb les claus d'aquests certificats hagin de deixar de ser accessibles quan el certificat caduca. Per tant, és necessari que les claus siguin emmagatzemades, encara que el corresponent certificat digital hagi caducat. Aquest procés es coneix amb el nom d'**història de claus** i és dut a terme, principalment, per a emmagatzemar claus privades que permetin desxifrar contingut que va ser xifrat en el passat. Normalment, la pròpia entitat final realitza aquest procés de manera local.

En canvi, quan es parla d'**arxivament de claus**, normalment es parla d'un servei ofert per una tercera part, que emmagatzema material de claus de vàries entitats finals. L'arxivament de claus consisteix en l'emmagatzemament a llarg termini de claus (ja siguin de xifratge o de verificació de certificats). L'arxivament de claus és útil, per exemple, quan s'intenta validar una signatura digital creada amb una clau associada a un certificat que ja ha expirat.

7.3 Els estàndards X.509

L'estàndard X.509 de l'ITU-T defineix un framework per als certificats de clau pública, incloent l'especificació de les dades utilitzades per representar els certificats en si, així com la informació sobre revocacions de certificats. Addicionalment, l'estàndard defineix també frameworks per a certificats d'atributs i serveis d'autenticació. Així, l'estàndard no ofereix la descripció de tots els components d'una PKI, sinó només d'una part, amb la intenció de servir com a base per a l'especificació i construcció de PKIs completes.

Certificats d'atributs

Un **certificat d'atributs** o AC (de l'anglès, *attribute certificate* és una estructura de dades signada digitalment que vincula uns valors d'uns atributs amb informació d'identificació del seu propietari.

D'altra banda, l'IETF dedica esforços a l'estandardització de les infraestructures de clau pública basades en X.509 a través del grup de treball PKIX. Inicialment, la feina del grup se centrava en perfilar les normes X.509 que produïa la ITU-T, però posteriorment el grup també va començar a desenvolupar iniciatives independents adreçades a cobrir les necessitats de la PKI a Internet. La IETF publica els seus documents tècnics en les anomenades RFC (de l'anglès, *Request For Comments*), algunes de les quals tenen caràcter estandarditzador.

ITU

La ITU (de l'anglès, *International Telecommunication Union* és l'agència de les Nacions Unides especialitzada en l'àmbit de les telecomunicacions, la informació i les tecnologies de la comunicació. En concret, la ITU-T és la branca de la ITU que coordina els estàndards sobre telecomunicacions.

IETF

La IETF (de l'anglès, *International Engineering Task Force* és una comunitat internacional que té com a objectiu millorar i evolucionar Internet, a través de la producció de documents tècnics que guiïn aquesta evolució.

La primera versió de l'estàndard internacional ITU-T X.509 va ser publicada el 1988 com a part de les recomanacions per directori X.500 i defineix un format estàndard per a certificats digitals. La versió descrita en aquest estàndard es coneix com a versió 1. El 1993, l'estàndard va ser revisat i es van afegir dos camps més als certificats, amb el que es coneix com la versió 2 del format. També durant el 1993 es van publicar les RFCs relacionades de Internet Privacy Enhanced Mail (PEM), que inclouen especificacions per a una PKI basada en els certificats x.509 v1 (RFC1422). L'experiència obtinguda a l'intentar fer desplegaments d'aquesta RFC va servir per mostrar les deficiències del format v1 i v2 dels certificats X.509. En resposta a les deficiències detectades, es va crear la versió 3 del format del certificats, que extèn la versió 2 afegint la possibilitat de crear camps d'extensions addicionals. L'estandardització del format v3 va ser completada al juny de 1996 i es considera vigent en l'actualitat.

7.3.1 Certificats de clau pública

Com hem vist, un certificat digital de clau pública és una estructura de dades que vincula una clau pública amb una identitat a través d'una signatura d'aquest vincle feta per una autoritat. En aquesta secció, descriurem l'estructura d'un certificat X.509.

Gestió de certificats X.509

L'Openssl inclou l'eina `x509` que incorpora tot de funcionalitats relatives als certificats X.509. Així, per exemple, l'eina permet visualitzar el contingut dels certificats, convertir certificats a diferents formats o signar peticions de certificat.

La Taula 7.1 mostra els camps d'un certificat X.509 versió 3. L'estructura de dades del certificat és similar a moltes de les estructures de dades on s'hi emmagatzema contingut signat: s'inclou un camp amb el contingut a signar, un camp amb l'identificador de l'algorisme que s'ha fet servir per resumir i signar el contingut i, finalment, el valor de la signatura.

Camp	**Descripció**
`TBSCertificate`	El certificat a signar (en anglès, *to-be-signed certificate*).
`signatureAlgorithm`	Identificador de l'algorisme de signatura utilitzat per l'emissor del certificat per signar-lo.
`signatureValue`	Cadena de bits amb el valor de la signatura.

Taula 7.1: Camps d'un certificat X.509.

L'identificador de l'algorisme de signatura

L'identificador de l'algorisme de signatura acostuma a especificar una funció hash amb la qual es fa un resum del contingut a signar i un algorisme de signatura.

La Taula 7.3.1 mostra els camps del certificat a signar d'un certificat X.509 versió 3.

Els nom distingits (o DN, de l'anglès, *Distinguished Name*) acostumen a ser representats com una cadena de caràcters, on diferents atributs i el seu valor són llistats separats per comes. Les claus reconegudes són el nom comú (CN, de l'anglès, *CommonName*); el nom de la localitat (L, de l'anglès, *LocalityName*); el nom de l'estat o província (ST, de l'anglès *StateOrProvinceName*); el nom de l'organització (O, de l'anglès, *OrganizationName*; el nom de la unitat organitzativa (OU, de l'anglès, *OrganizationalUnitName*); el nom del país (C, de l'anglès, *CountryName*); i el carrer (STREET, de l'anglès, *StreetAddres*).

Exemple 7.9 Exemple de certificat x.509

A continuació s'inclou un exemple d'un certificat x.509 emès per a un estudiant de l'assignatura de criptografia de la UOC.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 8793 (0x2259)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=CAT, ST=Barcelona, L=Barcelona, O=UOC, OU=EIMT,
            CN=Consultor Criptografia
        Validity
            Not Before: May 23 13:27:19 2016 GMT
            Not After : May 23 13:27:19 2018 GMT
        Subject: C=CAT, ST=Barcelona, O=UOC,
            OU=EstudiantsCriptografia,
            CN=estudiant/emailAddress=estudiant@uoc.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (361 bit)
                Modulus:
                    01:b4:50:f5:bc:50:66:5e:80:0f:a3:85:07:de:c5:
                    d0:d4:36:c6:54:b1:66:db:46:49:06:37:4d:85:e2:
                    e7:b3:e8:b4:39:d7:05:77:20:67:8c:68:be:f9:37:
                    9d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
```

```
            Netscape Cert Type:
                SSL Client, S/MIME
            X509v3 Key Usage:
                Digital Signature, Non Repudiation
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                32:6C:46:E0:A5:7A:97:E3:EC:E6:0F:3D:23:14:13:7B:
                    5B:E0:97:F3
            X509v3 Authority Key Identifier:
                keyid:D2:D1:3D:A7:69:53:C6:B3:8A:10:D6:3A:51:87:
                    EB:56:4C:7C:99:7A
                DirName:/C=CAT/ST=Barcelona/L=Barcelona/
                    O=UOC/OU=EIMT/CN=Consultor Criptografia
                serial:D5:16:AD:04:20:AA:8C:26

            Netscape CA Revocation Url:
                http://www.uoc.edu/criptografia/ca-crl.pem
    Signature Algorithm: sha1WithRSAEncryption
        a4:6f:89:4e:2c:fe:85:0b:a2:7e:02:e6:45:3f:81:79:22:fa:
        2f:a1:d8:bf:43:f8:42:b9:b1:6f:6c:66:93:96:a6:2e:af:cc:
        c0:40:5f:21:69:60:77:0b:4f:00:06:40:61:f7:ad:09:1a:f2:
        1d:55:3c:a6:f5:dc:c2:f6:39:81:57:59:d6:cc:c6:b5:ad:00:
        78:be:2f:ae:d4:b6:e6:71:ab:5a:03:76:3d:0c:55:3d:87:b7:
        ab:a8:8c:2a:ef:87:09:3e:f8:50:71:b4:67:5b:a2:72:8e:a2:
        3d:3c:06:d4:09:93:c6:d7:df:4c:b3:a9:6f:ba:b2:f9:3b:95:
        44:e3:15:3c:15:ce:24:1f:23:16:c9:07:72:91:90:ff:8d:e2:
        c6:1c:95:22:18:b1:d9:39:a1:31:97:4f:cb:cc:71:23:94:4d:
        ef:0b:f0:64:3d:f7:a0:70:4c:2e:0f:6c:54:f1:95:52:00:85:
        62:9c:a3:b2:28:ea:f0:21:58:ba:4c:24:38:d7:9b:9c:78:6a:
        a6:fc:cc:11:62:11:9b:55:59:66:08:9d:98:11:3b:4c:20:e0:
        31:81:ef:1b:6d:3b:97:75:de:1f:75:6c:e5:6a:95:96:a5:9b:
        2d:f9:78:f2:31:88:f3:36:b4:21:cd:20:d4:91:e2:b0:0b:48:
        ab:fc:64:57
```

Extensions dels certificats de clau pública

El camp d'extensions permet afegir nous camps a l'estructura de dades d'un certificat sense haver de modificar-ne la seva definició en ASN.1.

ASN.1

L'ASN.1 (de l'anglès, *Abstract Syntax Notation One*) és un llenguatge que permet definir estructures de dades de manera indepenent del dipositius o representacions internes que aquests facin servir.

Un camp d'extensió consisteix en:

- Un identificador d'extensió.
- Un flag de criticitat.
- Un valor codificat.

Les extensions, al ser camps addicionals, poden no ser reconegudes per totes les entitats que poden processar el certificat digital. El valor booleà del flag de criticitat condiciona com afectarà el reconeixement de l'extensió a la validació del certificat. Si el flag de criticitat conté el valor `FALSE` i l'entitat que ha de

Taula 7.2: Camps del certificat a signar.

Codi	Descripció
`version`	Indica la versió del certificat. Pot contenir els valors 0 (v1), 1 (v2) o 2 (v3).
`serialNumber`	És un valor enter designat per la CA que és únic per cada certificat emès per aquella CA en concret, és a dir, el parell de valors emissor i número de serie identifica de manera única un certificat digital de clau pública.
`signature`	Conté l'identificador de l'algorisme i la funció de hash fets servir per la CA per signar el certificat, per exemple, `sha-1WithRSAEncryption`. Aquest valor ha de ser el mateix que el del camp signatureAlgorithm.
`issuer`	Conté el nom distingit de la CA que emet el certificat, que no pot ser una cadena buida.
`validity`	Indica el període de validesa del certificat digital. Durant aquest interval, la CA garanteix que mantindrà informació sobre l'estat del certificat. El període de validesa s'indica amb dos camps de temps: `notBefore` i `notAfter`.
`subject`	Nom distingit que identifica al titular de la clau pública que està sent certificada. El camp pot estar buit si es tracta d'un certificat d'entitat final amb l'extensió `subjectAltName` inclosa i marcada com a crítica. En cas contrari, el camp no pot contenir una cadena buida.
`subjectPublicKeyInfo`	Conté dos components: `algorithm` i `subjectPublicKey`. El camp `algorithm` ha de contenir l'algorisme al que pertany la clau pública. El camp `subjectPublicKey` ha de contenir la clau pública que està sent certificada.
`issuerUniqueIdentifier`	Camp opcional que es fa servir per identificar de forma única l'emissor en cas de reutilització de noms.
`subjectUniqueIdentifier`	Camp opcional que es fa servir per identificar de forma única el titular del certificat en cas de reutilització de noms.
`extensions`	Camp opcional que permet afegir nous camps a l'estructura de dades.

processar el certificat no reconeix l'extensió, aleshores pot ignorar-la a l'hora de realitzar la validació. En canvi, si el flag té el valor `TRUE`, una extensió no reconeguda causa que el certificat es consideri invàlid. Si l'entitat que ha de processar el certificat reconeix l'extensió, aleshores l'estàndard especifica que aquesta hauria de processar l'extensió, independentment del valor de criticitat que tingui.

Per tant, totes les extensions que tenen el flag de criticitat a `FALSE` poden causar comportaments inconsistents entre les entitats que reconeixen l'extensió (i que, per tant, la processaran) i aquelles que no la reconeixen (i que, per tant, la poden ignorar).

L'estàndard X.509 defineix algunes extensions. En el següents paràgrafs, es descriuen algunes d'aquestes extensions.

L'extensió d'ús de la clau (`KeyUsage`) permet descriure la intenció d'ús del certificat. La Taula 7.3 descriu els possibles usos reconeguts per l'extensió. Un mateix certificat pot indicar diversos usos, tot i que aquest comportament pot suposar riscos de seguretat.

Concordança de les extensions

Si el booleà de `cA` de l'extensió `BasicConstraints` s'indica com a `FALSE`, aleshores l'extensió `KeyUsage` no pot tenir l'ús de `keyCertSign` actiu.

L'extensió de restriccions bàsiques (`BasicConstraints`) permet identificar si el titular del certificat és una autoritat de certificació i la profunditat màxima de les cadenes de certificació que inclouen el certificat en

Taula 7.3: Usos de clau reconeguts.

Codi	Descripció
`digitalSignature`	Verificació de signatures digitals.
`contentCommitment`	Verificació de signatures digitals on el signatari es compromet amb el contingut signat.
`keyEncipherment`	Xifratge de claus o d'algun altre tipus d'informació de seguretat.
`dataEncipherment`	Xifratge de dades d'usuari.
`keyAgreement`	Ús com a clau pública en un protocol d'establiment de claus.
`keyCertSign`	Verificació de signatures de certificats realitzades per Autoritats de Certificació.
`cRLSign`	Verificació de signatures de CRLs realitzades per autoritats.
`encipherOnly`	Ús com a clau pública en un protocol d'establiment de claus per utilitzar únicament xifratge de dades (cal indicar també l'ús `keyAgreement`).
`decipherOnly`	Ús com a clau pública en un protocol d'establiment de claus per utilitzar únicament desxifratge de dades (cal indicar també l'ús `keyAgreement`).

qüestió. D'una banda, l'extensió permet incloure un valor booleà que indica si la clau pública certificada pot ser utilitzada per verificar signatures de certificats digitals (camp `cA`). D'altra banda, i si el valor booleà revela que és un certificat de CA, l'extensió inclou un enter que indica el número màxim de certificats intermedis (no autoemesos) que poden seguir a aquest certificat en una cadena de certificació vàlida (camp `pathLenConstraint`).

L'extensió d'identificador de la clau de l'autoritat (`AuthorityKeyIdentifier`) permet identificar la clau privada utilitzada per signar un certificat digital. Això és particularment útil quan l'emissor del certificat disposa de diverses claus.

L'extensió d'identificador de la clau del titular (`SubjectKeyIdentifier`) permet identificar els certificats que tenen una clau pública donada. Així, quan una entitat final té diversos certificats (per exemple, de diferents emissors) amb la mateixa clau pública, el conjunt de certificats pot ser identificat fàcilment.

Exercici 7.2 Una aplicació ha de validar un certificat digital que conté una extensió marcada com a crítica. Indiqueu quin hauria de ser el resultat de la validació per les següents casuístiques (suposant que la resta de comprovacions que es realitzen per validar el certificat són correctes).

Nota: Un resultat `TRUE` indica una validació satisfactòria, mentre que un resultat `FALSE` indica que la validació no és correcta.

	Resultat de processar l'extensió	
	`TRUE`	`FALSE`
L'aplicació reconeix l'extensió		
L'aplicació no reconeix l'extensió		

Exercici 7.3 Una aplicació ha de validar un certificat digital que conté una extensió marcada com a **no** crítica. Indiqueu quin hauria de ser el resultat de la validació per les següents casuístiques (suposant que la resta de comprovacions que es realitzen per validar el certificat són correctes).

	Resultat de processar l'extensió	
	`TRUE`	`FALSE`
L'aplicació reconeix l'extensió		
L'aplicació no reconeix l'extensió		

Tipus de certificats de clau pública

Existeixen, principalment, dos tipus de certificats de clau pública:

- Certificats de clau pública d'**entitats finals**.
- Certificats de clau pública d'**autoritats de certificació** (CA).

Un certificat d'entitat final és un certificat emès per una autoritat de certificació a una entitat que no és emissora d'altres certificats. En canvi, un certificat de CA és un certificat emès per una CA a una entitat que és també una CA i, per tant, també és capaç d'emetre certificats. Un certificat de CA ha d'incloure l'extensió `basicConstraints` amb el component `cA` a `True`.

Els certificats de CA poden ser alhora classificats en tres tipus diferents:

- Un certificat **autoemès** és un certificat de CA on el titular i l'emissor són la mateixa CA. Aquest tipus de certificat es pot fer servir, per exemple, per a realitzar un canvi de claus, transferint la confiança de la clau antiga a la nova clau.
- Un certificat **autosignat** és un cas especial d'un certificat autoemès on la clau privada utilitzada per signar el certificat correspon a la clau pública que se certifica amb el certificat. Es poden fer servir certificats autosignats, per exemple, per donar a conèixer una clau pública o altra informació.
- Un certificat **creuat** és un certificat de CA on l'emissor i el titular són autoritats de certificació diferents. Un certificat creuat es pot fer servir, per exemple, per a reconèixer l'existència de la CA titular o bé per autoritzar-la.

7.3.2 Llistes de revocació de certificats

Les autoritats de certificació són responsables d'informar sobre l'estat de revocació dels certificats que emeten. Un dels mètodes per oferir aquesta informació de revocació és la publicació de llistes de revocació de certificats o CRLs (per les seves sigles en anglès, *Certificate Revocation List*). Normalment, la pròpia autoritat de certificació emet les CRLs, però també pot delegar aquesta responsabilitat a alguna altra entitat.

Una CRL és una llista dels números de sèrie dels certificats revocats, juntament amb la signatura de la CA (o l'emissor de la CRL) i una marca de temps. Normalment, les CRLs es publiquen periòdicament, per exemple, cada hora o un cop al dia. Quan un certificat és revocat, el seu número de sèrie s'afegeix a la CRL que s'emet després de la revocació. El número de sèrie no s'ha d'eliminar de la CRL fins que hagi aparegut en una CRL emesa amb posterioritat a la fi del període de validesa del certificat.

Com que les CRLs contenen una signatura de l'entitat que les emet, la integritat del seu contingut està garantida. D'aquesta manera, no és necessari confiar en què els servidors o els processos que distribueixen les CRLs no intentaran modificar-les.

Un dels inconvenients que presenta l'ús de CRLs és l'endarreriment que es crea a l'hora d'informar sobre la revocació d'un certificat. Entre la revocació d'un certificat i l'addició del seu número de sèrie a la propera CRL que es publica passa un interval de temps, que podrà ser menor o major en funció de la periodicitat de la publicació de la CRL. Durant aquest període, tot i que el certificat es troba revocat, la informació de revocació no estarà disponible. Aquest problema es pot minimitzar amb l'ús de protocols interactius que consulten l'estat d'un certificat digital en un moment concret.

Un altre dels inconvenients que presenta l'ús de CRLs és que són susceptibles a atacs de denegació de servei. Un atacant pot evitar que la informació de revocació d'un certificat arribi a les aplicacions blocant la distribució de la CRL. Així, mentre que un atacant no podrà modificar el contingut de la CRL (degut a la signatura sobre aquest), sí que pot atacar la disponibilitat del servei.

Generació de CRLs	L'Openssl inclou l'eina `ca`, que incorpora funcionalitats per gestionar una autoritat de certificació, entre les quals hi ha la creació de CRLs.

Així doncs, quan un sistema necessita validar un certificat digital, a més de comprovar-ne la seva signatura i el període de validesa, també serà necessari que es descarregui una CRL prou recent i comprovi que el número de sèrie del certificat no hi figura. La definició del que es considera prou recent dependrà de la política de cada sistema.

Els certificats poden contenir informació sobre com obtenir informació de revocació a través de CRLs fent servir l'extensió `crlDistributionPoint`.

Tipus de CRLs

Cada llista de revocació de certificats té un **abast**, és a dir, un conjunt de certificats que poden aparèixer en aquella CRL. Per exemple, l'abast d'una CRL poden ser tots els certificats emesos per una determinada CA o bé tots els certificats emesos per una CA per un motiu concret.

Una llista CRL **completa** enumera tots els certificats no expirats dins del seu abast que han estat revocats per alguna de les raons cobertes per l'abast. D'altra banda, direm que una CRL és **plena i completa** quan conté tots els certificats no expirats emesos per la CA que s'han revocat per qualsevol raó.

Els termes plena i completa	En anglès, es fan servir els termes *complete* i *full and complete* per diferenciar entre CRLs que contenen únicament certificats revocats per un dels motius indicats a l'abast o bé independentment del motiu.

Una CRL **indirecta** és una CRL amb un abast que inclou almenys un certificat emès per una entitat de certificació diferent de l'emissor de la CRL. Una CRL indirecta pot incloure en el seu abast certificats emesos per diverses autoritats de certificació. A més, si l'emissor de la CRL és una CA, aleshores l'abast de la CRL pot incloure també els certificats emesos per aquesta CA.

Una **delta** CRL només enumera els certificats dins del seu abast que han canviat d'estat de revocació des de l'emissió d'una CRL completa referenciada. La CRL completa referenciada s'anomena CRL **base**. Es considera que l'estat d'un certificat ha canviat si aquest està revocat, si ha deixat d'estar suspès o si la raó per la qual el certificat ha estat revocat ha canviat. L'abast de la delta CRL ha de ser el mateix que la CRL base que referencia. A més, la clau privada utilitzada per signar la delta CRL també ha de ser la mateixa que la feta servir per signar qualsevol CRL completa que pugui actualitzar. Generalment, les delta CRLs són més petites que les CRLs que actualitzen, de manera que fer servir delta CRLs pot ajudar a reduir el consum d'ample de banda d'un sistema que faci servir CRLs.

Una aplicació que fa servir delta CRLs ha de ser capaç de construir una CRL completa combinant una CRL completa emesa amb anterioritat i la delta CRL més recent. Addicionalment, l'aplicació també pot construir una CRL completa a partir de la delta CRL més recent i d'una CRL construïda localment que és completa per aquest abast.

És considera que una delta CRL és **actual** si el temps actual es troba en el període comprés entre els camps `thisUpdate` i `nextUpdate`. Per tant, podria passar que l'emissor de CRLs emetés més d'una delta CRL abans del `nextUpdate`, existint aleshores més d'una delta CRL considerada actual. En aquests casos, l'estàndard recomana (però no exigeix) que es faci servir la CRL que té el `thisUpdate` més actual.

Contingut d'una CRL

Una CRL és una llista de certificats revocats signada per una autoritat. Així, els camps que defineixen una CRL segons l'estàndard X.509 es descriuen a la Taula 7.4.

La llista de certificats revocats és alhora una seqüència de diversos camps. La Taula 7.5 descriu els camps

Taula 7.4: Camps d'una CRL.

Camp	Descripció
`tbsCertList`	La llista de certificats revocats (en anglès, *to-be-signed certificate list*).
`signatureAlgorithm`	Identificador de l'algorisme de signatura utilitzat per l'emissor de la CRL per signar-la.
`signatureValue`	Cadena de bits amb el valor de la signatura.

que la componen.

Taula 7.5: Camps de la llista a signar.

Camp	Descripció
`version`	Opcional, descriu la versió de la CRL codificada.
`signature`	Identificador de l'algorisme de signatura utilitzat per l'emissor de la CRL per signar-la. El valor d'aquest camp ha de coincidir amb el valor del camp `signatureAlgorithm`.
`issuer`	Nom de l'entitat que ha emès i signat la CRL.
`thisUpdate`	Data d'emissió d'aquesta CRL.
`nextUpdate`	Opcional, data d'emissió de la propera CRL. La propera CRL pot ser emesa abans de la data indicada pel camp `nextUpdate`, però no hauria de ser emesa més tard d'aquesta.
`revokedCertificates`	Opcional, la llista amb els certificats revocats. Si no hi ha certificats revocats, aleshores la llista no s'inclou. En cas contrari, els certificats revocats s'enumeren en base al seu número de sèrie, i s'especifica la data en la qual han estat revocats.
`crlExtensions`	Opcional, extensions que poden contenir atributs addicionals.

Extensions de les CRLs

El camp d'extensions és un camp opcional que pot aparèixer a partir de la versió 2 i que conté una seqüència d'una o més extensions, que permeten afegir atributs addicionals a les CRLs. De manera anàloga a les extensions dels certificats X.509, cada extensió d'una CRL pot ser marcada com a crítica o com a no crítica. Les aplicacions que no siguin capaces de reconèixer o processar una extensió marcada com a crítica en una CRL no han de fer ús d'aquella CRL. En canvi, si l'extensió està marcada com a no crítica, les aplicacions poden ignorar-la. En els següents paràgrafs, descriurem algunes de les extensions més populars que es fan servir en CRLs a Internet.

De la mateixa manera que als certificats de clau pública, l'extensió d'identificador de la clau de l'autoritat (`AuthorityKeyIdentifier`) és una extensió que permet afegir informació sobre la clau pública corresponent a la clau privada utilitzada per signar la CRL. Aquesta extensió és especialment útil quan un mateix emissor té vàries claus de signatura.

El número de CRL (`CRLNumber`) és una extensió que permet incloure un número de seqüència a la CRL. Donats un emissor de CRL i un abast concret, els números de seqüència són valors estrictament creixents.

L'indicador de delta CRL permet indicar que una CRL és una delta CRL. Per fer-ho, l'extensió inclou el número de CRL de la CRL base (`BaseCRLNumber`) que va ser utilitzada per crear aquesta delta CRL.

El codi de motiu (`CRLReason`) és una extensió que permet identificar el motiu per el qual s'ha revocat el certificat. La Taula 7.6 mostra els codis reconeguts. Exceptuant el codi `removeFromCRL`, la resta de codis indiquen que el certificat ha estat revocat.

Taula 7.6: Codis per especificar el motiu de revocació de certificats digitals.

Codi	Descripció
`unspecified`	Es pot fer servir per revocar certificats per algun motiu diferent als que tenen un codi específic.
`keyCompromise`	Es fa servir per revocar un certificat d'una entitat final i indica que la clau privada del titular del certificat ha estat compromesa.
`cACompromise`	Es fa servir per revocar certificats d'autoritats de certificació i indica que la clau privada de la CA ha estat compromesa.
`affiliationChanged`	Indica que el nom del titular o bé alguna altra informació del certificat ha estat modificada (però que no hi ha motiu per sospitar que la clau privada ha estat compromesa).
`superseded`	Indica que el certificat ha estat substituït (però que no hi ha motiu per sospitar que la clau privada ha estat compromesa).
`cessationOfOperation`	Indica que el certificat ja no és necessari per a l'objectiu pel qual va ser emès (però que no hi ha motiu per sospitar que la clau privada ha estat compromesa).
`certificateHold`	Indica que el certificat es troba suspès temporalment.
`removeFromCRL`	Aquest codi només pot aparèixer en delta CRLs i permet indicar que el certificat s'ha d'eliminar de la CRL, ja sigui perquè ha expirat o perquè ja no es troba suspès.
`privilegeWithdrawn`	Indica que el certificat (de clau pública o d'atributs) es troba revocat perquè un privilegi contingut al certificat ha estat retirat.
`aACompromise`	Es fa servir per revocar certificats d'autoritats d'atributs i indica que la clau privada de l'autoritat d'atributs ha estat compromesa.

Altres models d'emissió de delta CRLs

A l'exemple es mostra el model d'emissió de delta CRLs *tradicional*. Existeixen altres models, com ara el model de finestres lliscants (en anglès, *sliding windows*) que permeten estalviar més ample de banda en alguns escenaris.

Exemple 7.10 Exemple d'ús tradicional de CRLs

La següent taula es mostra un exemple de la manera tradicional d'emetre delta CRLs. En aquest exemple, s'emeten CRLs completes cada dues hores i delta CRLs cada 30 minuts. L'abast de la CRL comprèn tots els certificats emesos per l'autoritat de certificació C per qualsevol motiu excepte `superseded`.

La notació C_x indica el certificat emès per l'autoritat C amb número de sèrie x. Es fa servir l'hora amb precisió de minuts per indicar el temps, però en un cas real es faria servir la data i l'hora completa. A la llista de certificats revocats s'inclou el número de sèrie del certificat i el motiu de la revocació. S'omet la data en la qual ha estat revocat el certificat per simplificar-ne la visualització.

El certificat C_1 s'ha revocat per compromís de la clau privada en algun instant anterior a les `8:00`. Quan s'emet la CRL completa de les `8:00`, C_1 hi apareix amb motiu `keyCompromise`. Les delta CRL que tenen com a base la CRL 1 no inclouen aquest certificat, ja que ja apareix a la CRL base.

En algun moment entre les `8:30` i les `9:00`, C_{33} es revoca amb motiu `privilegeWithdrawn`, i apareix per tant a la primera delta CRL que es publica després de la revocació, la CRL 3. Entre les `9:00` i les `9:30` l'estat de revocació de C_{33} canvia a `affiliationChanged`, i aquest canvi es veu reflectit a la següent delta CRL que es publica, la CRL 4. En el mateix interval de temps el certificat C_{25} es revoca amb motiu `superseded`, però aquest certificat no apareix en cap de les CRLs ja que està fora de l'abast definit.

Entre les `9:30` i les `10:00` el certificat C_{22} es posa en espera. Aquest canvi es veu reflectit tant a la CRL completa com a la delta CRL que es publica a les `10:00`.

Entre les `10:00` i les `10:30` C_{55} és revocat per compromís de la clau privada, i apareix per tant a la delta CRL de les `10:30`. Tot i que el certificat expira a les `10:45`, C_{55} segueix apareixent a les delta CRL fins que apareix en una CRL completa (la CRL 9).

En l'interval de temps entre les `10:30` i les `11:00`, s'aixeca la suspensió sobre el certificat C_{22}. Per tant, a partir de la CRL 7, les delta CRL indiquen que es pot eliminar el certificat de la CRL amb el codi `removeFromCRL`. A partir de la CRL 10 aquesta notificació ja no s'inclou en les delta CRLs, ja que ja s'ha publicat una CRL completa on no hi apareix C_{22} (la CRL 9). És important notar que la CRL no inclou el certificat C_{22} amb codi `removeFromCRL` sinó que simplement no s'inclou el certificat a la llista de certificats revocats. El codi `removeFromCRL` no apareix en CRLs completes.

Temps actual t	Estat dels certificats en l'instant t	Contingut de la CRL completa	Contingut de la Delta CRL
8:00	$\{C_1 : keyCompromise\}$	CRLNumber : 1 thisUpdate: 8:00 nextUpdate: 10:00 revokedCertificates: $\{C_1 : keyCompromise\}$	CRLNumber : 1 thisUpdate: 8:00 nextUpdate: 8:30 BaseCRLNumber: 1 revokedCertificates: $\{\}$
8:30	$\{C_1 : keyCompromise\}$		CRLNumber : 2 thisUpdate: 8:30 nextUpdate: 9:00 BaseCRLNumber: 1 revokedCertificates: $\{\}$
9:00	$\{C_1 : keyCompromise,$ $C_{33} : privilegeWithdrawn\}$		CRLNumber : 3 thisUpdate: 9:00 nextUpdate: 9:30 BaseCRLNumber: 1 revokedCertificates: $\{C_{33} : privilegeWithdrawn\}$
9:30	$\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{25} : superseded\}$		CRLNumber : 4 thisUpdate: 9:30 nextUpdate: 10:00 BaseCRLNumber: 1 revokedCertificates: $\{C_{33} : affiliationChanged\}$
10:00	$\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{25} : superseded,$ $C_{22} : certificateHold\}$	CRLNumber : 5 thisUpdate: 10:00 nextUpdate: 12:00 revokedCertificates: $\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{22} : certificateHold\}$	CRLNumber : 5 thisUpdate: 10:00 nextUpdate: 10:30 BaseCRLNumber: 1 revokedCertificates: $\{C_{33} : affiliationChanged,$ $C_{22} : certificateHold\}$
10:30	$\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{25} : superseded,$ $C_{22} : certificateHold,$ $C_{55} : keyCompromise\}$		CRLNumber : 6 thisUpdate: 10:30 nextUpdate: 11:00 BaseCRLNumber: 5 revokedCertificates: $\{C_{55} : keyCompromise\}$
11:00	$\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{25} : superseded\}$ C_{55} expired on $t =$ 10:45		CRLNumber : 7 thisUpdate: 11:00 nextUpdate: 11:30 BaseCRLNumber: 5 revokedCertificates: $\{C_{55} : keyCompromise,$ $C_{22} : removeFromCRL\}$
11:30	$\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{25} : superseded\}$		CRLNumber : 8 thisUpdate: 11:30 nextUpdate: 12:00 BaseCRLNumber: 5 revokedCertificates: $\{C_{55} : keyCompromise,$ $C_{22} : removeFromCRL\}$
12:00	$\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{25} : superseded\}$	CRLNumber : 9 thisUpdate: 12:00 nextUpdate: 14:00 revokedCertificates: $\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{55} : keyCompromise\}$	CRLNumber : 9 thisUpdate: 12:00 nextUpdate: 12:30 BaseCRLNumber: 5 revokedCertificates: $\{C_{55} : keyCompromise,$ $C_{22} : removeFromCRL\}$
12:30	$\{C_1 : keyCompromise,$ $C_{33} : affiliationChanged,$ $C_{25} : superseded\}$		CRLNumber : 10 thisUpdate: 12:30 nextUpdate: 13:00 BaseCRLNumber: 9 revokedCertificates: $\{\}$

7.3.3 Online Certificate Status Protocol

El protocol OCSP (de l'anglès, *Online Certificate Status Protocol*) permet determinar l'estat de revocació actual d'un certificat digital a través d'un protocol interactiu. L'OCSP es pot fer servir ja sigui com a substitut o com a complement de les CRLs.

En el protocol hi participen dues parts, el client OSCP (que està interessat en saber l'estat d'un certificat digital) i el servidor OCSP (que respondrà a les consultes del client).

Quan un client necessita validar l'estat d'un certificat digital, aleshores envia una petició al servidor OCSP i suspèn l'acceptació del certificat fins que arriba la resposta. Una petició OCSP conté els següents camps:

- Versió del protocol.
- Identificador del certificat o certificats per als quals es demana l'estat de revocació.
- Opcionalment, extensions.

Una resposta OCSP (de tipus bàsic) conté la següent informació:

- Versió de la sintaxi de la resposta.
- Instant de temps en què s'ha generat la resposta.
- Conjunt de respostes (per cada un dels certificats que s'han demanat a la petició).
- Identificador de l'algorisme de signatura.
- Valor de la signatura.

Cada una de les respostes conté alhora quatre camps: l'identificador del certificat a la qual fa referència, l'estat del certificat, l'interval de validesa de la resposta i, opcionalment, les extensions.

Implementació de l'OCSP

L'Openssl inclou l'eina `ocsp` que incorpora funcionalitats per interactuar amb un servidor OCSP i, fins i tot, per operar com un petit servidor OCSP.

L'especificació d'OCSP de la RFC6960 defineix tres alternatives per indicar l'estat del certificat: bo (`good`), revocat (`revoked`) o desconegut (`unknown`). Cal anar amb compte, però, a l'hora d'interpretar aquestes respostes, ja que els noms utilitzats per designar-les poden generar confusió sobre els detalls del seu significat.

Una resposta bona indica que no hi ha cap certificat amb el número de sèrie especificat a la petició que es trobi revocat (i que estigui dins del seu periode de validesa). Això no implica necessàriament que el certificat existeixi (podria ser que mai hagués estat emès) ni que el certificat sigui vàlid en aquell moment (podria ser que la resposta del servidor OCSP no estigui compresa en l'interval de validesa del certificat). Una resposta de certificat revocat indica que el certificat ha estat revocat, ja sigui temporalment (especificant el codi de motiu `certificateHold`) o bé permanentment. Tot i això, un servidor OCSP també pot retornar aquesta resposta si la CA corresponent no ha emès mai un certificat amb aquest número de sèrie. Una resposta d'estat desconegut indica que el servidor no coneix el certificat.

Exercici 7.4 Indiqueu quines de les següents afirmacions són certes:

1. Per tal de comprovar la data de caducitat d'un certificat digital, podem utilitzar el protocol OCSP.
2. Per tal de comprovar la data de caducitat d'un certificat digital, podem utilitzar CRLs.
3. Per tal de comprovar la data de caducitat d'un certificat digital, només ens cal el propi certificat.
4. Una resposta OCSP `good` sempre indica que el certificat no està revocat.
5. Una resposta OCSP `revoked` sempre indica que el certificat està revocat.

7.3.4 Time Stamp Protocol

Com hem vist, les autoritats de segellat de temps o TSA són les autoritats d'una PKI encarregades de crear segells de temps. Per fer-ho, la TSA signa els segells que emet amb una clau específicament reservada per a aquest propòsit. En concret, el certificat digital corresponent ha de tenir una única instància del camp extended key usage amb keyPurposeId `id-kp-timeStamping` i l'extensió marcada com a crítica.

El TSP (Time-Stamp Protocol) és el protocol que es fa servir per interactuar amb una TSA. El procediment per aconseguir un segell de temps és el següent. En primer lloc, el sol·licitant envia una petició de segell de temps a la TSA. Després, la TSA respon a la petició amb un missatge de resposta. Per acabar, el sol·licitant hauria de comprovar l'estat d'error de la resposta. Si no hi ha errors, caldria validar el segell de temps retornat.

Nonce

Un nonce (de l'anglès, *number used once*) és un nombre arbitrari que es fa servir una única vegada en un protocol criptogràfic.

La petició de segell de temps que el sol·licitant envia a la TSA conté, entre d'altres:

- El hash de la dada que es vol segellar.
- L'identificador de l'algorisme de hash utilitzat.
- Opcionalment, un nonce. El nonce és un valor aleatori que té una probabilitat alta de ser generat una única vegada pel client. El nonce és opcional però, si s'inclou, la resposta de la TSA ha de contenir aquest mateix valor. El nonce permet detectar el reenviament d'un segell de temps, ja sigui realitzat de manera involuntària per errors en la transmissió o bé com a conseqüència d'un atac.
- Opcionalment, un booleà `certReq` que permet indicar que es desitja que la TSA inclogui el certificat corresponent a la signatura en la resposta. Si no s'inclou el camp o aquest és `False`, aleshores la resposta de la TSA no ha de contenir el certificat.

La resposta de la TSA conté l'estat de la petició i, en alguns casos, el segell de temps demanat.

L'estat està format per tres camps: un codi que l'identifica i que sempre es troba present i, opcionalment, un text i una explicació del motiu per el qual la petició ha fallat (si n'és el cas). Els codis d'estat reconeguts són:

- *concedit*: el segell de temps demanat s'adjunta a la resposta.
- *concedit amb modificacions*: s'adjunta un segell de temps, però aquest conté alguna modificació.
- *rebutjat*: es rebutja la petició de segell de temps.
- *en espera*: la creació del segell de temps es troba en espera.
- *en alerta per revocació*: el missatge conté un avís que la revocació és imminent.
- *notificació de revocació*: s'ha revocat el certificat.

La resposta ha de contenir el segell de temps només si l'estat és concedit o concedit amb modificacions. En cas contrari, caldrà indicar el motiu de la fallada. Els motius de fallada poden ser diversos, com ara per exemple, que l'algorisme de hash indicat no es reconegui, que el format de la petició sigui incorrecte, o que la font utilitzada per la TSA per aconseguir el temps no estigui disponible en aquell moment.

Sintaxi dels segells de temps

L'explicació sobre el contingut dels segells de temps abstrau els detalls reals de la sintaxis d'aquests. Els segells de temps són informació signada, que segueix la sintaxi CMS (Cryptographic Message Syntax).

El camp de temps

La codificació del temps en una resposta de la TSA sempre acaba en Z, el que indica que representa el temps Zulu. El temps Zulu és un sinònim del Temps Universal Coordinat que es fa servir en aviació civil.

Per la seva banda, el segell de temps estarà format per la signatura de la TSA (a la que s'adjuntarà l'identificador del certificat utilitzat per realitzar-la) i el contingut del segell de temps, que tindrà, entre d'altres, els següents camps:

- El hash de la dada que s'ha rebut.
- L'identificador de l'algorisme de hash utilitzat.
- Un número de sèrie, que serà assignat per la TSA a cada segell de temps i que serà únic entre els segells emesos per aquesta TSA.
- El temps, indicant el moment en què la TSA ha generat el timestamp expressat en UTC (Temps Universal Coordinat).
- Opcionalment, la precisió, que ens permet determinar l'interval de temps exacte en el qual s'ha creat el segell de temps. Si el camp no s'inclou, la precisió es pot anunciar d'altres maneres, com ara a partir de la política de la TSA.
- Opcionalment, el nonce. Si la petició contenia nonce, aleshores la resposta cal que també la contingui i que el valor sigui igual que el de la petició.
- Opcionalment, el nom de la TSA, que permet ajudar en la identificació de la TSA.

Implementació del TSP

L'Openssl inclou l'eina `ts` que incorpora les funcions bàsiques de client i servidor de TSA.

Finalment, quan es rep la resposta de la TSA, caldria validar-la. Per fer-ho, en primer lloc es comprova l'estat d'error i, si no s'ha produït cap error, aleshores es valida el segell de temps: es comproven els camps, es valida la signatura digital, es comprova que el segell de temps correspongui al que es va demanar (revisant tant el valor del hash com l'identificador de l'algorisme fet servir), l'estat del certificat de la TSA i el temps. El temps pot ser validat a partir de la referència d'un servei de temps local de confiança, o bé comprovant que el nonce inclòs en la petició es troba també en la resposta.

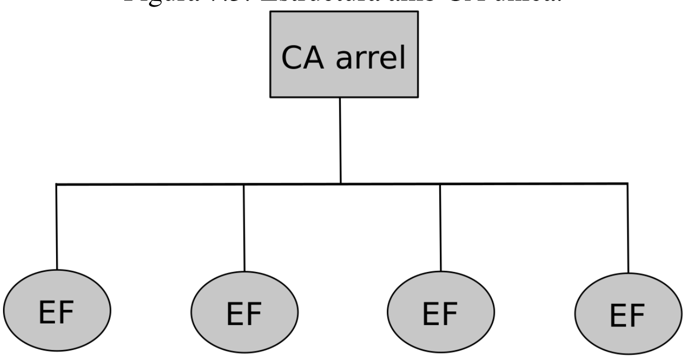
Exercici 7.5 Un ciutadà està recollint signatures digitals per a recolzar una proposta. Per tal que les signatures digitals siguin considerades vàlides, cal que estiguin realitzades en un interval de temps concret (t_{inici}, t_{fi}), és a dir, que s'hagin més tard de t_{inici} i abans de t_{fi}. Quin d'aquests segells de temps seria vàlid per demostrar que el conjunt de signatures s'han creat en el període establert?

1. Un segell de temps sobre totes les signatures realitzat en $t_s < t_{inici}$
2. Un segell de temps sobre totes les signatures realitzat en $t_s > t_{fi}$
3. Un segell de temps sobre totes les signatures realitzat en $t_{inici} < t_s < t_{fi}$

7.3.5 Estructures de PKI

Potser el model d'estructura de PKI més simple és el model de **CA única**. En aquest model, hi ha una única autoritat de certificació, que emet certificats per a totes les entitats finals que participen de la PKI. La Figura 7.3 mostra un exemple d'aquest model.

Figura 7.3: Estructura amb CA única.

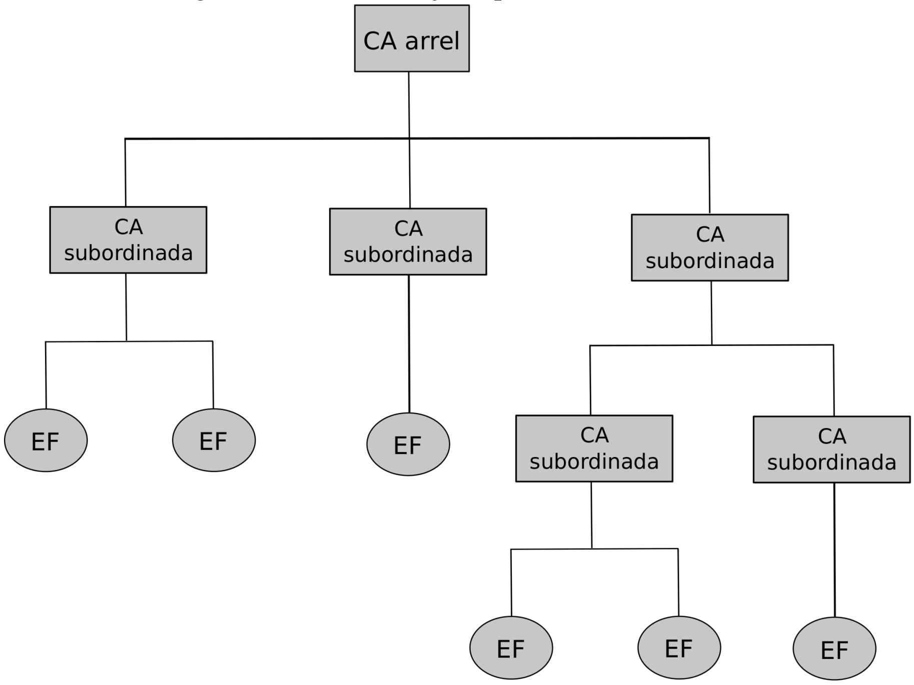


Una PKI amb un model **jeràrquic amb arrel única** té una una única CA arrel, però pot tenir altres CAs. Les diferents CAs es troben estructurades seguint una jerarquia, on la CA arrel certifica a un conjunt d'autoritats de certificació, que alhora poden crear certificats per a altres CAs (o per a entitats finals), creant els diferents nivells de la jerarquia. Les CAs intermèdies també són conegudes amb el nom d'autoritats subordinades. El model de CA única pot ser vist com un cas específic del model jeràrquic, on la única CA existent és la CA arrel. La Figura 7.4 mostra un exemple d'aquest model.

Una variant d'aquesta estructura és el model **jeràrquic amb llista de confiança**, que disposa d'una llista amb vàries CAs arrel. Cadascuna de les CAs arrel pot tenir autoritats subordinades seguint una jerarquia. L'ús més conegut d'aquest model és en els navegadors web, que contenen una llista amb uns centenars de CAs arrel. La Figura 7.5 mostra un exemple d'aquest model.

Existeixen altres estructures de PKI, com ara l'estructura de **malla**, on les autoritats de certificació emeten certificats creuats, o la interconnexió a través de **bridge** CAs, on una autoritat fa de nexe entre les diferents PKIs.

Figura 7.4: Estructura jeràrquica amb arrel única.



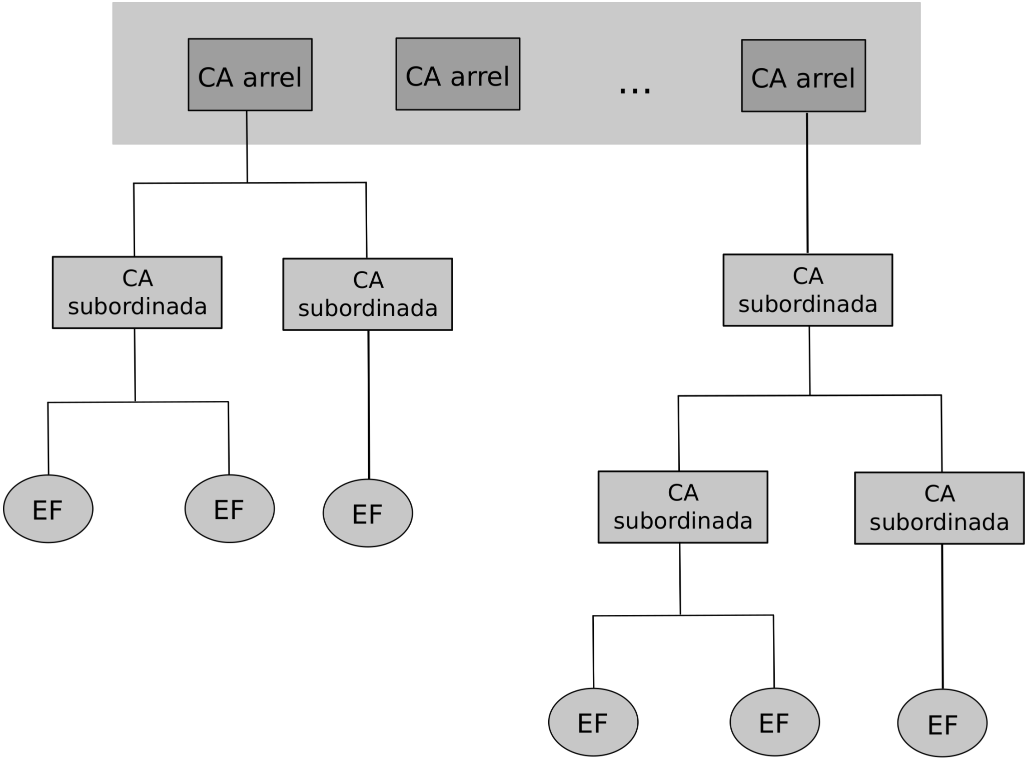
7.4 Les normes PKCS

Les normes PKCS (de l'anglès, *Public Key Cryptography Standards*) són un conjunt d'especificacions sobre criptografia de clau pública. Les especificacions les publica l'empresa RSA Laboratories i són elaborades conjuntament amb altres empreses del sector, com ara Apple, Microsoft, DEC, Lotus, Sun i MIT. Algunes d'aquestes especificacions han acabat esdevenint estàndards d'organismes internacionals com ara la IETF. L'objectiu d'aquestes publicacions és fomentar l'ús de la criptografia de clau pública i accelerar-ne el seu desplegament.

Actualment, hi ha 10 normes PKCS. Addicionalment, els PKCS#13 i #14, que cobreixen el xifrat i signatura fent servir criptografia de corbes el·líptiques i la generació de nombres pseudo-aleatoris, respectivament, no estan encara publicats. Els PKCS#2 i #4 es troben retirats des de 2010, quan van ser incorporats al PKCS#1 (tots dos descrivien també detalls sobre l'ús de l'RSA). El PKCS#6 descrivia la versió 1 dels certificats X.509 i està sent eliminat en favor de la versió 3 de X.509.

- PKCS#1: defineix mecanismes per xifrar i signar dades fent servir el criptosistema de clau pública RSA.
- PKCS#3: defineix el protocol d'establiment de claus de Diffie-Hellman.
- PKCS#5: descriu un mètode per xifrar una cadena amb una clau secreta derivada d'una contrasenya.
- PKCS#7: defineix una sintaxis general per missatges que inclouen atributs criptogràfics com ara signatures digitals o xifrat.
- PKCS#8: descriu un format per informació sobre claus privades, que inclou la clau privada per alguns algorismes de clau pública i, opcionalment, un conjunt d'atributs.
- PKCS#9: defineix tipus d'atributs per fer servir en altres estàndards PKCS.
- PKCS#10: descriu una sintaxi per peticions de certificació.
- PKCS#11: defineix una interfície de programació anomenada Cryptoki per a dispositius criptogràfics com targetes intel·ligents.
- PKCS#12 especifica un format portable per a emmagatzemar o transportar claus privades d'usuari, certificats, secrets, etc.
- PKCS#15 és un complement al PKCS#11 proveint un estàndard per credencials criptogràfiques emmagatzemades en tokens criptogràfics.

Figura 7.5: Estructura jeràrquica amb llista de confiança.



En aquest capítol, es detallen tres d'aquestes especificacions, els PKCS#1, #5 i #12.

7.4.1 PKCS#1

En el capítol anterior hem vist el funcionament bàsic de l'RSA. A la pràctica, però, no es fa servir l'RSA directament com s'ha descrit al capítol, ja que això resultaria insegur. Dos dels problemes més coneguts que té l'RSA amb la formulació que hem vist al capítol de Criptografia de Clau Pública són els atacs per l'ús d'exponents petits i el determinisme de l'algorisme.

D'una banda, fer servir exponents petits en combinació amb missatges m també petits fa que l'esquema sigui vulnerable. En concret, si $m^e < n$, aleshores podem desxifrar el missatge directament sense necessitat de conèixer la clau privada, calculant l'arrel e-èsima d'm sobre els enters. És a dir, si $c = m^e \mod n$ i $m^e < n$, aleshores $c = m^e$ i per tant $m = \sqrt[e]{c}$.

Exemple 7.11 Exemple d'atac per l'ús d'exponents petits

Suposem que tenim un parell de claus RSA de 64 bits formada pels valors:
`PubK = (e, n) = (3, 18230703860219055503)`
`PrivK = (d, n) = (616012317821603203, 18230703860219055503)`

Si volem xifrar un missatge $m = 55$, procediríem a elevar el missatge a l'exponent públic, com s'ha vist al capítol anterior:
`c` $= m^e \mod n = 55^3 \mod 18230703860219055503 = 166375$

Noteu com $55^3 = 166375$ i, per tant, $55^3 < n$.

Aleshores, un atacant que captura el missatge `c` i que coneix la clau pública, pot procedir a desxifrar el missatge, fent:

m = $c^{1/e} = 166375^{1/3} = 55$

D'altra banda, l'RSA és un algorisme determinista: el resultat de xifrar un text pla m amb una clau pública pub_k és sempre un mateix valor xifrat c. Si repetim el xifrat del mateix missatge amb la mateixa clau, el resultat és sempre el mateix valor c. Això fa que un atacant tingui un cert avantatge a l'hora d'intentar esbrinar el text pla m corresponent a un cert text xifrat c.

Suposem que un missatge m s'ha xifrat amb la clau pública pub_k fent servir RSA, donant com a resultat el valor xifrat c. Un atacant que coneix el valor xifrat c i la clau pública pub_k pot intentar desxifrar el missatge xifrant repetidament diversos valors m_i i anar comprovant si el resultat correspon al valor c que vol desxifrar. Si el conjunt de possibles missatges és petit, aquest atac sempre tindria èxit ja que l'atacant seria capaç de provar tots els possibles textos plans.

Exemple 7.12 Exemple d'atac pel determinisme de l'algorisme

Suposem que un elector que participa en unes eleccions on hi ha 4 candidats ($\{c_2, c_3, c_4, c_5\}$) ha d'enviar el seu vot xifrat amb la clau pública de la mesa electoral, indicant l'identificador del candidat votat.

Suposant que la mesa electoral té el següent parell de claus RSA també de 64 bits (on els exponents no són petits per evitar l'atac de l'exemple anterior):
`PubK = (e, n) = (7387905850005970831, 16517425874601317047)`
`PrivK = (d, n) = (6515200225287789487, 16517425874601317047)`
i que l'elector vol votar al candidat 4, el vot a enviar seria el valor:

vot = $m^e \mod n = 4^{7387905850005970831} \mod 16517425874601317047 =$
8163478232469599798

Un atacant que capturi el vot i vulgui conèixer-ne el seu contingut, només cal que generi ell mateix tots els possibles vots xifrats. Com que la clau pública de la mesa electoral és coneguda per tothom, l'atacant pot aconseguir aquesta informació i calcular:

$v_2 = 2^{7387905850005970831} \mod 16517425874601317047 = 11673347059272354770$
$v_3 = 3^{7387905850005970831} \mod 16517425874601317047 = 10980320764598560840$
$v_4 = 4^{7387905850005970831} \mod 16517425874601317047 = 8163478232469599798$
$v_5 = 5^{7387905850005970831} \mod 16517425874601317047 = 3701559658846578058$

Aleshores, l'atacant descobreix que l'elector ha votat al candidat 4, sense necessitat d'haver de desxifrar el vot.

Addicionalment, l'RSA ofereix xifratge homomòrfic. Donats dos missatges en pla, m_1 i m_2 i els seus corresponents textos xifrats c_1 i c_2 (amb la mateixa clau), el resultat de multiplicar els dos textos xifrats (c_1c_2) és precisament el mateix valor que s'obté al multiplicar els dos textos plans m_1 i m_2 i xifrar-los posteriorment.

En efecte, si tenim que:
$c_1 = E(m_1) = {m_1}^e \mod n$
$c_2 = E(m_2) = {m_2}^e \mod n$
aleshores:
$E(m_1) * E(m_2) = c_1 * c_2 = ({m_1}^e)({m_2}^e) \mod n = (m_1 * m_2)^e \mod n = E(m_1 * m_2)$

La propietat d'homomorfisme de l'RSA pot ser utilitzada per construir esquemes amb propietats interessants, però, d'altra banda, també pot suposar problemes de seguretat en segons quins escenaris. A partir d'un missatge xifrat $c_1 = E(m_1)$, un atacant podrà construir un segon missatge xifrat $c_2 = c_1 * E(\alpha)$ que correspondrà al missatge en pla $m_1 * \alpha$ sense conèixer la clau de xifratge ni el missatge en pla original m_1.

Així doncs, mentre que la propietat d'homomorfisme és útil per construir esquemes que manipulin dades mentre en preserven la seva confidencialitat, també pot ser una debilitat en segons quins desplegaments i depenent de l'ús que se'n faci.

Per tal de solucionar aquests problemes que apareixen amb la utilització de l'RSA en la seva definició bàsica, s'acostuma a afegir un conjunt de bits aleatoris com a *padding* al missatge en pla abans de xifrar-lo. D'aquesta manera, s'aconsegueix que el mateix missatge m pugui correspondre a diversos textos xifrats c, s'eviten els missatges m vulnerables per la seva representació i, alhora, es pot eliminar la propietat d'homomorfisme que presenta l'RSA.

Noteu que amb la introducció de bits aleatoris, l'RSA passa a ser un criptosistema probabilístic, on un mateix missatge xifrat amb una mateixa clau pot donar lloc a diversos textos xifrats. En canvi, el criptosistema d'ElGamal ja és probabilístic per definició.

L'estàndard PKCS#1 defineix tot un conjunt de recomanacions per implementar l'RSA. En concret, l'estàndard descriu primitives criptogràfiques, esquemes de xifrat, esquemes de signatura i detalls de codificació.

La versió 2.1 de l'estàndard PKCS#1 va ser republicada com a RFC 3447, amb unes petites correccions.

Esquemes de xifrat

Seguint la definició de l'estàndard, un esquema de xifrat consisteix en una operació de xifrat i una operació de desxifrat.

El PKCS#1 defineix dos esquemes de xifrat: RSAES-OAEP i RSAES-PKCS1-v1_5. En aquest capítol, veurem la descripció de l'esquema RSAES-OAEP. L'esquema RSAES-PKCS1-v1_5 s'inclou només per mantenir la compatibilitat amb les aplicacions ja existents, ja que actualment es coneixen atacs que fan que el seu ús no sigui recomanable. Abans però de descriure l'RSAES-OAEP, definirem i veurem un exemple de funció de generació de màscara, una construcció que es fa servir en RSAES-OAEP.

Funcions de generació de màscara

RSAES-OAEP Tot i incloure les sigles AES, aquest esquema no te res a veure amb el criptosistema de bloc.

L'esquema de xifrat RSAES-OAEP fa ús d'una funció de generació de màscara en dues ocasions per tal de generar el valor EM a xifrar.

> Una **funció de generació de màscara** (MGF per les seves sigles en anglès, *Mask Generation Function*) és una funció que rep com a paràmetres una cadena de mida variable i la mida de sortida desitjada i retorna una cadena de la mida especificada a l'entrada.

És a dir, una funció MGF és una funció que rep una entrada de mida m_i bits i un valor de mida de sortida m_o i retorna una sortida de mida m_o bits:

$$MGF(\{0,1\}^{m_i}, m_o) \rightarrow \{0,1\}^{m_o}$$

Les funcions de generació de màscara són deterministes, ja que la sortida de la funció queda determinada de manera única per la seva entrada. A més, la seva sortida ha de ser pseudoaleatòria, de manera que coneixent una part de la sortida no se'n pugui generar la resta.

L'MGF1 és una funció de generació de màscara basada en una funció hash. Donada una funció hash H amb sortida de mida $hLen$, l'MGF1 es defineix de la següent manera:

```
definició mgf1(seed, maskLen)
    si maskLen > 2^32*hLen aleshores
        retornar "Error: màscara massa llarga"
    fi si
    T = ""
    per iteracio = 0 fins a ceil(maskLen/hLen) - 1 fes
     comptador =  I2OSP(iteracio, 4)
        T = T || H( seed || comptador )
    fi per
    retornar maskLen octets més significatius de T
fi definició
```

La funció I2OSP	La funció `I2OSP(x,y)` retorna el valor x representat fent servir y octets.

És a dir, la funció va concatenant el resultat d'aplicar una funció hash a la concatenació del valor que rep com a llavor (`seed`) i un comptador d'iteracions que es va incrementant d'un en un. Aquest procés es repeteix fins a tenir prou octets com per generar una sortida de la mida especificada com a paràmetre (`maskLen`). Finalment, com que `maskLen` pot no ser múltiple de la mida del hash (`hLen`), es possible que s'hagin de descartar els octets menys significatius del resultat acumulat. A l'hora de concatenar el comptador amb la llavor, es concatena la seva representació en quatre octets.

La funció ceil	La funció `ceil(x)` retorna el menor enter major o igual que x.

Exemple 7.13 Exemple d'ús de l'MGF1 Suposem que volem fer servir l'MGF1 amb la funció hash sha-1 amb els següents valors d'entrada:
`maskLen = 45`
`seed = 0x5307`

La funció sha-1 produeix una sortida de 160 bits, és a dir, 20 octets. Per tant,
`hLen = 20`

En primer lloc, es comprova que la mida de sortida desitjada no sigui superior a $2^{32}hLen$. Com que no ho és ($maskLen < 2^{32}hLen$), es continua l'execució normalment, calculant el valor màxim que assumirà el comptador en el bucle:

`ceil(maskLen/hLen) - 1 = ceil(45/20) - 1 = 3 - 1 = 2`

Després, es procedeix a calcular el valor `T` a cada una de les iteracions del bucle:

Iteració 0:
`comptador = 0x00000000`
`seed` ‖ `comptador = 0x530700000000`
`H(seed` ‖ `comptador) = 0xc5230b3bcb615dbc0b9a63f6e975b0f327fc576c`
`T` ‖ `H( seed || comptador ) = 0xc5230b3bcb615dbc0b9a63f6e975b0f327fc576c`

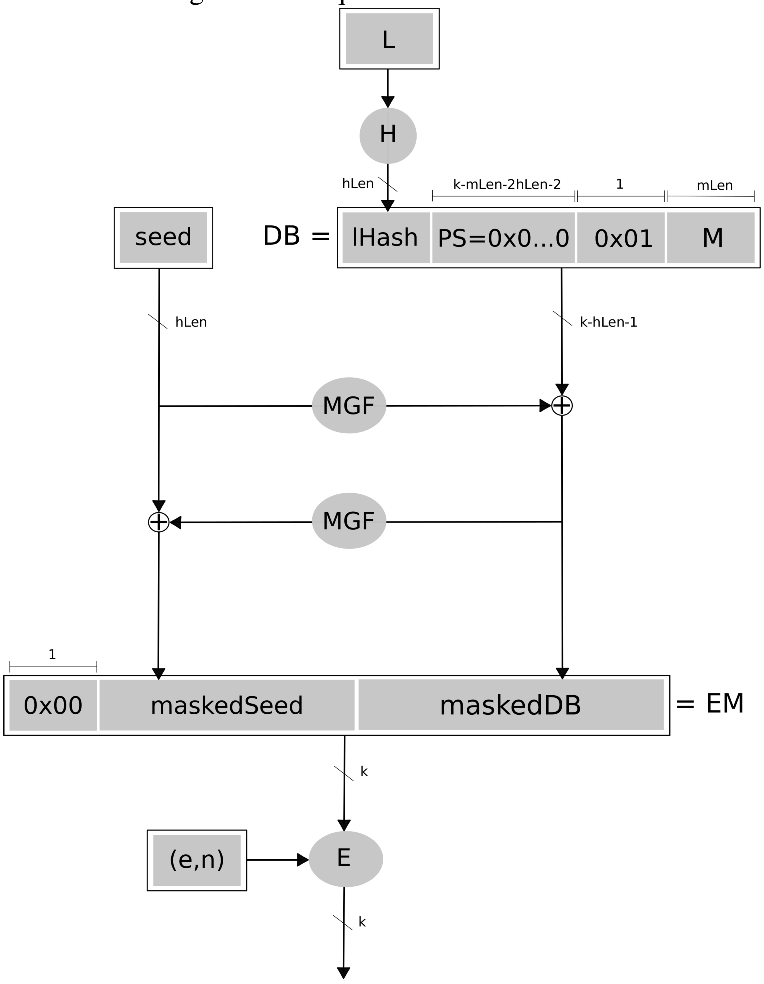
Iteració 1:
`comptador = 0x00000001`
`seed` ‖ `comptador = 0x530700000001`
`H(seed` ‖ `comptador) = 0x12f189b1d17e5b688d856fc700ffd2b20aabb14e`
`T` ‖ `H( seed || comptador ) = 0xc5230b3bcb615dbc0b9a63f6e975b0f327fc57`
`6c12f189b1d17e5b688d856fc700ffd2b20aabb14e`

Iteració 2:
```
comptador = 0x00000002
seed ‖ comptador = 0x530700000002
H(seed ‖ comptador) = 0x62f2e51fe23fdbfc3d200346f30157d5985d24ef
T ‖ H( seed || comptador ) = 0xc5230b3bcb615dbc0b9a63f6e975b0f327fc57
      6c12f189b1d17e5b688d856fc700ffd2b20aabb14e62f2e51fe23fdbfc3d2003
      46f30157d5985d24ef
```

Finalment, es retornen els 45 octets més significatius, de l'últim valor `T` calculat:
```
c5230b3bcb615dbc0b9a63f6e975b0f327fc576c12f189b1d17e5b688d856fc700ffd
      2b20aabb14e62f2e51fe2
```

L'esquema de xifrat RSAES-OAEP

Figura 7.6: Esquema de l'RSA-OAEP.



L'esquema de xifrat RSAES-OAEP defineix com xifrar un missatge M amb RSA afegint *padding* i introduint aleatorietat. L'esquema rep com a entrades tres valors, i retorna un valor xifrat:

`C = RSAES-OAEP(M, (e,n), L)`

on `M` és el missatge a xifrar; `(e,n)` és la clau pública RSA; i `L` és una etiqueta opcional (si no s'inclou, es pren com a valor la cadena buida).

A més de la funció de xifratge amb RSA que hem vist al capítol anterior, l'esquema també fa ús de dues funcions addicionals: `H`, una funció hash i `MGF`, una funció de generació de màscara

Octet	Un **octet** són 8 bits. En alguns estàndards es fa servir la paraula octet en comptes de byte per definir conjunts de 8 bits ja que, en realitat, la mida d'un byte depèn de la plataforma. Tot i que actualment gairebé tots els sistemes interpreten un byte com a 8 bits, en certs contextos el seu ús podria provocar ambigüitats.

Per descriure l'esquema, farem servir els termes `mLen` i `hLen` per referir-nos, respectivament, a la mida en octets del missatge `m` i de la sortida de la funció hash `H`. A més, el valor `k` representa la mida (també en octets) del capítol de la clau RSA utilitzada.

La Figura 7.6 mostra l'esquema de xifratge de l'RSA-OEAP. Com es pot apreciar, el primer pas de l'esquema és construir el *padding* per al missatge `M`. Això es fa concatenant quatre valors: `lHash`, el resultat d'aplicar la funció hash a l'etiqueta `L` (`lHash = H(L)`); `PS`, una cadena d'octets fixats a zero; un octet representant el valor `1`; i `M`, el missatge a xifrar. La longitud de la cadena `PS` és variable, i dependrà de la mida del missatge i de la sortida de la funció hash. De fet, es podria donar el cas que la cadena `PS` tingués longitud zero.

Així doncs,

`DB = lHash` ‖ `PS` ‖ `0x01` ‖ `M`

En segon lloc, es genera una llavor aleatòria de mida *hLen*, representada com a *seed* a l'esquema.

En els següents passos hi entren en joc, d'una banda, la funció xor i, d'altra banda, la funció de generació de màscara *MGF* triada. Utilitzant aquestes dues funcions es calculen els valors *maskedDB* i *maskedSeed*:

```
maskedDB = DB ⊕ MGF(seed, k-hLen-1)
maskedSeed = seed ⊕ MGF(DB, hLen)
```

i es calcula el valor *EM* com la concatenació d'un octet de zeros amb *maskedSeed* i *maskedDB*:

`EM = 0x00` ‖ `maskedSeed` ‖ `maskedDB`

Aquest valor `EM` té ara exactament `k` octets, i és el que es farà servir com a entrada de la funció de xifrat de l'RSA que hem vist al capítol anterior. Noteu que fent servir l'esquema RSAES-OAEP per xifrar se solucionen els problemes de l'RSA comentats anteriorment ja que, d'una banda, els valors a xifrar tenen ara sempre `k` octets i, d'altra banda, el xifrat deixa de ser determinista amb la inclusió del valor aleatori `seed`.

Per desxifrar un valor `C` xifrat amb RSAES-OAEP, es procedeix a desfer el camí realitzat a l'hora de xifrar-lo: en primer lloc, es desxifra el valor rebut fent servir la funció de desxifrat de l'RSA vista al capítol anterior. En segon lloc, assignarem el resultat del desxifrat a `EM`, i desfarem els passos realitzats a l'hora de xifrar fins a obtenir el valor original del missatge `M`.

Exemple 7.14 Exemple de xifratge fent servir RSAES-OAEP

En aquest exemple farem servir RSAES-OAEP per xifrar un missatge amb una clau de 512 bits i fent servir la funció sha1 com a funció hash. La clau pública que farem servir és:

```
n = 1217452315637823511017967267960235756306999862745378358866566861019 63
      66749040999892914856387609002413925126564321873531644179310315470260
      085900075427092633
e = 65537
```

El missatge a xifrar serà:
`message = 0x484f4c41`

Deixarem l'etiqueta buida (`L=''`) i farem servir com a llavor el valor:
`seed = 0000000000000000000000000000000000051ead`

Així doncs, la clau pública a utilitzar té 512 bits de manera que $k = 64$ octets. Alhora, la funció sha1 té una sortida de 160 bits, pel que `hlen = 20`. El missatge a xifrar té mida `mLen = 4` octets. Per tant, la mida de PS serà:
`k-mLen-2hLen-2 = 64-4-2*20-2 = 18` octets.

Sabent la mida de PS, podem calcular el valor DB:

```
lHash = H(‘’) = da39a3ee5e6b4b0d3255bfef95601890afd80709
DB = lHash || PS || 0x01 || M = da39a3ee5e6b4b0d3255bfef95601890afd807090000
      0000000000000000000000000000000001484f4c41
```

Ara, procedim a aplicar la funció de generació de màscara:

```
dbMask = MGF(seed, k-hLen-1) = MGF(seed, 43) = e68e82475a8216e69b2ac31ab2
      0b60a168c155da68b1e64c8972335f90ebd96d57905b0ffe49ae92b14f3f
maskedDB = DB ⊕ dbMask =
      da39a3ee5e6b4b0d3255bfef95601890afd80709000...001484f4c41 ⊕
      e68e82475a8216e69b2ac31ab20b60a168c155da68b...9ae92b14f3f =
      3cb721a904e95deba97f7cf5276b7831c71952d368b1e64c8972335f90ebd96d5790
      5b0ffe49afdafe037e
```

```
seedMask = MGF(maskedDB, hLen) = MGF(maskedDB, 20) =
      9e883239d0bc279884730611a7f07b5e65f17474
maskedSeed = seed ⊕ seedMask =
      0000000000000000000000000000000000051ead ⊕
      9e883239d0bc279884730611a7f07b5e65f17474 =
      9e883239d0bc279884730611a7f07b5e65f46ad9
```

Per tant, el valor que xifrarem amb la primitiva bàsica de xifratge RSA serà:

```
EM = 0x00 || maskedSeed || maskedDB = 009e883239d0bc279884730611a7f07b5e65f
      46ad93cb721a904e95deba97f7cf5276b7831c71952d368b1e64c8972335f90ebd96
      d57905b0ffe49afdafe037e
```

Per acabar, es procedeix a realitzar el xifratge. Com que tenim la clau pública expressada en enters en base decimal, una alternativa per a realitzar el càlcul és convertir prèviament la representació d'EM a base 10:

$\mathtt{c} = E_{(e,n)}(\mathtt{EM}) = m^e \mod n = 5362170382089168551579189988553345095874132024139077634594003538315944764911270805494744162555266564406433789864671246348638999851571397862373399515115506$

```
c = E(e,n)(EM) = m^e mod n = 536217038208916855157918998855334509587413202 41
      39077634594003538315944764911270805494744162555266564406433789864671
      246348638999851571397862373399515115506
```

Finalment, convertim de nou el resultat del xifratge a la representació hexadecimal de k octets:

```
C = 6661be2c1357ba21cbc074362ae9e8e08898779e7df987d4ae2993e1e02ab051cf9f7
      a109fc3389c96779e6206c49e695da846efa3945ba0c9e43adaddcc2ff2
```

7.4.2 PKCS#5

La norma PKCS#5 proveeix recomanacions per a la implementació de criptografia basada en contrasenyes. En concret, la norma descriu funcions de derivació de claus, esquemes de xifrat, esquemes d'autenticació de missatges i la sintaxi ASN.1 que identifica les diferents tècniques.

La versió 2.0 de l'estàndard PKCS#5 va ser republicada com a RFC 2898.

Sal criptogràfica i número d'iteracions

Avui en dia, l'ús de contrasenyes és un mètode molt freqüent per protegir accessos a sistemes o secrets. Les contrasenyes escollides pels usuaris, però, acostumen a no ser adequades per a ser utilitzades directament com a claus d'esquemes criptogràfics segurs. D'una banda, solen ser massa curtes i, d'altra banda, poden ser susceptibles a atacs per diccionari.

Bits de sal L'expressió bits de sal prové de la idea que la sal altera el gust dels aliments, de la mateixa manera que aquests bits modifiquen les contrasenyes.

L'ús d'una sal en criptografia basada en contrasenyes s'ha utilitzat tradicionalment per produir conjunts de claus grans a partir d'una única contrasenya. La clau corresponent a una contrasenya se selecciona de dins d'aquest conjunt de manera aleatòria a partir del valor de sal. Una clau individual se selecciona aplicant una funció de derivació de claus (KDF per les seves sigles en anglès, *Key Derivation Function*).

L'ús de bits de sal té, principalment, dos beneficis:

1. Es dificulta que un atacant pugui precalcular totes les claus corresponents a un diccionari de contrasenyes. Amb l'ús de n bits de sal, cada contrasenya té 2^n possibles claus, pel que el cost de precalcular-les augmenta considerablement.
2. S'aconsegueix que la probabilitat que la mateixa clau sigui seleccionada dues vegades sigui molt baixa. Això soluciona alguns dels problemes que apareixen quan es reutilitzen claus.

A més de fer servir uns bits de sal, una altra tècnica que s'utilitza habitualment en criptografia basada en contrasenyes és incrementar deliberadament el temps de càlcul necessari per calcular cada clau, de manera que aquest increment no sigui significatiu per a un usuari que necessita calcular una clau però sí que ho sigui per a un atacant que es troba fent una cerca exhaustiva. Aquest increment del temps de càlcul es fa augmentant el número d'iteracions que realitza la funció de derivació de clau. La norma PKCS#5 recomana fer servir com a mínim 1.000 iteracions. De totes maneres, moltes de les implementacions actuals ja superen amb escreix aquest valor.

Exemple 7.15 Emmagatzemament de contrasenyes en sistemes basats en Unix

Els sistemes basats en Unix tradicionalment fan servir sal criptogràfica per emmagatzemar les contrasenyes dels usuaris del sistema. Així, normalment les contrasenyes dels usuaris es troben emmagatzemades en el següent format:
`$id$salt$hashed`

on `id` és l'identificador de l'algorisme de hash utilitzat, `salt` és el valor de sal i `hashed` és el valor del hash contrasenya aplicant la sal.

Per exemple, les següents tres entrades serien vàlides per emmagatzemar la informació necessària per validar un usuari que faci servir la contrasenya *criptografia* per entrar al sistema:

```
$6$76YTAM$mbfTXZY1.D7qaIPbzKcQX8yBXuhwTr4D9u3vpctvU7XFcqPkUzgK3.z.93DTJV
      6.zzoMf9GaoDIugnJuY99CC1
$6$86YTAM$qG0.v6FvNTz9j4RKAXB5TMh1twEGhGPxvNlfZiCkpNhBYv4B4MBOYmUkdFUKIR
      IEB.Qfs2Gyqv2ohmxLtgN170
$1$86YTAM$VD8sYNaSgNCoEzFlf20hK.
```

Les dues primeres correspondrien a representacions de la mateixa contrasenya fent servir sha-512 com a funció hash i dos valors de sal diferents. És important notar com un petit canvi en el valor de sal (`76YTAM` en comptes de `86YTAM`) canvia radicalment el valor resultant.

En canvi la tercera entrada correspondria a l'emmagatzematge de la mateixa contrasenya amb la segona sal, però fent servir MD5. En aquest cas, podem veure com canviar la funció de hash també fa variar el resultat, encara que la contrasenya i la sal coincideixin.

Funcions de derivació de clau

Una **funció de derivació de claus** produeix una clau derivada a partir d'una clau base i d'altres paràmetres.

Una funció de derivació de claus basada en contrasenyes és un cas específic d'una funció de derivació de claus on la clau base és la contrasenya i els altres paràmetres són un valor de sal i un número d'iteracions.

La norma PKCS#5 defineix dues funcions de derivació de claus basades en contrasenyes: PKKDF1 i PBKDF2. La funció PBKDF1 s'inclou només per mantenir la compatibilitat amb aplicacions ja existents i es recomana l'ús de PBKDF2 per a les noves aplicacions.

7.4.3 PKCS#12

L'estàndard PKCS#12 especifica un format per emmagatzemar i transportar informació sobre l'identitat de persones, com ara claus privades, certificats i secrets, entre d'altres dades criptogràfiques.

Un dels usos més habituals d'aquest format és l'emmagatzemament d'una clau privada i del seu corresponent certificat digital x.509 o bé per emmagatzemar tots els certificats d'una cadena de confiança.

La versió 1.1 de l'estàndard PKCS#12 va ser republicada com a RFC 7292.

L'estàndard suporta diferents modes de privacitat i integritat per a la transferència d'informació personal. En concret, l'estàndard suporta quatre combinacions, dos modes de privacitat i dos d'integritat:

- Mode de privacitat de clau pública: la informació personal s'empaqueta i es xifra amb una clau pública de la plataforma de destí. Es poden recuperar les dades amb la clau privada corresponent.
- Mode de privacitat amb contrasenya: la informació personal es xifra amb una clau simètrica derivada del nom d'usuari i d'una contrasenya.
- Mode d'integritat amb clau pública: la integritat es garanteix amb una signatura digital sobre el contingut, realitzada amb la clau privada de la plataforma d'origen. La signatura és verificada a la destinació, fent servir la clau pública corresponent.
- Mode d'integritat amb contrasenya: la integritat és garanteix a través d'un Codi d'Autenticació de Missatge (MAC) derivat d'una contrasenya.

7.5 Formats de representació de dades

L'ASN.1 (per les seves sigles en anglès, *Abstract Syntax Notation number One*) és un estàndard que descriu una notació formal utilitzada per descriure dades en protocols de comunicacions. Aquesta notació permet representar dades de manera independent de les codificacions específiques de cada màquina, del sistema operatiu o del llenguatge de programació utilitzat. Les normes PKCS fan servir aquest estàndard per descriure com emmagatzemar i transmetre claus i altres tipus de material criptogràfic.

Organitzacions darrere l'ASN.1

L'ASN.1 és un estàndard conjunt de la ISO (*International Organization for Standardization*), l'IEC (*International Electrotechnical Commission*) i de l'ITU-T (*International Telecommunication Union Telecommunication Standardization Sector*).

L'estàndard permet definir **tipus de dades** i **valors**. Un tipus de dades és una categoria d'informació (per exemple, numèrica o textual). Un valor és una instància d'un tipus concret. La sintaxi ASN.1 defineix tres categories de tipus de dades: tipus simples, que són atòmics; tipus estructurats, que tenen components; i tipus etiquetats, que són derivats d'altres. Es pot assignar un nom als tipus i valors ASN.1, de manera que aquest nom es pot fer servir per definir altres tipus i valors.

Així, per exemple, els enters (`INTEGER`), les cadenes de bits (`BIT STRING`) o el valor null (`NULL`) són tipus simples. En canvi, la seqüència (`SEQUENCE`), una col·lecció ordenada d'un o més valors d'altres tipus, i el conjunt (`SET`), una col·lecció desordenada d'un o més valors d'altres tipus, són tipus estructurats.

La notació ASN.1 es complementa per l'especificació d'un conjunt d'algorismes anomenats **regles de codificació** (en anglès es coneixen com *encoding rules*) que determinen la representació exacta de cada missatge en octets. Tres de les famílies de regles de codificació estandarditzades són: *Basic Encoding Rules* (BER), *Packed Encoding Rules* (PER) i *XML Encoding Rules* (XER).

Codificacions canòniques

De la mateixa manera que la codificació DER ens ofereix una codificació única dins de BER, existeixen representacions canòniques de les regles PER (anomenades CANONICAL-PER) i XER (conegudes com a Canonical XML Encoding Rules o CXER).

En criptografia sovint necessitem una representació única d'una certa dada. Els estàndards de codificació comentats ofereixen però diverses maneres de codificar un mateix valor, pel que no són adients per fer servir en criptografia. Les regles de codificació DER (de l'anglès, *Distinguished Encoding Rules*) són un subconjunt de les regles BER que ofereixen una codificació única a cada valor ASN.1. En concret, les regles DER especifiquen, per a cada valor a codificar, quin dels possibles mètodes de codificació BER s'ha d'utilitzar en aquell cas, assegurant així una codificació única. Així doncs, codificar fent servir les regles DER ens permet garantir que els processos criptogràfics que realitzem no es veuen alterats per l'ús de diferents representacions d'una mateix valor.

Exemple 7.16 Exemple del resultat de codificar en BER i DER

Fent servir la codificació BER, el valor booleà `Cert` pot ser codificat de 255 maneres diferents, que corresponen als 255 valors diferents de zero que es poden representar amb un byte ($2^8 - 1 = 255$). La sintaxi DER ens indica quina d'aquestes 255 maneres hem de triar per tal de codificar el booleà `Cert`.

Així, molts estàndards relacionats amb la criptografia fan servir DER per codificar dades. Per exemple, les dades que són signades dels certificats X.509 que hem vist a la Secció 7.3.1 o les de les llistes de revocació de certificats descrites a la Secció 7.3.2 es codifiquen en DER. Això permet que les comprovacions de les signatures digitals retornin els resultats esperats.

A vegades però, fer servir fitxers binaris per transmetre contingut criptogràfic no és el més adient. PEM és una codificació printable que fa servir 64 caràcters que són universalment representables: les lletres de la `a` a la `z` en minúscula i majúscula, els dígits del `0` al `9` i els símbols `+` i `/`. Així, cada caràcter permet codificar 6 bits d'informació ($2^6 = 64$). Addicionalment, el caràcter `=` es fa servir com a caràcter especial per a indicar com tractar el padding de cada missatge. El text resultant d'una codificació PEM consisteix en un conjunt de línies de 64 caràcters, exceptuant l'última línia que pot contenir un nombre de caràcters menor.

L'origen del format PEM

El protocol Privacy-enhanced Electronic Mail (PEM) va ser el primer estàndard en proposar l'ús d'una codificació en base 64 amb caràcters printables i línies curtes. Tot i que el protocol va caure en desús, la codificació encara es fa servir per transferir material criptogràfic de manera printable. Actualment, es fa servir el nom fitxer PEM per indicar fitxers amb material criptogràfic codificats en base 64 (inspirats amb la codificació original de l'RFC PEM), però que van més enllà del que s'especifica a l'estàndard. Per exemple, l'ús de les etiquetes `--BEGIN CERTIFICATE--` i `--END CERTIFICATE--` no es troba especificada a l'estàndard.

Així doncs, el sistema de codificació PEM ens permet representar una cadena d'octets qualsevol en un conjunt de caràcters printables representats en línies curtes, la majoria de les quals tenen la mateixa mida. La cadena de caràcters representant el material criptogràfic codificat en base 64 es troba emmarcada dins d'unes etiquetes que n'indiquen l'inici i el final. Per exemple, un certificat digital representat en PEM es trobaria emmarcat entre les etiquetes `--BEGIN CERTIFICATE--` i `--END CERTIFICATE--`:

```
--BEGIN CERTIFICATE--
MIIBOTCCATqgAwIBAgIQUq+2SdEkLr5K6xqjSEvRsDANBgkqhkiG9w0BAQUFADAU
MRIwEAYDVQQDEwlsb2NhbGhvc3QwHhcNMTIwODA0MDA0OTEyWhcNMTcwODA0MDAw
MDAwWjAUMRIwEAYDVQQDEwlsb2NhbGhvc3QwgZ8wDQYJKoZIhvcNAQEBBQADgY0A
```

```
[...]
Y2nd44bYEpmaBy7XJ5UIGEkuD3VIxT2S+2bCwkRR+9/+7vggR2q7l7YEktM2mFBI
yqOMOroAw+5cdc06c/B7UimwKFczsyhi9LUIr3rXI42FdXBHWw==
--END CERTIFICATE--
```


7.6 Els problemes de la PKI en desplegaments reals

Tot i que la PKI semblava que hauria de resoldre molts dels problemes de seguretat als quals ens enfrontem, el seu desplegament no ha arribat a complir amb les expectatives que s'hi havien dipositat: avui en dia les PKIs no són tant populars com es creia que arribarien a ser i les garanties de seguretat que ofereixen les infraestructures de clau pública no sempre estan a l'alçada del que, a nivell teòric, haurien d'oferir.

Així, per exemple, l'ús del protocol HTTPS es troba molt estès avui en dia. L'HTTPS fa servir una infraestructura de clau pública per autenticar els servidors web, normalment a través del seu domini. A més, la clau pública del certificat del servidor es fa servir per establir un canal de comunicació segur entre el servidor i el client.

Qui controla les CAs de confiança dels navegadors?

Un estudi publicat l'any 2013 sobre l'estat de l'ecosistema HTTPS reporta que només un 20% dels certificats d'autoritats de certificació de confiança dels principals navegadors corresponen a CAs comercials. La resta d'autoritats es troben controlades per empreses, institucions financeres, institucions religioses, museus i biblioteques.

Per tal de poder validar els certificats dels servidors, els navegadors tenen una llista d'autoritats de certificació de confiança en les quals confien. Aquesta llista acostuma a tenir uns pocs centenars de certificats arrel, el que porta a confiar en uns pocs milers de certificats de CA. Depenent de la configuració del sistema, la llista pot provenir del sistema operatiu o del navegador, i hi ha diferències notables en les llistes de certificats de les diferents configuracions.

Els sectors més crítics amb aquest model defensen que un dels principals problemes que hi ha és que qualsevol CA de confiança té el poder de signar qualsevol domini. Certament, amb el model actual, una CA d'algun país remot que tingui el seu certificat arrel en el navegador d'un usuari gaudirà de la mateixa confiança que una CA espanyola a l'hora d'emetre un certificat per a una pàgina web amb un domini del govern espanyol. Però no només això, les CAs arrel tenen també el poder de crear autoritats de certificació intermèdies que, excepte en entorns molt específics, també tindran el poder d'emetre certificats per a qualsevol domini. En aquesta línia, en els últims anys hi ha hagut diversos incidents de seguretat relacionats amb la PKI de la Web.

Al 2012, l'autoritat de certificació Trustwave (en la qual confien els principals navegadors) va emetre un certificat de CA i va incloure la clau privada corresponent dins d'un dispositiu hardware segur, que va llogar a una empresa amb l'objectiu que aquesta pogués espiar les connexions xifrades amb TLS dels seus empleats. La pròpia CA va reconèixer que havia dut a terme aquesta pràctica. Aquest incident va alertar d'una pràctica que es rumoreja que és habitual entre les autoritats de certificació, i que posa en perill la seguretat d'Internet.

Google ha detectat en nombroses ocasions certificats fraudulents afectant a algun dels seus dominis. Així, al 2014 va denunciar el National Informatics Centre (NIC) de l'Índia (una autoritat subordinada de l'Indian Controller of Certifying Authorities) estava emetent certificats no autoritzats que afectaven alguns dels seus dominis, algun domini de Yahoo i altres dominis. Al desembre de 2013, Google ja havia detectat certificats fraudulents per als seus dominis emesos per l'entitat de certificació francesa ANSSI i, uns mesos abans, al gener del mateix any, per l'entitat turca Türktrust.

Google és capaç de detectar aquests atacs gràcies a vàries mesures que ha anat desplegant amb el temps. Així, per exemple, el navegador Chrome porta incorporat *pinning* de certificats per a alguns dels dominis de google.

El *pinning* és un procediment pel qual s'associa un host a una identitat criptogràfica (una o vàries claus públiques o certificats en una cadena de certificats X.509). La validació d'un pin consisteix a comprovar que almenys una de les claus públiques especificades es troba en la cadena de certificació del host.

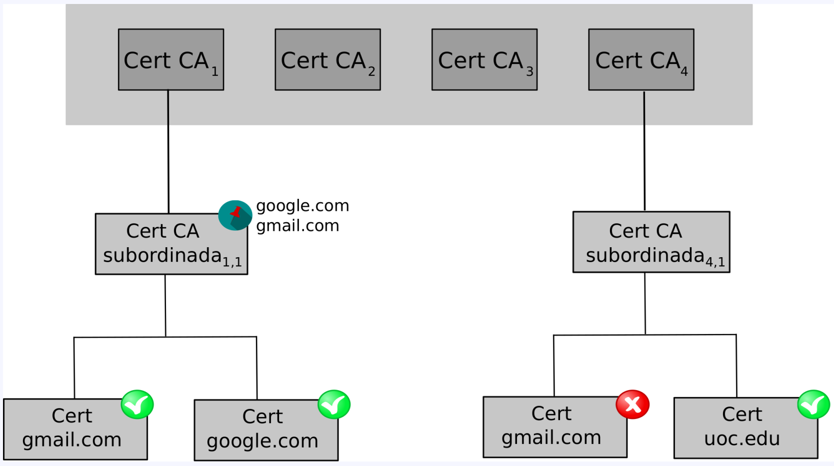
Exemple 7.17 Pinning de certificats en el navegador Chrome

El navegador Chrome inclou un conjunt de claus públiques per a dominis de Google, fet que permet als seus usuaris detectar, per exemple, quan es troben davant d'un certificat de gmail que aparenta ser vàlid (es troba dins d'una cadena de certificació vàlida atenent als certificats de confiança del navegador) però que no inclou cap de les claus públiques especificades, com es donaria en els casos comentats anteriorment.

Així, per exemple, suposem una instància del navegador Chrome que té quatre certificats a la seva llista de confiança, com es mostra a la figura següent. Si no hi ha més restriccions, qualsevol certificat signat amb la clau privada corresponent a algun dels certificats de la llista confiança (o alguna autoritat de certificació subordinada) serà considerat com a vàlid (si totes les comprovacions esmentades a la Secció 7.2.5 són satisfactòries).

Ara bé, si s'afegeix un pin que vinculi el certificat de la CA subordinada$_{1,1}$ amb els dominis google.com i gmail.com, caldrà que la cadena de certificació dels certificats d'aquests dominis passi pel certificat amb el pin per donar el certificat per vàlid. Així, els certificats de gmail.com i google.com emesos per la CA subordinada$_{1,1}$ són vàlids, mentre que el certificat de gmail.com emès per la CA subordinada$_{4,1}$ no ho serà. En canvi, un certificat d'un altre domini que no tingui cap pin (com ara uoc.edu) emès per la mateixa CA subordinada$_{4,1}$, serà considerat com a vàlid.

Noteu que l'existència d'un pin no elimina el requeriment de validar la cadena de certificació, és a dir, el pin afegeix comprovacions a l'hora de validar un certificat, però no n'elimina.



Altres dels problemes que sorgeixen amb els desplegaments pràctics de les infraestructures de clau pública i que no estan limitats a l'https són fruit dels conflictes d'interès entre els diferents actors que participen de la PKI, la falta d'incentius per a segons quines accions crítiques per al bon funcionament de la infraestructura, la confusió que generen certes especificacions o la falta d'usabilitat. Així, per exemple, si analitzem les alternatives de consulta de l'estat de revocació d'un certificat, trobem que el protocol OCSP comporta dificultats alhora d'interpretar les seves respostes (com es comentava a la Secció 7.3.3), mentre que l'emissió de CRLs és un procediment costós per a la CA per al qual no en té un incentiu directe. Pel que fa la usabilitat, potser un dels exemples més clars de sistemes que no arriben a ser utilitzats en tot el seu potencial és el DNI electrònic: mentre que gairebé tota la ciutadania espanyola disposa de certificats digitals en el seu document d'identitat, la necessitat de tenir un dispositiu hardware capaç de llegir el DNI i la dificultat d'instal·lar-lo, configurar-lo i fer-lo servir en un equip domèstic, fan que en molts casos aquests certificats no es facin servir.

7.7 Resum

En aquest capítol, s'ha presentat la infraestructura de clau pública, tot repassant les entitats que hi participen i el seu paper dins de la infraestructura, les fases per les quals passa un certificat digital des de la seva creació fins a la finalització del seu ús i els estàndards més importants que detallen diferents processos de la PKI. Finalment, s'han repassat els formats més habituals per codificar informació criptogràfica i s'ha discutit sobre els problemes que presenten els deplegaments reals de les infraestructures de clau pública, més enllà dels conceptes teòrics detallats als estàndards.

7.8 Solucions dels exercicis

Exercici 7.1:

a i d: Per tal de garantir el no-repudi, la clau privada no pot ser coneguda per ningú, a part del subscriptor del certificat. Per tant, caldrà que les claus s'hagin generat o bé pel propi subscriptor, o bé en un dispositiu hardware segur.

Exercici 7.2:

c: Les afirmacions a i b són falses, ja que tant l'OCSP com les CRLs ens permeten comprovar l'estat de revocació d'un certificat digital, però no la seva validesa (que es troba explicitada en el propi certificat). Les respostes d i e també són falses, ja que podem obtenir també aquestes respostes en altres situacions, per exemple, preguntant per certificats que no hagin estat mai emesos.

Exercici 7.3:

El resultat de la validació del certificat és:

	Resultat de processar l'extensió	
	TRUE	FALSE
L'aplicació reconeix l'extensió	TRUE	FALSE
L'aplicació no reconeix l'extensió	FALSE	FALSE

Exercici 7.4:

El resultat de la validació del certificat és:

	Resultat de processar l'extensió	
	TRUE	FALSE
L'aplicació reconeix l'extensió	TRUE	FALSE
L'aplicació no reconeix l'extensió	TRUE	TRUE

Exercici 7.5:

Cap dels segells de temps proposats ens permetria demostrar que les signatures han estat recollides en el període de temps indicat. El segell de temps a demostraria que les signatures han estat creades abans de l'inici del període establert. El segell de temps b no podria garantir que no s'han creat signatures ni abans ni després del període de temps establert. El segell de temps c no garantiria que les signatures no han estat creades abans de l'inici del període. Un segell de temps només permet garantir que una dada existeix en un instant de temps concret.

7.9 Bibliografia

Adams, C., Steve L. (2003). *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley Professional.

Choudhury, S. Bhatnagar, K., Haque, W. (2002). *Public key infrastructure implementation and design.*. John Wiley & Sons, Inc.

Kuhn, D. Richard, et al. (2001). *Introduction to public key technology and the federal PKI infrastructure*. National Institute of Standards and Technology.

Obaidat, M., Boudriga, N. (2007). *Security of E-Systems and Computer Networks*. Cambridge University Press.

International Telecommunication Union (2000). *Recommendation X.509 - The Directory: Public-key and attribute certificate frameworks.*

International Telecommunication Union (2015). *Recommendation X.680 - Abstract Syntax Notation One (ASN. 1): Specification of Basic Notation*

Jonsson, J., Kaliski, B. (2003). *Public-key cryptography standards (PKCS)# 1: RSA cryptography specifications version 2.1.*

Nystrom, M., et al. (2014). *PKCS# 12: Personal information exchange syntax v. 1.1.*

Kaliski, B. (2000). *PKCS# 5: Password-based cryptography specification version 2.0.*

J. Linn (1993). *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*.
`https://tools.ietf.org/html/rfc1421`

S. Kent (1993). *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management*.
`https://tools.ietf.org/html/rfc1422`

D. Balenson (1993). *Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers*.
`https://tools.ietf.org/html/rfc1423`

B. Kaliski (1993). *Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services*.
`https://tools.ietf.org/html/rfc1424`

S. Kille (1995). *A String Representation of Distinguished Names*.
`https://tools.ietf.org/html/rfc1779`

C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen (1999). *SPKI Certificate Theory*.
`https://tools.ietf.org/html/rfc2693`

B. Kaliski (2000). *PKCS #5: Password-Based Cryptography Specification Version 2.0*.
`https://tools.ietf.org/html/rfc2898`

C. Adams, P. Cain, D. Pinkas, R. Zuccherato (2001). *Internet X.509 Public Key Infrastructure Time-Stamp Protocol*.
`https://tools.ietf.org/html/rfc3161`

W. Polk, R. Housley, L. Bassham (2002). *Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.
`https://tools.ietf.org/html/rfc3279`

J. Jonsson, B. Kaliski (2003). *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography*

Specifications Version 2.1.
`https://tools.ietf.org/html/rfc3447`

M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, R. Nicholas (2005). *Internet X.509 Public Key Infrastructure: Certification Path Building*.
`https://tools.ietf.org/html/rfc4158`

C. Adams, S. Farrell, T. Kause, T. Mononen (2005). *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*.
`https://tools.ietf.org/html/rfc4210`

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.
`https://tools.ietf.org/html/rfc5280`

R. Housley (2009). *Cryptographic Message Syntax (CMS)*.
`https://tools.ietf.org/html/rfc5652`

K. Moriarty, Ed., M. Nystrom, S. Parkinson, A. Rusch, M. Scott (2014). *PKCS #12: Personal Information Exchange Syntax v1.1*.
`https://tools.ietf.org/html/rfc7292`

OpenSSL Software Foundation (Consultat per última vegada: setembre de 2016). *OpenSSL Cryptography and SSL/TLS Toolkit Documentation*.
`https://www.openssl.org/docs/`

S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams (2013). *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*.
`https://tools.ietf.org/html/rfc6960`

8. Criptografia de corbes el·líptiques

En capítols anteriors s'ha vist com la criptografia de clau pública permetia solucionar alguns dels problemes presentats per la criptografia simètrica (com ara la distribució de claus) i alhora oferia propietats addicionals més enllà del xifratge (com ara el no-repudi a través de signatures digitals). Ara bé, la criptografia de clau pública requereix de recursos computacionals més elevats que la criptografia simètrica, cosa que en dificulta la seva execució en dispositius amb poc recursos i en limita el nivell de seguretat com a conseqüència del compromís amb el temps d'execució. De fet, en el cas del xifratge, sovint es combina l'ús de la criptografia de clau pública amb criptografia simètrica a través de la tècnica del sobre digital, cosa que permet xifrar continguts de gran mida amb claus públiques més petites.

En aquest capítol es presenta la criptografia de corbes el·líptiques. Entre els seus principals avantatges hi trobem que la criptografia de corbes el·líptiques ofereix el mateix nivell de seguretat que la criptografia de clau pública tradicional però amb claus més petites. D'aquesta manera, s'aconsegueix també que les operacions criptogràfiques siguin més ràpides d'executar i requereixin de menys recursos computacionals, fent-les possibles en dispositius amb recursos limitats. A més, la criptografia de corbes el·líptiques permet la definició de *pairings*, amb els quals es poden crear construccions criptogràfiques amb propietats addicionals a les que ens oferia la criptografia de clau pública bàsica.

Així doncs, en primer lloc aquest capítol detalla els beneficis de la criptografia de corbes el·líptiques. A continuació, es presenten les corbes el·líptiques, la seva aritmètica, i es descriu com es fan servir les corbes el·líptiques en criptografia. Després s'explica el problema del logaritme discret sobre corbes el·líptiques i es detallen els algorismes criptogràfics més populars basats en aquest problema.

8.1 L'origen de la criptografia de corbes el·líptiques

Com hem vist, la criptografia de clau pública es va donar a conèixer a la dècada dels 70, amb el protocol d'intercanvi de claus de Diffie-Hellman, que permet a dues parts establir un secret compartit sense necessitat d'haver-se intercanviat cap informació prèviament. La seguretat del protocol de Diffie-Hellman es basa en la dificultat de calcular el logaritme discret en el grup dels enters mòdul un primer. Poc després, Rivest, Shamir, i Adleman van proposar l'RSA, un sistema de xifratge de clau pública que es basa en un altre problema, la factorització d'enters. Així doncs, aquests primers algorismes de clau pública es basaven en problemes computacionalment difícils de resoldre que es definien sobre els enters o sobre grups d'enters mòdul un primer.

L'ús de corbes el·líptiques en el disseny de criptosistemes de clau pública va ser proposat per primera vegada l'any 1985 per Neal Koblitz i Victor Miller, de manera independent. Ambdós van proposar fer servir el grup de punts d'una corba el·líptica definida sobre un cos finit en criptosistemes basats en el problema del logaritme discret, donant llum així a la criptografia de corbes el·líptiques (o ECC, de les seves sigles en anglès, *Elliptic Curve Cryptography*).

Més d'una dècada després, els primers estàndards que descrivien algorismes de criptografia de corbes el·líptiques i paràmetres per a les corbes sobre les quals construir-los es van començar a publicar.

Primers estàndards d'ECC

El primer estàndard publicat sobre corbes el·líptiques va ser l'*ANSI X9.62: The Elliptic Curve Digital Signature Algorithm (ECDSA)* l'any 1999. Un any després, al 2000, el NIST també incloïa l'ECDSA a l'ara obsolet *NIST FIPS PUB 186-2: Digital Signature Standard (DSS)*.

L'adopció de la criptografia de corbes el·líptiques no ha estat, però, lliure de polèmica. L'algorisme Dual_EC_DRBG (*Dual Elliptic Curve Deterministic Random Bit Generator*) va ser estandarditzat pel NIST l'any 2006, juntament amb uns altres tres algorismes, per a la generació de números pseudoaleatoris. L'estàndard explicitava no només l'algorisme de generació de números pseudoaleatoris, sinó també la corba el·líptica concreta i els punts de la corba a fer servir per l'algorisme. Altres organismes d'estandardització també van incloure aquests mateixos paràmetres als seus estàndards. Ja durant el procés d'estandardització, alguns investigadors van començar a mostrar preocupacions per les possibles vulnerabilitats de l'algorisme. En particular, els investigadors se'n van adonar que el coneixement d'un cert secret podia permetre recuperar l'estat intern del generador a partir de només 256 bits de la sortida. A més, també van notar que era possible generar els paràmetres de l'algorisme de manera que qui ho fes conegués aquest secret. Dit d'una altra manera, els investigadors van advertir que l'algorisme podia incorporar una porta del darrere (o *backdoor*). Tot i així, diverses implementacions de llibreries criptogràfiques comercials van incorporar l'algorisme amb els paràmetres recomanats pel NIST i, en alguns casos, fins i tot van configurar-lo com a algorisme per defecte. Anys després, les sospites dels investigadors van quedar confirmades quan les revelacions d'Edward Snowden apuntaven que l'NSA havia introduït intencionadament una porta del darrere a l'algorisme Dual_EC_DRBG. Aquests fets van precipitar que l'algorisme fos retirat de l'estàndard del NIST l'any 2014.

La història del Dual EC DRBG

Per a conèixer amb més detalls els fets que van portar a l'estandardització de l'algorisme Dual_EC_DRBG i el paper que les diferents institucions i investigadors hi van jugar, us recomanem la lectura de l'article *Dual EC: A standardized back door*, de Daniel Bernstein, Tanja Lange i Ruben Niederhagen.

Nombres *nothing-up-my-sleeve* (NUMS)

En anglès es fa servir l'expressió *nothing-up-my-sleeve numbers* (literalment, nombres sense res tret de la màniga) per designar els nombres que es troben lliures de sospita de tenir propietats ocultes. Aquests nombres s'utilitzen com a constants en els algorismes criptogràfics, per exemple, per a la inicialització o la definició de paràmetres, amb l'objectiu d'assegurar que no han estat triats intencionadament per a debilitar l'algorisme o incorporar-hi portes del darrere. Per aconseguir-ho, els nombres se seleccionen usant fonts conegudes que deixin poc marge de maniobra per a manipular l'algorisme, com ara els primers dígits decimals de pi.

Tot i aquests daltabaixos amb les agències d'estandardització, la criptografia de corbes el·líptiques s'ha anat fent lloc en la societat, oferint esquemes de signatura digital, de xifratge híbrid, d'intercanvi de claus, i de generació de números pseudoaleatoris, tots ells basats en el problema del logaritme discret sobre corbes el·líptiques.

Actualment, la criptografia de corbes el·líptiques es fa servir àmpliament en protocols com ara l'IPsec o el TLS. L'empresa americana F5 (especialitzada en xarxes de lliurament d'aplicacions, gestió del núvol i seguretat a la xarxa) publica informes anuals sobre l'ús que se n'està fent del protocol TLS a Internet. El seu informe de 2019 (*The 2019 TLS Telemetry report*) reporta que del milió de pàgines web millor situades al rànquing Alexa, prop d'un 20% estan fent servir claus basades en corbes el·líptiques. En concret, un 18.23% fan servir una clau de 256 bits sobre una corba el·líptica. Tot i així, encara predomina l'ús de l'RSA (un 73.57% de les pàgines fan servir claus RSA de 2048 bits).

La criptografia de corbes el·líptiques també s'ha popularitzat per ser utilitzada en esquemes de signatura digital en algunes criptomonedes basades en cadena de blocs (*blockchain*). Així, per exemple, Bitcoin fa servir ECDSA (*Elliptic Curve Digital Signature Algorithm*) com a algorisme de signatura; i Ethereum 2.0 fa servir signatures BLS (*Boneh–Lynn–Shacham*).

8.2 Beneficis de la criptografia de corbes el·líptiques

El principal avantatge de la criptografia de corbes el·líptiques és que utilitza claus més petites per arribar als mateixos nivells de seguretat que els algorismes de criptografia de clau pública tradicionals. Aquesta disminució en la mida de la clau comporta una millora en la velocitat d'execució d'algunes de les primitives bàsiques (per exemple, en la generació de la clau) i, alhora, un estalvi de recursos, de manera que es pot utilitzar en dispositius amb recursos limitats.

El nivell de seguretat d'un algorisme és una mesura creada per comparar la seguretat que ofereixen diferents algorismes criptogràfics quan es fan servir amb diverses mides de clau.

El **nivell de seguretat** d'un algorisme és n quan el millor atac conegut contra l'algorisme requereix 2^n passos. El nivell de seguretat d'un algorisme també es coneix com a la **mida efectiva** de la clau i, en conseqüència, és habitual veure'l expressat en bits.

La Taula 8.1 detalla el nivell de seguretat ofert per diferents algorismes criptogràfics en funció de la mida de la clau utilitzada. Tant el nivell de seguretat com les mides de les claus estan expressades en bits. La mida de la clau correspon a la mida del mòdul per als algorismes basats en el problema de la factorització d'enters; a la mida de la clau pública per als algorismes basats en el logaritme discret; i a l'ordre del punt base per als algorismes basats en corbes el·líptiques.

Així, per exemple, l'AES amb una mida de clau de 128 bits ofereix un nivell de seguretat de 128 bits (en general, en els algorismes simètrics la mida de la clau coincideix amb el nivell de seguretat). Per aconseguir aquest mateix nivell de seguretat fent servir RSA, caldrà utilitzar un mòdul de 3072 bits. En canvi, el mateix nivell de seguretat requereix només d'una clau de 256 bits per a l'ECDSA.

Taula 8.1: Comparativa del nivell de seguretat proporcionat per diferents mides de clau (en bits) depenent de l'algorisme criptogràfic.

	Algorismes criptogràfics			
	Clau simètrica	Factorització d'enters	Logaritme discret	Corbes el·líptiques
Nivell de seguretat	AES, 3DES	RSA	DSA, DH, ElGamal	ECDSA, ECDH
80	80	1024	1024	160
112	112	2048	2048	224
128	128	3072	3072	256
192	192	7680	7680	384
256	256	15360	15360	512

És interessant destacar no només la diferència en la mida de la clau, sinó també en el creixement d'aquesta mida en funció del nivell de seguretat. El creixement de la mida de la clau és molt més ràpid per a algorismes basats en la factorització d'enters i el logaritme discret que en algorismes basats en corbes el·líptiques, de manera que les diferències en les mides de clau entre aquests grups d'algorismes s'incrementen amb l'augment del nivell de seguretat.

També cal remarcar que el nivell de seguretat d'un algorisme criptogràfic es calcula en base a la complexitat del millor algorisme que es coneix en un moment donat per a trencar-lo. Per tant, els nivells de seguretat detallats en aquesta taula reflecteixen el coneixement que es té actualment sobre possibles tècniques de factorització i/o càlcul del logaritme discret. Amb els avenços en la investigació criptogràfica es poden descobrir noves maneres d'atacar aquests problemes, que facin variar aquests nivells de seguretat.

El cost de bullir aigua

La definició que acabem de proporcionar del nivell de seguretat d'un algorisme criptogràfic és poc intuïtiva, en tant que és difícil valorar l'esforç necessari per trencar un criptosistema d'un determinat nivell de seguretat.

Arjen K. Lenstra i altres van proposar fer servir com a mesura més informal de la seguretat d'un algorisme la quantitat d'aigua que s'aconseguiria fer bullir amb l'energia necessària per trencar l'algorisme. Així, l'energia necessària per a trencar una clau RSA de 242 bits és l'equivalent a la necessària per a fer bullir l'aigua que hi cap en una cullereta de cafè. Trencar l'RSA de 745 bits és l'equivalent a fer bullir l'aigua d'una piscina, i per trencar l'RSA de 2380 bits caldria tanta energia com la necessària per fer bullir tota l'aigua del planeta Terra!

Tot i això, l'ús de la criptografia de corbes el·líptiques té també alguns inconvenients en relació als algorismes de clau pública basats en els problemes tradicionals. D'una banda, certs aspectes legals poden dificultar-ne la seva adopció. Algunes empreses tenen patentats diferents aspectes de la criptografia de corbes el·líptiques, cosa que pot dificultar-ne el seu desplegament. D'altra banda, el fet que siguin algorismes més recents fa que els estàndards estiguin menys desenvolupats, i també genera dubtes sobre possibles vulnerabilitats no conegudes. Així, per exemple, encara hi ha poca recerca en aspectes com ara els atacs de canal lateral. Finalment, la complexitat de les matemàtiques rere de les corbes el·líptiques també en dificulta la seva comprensió i accessibilitat, cosa que pot afectar a la seguretat de les implementacions.

8.3 Corbes el·líptiques

La criptografia de corbes el·líptiques proporciona algorismes de clau pública basats en l'estructura algebraica de les corbes el·líptiques definides sobre cossos finits.

Definició 8.1 La **corba el·líptica** $E/\mathbb{Z}_p$ (amb $p > 3$) és el conjunt de tots els parells $(x,y) \in \mathbb{Z}_p$ tals que:

$$y^2 = x^3 + ax + b \mod p$$

juntament amb un punt imaginari a l'infinit $\mathscr{O}$, amb $a, b \in \mathbb{Z}_p$ i $\Delta = -16(4a^3 + 27b^2) \neq 0 \mod p$.

L'expressió que defineix la corba el·líptica tal com l'acabem d'exposar es coneix com la forma curta de Weierstrass.

El valor Δ correspon al discriminant de la corba. Geomètricament, la condició que el discriminant sigui diferent de zero assegura que la corba no té cap vèrtex ni es creua amb sí mateixa, és a dir, no té cap punt que tingui dues o més rectes tangents. Això faria que la corba no fos adequada per al seu ús en els algorismes criptogràfics que descriurem a continuació.

Exemple 8.1 Exemple de punts sobre una corba el·líptica

La corba el·líptica $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$ té 17 elements:

$$[\mathscr{O}, (0,4), (0,7), (1,1), (1,10), (2,5), (2,6), (4,4), (4,7), (6,2), (6,9), (7,4), (7,7), (8,2), (8,9), (10,3), (10,8)]$$

Podem comprovar que aquests punts efectivament pertanyen a la corba verificant que compleixen l'equació que la defineix. Així, per exemple, per al punt $(0,4)$, tenim que:

$$y^2 = x^3 - 5x + 5 \mod 11$$
$$4^2 = 0^3 - 0x + 5 \mod 11$$
$$16 = 5 \mod 11$$

També podem verificar que la corba té discriminant diferent de zero:

$$-16(4a^3 + 27b^2) \mod p = -16(4(-5)^3 + 27(5)^2) \mod 11 = 5 \neq 0 \mod 11$$

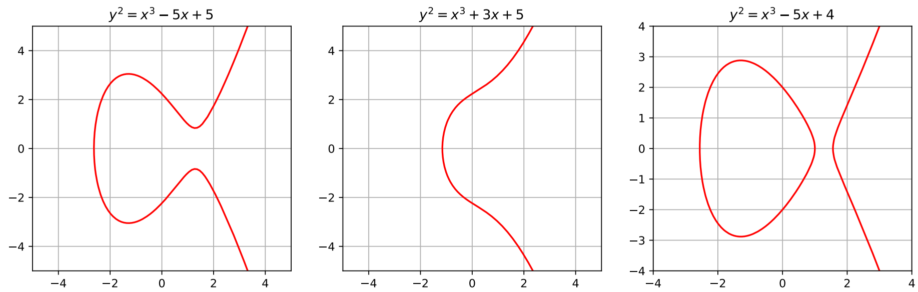
Exercici 8.1 Calculeu tots els punts de la corba el·líptica $E/\mathbb{Z}_5 : y^2 = x^3 - 4x + 1$.

En criptografia es fan servir principalment corbes el·líptiques sobre cossos finits, tal com les acabem de definir. Ara bé, la representació geomètrica de les corbes sobre els reals ens permet obtenir una visualització més intel·ligible d'aquestes. Així, en els propers paràgrafs presentarem les corbes el·líptiques sobre els reals, per tal d'apropar-nos a la seva descripció i les seves propietats.

8.3.1 Corbes el·líptiques sobre els reals

Les figures següents mostren tres exemples de corbes el·líptiques definides sobre els nombres reals, és a dir, corbes de la forma $y^2 = x^3 + ax + b$ on $(x,y) \in \mathbb{R}$:

Figura 8.1: Exemples de corbes el·líptiques sobre $\mathbb{R}$.



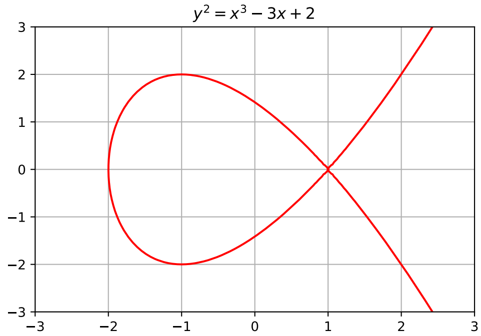
Com podem observar, les corbes són simètriques respecte a l'eix de les abscisses. Això és així ja que y és el resultat d'una arrel quadrada, de manera que per cada valor d'x avaluat, obtindrem dos valors per a y, que correspondran al valor positiu i al negatiu de l'arrel (sempre que aquesta sigui diferent de 0).

D'altra banda, les corbes de la figura anterior no es creuen amb sí mateixes ni tenen cap vèrtex, ja que el discriminant de totes elles és diferent de zero. En canvi, la corba de la Figura 8.2 té discriminant zero ($a = -3, b = 2$ i, per tant, $\Delta = 4(-3)^3 + 27(2)^2 = 0$) i es creua amb sí mateixa en el punt $(1,0)$:

La criptografia de corbes el·líptiques treballa sobre un grup. Per tant, a més dels punts de la corba, que seran els elements d'aquest grup, necessitem definir una **operació de grup**.

Grup	Tal com s'ha presentat al capítol de Fonaments matemàtics, un **grup** és una estructura algebraica en què l'operació definida compleix la propietat associativa i, a més, el conjunt sobre el qual està definida l'operació conté l'element neutre i l'element invers d'aquesta operació.

Figura 8.2: Exemple de corba el·líptica amb discriminant zero.

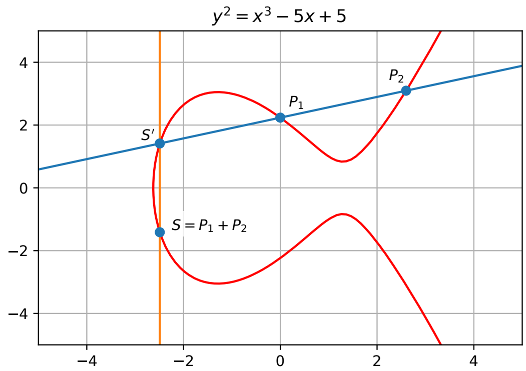


Siguin $P_1 = (x_1, y_1)$ i $P_2 = (x_2, y_2)$ dos punts sobre una corba el·líptica, definirem una operació *suma* de la manera següent.

Si els dos punts són diferents (és a dir, $P_1 \neq P_2$), per calcular el punt resultant de la suma, $S = P_1 + P_2$, traçarem la recta entre P_1 i P_2; trobarem el tercer punt d'intersecció S' d'aquesta recta amb la corba el·líptica; i buscarem el punt simètric d'S' respecte a l'eix de les x, S. Aquest punt simètric S serà el resultat de la suma.

La Figura 8.3 mostra un exemple d'una suma de dos punts diferents, P_1 i P_2. La línia blava correspon a la recta que passa pels dos punts. El tercer punt d'intersecció de la recta amb la corba el·líptica és el punt S'. La línia taronja és una recta vertical que passa pel punt S', i que ens permet calcular el punt simètric S respecte a l'eix de les x. Aquest punt simètric S és el resultat de la suma $P_1 + P_2$.

Figura 8.3: Suma de dos punts diferents.



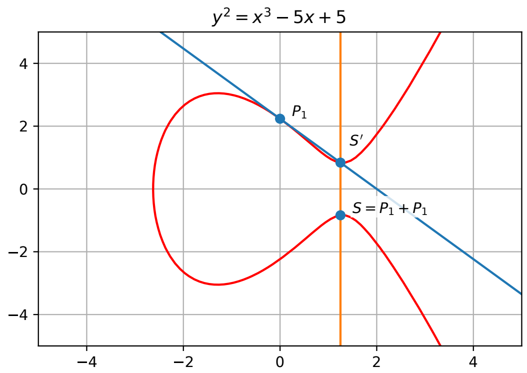
Si els dos punts són iguals (és a dir, $P_1 = P_2$), per calcular el punt suma, $S = P_1 + P_1$, caldrà fer una petita modificació al procediment: la línia a traçar en el primer pas del procediment serà la recta tangent a la corba el·líptica en el punt P_1. Després, es procedeix anàlogament a l'operació de suma de punts diferents: es troba el punt d'intersecció S' de la recta tangent amb la corba el·líptica i, de nou, es busca el punt simètric S respecte a l'eix de les x.

La Figura 8.4 mostra un exemple d'una suma d'un punt P_1 amb ell mateix. La línia blava correspon a la recta tangent a la corba el·líptica en el punt P_1. El punt d'intersecció de la recta amb la corba és el punt S'. De nou, la línia taronja és una recta vertical que passa pel punt S' i que permet calcular el punt simètric S, que és el resultat de la suma $P_1 + P_1$.

El mètode que acabem de descriure per a sumar punts d'una corba és coneix amb el nom de **mètode de la corda i la tangent**.

Ja tenim doncs el conjunt d'elements del grup (els punts de la corba) i una operació de grup (la *suma* que

Figura 8.4: Suma d'un punt amb ell mateix.



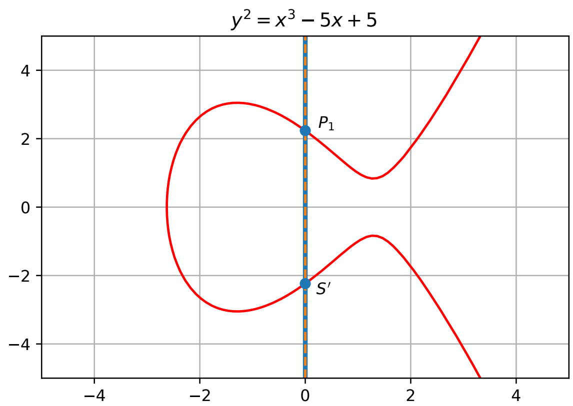
acabem de definir). Per tal d'acabar de definir el grup, caldrà disposar d'un element neutre respecte a l'operació *suma*, és a dir, un element tal que:

$$P_1 + \mathcal{O} = P_1$$

Doncs bé, aquest element neutre és precisament el punt imaginari a l'infinit $\mathcal{O}$ que afegíem com a element de la corba en la definició de l'inici del capítol. Podem imaginar aquest punt com a un punt situat a l'infinit als finals de l'eix de les y. Aquest element és necessari, ja que no hi ha cap altre punt sobre la corba que compleixi que sumat a un altre punt P_1 obtinguem com a resultat el mateix P_1.

Podem fer servir la mateixa estratègia que hem descrit anteriorment per sumar un punt P_1 amb l'element $\mathcal{O}$, i comprovar com efectivament el resultat és el mateix P_1. Així, per calcular $S = P_1 + \mathcal{O}$, en primer lloc es traça una recta entre el punt P_1 i $\mathcal{O}$. Aquesta recta serà la recta vertical que passi pel punt P_1 (en l'exemple de la Figura 8.5, correspon a la línia blava). A continuació, es troba el segon punt d'intersecció S' d'aquesta recta amb la corba el·líptica. Finalment, es busca el punt simètric d'S' respecte a l'eix de les x. Aquest punt simètric S serà el resultat de la suma, i serà precisament el mateix punt P_1.

Figura 8.5: Suma d'un punt amb l'element neutre $\mathcal{O}$.

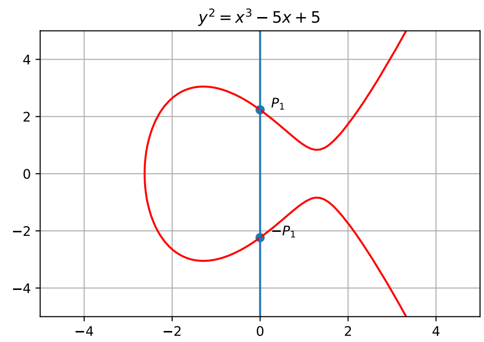


En un grup cal que els elements tinguin inversos. Una vegada tenim el punt neutre definit, podem definir l'invers additiu de qualsevol punt P_1 de la corba com l'element $-P_1$ tal que:

$$P_1 + (-P_1) = \mathcal{O}$$

Donat un punt $P_1 = (x_1, y_1)$, el seu invers additiu $-P_1$ és simplement $(x_1, -y_1)$. Efectivament, si sumem P_1 i $-P_1$ fent servir l'operació suma que hem definit anteriorment, obtenim com a resultat el punt a l'infinit $\mathcal{O}$.

Figura 8.6: L'invers d'un punt.



Això fa que calcular l'invers d'un punt a la corba sigui molt eficient, ja que per a un punt $P_1 = (x_1, y_1)$, el seu invers additiu $-P_1$ és simplement $(x_1, -y_1)$.

Semàntica

En aquest capítol fem servir els termes *suma*, *neutre* i *invers additiu* (o simplement *invers*) per a referir-nos a l'operació que definim sobre els punts de la corba el·líptica, l'element tal que sumat a un punt dona el mateix punt, i el punt tal que sumat a un altre punt dona l'element neutre. Aquests termes són, però, una mica arbitraris. Podríem haver utilitzat algun altre nom per a descriure l'operació (per exemple, multiplicació) i altres termes com ara element identitat i element negatiu per referir-nos a l'element neutre i a l'invers d'un punt.

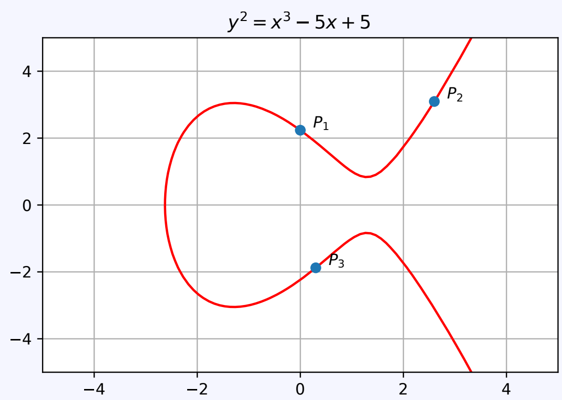
Per últim, per a tenir estructura de grup l'operació *suma* ha de ser associativa, és a dir, donats tres punts P_1, P_2 i $P_3 \in E$, aquests han de complir que:

$$(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$$

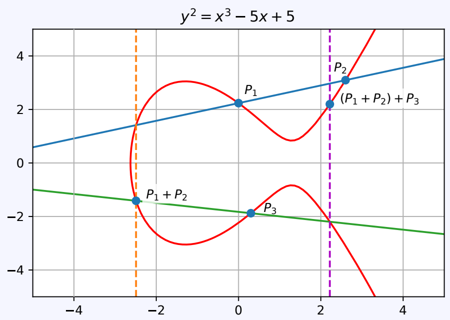
La demostració d'aquesta propietat queda fora de l'abast d'aquest document, però en veurem un exemple.

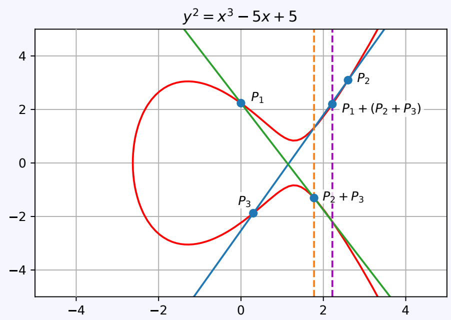
Exemple 8.2 **Exemple de propietat associativa de la suma**

Donats tres punts P_1, P_2 i $P_3 \in E/\mathbb{R} : y^2 = x^3 - 5x + 5$, calcularem el resultat de la suma $P_1 + P_2 + P_3$ executant les sumes entre dos punts en ordres diferents, i comprovarem que el resultat és el mateix.



Les dues imatges següents mostren gràficament la suma dels tres punts. A la imatge de l'esquerra s'efectua la suma $(P_1 + P_2) + P_3$; a la imatge de la dreta es calcula $P_1 + (P_2 + P_3)$.





Per a calcular $(P_1 + P_2) + P_3$ (imatge de l'esquerra) se suma primer $P_1 + P_2$ (la línia blava mostra la recta entre els dos punts i la línia taronja el punt simètric al tercer punt d'intersecció amb la corba). A continuació se suma el resultat amb P_3: la línia verda mostra la recta entre els dos punts a sumar i la línia violeta el punt simètric al tercer punt d'intersecció amb la corba, que és el resultat de la suma dels tres punts.

Anàlogament, per calcular $P_1 + (P_2 + P_3)$ (imatge de la dreta) se suma $P_2 + P_3$ (línies blava i taronja) i al resultat se li suma P_1 (línies verda i violeta).

En efecte, el resultat d'ambdues operacions és el mateix punt.

Així doncs, els punts sobre una corba el·líptica (juntament amb el punt a l'infinit) i l'operació suma que acabem de definir (amb el punt a l'infinit com a element neutre que permet definir els inversos dels elements) formen un grup. Ara bé, per tal que aquest grup pugui ser usat en criptografia, caldrà deixar enrere la representació sobre els reals i tornar als cossos finits.

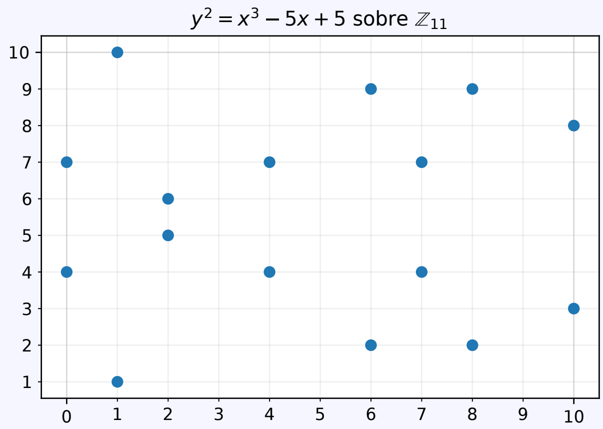
8.3.2 Corbes el·líptiques sobre cossos finits

Deixant enrere la representació de les corbes el·líptiques sobre els reals, que ajuda a comprendre'n les seves característiques però que no és gaire útil per a la criptografia, reprenem ara les corbes el·líptiques sobre cossos finits, tal com s'han definit a l'inici del capítol.

Es poden representar gràficament els punts que conformen una corba el·líptica sobre un cos finit de manera similar a com es fa sobre els reals. En aquest cas, però, es deixa de visualitzar la forma de la corba, i simplement es podrà observar el conjunt de punts i algunes propietats de la seva estructura. En particular, se segueix mantenint la simetria respecte l'eix de les x.

Exemple 8.3 Exemple de representació gràfica d'una corba el·líptica sobre un cos finit

A continuació es representa gràficament la corba el·líptica de l'Exemple 8.1, $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$, que com s'ha vist té 17 elements. Val a dir que a la figura s'observen només 16 punts, ja que el 17è element correspon al punt a l'infinit $\mathcal{O}$.

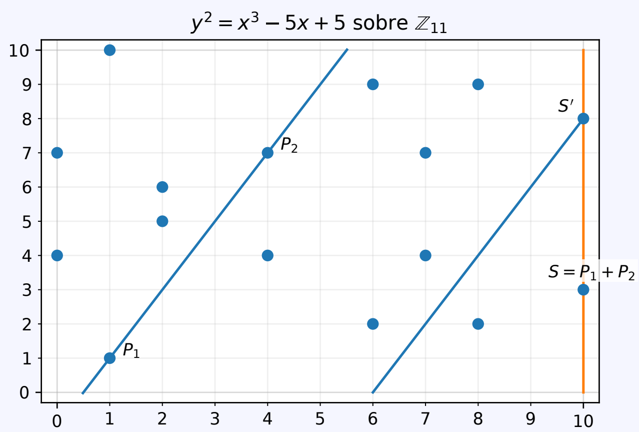


La llei de grup

Podem mantenir la definició de l'operació *suma* que s'ha presentat a la secció anterior, treballant ara sobre el cos finit, com a operació de grup. Gràficament, es pot seguir aplicant el mateix procediment per tal de calcular el resultat d'una suma de dos punts, considerant ara però que les rectes que es tracen són mòdul el primer.

Exemple 8.4 Exemple de suma de punts d'una corba el·líptica sobre un cos finit

Seguint amb la corba dels exemples anteriors, $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$, calculem la suma entre els punts $P_1 = (1,1)$ i $P_2 = (4,7)$ gràficament. Per fer-ho, es traça la recta que els uneix, prolongant-la si cal considerant el mòdul, fins a trobar el tercer punt d'intersecció de la recta amb els punts de la corba, $S' = (10,8)$. Finalment, es calcula el punt simètric d'S', S, que correspon al resultat de la suma ($S = P_1 + P_2 = (10,3)$).



A la pràctica, però, quan s'opera amb punts sobre corbes el·líptiques, es fan servir expressions analítiques per tal de calcular els resultats de les operacions. A continuació es detallen les expressions que permeten calcular la suma de dos punts sobre una corba el·líptica mòdul un primer.

Donats dos punts, $P_1 = (x_1, y_1)$ i $P_2 = (x_2, y_2)$, que pertanyen a una corba el·líptica $E/\mathbb{Z}_p$, el punt P_3 resultant de la suma, $P_3 = P_1 + P_2 = (x_3, y_3)$, es pot calcular com:

Si $P_1 \neq P_2$:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \mod p$$

Si $P_1 = P_2$:

$$m = \frac{3x_1^2 + a}{2y_1} \mod p$$

i en ambdós casos:

$$x_3 = m^2 - x_1 - x_2 \mod p$$
$$y_3 = m(x_1 - x_3) - y_1 \mod p$$

Convé esmentar que el valor m que es calcula en el primer pas de la suma correspon al pendent de la recta entre els dos punts (quan els punts a sumar són diferents entre ells) o bé de la recta tangent a la corba que passa pel punt (quan els punts a sumar són iguals).

D'altra banda, pel que fa a la terminologia, a vegades es distingeix entre la *suma de punts* i el *doblat d'un punt* per referir-se, respectivament, a la suma de punts diferents (és a dir, el cas $P_1 \neq P_2$) i de punts iguals (és a dir, el cas $P_1 = P_2$).

Exemple 8.5 Suma de punts

Donats els punts $P_1 = (x_1, y_1) = (1, 10)$ i $P_2 = (x_2, y_2) = (4, 7)$ de la corba $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$, podem calcular el punt $P_3 = P_1 + P_2$ de la manera següent.

Com que $P_1 \neq P_2$, aleshores el pendent m és:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \mod p = \frac{7 - 10}{4 - 1} \mod 11 = \frac{8}{3} \mod 11 = 10$$

Després, es calculen les coordenades del punt:

$$x_3 = m^2 - x_1 - x_2 \mod p = 10^2 - 1 - 4 \mod 11 = 7$$
$$y_3 = m(x_1 - x_3) - y_1 \mod p = 10(1 - 7) - 10 \mod 11 = 7$$

Finalment, es pot comprovar com $P_3 = (x_3, y_3) = (7, 7)$ es troba efectivament a la corba E:

$$\begin{aligned} y^2 &= x^3 - 5x + 5 \mod 11 \\ 7^2 &= 7^3 - 5 \cdot 7 + 5 \mod 11 \\ 49 &= 313 \mod 11 \\ 5 &= 5 \mod 11 \end{aligned}$$

Exercici 8.2 Donats els punts $P_1 = (x_1, y_1) = (1,1)$ i $P_2 = (x_2, y_2) = (4,7)$ de la corba $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$, calculeu analíticament el punt resultant de la suma $P_3 = P_1 + P_2$, i comproveu que el resultat coincideix amb el que hem calculat gràficament a l'Exemple 8.4.

Doncs bé, els punts de la corba sobre un cos finit, amb l'operació suma que acabem de definir, poden formar un grup cíclic, sobre el qual es poden construir algorismes criptogràfics basats en el problema del logaritme discret.

Grup cíclic	Tal com s'ha presentat al capítol de Fonaments matemàtics, un **grup cíclic** és un grup que conté un element g (que s'anomena generador) tal que les seves potències generen tots els elements del grup, llevat del zero.

Exemple 8.6 Exemple de grup cíclic sobre una corba el·líptica

Com hem vist a l'exemple anterior, la corba el·líptica $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$ té 17 elements:

$$[\mathcal{O}, (0,4), (0,7), (1,1), (1,10), (2,5), (2,6), (4,4), (4,7), (6,2), (6,9), (7,4), (7,7), (8,2), (8,9), (10,3), (10,8)]$$

Aquests punts formen un grup cíclic d'ordre 17. Com ja hem vist al capítol de Fonaments Matemàtics, com que 17 és primer, tots els elements són primitius i, per tant, podem generar tots els punts de la corba sumant un punt amb sí mateix iterativament. Per exemple, per a $P = (0,4)$:

$$\begin{array}{lll}
P = (0,4) & 8P = (6,9) & 15P = (4,7) \\
2P = P + P = (4,4) & 9P = (6,2) & 16P = (0,7) \\
3P = P + P + P = (7,7) & 10P = (10,3) & 17P = \mathcal{O} \\
4P = (8,2) & 11P = (2,5) & 18P = (0,4) \\
5P = (1,10) & 12P = (1,1) & 19P = \cdots \\
6P = (2,6) & 13P = (8,9) & \\
7P = (10,8) & 14P = (7,4) &
\end{array}$$

Exercici 8.3 Genereu tots els punts de la corba el·líptica $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$ fent servir el punt $P = (8,9)$ com a generador.

Exercici 8.4 Calculeu quin és l'invers del punt $P_1 = (4,7)$ de la corba $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$.

Exercici 8.5 La corba el·líptica $E/\mathbb{Z}_{11} : y^2 = x^3 - 3x + 6$ té 9 elements:

$$[\mathcal{O}, (1,2), (1,9), (4,5), (4,6), (7,3), (7,8), (9,2), (9,9)]$$

Podem generar tots els punts de la corba a partir del punt $(4,5)$?

Multiplicació escalar

Cal destacar que a l'exemple anterior implícitament acabem de definir l'operació de **multiplicació escalar** en una corba el·líptica com a l'operació de suma reiterada d'un element amb ell mateix (escrivíem $2P$ per a representar $P+P$; $3P$ per a representar $P+P+P$, etc.).

Per tal de calcular de manera eficient operacions de multiplicació escalar, es pot utilitzar una generalització de l'algorisme de multiplicar i elevar. Com que s'ha fet servir notació additiva per a definir l'operació de grup en corbes el·líptiques però, en canvi, la versió tradicional de l'algorisme de multiplicar i elevar fa servir notació multiplicativa, caldrà adaptar l'algorisme. En concret, caldrà substituir les operacions de multiplicació per l'operació *suma* de punts que s'ha definit. A més, com que en corbes el·líptiques sovint es fa servir notació additiva (tal com estem fent en aquest capítol), l'algorisme per a corbes el·líptiques es coneix habitualment amb el nom d'**algorisme de doblar i sumar**.

Així, per tal de calcular la multiplicació escalar entre un punt (`point`) d'una corba (`ec`) i un enter (`scalar`), podem fer servir l'algorisme següent:

```
def double_and_add(point, scalar, ec):
    """
␣␣␣␣Donat␣un␣punt␣(point)␣que␣pertany␣a␣una␣corba␣(ec)␣i␣un␣enter
␣␣␣␣␣␣␣␣␣(scalar),␣retorna␣la␣multiplicació␣escalar␣del␣punt␣per
␣␣␣␣␣␣␣␣␣␣l'enter.
␣␣␣␣"""

    # Obtenim la representació binària de l'enter
    bin_scalar = bin(scalar)[2:]

    # Inicialitzem el resultat amb l'element neutre
    result = ec.neutre

    # Recorrem la representació binària des del dígit
    # menys significatiu al més significatiu
    for i, e in enumerate(bin_scalar[::-1]):
        if e == "1":
            result = (result + point) # sumar
        if i != len(bin_scalar) - 1: # si no és l'última iteració
            point = (point + point) # doblar
    return result
```

Per a un escalar s, l'algorisme requereix $\log_2 s$ iteracions del bucle, cadascuna de les quals pot comportar una o dues sumes de punts, en funció del valor del bit de l'enter que s'està processant a cada iteració. Per tant, en el pitjor cas l'algorisme requereix de $2\log_2 s$ sumes.

A més de permetre calcular la multiplicació de manera més eficient que sumant repetidament el punt amb sí mateix, l'algorisme de doblar i sumar té un altre avantatge. Quan el punt P a multiplicar és un punt fixat (per exemple, si es fa servir una corba estandarditzada com les que veurem més endavant), es pot accelerar el temps de computació precalculant i emmagatzemant alguns valors que es reutilitzen sovint. En concret, s'emmagatzemen els resultats d'anar doblant el punt P (és a dir, $2P, 4P, 8P, \ldots$), de manera que el pitjor cas només requereixi de $\log_2 s$ sumes. Aquesta reducció en el temps d'execució ve, però, a canvi d'un increment en l'espai necessari per córrer l'algorisme, doncs cal desar els resultats precalculats.

Exemple 8.7 Exemple de multiplicació escalar

Procedim a calcular $10P$ per a $P = (0,4) \in E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$ fent servir l'algorisme de doblar i sumar.

La representació binària de l'enter 10 és 1010. Per tant, es faran quatre iteracions del bucle.

En primer lloc s'inicialitza `result` a $\mathcal{O}$. A continuació s'executen les iteracions:

Iteració	e	Còmput
1	$e=0$	`point = point + point` $=(0,4)+(0,4)=(4,4)$
2	$e=1$	`result = result + point` $=\mathcal{O}+(4,4)=(4,4)$
		`point = point + point` $=(4,4)+(4,4)=(8,2)$
3	$e=0$	`point = point + point` $=(8,2)+(8,2)=(6,9)$
4	$e=1$	`result = result + point` $=(4,4)+(6,9)=(\mathbf{10},\mathbf{3})$

Per tant, $10P=(10,3)$. Cal remarcar, d'una banda, que el càlcul ha requerit únicament de quatre operacions de suma entre punts (en comptes de les nou que caldrien si haguéssim anat sumant repetidament P amb sí mateix). D'altra banda, és interessant notar com s'ha arribat al resultat: la variable `point` conté, a cada iteració, el resultat de doblar P (els valors $2P=(4,4)$, $4P=(8,2)$ i $8P=(6,9)$); i la variable `result` acumula els valors que permeten calcular el resultat i que vénen indicats per la representació binària de l'escalar (en aquest cas, $10P=8P+2P$).

Exercici 8.6 Calculeu $12P$ per a $P=(7,2)\in E/\mathbb{Z}_{23}: y^2=x^3+3x+8$ fent servir l'algorisme de doblar i sumar.

El nombre de punts d'una corba el·líptica

Comptar el nombre de punts d'una corba el·líptica sobre un cos finit $\#E/\mathbb{Z}_p$ no és senzill. El teorema de Hasse, provat l'any 1933, proporciona uns llindars que permeten acotar aquest valor.

Teorema 8.1 Donada una corba el·líptica $E/\mathbb{Z}_p$ el nombre de punts de la corba $\#E$ compleix que:

$$|\#E-(p+1)|\leq 2\sqrt{p}$$

Exercici 8.7 Proporcioneu una estimació del nombre de punts de la corba el·líptica $E/\mathbb{Z}_{11}: y^2=x^3-3x+6$ fent servir el teorema de Hasse.

Però el teorema de Hasse no ens permet calcular el nombre exacte de punts i, com hem vist, aquest és important per caracteritzar el grup que es genera. El mètode més simple per trobar el nombre exacte de punts d'una corba consisteix a calcular per cadascun dels possibles valors d'$x\in\mathbb{Z}_p$, el nombre de solucions que té l'equació de la corba (tal com es proposa a la solució de l'Exercici 8.1). Ara bé, aquest mètode no és computacionalment viable per a les corbes que són útils en criptografia.

Durant anys no es coneixia cap algorisme eficient per al càlcul exacte del nombre de punts d'una corba el·líptica (els algorismes existents eren exponencials). Això va canviar l'any 1985 amb la publicació de l'algorisme de Schoof, el primer algorisme amb temps d'execució polinomial que permetia comptar els punts d'una corba el·líptica. Aquesta versió de l'algorisme seguia sent ineficient per a corbes amb l'ordre necessari per a aplicacions criptogràfiques, però una versió millorada d'aquest, l'algorisme de Schoof-Elkies-Atkin (SEA), sí que es pot fer servir a la pràctica per a aquestes corbes. L'algorisme de SEA és actualment el millor algorisme genèric conegut per a comptar punts de corbes el·líptiques.

Estructura dels grups generats per corbes el·líptiques

Sabem doncs que els punts d'una corba el·líptica sobre un cos finit amb l'operació suma poden formar un grup cíclic. Ara bé, ens podem preguntar si això és sempre així. La resposta és negativa. Si l'ordre de la corba, $\#E/\mathbb{Z}_p$, es pot factoritzar en el producte de primers diferents, aleshores el grup $E/\mathbb{Z}_p$ és cíclic. En cas contrari, el grup és isomorf al producte directe de dos grups cíclics.

Grups isomorfs i producte directe

Informalment, diem que dos grups són isomorfs si tenen la mateixa estructura, és a dir, si les diferències entre els dos grups són només *cosmètiques* (per exemple, en el nom dels elements). D'altra banda, el producte directe de dos grups és un grup que té com a elements els membres del producte Cartesià entre els dos grups. L'operació de grup opera component a component, utilitzant l'operació definida en cada grup d'origen. Per a una introducció més formal i completa a aquests termes, us recomanem la lectura de *Further pure mathematics: Group theory* de *The Open University* (2016).

El cas particular en què el nombre d'elements és primer és el que hem anat veient en la majoria d'exemples d'aquest capítol. En aquest cas, el grup $E/\mathbb{Z}_p$ és cíclic i, a més, tots els elements (excepte el punt a l'infinit $\mathcal{O}$) en són generadors. A continuació estudiarem l'estructura dels grups formats per corbes el·líptiques amb nombre d'elements no primer i, per fer-ho, veurem algunes definicions i teoremes que descriuen l'estructura de grups finits (no necessàriament definits per corbes el·líptiques).

Definició 8.2 Donat un grup $(G,\cdot)$, un subconjunt d'elements $H \subseteq G$ és un **subgrup** de G si $(H,\cdot)$ és un grup. Per denotar que H és un subgrup de G, escrivim $H \leq G$.

Aquesta definició segueix el que entendríem intuïtivament com a un subgrup, és a dir, un subgrup no és res més que un subconjunt d'elements d'un grup que manté les propietats de grup amb la mateixa operació (associativitat, element neutre i element invers).

Definició 8.3 Donat un grup $(G,\cdot)$ i un element $e \in G$, el subgrup H format per les potències de l'element e és un **subgrup cíclic** de G.

Això es deriva directament de les definicions de subgrup i grup cíclic. L'element e les potències del qual generen el subgrup cíclic és doncs el generador del subgrup.

Teorema 8.2 L'ordre d'un element $e \in G$ és igual a l'ordre del subgrup cíclic que genera.

Donat un grup qualsevol, ens podem preguntar com són els subgrups que conté. El teorema de Lagrange ens descriu l'ordre d'aquests subgrups.

Teorema 8.3 El teorema de Lagrange estableix que per tot grup finit G, l'ordre de cada subgrup de G és un divisor de l'ordre de G.

Fixeu-vos que el teorema de Lagrange ens diu que l'ordre dels subgrups de G (i, per tant, l'ordre dels elements de G) és un divisor de l'ordre de G, però no ens descriu si per a cada divisor de l'ordre de G hi ha un element que té aquell ordre.

Si el grup G és cíclic, podem determinar exactament quants elements amb cada ordre possible hi ha.

Teorema 8.4 Sigui G un grup cíclic d'ordre n, si $d|n$ aleshores hi ha $\phi(d)$ elements $g \in G$ d'ordre d.

Exemple 8.8 Exemple de grup cíclic amb ordre no primer amb enters

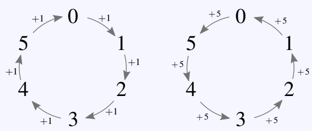
Sigui $G = (\mathbb{Z}_6,+)$ el grup format pels enters mòdul 6 amb l'operació suma modular. Veiem alguns exemples de les propietats de $(\mathbb{Z}_6,+)$ com a grup:

- L'operació suma és associativa, per exemple, $2+(3-5) \mod 6 = (2+3)-5 \mod 6$.
- Té element neutre respecte a la suma modular, el 0, ja que qualsevol element sumat a 0 dona el mateix element.
- Per a qualsevol element e del grup, l'element $-e \mod 6$ és l'invers additiu, ja que $e-e=0$

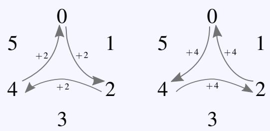
mod 6.

El grup $(\mathbb{Z}_6, +)$ és un grup cíclic, ja que es pot generar a partir de les *potències* dels elements 1 i 5. Noteu que en aquest cas, com que estem fent servir notació additiva, les *potències* són la suma repetida (i no pas la multiplicació).

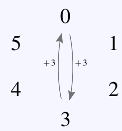
El grup $(\mathbb{Z}_6, +)$ té dos subgrups (més enllà del subgrup amb l'element neutre i d'ell mateix). L'element 2 genera un subgrup cíclic d'ordre 3, format pels elements $\{2, 4, 0\}$. L'element 4 també genera aquest mateix subgrup. D'altra banda, l'element 3 genera el subgrup cíclic d'ordre 2: $\{3, 0\}$. Noteu com tots els subgrups de $(\mathbb{Z}_6, +)$ contenen l'element 0.



Subgrup d'ordre 6
Generadors: 1, 5



Subgrup d'ordre 3
Generadors: 2, 4



Subgrup d'ordre 2
Generadors: 3

Tal com ens diu el teorema de Lagrange, l'ordre dels subgrups és un divisor de l'ordre de G ($2|6$ i $3|6$).

A més, com que G és cíclic, sabem que hi ha $\phi(2) = 1$ element d'ordre 2 (l'element 3) i $\phi(3) = 2$ elements d'ordre 3 (els elements 2 i 4).

Ordre	Núm. d'elements	Elements	Subgrup
1	$\phi(1) = 1$	0	$\{0\}$
2	$\phi(2) = 1$	3	$\{0, 3\}$
3	$\phi(3) = 2$	2, 4	$\{0, 2, 4\}$
6	$\phi(6) = 2$	1, 5	$\{0, 1, 2, 3, 4, 5\}$

Exemple 8.9 Exemple de grup cíclic amb ordre no primer amb corba el·líptica

La corba el·líptica $E/\mathbb{Z}_{11} : y^2 = x^3 + 10x + 4$ té 15 elements:

$$[\mathscr{O}, (0,2), (0,9), (1,2), (1,9), (4,3), (4,8), (5,5), (5,6), (6,4), (6,7), (9,3), (9,8), (10,2), (10,9)]$$

El grup format pels punts de la corba el·líptica amb l'operació suma tal com l'hem definida és un grup cíclic ja que 15 es pot factoritzar en el producte de primers diferents (3 i 5). L'element $(0, 2)$ n'és un dels generadors.

Seguint el teorema de Lagrange, l'ordre dels subgrups són els divisors de 15, és a dir, 3 i 5. Com que G és cíclic, sabem que hi ha $\phi(3) = 2$ elements d'ordre 3 i $\phi(5) = 4$ elements d'ordre 5.

Ordre	Núm. d'elem.	Elements	Subgrup
1	$\phi(1) = 1$	$\mathscr{O}$	$\{\mathscr{O}\}$
3	$\phi(3) = 2$	(1, 2), (1, 9)	$\{\mathscr{O}, (1,2), (1,9)\}$
5	$\phi(5) = 4$	(5, 5), (5, 6), (10, 2), (10, 9)	$\{\mathscr{O}, (5,5), (5,6), (10,2), (10,9)\}$
15	$\phi(15) = 8$	(0, 2), (0, 9), (4, 3), (4, 8), (6, 4), (6, 7), (9, 3), (9, 8)	$E/\mathbb{Z}_{11} : y^2 = x^3 + 10x + 4$

És molt habitual que les corbes que es fan servir en criptografia tinguin ordre primer (i, per tant, tots els elements generin tot el grup cíclic). En algunes construccions, però, es triaran corbes que no tinguin aquesta propietat. Donat un punt base G d'ordre n, es defineix el cofactor h d'una corba el·líptica, que mesura la proporció de punts útils de la corba:

Definició 8.4 El cofactor h d'una corba el·líptica $E/\mathbb{Z}_p$ d'ordre $\#E$ per a un punt base $G \in E$ d'ordre n és:

$$h = \frac{\#E}{n}$$

A les corbes amb ordre $\#E$ primer, tenim que $\#E = n$ i, per tant, el cofactor és sempre 1.

Exemple 8.10 Exemple de cofactor diferent d'1

Recuperem la corba el·líptica de l'exemple 8.9 (els subgrups de la corba es troben detallats al propi exemple). Com hem vist, la corba $E/\mathbb{Z}_{11} : y^2 = x^3 + 10x + 4$ té ordre 15 i l'element $(1,2)$ té ordre 3.

Per tant, el cofactor h de la corba E per al punt base $G = (1,2) \in E$ és:

$$h = \frac{\#E}{n} = \frac{15}{3} = 5$$

8.4 Corbes el·líptiques per a usos criptogràfics

Com s'ha comentat anteriorment, en criptografia es fan servir corbes el·líptiques sobre cossos finits, i no pas corbes definides sobre els reals. Ara bé, són totes les corbes el·líptiques sobre cossos finits adequades per a usos criptogràfics? La resposta és, de nou, negativa. No totes les corbes el·líptiques permeten construir criptosistemes segurs: algunes d'elles són vulnerables a atacs coneguts. Anomenem corbes criptogràficament fortes a les corbes que són adequades per a usos criptogràfics.

Abans de descriure les propietats de les corbes criptogràficament fortes, detallarem els paràmetres que defineixen els criptosistemes basats en corbes el·líptiques:

Definició 8.5 Els **paràmetres de domini** d'un criptosistema basat en corbes el·líptiques són els paràmetres que determinen la corba el·líptica E i el punt base G:

- Un primer p que especifica el cos finit sobre el qual es defineix la corba.
- Els dos coeficients $a, b \in \mathbb{Z}_p$ que defineixen la corba $E/\mathbb{Z}_p : y^2 = x^3 + ax + b$.
- Un punt base $G \in E/\mathbb{Z}_p$ que genera el subgrup cíclic sobre el que es construeix el problema del logaritme discret.
- L'ordre n primer del punt base G.
- El cofactor $h = \#E/n$.

Habitualment els paràmetres de domini s'especifiquen com una tupla (p,a,b,G,n,h).

Considerant els atacs que es coneixen actualment, es requereix que els paràmetres de domini de les corbes el·líptiques per a usos criptogràfics compleixin les condicions següents:

1. És necessari que el nombre de punts de la corba $\#E$ sigui divisible per un primer n suficientment gran (en general, com a mínim major que 2^{160}). Això s'aconsegueix habitualment seleccionant corbes amb $\#E$ primer o bé amb un cofactor petit (normalment, el cofactor h és $1,2,3$ o 4).
2. Cal assegurar que l'ordre de la corba no coincideixi amb el del cos finit sobre el qual es defineix, és a dir, per a una corba $E/\mathbb{Z}_p$, cal assegurar que $\#E \neq p$.
3. Cal assegurar que n no divideix $p^k - 1$ per a tots els enters k entre 1 i 20.

Ara bé, a partir d'aquestes condicions, com es poden generar corbes el·líptiques per a usos criptogràfics? Una de les alternatives que es fan servir és seleccionar corbes aleatòriament, descartant aquelles corbes que no compleixen els tres requisits especificats al paràgraf anterior. El procediment consisteix doncs en repetir el procés de seleccionar aleatòriament una corba i verificar que aquesta compleixi els requisits fins a trobar-ne una que els compleixi. La selecció aleatòria de la corba proporciona un cert nivell de seguretat, ja que la probabilitat de generar una corba que pertanyi a una classe especial que sigui vulnerable a un atac específic (potencialment encara per descobrir) és molt baixa.

Tanmateix, com que algunes de les condicions que cal que les corbes compleixin per a ser segures per a usos criptogràfics no són trivials de verificar (sobretot per part de desenvolupadors sense coneixements específics en aquest tipus de criptografia) i, a més, el procediment de generar les corbes és computacionalment costós (implica comptar el nombre de punts de la corba, que com ja hem vist, no és trivial), sovint es fan servir corbes estandarditzades que han estat validades per experts i per a les quals el nombre de punts és conegut. Existeixen diferents organismes que han estandarditzat corbes el·líptiques per a usos criptogràfics, com ara el NIST, el consorci alemany Brainpool o Certicom Research. Aquestes corbes han estat suposadament generades per a evitar els atacs coneguts i són suposadament segures per a usos criptogràfics. Però fer ús d'una corba dissenyada per un tercer implica confiar que aquest tercer no l'ha dissenyada malintencionadament i, com es descriu a l'inici d'aquest capítol (Secció 8.1), això històricament no sempre ha estat així. Per aquest motiu, alguns estàndards inclouen corbes seleccionades pseudoaleatòriament utilitzant un algorisme que permet a tercers verificar que realment s'ha seguit aquest algorisme per a generar-les. D'aquesta manera, hom pot verificar que la corba ha estat generada pseudoaleatòriament (cosa que en dificulta la possible inclusió de portes del darrere) i alhora confiar que la corba compleix els requisits de seguretat mínims (doncs ha passat per un escrutini públic extens abans de ser incorporada a l'estàndard).

L'ús de corbes estandarditzades pot simplificar els algorismes de generació de claus (que veurem més endavant en aquest mateix capítol) i també permet en certes ocasions precalcular valors necessaris per a la creació de signatures digitals. Així, es poden aprofitar moments en què el processador no està ocupat per precalcular aquests valors (basats en els paràmetres del domini especificats pels estàndards) i fer-los servir quan es requereixi generar una signatura, reduint així el temps necessari per calcular-la.

8.4.1 Selecció verificablement pseudoaleatòria de corbes

Alguns estàndards recomanen l'ús de corbes el·líptiques que s'han generat pseudoaleatòriament, seguint un procediment que permet a terceres parts comprovar que efectivament s'han generat seguint aquest procediment. En aquesta secció presentarem el procediment de generació de corbes pseudoaleatòries verificable que es descriu a l'estàndard l'ANSI X9.62 i a l'estàndard NIST.FIPS.186-4.

El procediment consta de dos algorismes: l'algorisme de selecció, que selecciona una corba pseudoaleatòria, i l'algorisme de verificació, que permet a un tercer verificar que la corba s'ha triat amb l'algorisme de selecció.

L'algorisme de **selecció verificablement aleatòria** de corbes el·líptiques rep com a entrada un primer p que definirà el cos finit i una funció hash H de mida l bits. A partir d'aquests valors, l'algorisme retorna una corba el·líptica $E/\mathbb{Z}_p : y^2 = x^3 + ax + b$ i una llavor S que s'utilitza en el procés de verificació.

L'algorisme de selecció verificablement aleatòria de corbes el·líptiques segueix els passos següents:

1. Es calcula $t = \lceil \log_2 p \rceil$, $s = \lfloor \frac{t-1}{l} \rfloor$ i $v = t - sl$.
2. Es genera una cadena binària aleatòria S de g bits, amb $g \geq l$.
3. Es calcula $c_0 = H(S)_{[l-v \dots l]}$, és a dir, c_0 correspon als v últims bits del hash de la llavor S.
4. Es calcula $W_0 = 0 \mid\mid c_{0[1 \dots v]}$, és a dir, W_0 és el resultat de fixar el primer bit de c_0 a 0.
5. Per i des de 1 fins a s:
   - Es calcula $s_i = (S + i) \mod 2^g$.
   - Es calcula $W_i = H(s_i)$. A l'hora de calcular el hash, el valor s_i es representa com una cadena binària de g bits.
6. Es calcula $r = W_0 \mid\mid W_1 \mid\mid \dots \mid\mid W_s$.
7. Si $r = 0$ o bé $4r + 27 = 0 \mod p$, es torna al pas 2.
8. Es trien valors a i $b \in \mathbb{Z}_p$ arbitraris, de manera que com a mínim un d'ells sigui diferent de 0 i tals que $r \cdot b^2 = a^3 \mod p$.
9. L'algorisme retorna els coeficients a i b seleccionats i la llavor S generada al pas 2.

Notació

La notació $X_{[a \dots b]}$ indica els bits des de la posició a fins a la posició b del valor X. El símbol $\mid\mid$ expressa la concatenació.

Així, per exemple, per al valor $X = 101100$ tenim que $X_{[0 \dots 2]} = 10$, $X_{[3 \dots 6]} = 100$ i $0 \mid\mid X_{[1 \dots 6]} = 001100$.

La sortida de l'algorisme són els coeficients de la corba $E/\mathbb{Z}_p$ seleccionada (amb el valor p especificat a l'entrada) i la cadena binària S que s'ha fet servir com a llavor del procés pseudoaleatori. Aquesta llavor S es farà servir posteriorment en el procés de verificació, i permet assegurar que els coeficients de la corba no s'han dissenyat manualment.

Els paràmetres que defineixen la corba (els coeficients a i b) queden gairebé determinats pel valor r. En concret, per a un valor d'r fixat, hi ha essencialment dos possibles valors a triar per al parell a, b. Com que r es deriva de l'aplicació d'una funció hash, no és computacionalment factible per a un atacant trobar una llavor S que generi uns paràmetres a, b seleccionats manualment per l'atacant.

L'elecció d'a i b

El lector interessat a entendre perquè r determina els valors a i b pot consultar la *Guide to Elliptic Curve Cryptography* de Darrel Hankerson, Alfred Menezes i Scott Vanstone.

És interessant notar que les condicions validades al pas 7 ($r \neq 0$ i $r \neq -\frac{27}{4} \mod p$) permeten assegurar que el discriminant de la corba és diferent de zero, ja que $r = \frac{a^3}{b^2} \mod p$ per construcció (pas 8).

L'algorisme que acabem de descriure selecciona una corba pseudoaleatòria de manera que després es pugui verificar que s'ha seleccionat així. Ara bé, l'algorisme de selecció no té en compte les condicions necessàries per a que la corba sigui segura per a usos criptogràfics. Per tal de generar uns paràmetres de domini segurs, es procedeix a executar l'algorisme anterior i, a continuació, es comprova que la corba generada compleixi les tres condicions que la fan criptogràficament forta (especificades a la secció anterior). Si no les compleix, es torna a executar l'algorisme tantes vegades com calgui, fins a aconseguir seleccionar uns paràmetres adequats.

A més a més de la corba el·líptica E, la generació de paràmetres de domini per a criptografia de corbes el·líptiques requereix d'un punt base G. Per tal de calcular aquest punt, es tria un punt $G' \in E$ arbitrari i es calcula $G = hG'$, on $h = \#E/n$ (recordeu que n és el primer gran que divideix $\#E$). L'única condició que cal que G satisfaci és que sigui diferent de $\mathcal{O}$ (si no ho és, simplement es torna a repetir el procediment fins a trobar un $G \neq \mathcal{O}$). L'ordre del punt G és n.

Donada una corba E (especificada pels paràmetres a, b i p) i la llavor S generades per l'algorisme de generació de corbes verificablement aleatòries, juntament amb la funció hash H utilitzada, l'algorisme de **verificació de la selecció aleatòria** permet comprovar que una corba ha estat creada seguint l'algorisme anteriorment.

L'algorisme consta dels passos següents:

1. S'executen els passos 1 i de 3 a 6 de l'algorisme de selecció. El pas 2 no és necessari, ja que el valor S que s'utilitza és el que es rep a l'entrada.
2. Es comprova que $r \cdot b^2 = a^3 \mod p$. Si la comprovació és satisfactòria, l'algorisme dona per vàlida la corba. En cas contrari, la verificació falla.

Noteu com l'ús de l'algorisme de selecció verificablement aleatòria en la creació de corbes dificulta la creació de corbes especialment vulnerables, ja que els paràmetres d'aquestes es deriven d'una funció hash i, per tant, no es poden construir deliberadament vulnerables. Ara bé, ningú no impedeix a un possible generador de corbes malintencionat d'executar l'algorisme repetidament fins a trobar una corba que compleixi alguns requisits que li siguin d'interès. És per això que l'ús d'aquest algorisme en la creació de corbes estandarditzades no està totalment lliure de sospita.

8.4.2 Corbes estandarditzades

Les corbes generades i publicades per organismes d'estandardització es coneixen com a corbes estàndard o bé com a corbes amb nom.

La Taula 8.2 anomena les corbes amb nom més conegudes així com l'estàndard que les defineix:

Taula 8.2: Estàndards que inclouen la definició de corbes el·líptiques.

Organisme	Estàndard	Corbes sobre $\mathbb{Z}_p$
NIST	FIPS PUB 186-4: Digital Signature Standard (DSS)	P-192, P-224, P-256, P-384, P-521
Certicom Research	SEC 2: Recommended Elliptic Curve Domain Parameters	secp192k1, secp192r1, secp224k1, secp224r1, secp256k1, secp256r1, secp384r1, secp521r1
Brainpool	ECC Brainpool Standard Curves and Curve Generation	brainpoolP160r1, brainpoolP192r1, brainpoolP224r1, brainpoolP256r1, brainpoolP320r1, brainpoolP384r1, brainpoolP512r1

El NIST, en el seu estàndard per a signatures digitals (*FIPS PUB 186-4: Digital Signature Standard (DSS)*) publicat al juliol de 2013 defineix cinc corbes el·líptiques sobre $\mathbb{Z}_p$, fent servir primers de diferents mides (192, 224, 256, 384 i 521 bits). Les corbes s'anomenen anteposant el prefix *P-* a la mida. Aquest prefix indica que les corbes estan definides sobre cossos finits d'ordre primer. Les corbes han estat generades amb l'algorisme de generació de corbes verificablement aleatòries descrit a la secció anterior, fent servir SHA-1 com a funció hash.

Totes elles tenen en comú que fan servir el coeficient $a = -3$, el que permet optimitzar la suma de punts en una de les representacions habituals. A més, els primers p seleccionats són primers de quasi-Mersenne, que permeten optimitzar la reducció modular. Les cinc corbes també tenen en comú el cofactor, $h = 1$.

Primers de quasi-Mersenne	Els nombres primers de quasi-Mersenne són primers de la forma $2^n - c$, amb $c \ll 2^n$. Per exemple, $2^{192} - (2^{64}+1)$ és un primer de quasi-Mersenne, ja que $(2^{64}+1) \ll 2^{192}$.

A tall d'exemple, a continuació es detallen els paràmetres de la corba P-256, una de les 5 corbes sobre $\mathbb{Z}_p$ definides a l'estàndard del NIST. L'especificació de la corba consta dels paràmetres següents: el mòdul primer p, l'ordre primer n de la corba, la llavor S que s'ha utilitzat en l'algorisme de generació de corbes verificablement aleatòries per tal de generar-la, el valor c que correspon al valor r calculat en l'algorisme de generació, el coeficient b de l'equació de la corba (el coeficient a és -3 per a totes elles), i les coordenades x i y del punt base G.

La corba P-256 del NIST queda definida pels paràmetres següents:

$$
\begin{aligned}
p &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\
n &= 115792089210356248762697446949407573529996955224135760342422259061068512044369 \\
SEED &= \texttt{(0x) c49d3608 86e70493 6a6678e1 139d26b7 819f7e90} \\
c &= \texttt{(0x) 7efba166 2985be94 03cb055c 75d4f7e0 ce8d84a9 c5114abc af317768} \\
&\quad \texttt{0104fa0d} \\
b &= \texttt{(0x) 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e} \\
&\quad \texttt{27d2604b} \\
G_x &= \texttt{(0x) 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945} \\
&\quad \texttt{d898c296} \\
G_y &= \texttt{(0x) 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068} \\
&\quad \texttt{37bf51f5}
\end{aligned}
$$

Exemple 8.11 Exemple de verificació de la generació aleatòria d'una corba el·líptica

A continuació procedirem a verificar que la corba del NIST P-256 ha estat efectivament generada pseudoaleatòriament amb l'algorisme descrit anteriorment.

Com que la generació ha fet servir SHA-1 com a funció hash, aleshores $l = 160$.

1. Es calcula $t = \lceil \log_2 p \rceil = 256$, $s = \lfloor \frac{t-1}{l} \rfloor = \lfloor \frac{256-1}{160} \rfloor = 1$ i $v = t - sl = 96$.
2. La llavor S és `0xc49d360886e704936a6678e1139d26b7819f7e90`, amb $g = 160 \geq l$.
3. Es calcula:

$$
\begin{aligned}
c_0 &= H(S)_{[l-v\ldots l]} = \\
&= \texttt{0x3f07c5eac96ecc0bfefba1662985be9403cb055c}_{[64\ldots160]} = \\
&= \texttt{0xfefba1662985be9403cb055c}
\end{aligned}
$$

4. Es calcula $W_0 = 0 \,||\, c_{0[1\ldots v]} = \texttt{0x7efba1662985be9403cb055c}$.
5. Per i des de 1 fins a s:

$$s_1 = (S+1) \mod 2^{160} = \texttt{0xc49d360886e704936a6678e1139d26b7819f7e91}$$

$$
\begin{aligned}
W_1 = H(s_1) &= H(\texttt{0xc49d360886e704936a6678e1139d26b7819f7e91}) = \\
&= \texttt{0x75d4f7e0ce8d84a9c5114abcaf3177680104fa0d}
\end{aligned}
$$

6. Es calcula:

$$r = W_0||W_1 =$$
$$= \texttt{0x7efba1662985be9403cb055c75d4f7e0ce8d84a9c5114abcaf3177680104fa0d}$$

7. Es comprova que $r \cdot b^2 = a^3 \mod p$:

$$\texttt{0x7efb...fa0d} \cdot \texttt{0x5ac6...604b}^2 = (-3)^3 \mod 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$

Com que la igualtat es compleix, podem verificar que la corba ha estat generada pseudoaleatòriament seguint l'algorisme descrit.

Fixeu-vos que com que $s = 1$, només s'executa una única iteració del bucle que calcula les W_i. A més, cal tenir en compte que al calcular els hashos (pas 3 i pas 5), cal representar les entrades de la funció hash (els valors S i s_1) com a cadenes binàries.

Exercici 8.8 Verifiqueu que la corba del NIST P-192 ha estat generada pseudoaleatòriament amb l'algorisme verificable de generació de corbes.

La corba P-192 del NIST queda definida pels paràmetres següents:

$$
\begin{aligned}
p &= 2^{192} - 2^{64} - 1\\
n &= 6277101735386680763835789423176059013767194773182842284081\\
SEED &= \texttt{(0x) 3045ae6f c8422f64 ed579528 d38120ea e12196d5}\\
c &= \texttt{(0x) 3099d2bb bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65}\\
b &= \texttt{(0x) 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1}\\
G_x &= \texttt{(0x) 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012}\\
G_y &= \texttt{(0x) 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811}
\end{aligned}
$$

Certicom Research, a l'estàndard *SEC 2: Recommended Elliptic Curve Domain Parameters* publicat el gener de 2010 (versió 2.0), descriu vuit corbes el·líptiques sobre $\mathbb{Z}_p$, fent servir primers de les mateixes mides que l'estàndard del NIST (192, 224, 256, 384 i 521 bits). En general, per a cada mida del primer, l'estàndard descriu dos tipus de corbes: una corba generada pseudoaleatòriament de manera verificable i una corba de Koblitz. L'excepció són les dues mides superiors (384 i 521 bits), per a les quals només es proporciona la corba pseudoaleatòria.

Els noms de les corbes del SEC 2 segueixen un patró que permet identificar-ne les característiques fàcilment: els primers tres caràcters són *sec*, que denota *Standards for Efficient Cryptography*; a continuació s'inclou una p, que descriu corbes sobre cossos finits $\mathbb{Z}_p$; després hi ha un número, que descriu la mida del primer utilitzat; a continuació hi ha una lletra, k o r, especificant si es tracta d'una corba aleatòria o de Koblitz; i finalment hi ha un número de seqüència.

Les corbes aleatòries (corbes r) especificades al SEC2 són equivalents a les corbes del NIST de la mateixa mida especificades a la Taula 8.2 (la corba P-192 és equivalent a la secp192r1, la corba P-224 a la secp224r1, etc.). Per tant, les corbes aleatòries que especifica el SEC 2 han estat generades amb l'algorisme verificable de generació de corbes fent servir SHA-1 com a funció hash.

Les corbes de Koblitz sobre cossos finits $\mathbb{Z}_p$ tenen la particularitat d'estar definides per uns paràmetres que permeten una implementació especialment eficient (el càlcul del doblat d'un punt en aquest tipus de corbes és molt eficient). Les corbes de Koblitz especificades al SEC 2 han estat seleccionades repetint el procés de seleccionar paràmetres eficients fins a aconseguir trobar uns paràmetres que descriguin una corba d'ordre primer i tenen $a = 0$, de manera que l'equació que les defineix és de la forma $E/\mathbb{Z}_p : y^2 = x^3 + b$.

Corbes de Koblitz

És important tenir en compte que el mot *corbes de Koblitz* és també utilitzat (i, de fet, més comunament) per a referir-se a corbes sobre $\mathbb{F}_{2^m}$. A l'estàndard de Certicom, es generalitza el terme i es fa servir per referir-se també a les corbes sobre un cos finit d'ordre primer que tenen una propietat concreta que permet la implementació eficient del càlcul del doblat d'un punt.

La corba *secp256k1*

La corba *secp256k1* és actualment una corba molt coneguda, ja que és la que fa servir la criptomoneda Bitcoin per a les signatures digitals que autoritzen les transaccions. Anteriorment al seu ús a Bitcoin, la corba era poc utilitzada. Els valors especialment petits dels coeficients de la corba semblen indicar que és una corba *nothing-up-my-sleeve*.

La corba *secp256k1* queda definida pels paràmetres següents:

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$
$$n = (\texttt{0x})\ \texttt{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141}$$
$$a = 0$$
$$b = 7$$
$$G_x = (\texttt{0x})\ \texttt{79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798}$$
$$G_y = (\texttt{0x})\ \texttt{483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8}$$
$$h = 1$$

L'estàndard *ECC Brainpool Standard Curves and Curve Generation* publicat el 2005 inclou set corbes el·líptiques de set mides diferents, quatre de les quals coincideixen amb les del NIST i Certicom. Les altres tres mides són 160, 320, i 512 bits (aquesta última substitueix les de 521 del NIST i Certicom). Totes elles han estat generades de manera verificablement pseudoaleatòria, amb un mètode similar al que s'ha explicat anteriorment.

A diferència de les corbes del NIST, les de Brainpool no fan servir primers de quasi-Mersenne, de manera que en general són menys eficients. Aquesta decisió va ser presa per tal d'evitar problemes de patents amb els algorismes de càlculs aritmètics ràpids.

De manera similar als altres estàndards, els noms de les corbes de Brainpool segueixen un patró que en descriu les seves propietats: els primers caràcters són *brainpool*, que denota l'organisme de certificació; a continuació s'inclou una P, que descriu corbes sobre cossos finits $\mathbb{Z}_p$; després hi ha un número, que descriu la mida del primer utilitzat; a continuació hi ha una lletra, r, que especifica que es tracta d'una corba aleatòria; i finalment hi ha un número de seqüència.

L'informe de 2017 sobre l'ús del TLS de F5 (*The 2017 TLS Telemetry report*) resumeix les observacions sobre més de 20 milions de *hosts* TLS repartits arreu del món. L'informe reporta que un 74% dels intercanvis de clau de Diffie-Hellman sobre corbes el·líptiques que es porten a terme fan servir la corba del NIST P-256, sent clarament per tant la més popular de totes les corbes estandarditzades. Part d'aquesta popularitat ve donada pel fet de ser la corba per defecte a l'OpenSSL en aquell moment i perquè el TLS 1.3 requereix que totes les implementacions suportin aquesta corba per a l'intercanvi de claus de Diffie-Hellman. La segona corba més popular és la *Curve25519*, una corba proposada pel criptògraf Daniel J. Bernstein sobre un primer de 256 bits (el primer $2^{255} - 19$, que li dona nom). Aquesta corba no està coberta per cap patent i la implementació de referència és codi lliure. El 2017 el NIST va anunciar que inclouria aquesta corba al seu estàndard de signatura digital i, de fet, la propera versió d'aquest, encara en format d'esborrany (*FIPS PUB 186-5 (Draft)*), ja la contempla.

8.4.3 Funcions hash que retornen punts de corbes el·líptiques

Alguns dels esquemes criptogràfics basats en corbes el·líptiques requereixen d'una funció hash que rebi una entrada arbitrària (per exemple, la cadena de caràcters a signar) i retorni un punt d'una corba el·líptica concreta. En aquesta secció descriurem com es poden construir aquestes funcions hash.

Donada una corba $E/\mathbb{Z}_p : y^2 = x^3 + ax + b$ i un punt $G \in E/\mathbb{Z}_p$ d'ordre n, una construcció ingènua d'una funció hash H que rebi com a entrada cadenes arbitràries i retorni punts de la corba $E/\mathbb{Z}_p$ és la següent:

$$H(m) = H'(m) \cdot G = P \in E/\mathbb{Z}_p$$

on H' és una funció hash que rep entrades arbitràries i retorna un valor a $\mathbb{Z}_n$.

És a dir, la funció hash H es construeix multiplicant un enter (resultant d'una altra funció hash) per un punt de la corba, de manera que s'obté un punt de la corba desitjada. Noteu que la funció H' és fàcil de construir. Per exemple, podríem definir H' com el resultat de la funció hash SHA-1 mòdul n (és a dir, $H'(m) = SHA\text{-}1(m) \mod n$).

Tanmateix, aquesta construcció presenta problemes de seguretat a diferents nivells, motiu pel qual no es fa servir a la pràctica. Més endavant (a l'Exercici 9.4) veurem un exemple concret d'aquests problemes de seguretat en l'ús d'aquesta construcció en les signatures BLS.

Una construcció més elaborada per a aconseguir funcions hash que retornin punts de corbes el·líptiques és el mètode d'intentar-i-incrementar (en anglès, es coneix amb el nom de *try-and-increment*). El mètode consisteix a interpretar el missatge m com a la coordenada x del punt de la corba i calcular el valor d'y corresponent, tenint en compte però dos detalls que podrien ser problemàtics. D'una banda, pot ser que un missatge donat m no correspongui a la coordenada x de cap punt de la corba. D'altra banda, donada una coordenada x, ja hem vist que existiran dos possibles valors d'y.

L'**algorisme de intentar-i-incrementar** permet convertir un missatge m en un punt d'una corba el·líptica $E/\mathbb{Z}_p : y^2 = x^3 + ax + b$ seguint els passos següents:

1. Inicialitzar el comptador a zero: $c = 0$.
2. Calcular $(x, s) = H''(c||m)$.
3. Calcular $t = x^3 + ax + b$.
4. Si t és un residu quadràtic mòdul p:
   (a) Calcular l'arrel quadrada de t: $y = (-1)^s \cdot t^{1/2} \mod p$.
   (b) Retornar el punt (x, y).
   En cas contrari:
   (a) Incrementar c: $c = c + 1$.
   (b) Tornar al pas 2.

La funció hash auxiliar H'' rep el valor d'un comptador concatenat amb el missatge i retorna dos valors, un valor a $\mathbb{Z}_p$, que correspon a la coordenada x del punt, i un bit s que es farà servir per decidir quin dels dos possibles valors per a la coordenada y se selecciona (en cas que efectivament existeixin). D'aquesta manera, si el primer valor x no correspon a cap punt de la corba, es pot incrementar el comptador i tornar-ho a intentar, esquivant així el primer dels detalls problemàtics que esmentàvem. D'aquest procediment en prové el nom de l'algorisme. Un cop calculat el valor x, calculem el valor y com l'arrel quadrada d'x, triant sempre el valor més petit dels dos possibles (es tria una ordenació qualsevol, ja que no és important per l'algorisme). Després, el bit s es fa servir per decidir si es retorna el valor més petit o es canvia per l'altre valor.

Exemple 8.12 Exemple d'execució de l'algorisme de intentar-i-incrementar

Calculem un punt de la corba $E/\mathbb{Z}_{11}: y^2 = x^3 - 5x + 5$ que representa el missatge *CRIPTOGRAFIA*. Per fer-ho, farem servir la següent funció hash auxiliar:

$$H''(m) = (SHA\text{-}1(m) \mod p, \quad SHA\text{-}1(m) \mod 2)$$

i triarem com a resultat de l'arrel quadrada el valor més petit considerat sobre els enters.

1. Inicialitzem el comptador a zero: $c = 0$.
2. Calculem la funció hash auxiliar:

$$\begin{aligned}(x,s) = H''(c||m) = H''(0||\texttt{CRIPTOGRAFIA}) =\\ &= (SHA\text{-}1(\texttt{0CRIPTOGRAFIA}) \mod 11, \quad SHA\text{-}1(\texttt{0CRIPTOGRAFIA}) \mod 2)\\ &= (\texttt{0x2f48}\cdots\texttt{8086} \mod 11, \texttt{0x2f48}\cdots\texttt{8086} \mod 2) =\\ &= (8,0)\end{aligned}$$

3. Calculem $t = x^3 + ax + b = 8^3 - 5 \cdot 8 + 5 = 477 \mod 11 = 4$.
4. El valor $t = 4$ és un residu quadràtic a $\mathbb{Z}_{11}$ (ja que $2^2 = 4 \mod 11$ i $9^2 = 4 \mod 11$).
   (a) Les dues arrels de 4 a $\mathbb{Z}_{11}$ són 2 i 9. Com que $2 < 9 \in \mathbb{Z}$, aleshores $y = (-1)^s \cdot t^{1/2} \mod p = (-1)^0 \cdot 2 \mod 11 = 2$.
   (b) Es retorna el punt $(8,2)$.

El punt $(8,2)$ pertany, per tant, a la corba E, i es pot fer servir com a representació del missatge *CRIPTOGRAFIA* en qualsevol algorisme criptogràfic que la faci servir en els seus paràmetres de domini.

8.5 El problema del logaritme discret sobre corbes el·líptiques

Utilitzant la multiplicació escalar sobre corbes el·líptiques i de manera anàloga al problema del logaritme discret a $\mathbb{Z}_p$, podem definir ara el problema del logaritme discret sobre una corba el·líptica (ECDLP, de l'anglès, *Eliptic Curve Discrete Logarithm Problem*).

Donada una corba el·líptica $E/\mathbb{Z}_p$ amb ordre $\#E$, un element primitiu $G \in E$ i un altre element $T \in E$, el problema del **logaritme discret sobre corbes el·líptiques** consisteix a trobar un enter $1 \leq i \leq \#E$ tal que:

$$i \cdot G = T$$

Notació

Convé esmentar que la formulació que es fa servir s'allunya una mica de la formulació tradicional del problema ja que s'ha triat la notació additiva per a expressar l'operació de grup entre els punts de la corba el·líptica. En canvi, si s'hagués fet servir la notació multiplicativa, la definició del logaritme discret seria més similar a la definició clàssica. Tot i això, el problema és equivalent.

Exercici 8.9 Calculeu el logaritme discret del punt $T = (0,7)$ en base $G = (8,9)$ per a la corba $E/\mathbb{Z}_{11}: y^2 = x^3 - 5x + 5$. Podeu fer servir el resultat de l'Exercici 8.3 per a resoldre aquesta activitat.

Per quantificar el nivell de dificultat del problema és, per tant, necessari conèixer el número de punts de la corba, $\#E$.

El problema del logaritme discret sobre corbes el·líptiques es pot generalitzar eliminant la restricció que el punt G hagi de ser primitiu. En aquest cas, donat un punt base G d'ordre n primer, cal trobar l'enter $1 \leq i \leq n$ tal que $i \cdot G = T$. El logaritme discret es calcula doncs sobre el subgrup cíclic de E generat per G, al qual el punt T també pertany. L'ordre d'aquest punt, n, passa a tenir importància en la seguretat dels protocols que se'n deriven.

El millor algorisme que es coneix actualment (2025) per solucionar el problema del logaritme discret sobre corbes el·líptiques (excepte per casos molt específics o assumint l'existència d'ordinadors quàntics) és l'algorisme ro de Pollard. Aquest algorisme permet calcular el logaritme discret amb un temps d'execució $\mathcal{O}(\sqrt{n})$, on n és l'ordre del punt. D'aquí se'n deriven els nivells de seguretat que ofereixen les corbes el·líptiques per a diferents mides de la clau que s'han detallat a la comparativa de la Taula 8.1.

Rècord de còmput	El rècord de còmput de l'ECDLP amb l'algorisme ro de Pollard és sobre una corba definida sobre un cos primer de 112 bits de l'any 2012.

8.6 Criptografia basada en el problema del logaritme discret sobre corbes

En aquesta secció es descriuen algorismes criptogràfics que es basen en el problema del càlcul del logaritme discret sobre corbes el·líptiques. En primer lloc, veurem la versió de l'intercanvi de claus de Diffie-Hellman sobre corbes. A continuació, presentarem l'algorisme de signatura ECDSA. Finalment, descriurem l'algorisme de xifratge ECIES.

8.6.1 Intercanvi de claus de Diffie-Hellman amb corbes el·líptiques

L'algorisme d'intercanvi de claus de Diffie-Hellman sobre corbes el·líptiques (ECDH, de l'anglès, *Elliptic Curve Diffie-Hellman Key Exchange*) és una variant de l'algorisme d'intercanvi de claus de Diffie-Hellman (que ja hem presentat al Capítol 6) que treballa sobre corbes el·líptiques fent servir les operacions de *suma* i *multiplicació* que hem definit en aquest capítol. De la mateixa manera que la variant clàssica, l'objectiu de l'ECDH és aconseguir que dos usuaris que es comuniquen per un canal insegur derivin una clau compartida.

El procés d'inicialització de l'algorisme consisteix en la selecció dels paràmetres de domini: els elements que defineixen la corba el·líptica E (un primer p i els coeficients a i $b \in \mathbb{Z}_p$) i un element $G = (x_g, y_g) \in E$ d'ordre n.

Una vegada fixats els paràmetres de domini, l'algorisme d'intercanvi de claus s'executa de manera anàloga a la variant clàssica:

L'algorisme d'**intercanvi de claus de Diffie-Hellman sobre corbes el·líptiques** entre dos usuaris, A i B, consta dels passos següents:

1. A tria un valor aleatori $k_{privA} = a \in (1, \ldots, n)$ i calcula $k_{pubA} = a \cdot G = (x_a, y_a)$.
2. B tria un valor aleatori $k_{privB} = b \in (1, \ldots, n)$ i calcula $k_{pubB} = b \cdot G = (x_b, y_b)$.
3. A i B intercanvien els seus valors k_{pub}, és a dir, A envia a B el valor k_{pubA} i B envia a A el valor k_{pubB}.
4. A deriva la clau compartida $k_{AB} = k_{pubB} \cdot k_{privA} = (x_{AB}, y_{AB})$.
5. B deriva la clau compartida $k_{AB} = k_{pubA} \cdot k_{privB} = (x_{AB}, y_{AB})$.

Les claus privades (els valors a i b) són enters, mentre que tant les claus públiques (k_{pubA} i k_{pubB}) com la clau compartida (k_{AB}) són punts de la corba el·líptica. Per tant, a cada pas (a excepció del pas 3) es calcula una multiplicació escalar.

Així doncs, efectivament, el punt k_{AB} derivat per les dues parts participants en el protocol és el mateix, ja que, d'una banda, l'usuari A calcula:

$$k_{AB} = k_{privA} \cdot k_{pubB} = a(bG)$$

i, d'altra banda, l'usuari B calcula:

$$k_{AB} = k_{privB} \cdot k_{pubA} = b(aG)$$

Com que la suma de punts és associativa, les dues parts calculen el mateix valor.

Pel que fa a la seguretat davant d'un atacant que escolti el canal, l'atacant coneixerà els dos valors intercanviats pel canal, k_{pubA} i k_{pubB}, a més dels paràmetres públics que defineixen E (a, b i p) i el punt G, però calcular k_{AB} a partir d'aquests valors és un procés computacionalment difícil.

Exemple 8.13 Exemple d'intercanvi de claus de Diffie-Hellman amb corbes el·líptiques

Els usuaris A i B disposen d'un canal insegur amb el qual comunicar-se, i volen aconseguir crear una clau compartida k_{AB}:

Se seleccionen els paràmetres de domini $a = 3, b = 8, p = 23$ (i, per tant, $E/\mathbb{Z}_{23} : y^2 = x^3 + 3x + 8$) i $G = (19,1) \in E/\mathbb{Z}_{23}$ (de manera que $\#E = n = 29$).

1. A tria el valor aleatori $k_{privA} = a = 15$ i calcula:
   $k_{pubA} = a \cdot G = 15(19,1) = (6,14)$.
2. B tria un valor aleatori $k_{privB} = b = 18$ i calcula:
   $k_{pubB} = b \cdot G = 18(19,1) = (13,17)$.
3. A i B intercanvien els seus valors k_{pub}.
4. A deriva la clau compartida, calculant:
   $k_{AB} = k_{pubB} \cdot k_{privA} = 15(13,17) = (17,2)$.
5. B deriva la clau compartida, calculant:
   $k_{AB} = k_{pubA} \cdot k_{privB} = 18(6,14) = (17,2)$.

El protocol finalitza amb A i B compartint el mateix valor secret $k_{AB} = (17,2)$.

8.6.2 L'esquema de signatura ECDSA

L'ECDSA (per les seves sigles en anglès, *Elliptic Curve Digital Signature Algorithm*) és una variant de l'algorisme de signatura DSA (Digital Signature Algorithm) que es basa en el problema del logaritme discret sobre corbes el·líptiques. L'ECDSA és anàleg al DSA, però operant sobre corbes el·líptiques en comptes de sobre els enters. L'ECDSA va ser proposat l'any 1992 per Scott Vanstone, i des d'aleshores ha estat inclòs en diversos estàndards.

L'algorisme de generació de claus de l'ECDSA coincideix amb els primers passos de l'intercanvi de claus de Diffie-Hellman amb corbes el·líptiques.

En primer lloc, s'executa el procés d'inicialització, en què se seleccionen els paràmetres de domini: la corba $E/\mathbb{Z}_p$ (amb mòdul p i coeficients a i b) i un punt G que genera un grup cíclic d'ordre primer n. Els paràmetres de domini són doncs la tupla (p,a,b,G,n). Si es fan servir corbes estandarditzades, no cal executar el procés d'inicialització, ja que els paràmetres de domini ja venen definits.

Una vegada generats els paràmetres de domini, es procedeix a la generació de claus:

L'algorisme de **generació de claus ECDSA** consta dels passos següents:

1. Es tria un enter aleatori $k_{priv} = d \in_R (1, n)$.
2. Es calcula $k_{pub} = B = d \cdot G$.
3. La clau pública k_{pub} és el punt B, mentre que la clau privada k_{priv} és el valor d.

Les claus generades per l'algorisme anterior es poden fer servir per a generar i validar signatures digitals fent servir els algorismes següents:

A partir d'un missatge en clar m, la clau privada de l'emissor $k_{priv} = d$, i els paràmetres de domini (p, a, b, G, n), es calcula la **signatura digital ECDSA** del missatge:

1. Es tria una clau efímera $k_e \in_R (0, n)$.
2. Es calcula $R = k_e \cdot G$. Com que R és un punt de la corba el·líptica, tenim que $R = (x_R, y_R)$.
3. Es calcula $r = x_R \mod n$. Si $r = 0$, es torna al pas 1.
4. Es calcula $e = H(m)$.
5. Es calcula $s = (e + d \cdot r) \cdot k_e^{-1} \mod n$. Si $s = 0$, es torna al pas 1.
6. La signatura és la tupla (r, s).

L'algorisme de signatura ECDSA requereix de l'ús d'una funció hash, al pas 4, que s'aplica al missatge abans de signar-lo, com és habitual en les signatures digitals. Per a l'algorisme ECDSA la funció hash ha de retornar un enter i, per tant, no és necessari fer servir un esquema que permeti obtenir punts a partir de funcions hash (a diferència d'altres esquemes criptogràfics que veurem més endavant).

En els passos 3 i 5 es fan comprovacions sobre els valors que formen part de la signatura digital, i que asseguren que la signatura resultant de l'execució no contingui zeros. Això permet evitar certs atacs.

Com que els tres primers passos de la signatura no requereixen l'ús del missatge m a signar, si es fan servir corbes estandarditzades (i, per tant, els paràmetres de domini són coneguts anticipadament) es poden precalcular valors r aprofitant moments en què la càrrega del sistema sigui baixa i el processador estigui disponible. Aquests valors poden ser utilitzats després, quan es requereixi fer una signatura digital, reduint el temps de còmput necessari per calcular-la.

La variant de l'algorisme de signatura ECDSA que acabem de descriure és probabilística. Per a un mateix missatge i una mateixa clau privada, existeixen diverses signatures vàlides. Aquesta variabilitat s'introdueix en la generació d'una clau efímera (k_e), que se selecciona aleatòriament i que serà diferent per a cada signatura. De fet, és indispensable per a la seguretat de l'esquema que la clau efímera sigui única, doncs múltiples signatures fetes amb una mateixa clau privada i una mateixa clau efímera rebel·len la clau privada. Quan no es pot assegurar que es disposa d'una font d'aleatorietat prou bona per a complir aquest requisit, s'utilitzen versions deterministes de l'ECDSA, que deriven la clau efímera del missatge a signar.

A partir d'un missatge en clar m, la clau pública k_{pub}, els paràmetres de domini $((p,a,b,G,n))$ i una signatura del missatge (r,s), els passos següents permeten **verificar una signatura ECDSA**:

1. Es verifica que r i s es trobin a l'interval $(0,n)$.
2. Es calcula $e = H(m)$.
3. Es calcula $w = s^{-1} \mod n$.
4. Es calcula $u_1 = w \cdot e \mod n$ i $u_2 = w \cdot r \mod n$.
5. Es calcula $P = u_1 \cdot G + u_2 \cdot B$. Com que P és un punt de la corba el·líptica, tenim que $P = (x_P, y_P)$.
6. Si $x_P = r \mod n$, aleshores la signatura és vàlida i la verificació finalitza correctament. En cas contrari, la signatura es considera invàlida i la verificació fracassa.

Noteu com l'algorisme de verificació de signatures ECDSA és correcte. Si la signatura (r,s) ha estat creada per un signant legítim, aleshores $s = (e + d \cdot r) \cdot k_e^{-1} \mod n$ (pas 5 de l'algorisme de signatura) i, per tant:

$$k_e = s^{-1}(e + d \cdot r) \mod n = s^{-1} \cdot e + s^{-1} \cdot d \cdot r \mod n = w \cdot e + w \cdot d \cdot r \mod n =$$
$$= u_1 + u_2 \cdot d \mod n$$

Aleshores, tenim que:

$$P = u_1 \cdot G + u_2 \cdot B = u_1 \cdot G + u_2 \cdot d \cdot G = G(u_1 + u_2 \cdot d) = G \cdot k_e$$

i, per tant, $x_P = r$.

Exemple 8.14 Exemple de signatura i verificació amb ECDSA

L'usuari A vol enviar el missatge $m = 567$ signat amb ECDSA a un altre usuari.

En primer lloc, se seleccionen els valors públics $a = -5, b = 5, p = 11$ ($E/\mathbb{Z}_{11} : x^3 - 5x + 5$) i $G = (2,6) \in E$ (de manera que $\#E = n = 17$).

A continuació, cal que A disposi d'un parell de claus ECDSA. L'algorisme de creació de claus és el mateix que en l'ECDH:

1. A tria el valor aleatori $k_{privA} = d = 5$.
2. A calcula $k_{pub} = B = a \cdot G = 5(2,6) = (8,9)$.
3. La clau pública k_{pub} és el punt $B = (8,9)$, mentre que la clau privada k_{priv} és el valor $d = 5$.

Per tal de signar el missatge $m = 567$, A executa l'algorisme de signatura. Per simplicitat, a l'exemple es farà sevir la funció identitat (que retorna a la sortida el valor que rep a l'entrada) com a funció hash H.

1. A tria una clau efímera $k_e = 10 \in_R (0,17)$.
2. A calcula $R = k_e \cdot G = 10(2,6) = (6,2) = (x_R, y_R)$.
3. A calcula $r = x_R \mod n = 6$. Com que $r \neq 0$, segueix l'execució de l'algorisme al pas següent.
4. A calcula $e = H(m) = 567$.
5. A calcula $s = (e + d \cdot r) \cdot k_e^{-1} \mod n = (567 + 5 \cdot 6) \cdot 10^{-1} \mod 17 = 597 \cdot 12 \mod 17 = 7$. Com que $s \neq 0$, segueix l'execució de l'algorisme al pas següent.
6. La signatura és la tupla $(r,s) = (6,7)$.

A partir del missatge en clar $m = 567$, la clau pública k_{pub}, els paràmetres de domini, i la signatura del missatge $(r,s) = (6,7)$, el receptor del missatge pot verificar-ne la signatura:

1. Verifica que $6 \in (0,17)$ i $7 \in (0,17)$.

2. Calcula $e = H(m) = 567$.
3. Calcula $w = s^{-1} \mod n = 7^{-1} \mod 17 = 5$.
4. Calcula $u_1 = w \cdot e \mod n = 5 \cdot 567 \mod 17 = 13$ i $u_2 = w \cdot r \mod q = 5 \cdot 6 \mod 17 = 13$.
5. Calcula $P = u_1 \cdot G + u_2 \cdot B = 13(2,6) + 13(8,9) = (6,2) = (x_P, y_P)$.
6. Comprova si $x_P = r \mod q$. Com que efectivament $6 = 6 \mod 17$, la signatura és vàlida.

Exercici 8.10 Donada la corba i claus generades per l'algorisme de generació de claus de l'Exemple 8.14 i la mateixa funció hash que en aquest exemple, verifiqueu la validesa de la signatura ECDSA $(10,3)$ per al missatge $m = 876$.

Exercici 8.11 Donada la corba $E/\mathbb{Z}_{23} : y^2 = x^3 + 3x + 8$ i el punt base $G = (13,6)$ d'ordre 29:

1. genereu un parell de claus ECDSA fent servir aquests paràmetres de domini,
2. proporcioneu una signatura amb ECDSA del missatge $m = 25504446$ considerant com a funció hash la funció $H(x) = x \mod 10$, i
3. valideu la signatura que acabeu de generar.

8.6.3 L'esquema de xifratge integrat de corbes el·líptiques (ECIES)

L'esquema de xifratge integrat de corbes el·líptiques (ECIES) (de l'anglès, *Elliptic Curve Integrated Encryption Scheme*) va ser proposat per Mihir Bellare, Michel Abdalla i Phillip Rogaway a finals dels noranta i és l'algorisme de xifratge basat en corbes el·líptiques més estès actualment.

Diferents versions de l'ECIES (similars però no totalment compatibles entre elles) es troben estandarditzades per diversos organismes als estàndards ANSI X9.63, SEC 1, ISO/IEC 15946-3, i IEEE P1363a.

L'ECIES és un algorisme de xifratge híbrid, que fa ús de la criptografia de clau pública per a generar una clau que s'utilitza en un algorisme de xifratge simètric. D'aquesta manera, s'aprofita l'eficiència del xifratge simètric i les propietats que ofereix la criptografia de clau pública.

A grans trets, el funcionament de l'ECIES es basa en generar una clau secreta compartida entre emissor i receptor de manera anàloga a l'algorisme de Diffie-Hellman i a partir d'aquesta clau se'n deriven dues claus simètriques. La primera d'aquestes claus simètriques es fa servir per a xifrar el missatge fent servir un criptosistema de clau simètrica, i la segona es fa servir per a autenticar el text xifrat.

L'ECIES fa servir tres primitives criptogràfiques: un algorisme de xifratge de clau simètrica (tal que $m = D_k(E_k(m))$, una funció de derivació de clau (KDF) i un codi d'autenticació de missatges (MAC).

A partir d'un missatge en clar m, la clau pública $k_{pub} = B$ (generada amb el mateix algorisme de generació de claus que per a l'ECDSA) i els paràmetres de domini (p,a,b,G,n,h), els passos següents permeten **xifrar el missatge amb ECIES** (l'esquema de xifratge integrat de corbes el·líptiques):

1. Es tria aleatòriament una clau efímera $k_e \in_R (0,n)$.
2. Es calcula $R = k_e \cdot G$ i $Z = h \cdot k_e \cdot B$. Si $Z = \mathcal{O}$, es torna al pas 1. Com que Z és un punt de la corba el·líptica, tenim que $Z = (x_Z, y_Z)$.
3. Es calcula $(k_1, k_2) = KDF(x_Z, R)$.
4. Es calcula $C = E_{k_1}(m)$.
5. Es calcula $t = MAC_{k_2}(C)$.
6. Es retornen els valors (R, C, t).

Noteu com del secret compartit Z se'n deriva la clau simètrica k_1, que es fa servir per a xifrar el missatge, i la clau simètrica k_2, que es fa servir per autenticar-lo.

A partir d'un missatge xifrat (R,C,t), la clau privada $k_{priv} = (d)$, i els paràmetres de domini (p,a,b,G,n), els passos següents permeten **desxifrar el missatge amb ECIES** (l'esquema de xifratge integrat de corbes el·líptiques):

1. Es valida que R sigui un punt vàlid de la corba diferent de $\mathcal{O}$.
2. Es calcula $Z = h \cdot d \cdot R$. Si $Z = \mathcal{O}$, es rebutja el text xifrat. Com que Z és un punt de la corba el·líptica, tenim que $Z = (x_Z, y_Z)$.
3. Es calcula $(k_1,k_2) = KDF(x_Z,R)$.
4. Es calcula $t' = MAC_{k_2}(C)$. Si $t \neq t'$, es rebutja el text xifrat.
5. Es calcula $m = D_{k_1}(C)$.
6. Es retorna el text en clar m.

És interessant notar com el punt Z es deriva del secret de Diffie-Hellman compartit entre emissor i receptor (l'emissor calcula $k_e \cdot B$ fent servir la clau pública del receptor B i el receptor calcula $d \cdot R$ fent servir la seva clau privada).

Si el text xifrat (R,C,t) és realment el resultat de xifrar m seguint l'algorisme de xifratge, aleshores:

$$Z = h \cdot d \cdot R = h \cdot d \cdot k_e \cdot G = h \cdot k_e \cdot d \cdot G = h \cdot k_e \cdot B$$

i, per tant, el receptor deriva les mateixes dues claus simètriques k_1 i k_2 que l'emissor ha utilitzat per xifrar, i pot validar i desxifrar el missatge.

8.7 Resum

En aquest capítol s'ha presentat la criptografia de corbes el·líptiques. En primer lloc, s'ha explicat el seu origen i els seus avantatges en relació a la criptografia de clau pública tradicional. A continuació, s'ha descrit com es fan servir les corbes el·líptiques per crear grups d'interès per a la criptografia, s'han presentat els principals estàndards que cobreixen la criptografia de corbes el·líptiques, i s'ha exposat la variant del problema del logaritme discret sobre corbes el·líptiques. Per últim, s'han detallat alguns dels esquemes criptogràfics més populars que fan servir aquest tipus de criptografia: l'intercanvi de claus de Diffie-Hellman sobre corbes el·líptiques, l'esquema de signatura ECDSA, i l'esquema de xifratge ECIES.

8.8 Solucions dels exercicis

Exercici 8.1:

A continuació es proposa una aproximació que permet resoldre el problema plantejat amb els coneixements exposats fins a la secció en què es proposa l'exercici. Més endavant s'expliquen altres alternatives més eficients per al càlcul dels punts d'una corba.

Els punts de la corba seran tots aquells que compleixin l'equació $y^2 = x^3 - 4x + 1$ a $\mathbb{Z}_5$. Per tant, en primer lloc avaluem l'expressió per a tots els possibles valors $x \in \mathbb{Z}_5$:

x	y^2
0	1
1	3
2	1
3	1
4	4

Ara, ens cal saber quins dels valors d'y^2 són residus quadràtics a $\mathbb{Z}_5$. Com que el cos és petit, els podem calcular per força bruta, provant tots els possibles casos:

y	y^2
0	0
1	1
2	4
3	4
4	1

Veiem doncs que els valors 1 i 4 són residus quadràtics, mentre que el 3 no ho és.

Per tant, trobem que els punts de la corba són:

$$[(0,1),(0,4),(2,1),(2,4),(3,1),(3,4),(4,2),(4,3),\mathcal{O}]$$

Exercici 8.2:

Com que $P_1 \neq P_2$, aleshores el pendent m és:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \mod p = \frac{7-1}{4-1} \mod 11 = \frac{6}{3} \mod 11 = 2$$

Després, es calculen les coordenades del punt:

$$x_3 = m^2 - x_1 - x_2 \mod p = 2^2 - 1 - 4 \mod 11 = 10$$

$$y_3 = m(x_1 - x_3) - y_1 \mod p = 2(1-10) - 1 \mod 11 = 3$$

Per tant, el resultat de la suma és $P_3 = (10,3)$. Efectivament, el resultat coincideix amb el càlcul fet gràficament a l'Exemple 8.4.

Exercici 8.3:

En primer lloc, calculem $P+P$. Per fer-ho, es calcula el valor del pendent m:

$$m = \frac{3x_1^2 + a}{2y_1} \mod p$$

$$m = \frac{3 \cdot 8^2 - 5}{2 \cdot 9} \mod 11 = \frac{187}{18} \mod 11 = 0$$

i després, es calculen les coordenades del punt:

$$x_3 = m^2 - x_1 - x_2 \mod p = 0 - 8 - 8 \mod 11 = 6$$
$$y_3 = m(x_1 - x_3) - y_1 \mod p = 0(8-6) - 9 \mod 11 = 2$$

Per tant, $2P = P + P = (6,2)$.

Seguint el mateix procediment, podem calcular la resta de punts:

$$\begin{aligned}
P &= (8,9) & 8P &= (4,4) & 15P &= (6,9)\\
2P &= P+P = (6,2) & 9P &= (4,7) & 16P &= (8,2)\\
3P &= P+P+P = (1,10) & 10P &= (2,5) & 17P &= \mathcal{O}\\
4P &= (0,4) & 11P &= (10,8) & 18P &= (8,9)\\
5P &= (7,4) & 12P &= (7,7) & 19P &= \cdots\\
6P &= (10,3) & 13P &= (0,7) & &\\
7P &= (2,6) & 14P &= (1,1) & &
\end{aligned}$$

Exercici 8.4:

L'invers del punt $P_1 = (x_1, y_1)$ serà el punt P_2 tal que $P_1 + P_2 = \mathcal{O}$.

D'una banda, sabem que aquest punt P_2 és precisament $P_2 = -P_1 = (x_1, -y_1) = (4, -7) = (4,4)$.

Alternativament, si prenem com a referència la solució de l'exercici anterior, veiem que:

$$P_1 = (4,7) = 9P$$
$$9P + 8P = 17P = \mathcal{O}$$
$$P_2 = 8P = (4,4)$$

Exercici 8.5:

Com que 9 no és primer, no tots els elements del grup són generadors. En particular, el punt $(4,5)$ no ho és, ja que no genera tots els 9 elements del grup:

$$\begin{aligned}
P &= (4,5)\\
2P &= P+P = (4,6)\\
3P &= P+P+P = \mathcal{O}
\end{aligned}$$

Exercici 8.6:

La representació binària de l'enter 12 és 1100. En primer lloc s'inicialitza `result` a $\mathcal{O}$. A continuació s'executen les iteracions:

Taula 8.3: Execució de l'algorisme de sumar i doblar

Iteració	e	Còmput
1	$e=0$	`point = point + point` = $(7,2)+(7,2)=(18,11)$
2	$e=0$	`point = point + point` = $(18,11)+(18,11)=(22,2)$
3	$e=1$	`result = result + point` = $\mathcal{O}+(22,2)=(22,2)$
		`point = point + point` = $(22,2)+(22,2)=(10,16)$
4	$e=1$	`result = result + point` = $(22,2)+(10,16)=(\mathbf{16},\mathbf{14})$

Per tant, $12P=(16,14)$.

Exercici 8.7:

Substituint els valors al teorema de Hasse tenim que:

$$|\#E-12|\leq 2\sqrt{11}$$
$$5.36\leq \#E\leq 18.63$$

Per tant, les corbes el·líptiques definides sobre $\mathbb{Z}_{11}$ tindran entre 6 i 18 punts.

Exercici 8.8:

1. Es calcula $t=\lceil\log_2 p\rceil=192$, $s=\lfloor\frac{t-1}{l}\rfloor=\lfloor\frac{192-1}{160}\rfloor=1$ i $v=t-sl=32$.
2. La llavor S és `0x3045ae6fc8422f64ed579528d38120eae12196d5`, amb $g=160\geq l$.
3. Es calcula:

$$\begin{aligned}c_0 &= H(S)_{[l-v\ldots l]}=\\ &= \texttt{0x7f6da10026c7ff92c5ac2e890bd59b44b099d2bb}_{[128\ldots160]}=\\ &= \texttt{0xb099d2bb}\end{aligned}$$

4. Es calcula $W_0=0\,||\,c_{0[1\ldots v]}=\texttt{0x3099d2bb}$.
5. Per i des de 1 fins a s:

$$s_1=(S+1)\mod 2^{160}=\texttt{0x3045ae6fc8422f64ed579528d38120eae12196d6}$$

$$\begin{aligned}W_1=H(s_1)&=H(\texttt{0x3045ae6fc8422f64ed579528d38120eae12196d6})=\\ &=\texttt{0xbfcb2538542dcd5fb078b6ef5f3d6fe2c745de65}\end{aligned}$$

6. Es calcula:

$$\begin{aligned}r&=W_0||W_1=\\ &=\texttt{0x3099d2bbbfcb2538542dcd5fb078b6ef5f3d6fe2c745de65}\end{aligned}$$

7. Es comprova que $r\cdot b^2=a^3\mod p$:

$$\texttt{0x3099...de65}\cdot\texttt{0x6421...b9b1}^2=(-3)^3\mod 2^{192}-2^{64}-1$$

   Com que la igualtat es compleix, podem verificar que la corba ha estat generada aleatòriament seguint l'algorisme descrit.

Exercici 8.9:

Com que la corba ($E: y^2=x^3-5x+5$ a $\mathbb{Z}_{11}$) coincideix amb la de l'exercici 8.3 i la base $G=(8,9)$ és precisament el punt P, aleshores podem observar directament de la solució de l'exercici 8.3 que:

$$13P=(0,7)=T$$

i, per tant, el logaritme és 13.

Exercici 8.10:

1. Es verifica que $10 \in (0,17)$ i $3 \in (0,17)$.
2. $e = H(m) = 876$.
3. $w = s^{-1} \mod q = 3^{-1} \mod 17 = 6$.
4. $u_1 = w \cdot e \mod q = 6 \cdot 876 \mod 17 = 3$ i $u_2 = w \cdot r \mod q = 6 \cdot 10 \mod 17 = 9$.
5. $P = u_1 \cdot G + u_2 \cdot B = 3(2,6) + 9(8,9) = (0,7) = (x_P, y_P)$.
6. Es comprova si $x_P = r \mod q$. Com que $0 \neq 10 \mod 17$, la signatura és invàlida.

Exercici 8.11:

1. La generació del parell de claus consta dels passos següents:

   1. Es tria el valor aleatori $k_{privA} = d = 6$.
   2. Es calcula $B = aG = 6(13,6) = (18,11)$.
   3. La clau pública k_{pub} és el punt $B = (18,11)$, mentre que la clau privada k_{priv} és el valor $d = 6$.

2. La signatura del missatge $m = 25504446$ consta dels passos següents:

   1. Es tria una clau efímera $k_e = 19 \in_R (0,29)$.
   2. Es calcula $R = k_e \cdot G = 19(13,6) = (12,1) = (x_R, y_R)$.
   3. Es calcula $r = x_R \mod n = 12$. Com que $r \neq 0$, segueix l'execució de l'algorisme al pas següent.
   4. Es calcula $e = H(m) = H(25504446) = 6$.
   5. Es calcula $s = (e + d \cdot r)k_e^{-1} \mod n = (6 + 6 \cdot 12)19^{-1} \mod 29 = 20 \cdot 26 \mod 29 = 27$. Com que $s \neq 0$, segueix l'execució de l'algorisme al pas següent.
   6. La signatura és la tupla $(r,s) = (12,27)$.

3. La verificació de la signatura $(r,s) = (12,27)$ per al missatge $m = 25504446$ consta dels passos següents:

   1. Es verifica que $12 \in (0,29)$ i $27 \in (0,29)$.
   2. Es calcula $e = H(25504446) = 6$.
   3. Es calcula $w = s^{-1} \mod q = 27^{-1} \mod 29 = 14$.
   4. Es calcula $u_1 = w \cdot e \mod q = 14 \cdot 6 \mod 29 = 26$ i $u_2 = w \cdot r \mod q = 14 \cdot 12 \mod 29 = 23$.
   5. Es calcula $P = u_1 \cdot G + u_2 \cdot B = 26(13,6) + 23(18,11) = (12,1) = (x_P, y_P)$.
   6. Es comprova si $x_P = r \mod q$. Com que efectivament $12 = 12 \mod 29$, la signatura és vàlida.

8.9 Bibliografia

American Bankers Association (1998). *ANSI X9.62: The Elliptic Curve Digital Signature Algorithm (ECDSA)*.

Antonopoulos, Andreas M. (2014). *Mastering Bitcoin: unlocking digital cryptocurrencies*. Capítol 4: Keys, addresses. O'Reilly Media, Inc..

Arjen, Lenstra; Thorsten, Kleinjung; i Thomé, Emmanuel (2013). *Universal security: from bits and mips to pools, lakes - and beyond*. Number Theory and Cryptography, Nov 2013, Darmstadt, Germany, pp.121-124.

Bernstein, Daniel. J.; Lange, Tanja; i Niederhagen, Ruben (2015). *Dual EC: a standardized back door*. The New Codebreakers, 256-281, Springer.

Boneh, Dan; i Victor Shoup (2020). *A graduate course in applied cryptography*.

Hales, Thomas C.(2013). *The NSA back door to NIST*. Notices of the AMS 61.2: 190-192.

Hankerson, Darrel; Menezes, Alfred J.; i Vanstone, Scott (2006). *Guide to elliptic curve cryptography*. Springer Science & Business Media.

Holmes, David (2018). *The 2017 TLS Telemetry Report*. F5.

Kerry, Cameron F.; i Charles Romine (2013). *NIST FIPS PUB 186-4: Digital Signature Standard (DSS)*.

Open University, The (2016). *Further pure mathematics: Group theory*.

Paar, Christof, and Jan Pelzl (2009). *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media.

Perlroth, Nicole (2013). *Government announces steps to restore confidence on encryption standards*. The New York Times.

Schwennesen, Ben; i Hubert Bray (2016). *Elliptic curve cryptography and government backdoors*.

Warburton, David (2019). *The 2019 TLS Telemetry Report*. F5.

9. Criptografia basada en pairings

En el capítol anterior s'han presentat les corbes el·líptiques i alguns dels seus usos en criptografia. Tot i això, no ens hem endinsat en una de les construccions més populars en els últims anys en criptografia basada en corbes el·líptiques: els *pairings*.

Algunes corbes el·líptiques tenen una estructura addicional anomenada aparellament (en anglès, es coneix com a *pairing*), que obre la porta a tot un nou conjunt d'eines criptogràfiques. En particular, els *pairings* es fan servir en criptografia per a atacar criptosistemes basats en el logaritme discret i també en la construcció de noves primitives criptogràfiques.

En aquest capítol descriurem les propietats dels *pairings* i la seva definició (tot explicant les eines matemàtiques necessàries per a construir-los), i veurem alguns algorismes criptogràfics basats en aquests.

9.1 Propietats dels *pairings*

Existeixen diferents tipus de *pairings*, però no tots són adequats per a usos criptogràfics. Els únics *pairings* coneguts que són útils en criptografia i, a més, eficientment computables, són els *pairings* de Weil i de Tate sobre corbes el·líptiques.

Més enllà de la seva definició explícita, que veurem més endavant, a continuació en descrivim les propietats que els caracteritzen.

Definició 9.1 Siguin $\mathbb{G}_0, \mathbb{G}_1$ i $\mathbb{G}_T$ grups cíclics d'ordre primer q amb $G_0 \in \mathbb{G}_0$ i $G_1 \in \mathbb{G}_1$ elements generadors dels grups. Diem que $\mathbb{G}_0$ i $\mathbb{G}_1$ són els grups d'origen i $\mathbb{G}_T$ el grup objectiu.

Un aparellament (conegut en anglès com a ***pairing***) és una aplicació $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ que satisfà dues propietats:

1. És **bilineal**, és a dir, per a tot $P_0, Q_0 \in \mathbb{G}_0$ i $P_1, Q_1 \in \mathbb{G}_1$:

$$e(P_0, P_1 + Q_1) = e(P_0, P_1) * e(P_0, Q_1)$$

$$e(P_0 + Q_0, P_1) = e(P_0, P_1) * e(Q_0, P_1)$$

2. És **no degenerat**, és a dir, $g_T = e(G_0, G_1)$ és un generador de $\mathbb{G}_T$.

És a dir, l'aplicació e és lineal per a les dues entrades. La bilinealitat dels *pairings* implica la següent propietat, en la qual es basen les construccions criptogràfiques que els fan servir:

$$e(\alpha P_0, \beta P_1) = e(P_0, P_1)^{\alpha\beta} = e(\beta P_0, \alpha P_1)$$

La propietat es deriva directament de la definició de bilinealitat:

$$e(\alpha P_0, \beta P_1) = e(P_0, \beta P_1)^{\alpha} = e(\alpha P_0, P_1)^{\beta} = e(P_0, P_1)^{\alpha\beta}$$

Notació

Noteu que s'ha fet servir notació additiva a $\mathbb{G}_0$ i $\mathbb{G}_1$ i multiplicativa a $\mathbb{G}_T$. Això és així ja que com veurem a continuació, els grups $\mathbb{G}_0$ i $\mathbb{G}_1$ estan definits per punts de corbes el·líptiques (i en aquest capítol anomenem suma a l'operació de grup) i, en canvi, el grup $\mathbb{G}_T$ serà un grup multiplicatiu.

Diem que un aparellament és **simètric** si $\mathbb{G}_0 = \mathbb{G}_1$ i **asimètric** en cas contrari.

La definició explícita dels *pairings* de Weil i de Tate és complexa i, per això, sovint s'obvia aquesta definició i es descriuen únicament les propietats que els caracteritzen. Entendre les propietats dels *pairings* és suficient per poder comprendre els esquemes criptogràfics que se'n deriven però, d'altra banda, entendre com es calculen ens permet aproximar-nos més detalladament a la seva estructura. Les dues seccions següents estan centrades en la definició explícita dels *pairings*: en primer lloc, es detallen un conjunt d'eines matemàtiques que permeten definir els *pairings*; a continuació, es presenta la definició explícita dels *pairings* de Weil i de Tate, tot exemplificant el càlcul amb una corba petita. El lector interessat pot doncs aprofundir en la construcció dels *pairings* seguint la lectura de la propera secció. D'altra banda, el lector que no desitgi aprofundir en la construcció dels *pairings*, pot passar directament a la Secció 9.4 per focalitzar-se directament en els algorismes criptogràfics que es basen en les propietats que proporcionen els *pairings*.

9.2 Eines matemàtiques per a la construcció dels *pairings*

En aquesta secció es presenten les eines matemàtiques que permeten definir explícitament els *pairings* de Weil i de Tate. En primer lloc, s'estén la definició de corbes el·líptiques que hem vist fins ara sobre $\mathbb{Z}_p$ a cossos finits amb un nombre d'elements no primer. A continuació, es defineix la r-torsió dels punts d'una corba el·líptica i s'observa l'estructura dels subgrups que conforma. Finalment, es descriu el concepte de divisor d'una funció i es presenta el seu ús en el context de la criptografia de corbes el·líptiques.

9.2.1 Corbes el·líptiques sobre cossos estesos

Fins ara hem fet servir corbes el·líptiques definides sobre cossos finits amb un nombre primer d'elements ($\mathbb{Z}_p$). Ara bé, també es poden construir corbes el·líptiques sobre altres cossos finits, com ara cossos amb un nombre d'elements potència d'un primer $\mathbb{F}_{p^d}$. Al capítol de *Fonaments matemàtics* ja hem vist com construir aquests cossos finits fent servir polinomis irreductibles. L'ús d'aquests cossos per a la creació de corbes el·líptiques és anàleg al cas de cossos finits amb nombre primer d'elements.

Exemple 9.1 Exemple de corba el·líptica sobre cos estès

Com ja hem vist a l'Exemple 8.6, la corba el·líptica $E/\mathbb{Z}_{11} : y^2 = x^3 - 5x + 5$ té 17 elements:

$$[\mathscr{O}, (0,4), (0,7), (1,1), (1,10), (2,5), (2,6), (4,4), (4,7), (6,2), (6,9), (7,4), (7,7), (8,2), (8,9), (10,3), (10,8)]$$

La corba està definida sobre $\mathbb{Z}_{11}$, un cos finit amb un nombre primer d'elements (11).

La corba el·líptica $E/(\mathbb{Z}_{11}/z^2+1) : y^2 = x^3 - 5x + 5$ té en canvi 119 elements. La corba està definida sobre el cos finit d'11^2 elements fent servir el polinomi z^2+1, irreductible a $\mathbb{Z}_{11}$ i de grau 2.

Alguns dels 119 elements d'aquesta corba el·líptica són:

$$[\mathscr{O}, (0,4), (0,7), (1,1), (1,10), (2,5), (2,6), (4,4), (4,7), (6,2), (6,9), (7,4), (7,7), (8,2), (8,9), (10,3), (10,8), (5,4z), (5,7z), (9,2z), (9,9z), (z+2, z+3), (z+2, 10z), (z+4, 4z+8), (z+4, 7z+3), \ldots]$$

Podem comprovar que aquests punts efectivament pertanyen a la corba verificant que compleixen l'equació que la defineix. Així, per exemple, per al punt $(5,4z)$, tenim que:

$$\begin{aligned}
y^2 &= x^3 - 5x + 5 \mod 11\\
(4z)^2 &= 5^3 - 5\cdot 5 + 5 \mod 11\\
16z^2 &= 105 \mod 11\\
5z^2 &= 6 \mod 11\\
5z^2 - 6 &= 0 \mod 11
\end{aligned}$$

Efectivament, el residu de dividir $5z^2 - 6$ entre z^2+1 a $\mathbb{Z}_{11}$ és 0 (ja que $5(z^2+1) = 5z^2 + 5 = 5z^2 - 6 \mod 11$).

9.2.2 Els punts de la r-torsió

En gran part d'aquest capítol hem treballat amb corbes el·líptiques definides sobre cossos finits amb un nombre primer d'elements. A la secció anterior hem vist que també podem definir corbes sobre cossos estesos amb ordre la potència d'un primer. A continuació veurem una de les construccions que es poden definir quan treballem amb grups sobre cossos estesos, els grups formats per r-torsions.

Definició 9.2 Sigui E una corba el·líptica definida sobre un cos finit $\mathbb{Z}_p$ i n un primer divisor de $\#E/\mathbb{Z}_p$. El **grau d'immersió** (en anglès, parlem d'*embedding degree*) d'E respecte a n és el menor enter k tal que n divideix $p^k - 1$.

Per al cas $n = \#E/\mathbb{Z}_p$, direm simplement que k és el grau d'immersió d'E.

Una vegada definit el grau d'immersió, passem a definir la r-torsió:

Definició 9.3 Sigui r un primer diferent de p. Es defineixen els punts de la **r-torsió**, $E[r]$, com el conjunt de punts P que pertanyen a $E/\mathbb{F}_{p^k}$ tals que $rP = \mathscr{O}$. És a dir,

$$E[r] = \{P \in E/\mathbb{F}_{p^k} \text{ tals que } rP = \mathscr{O}\}$$

amb k el grau d'immersió d'E respecte a r.

Exemple 9.2 Exemple de 3-torsió

La corba $E/\mathbb{Z}_{11} : y^2 = x^3 + 10x + 4$ té 15 elements ($\#E/\mathbb{Z}_{11} = 15$):

$$[\mathcal{O},(0,2),(0,9),(1,2),(1,9),(4,3),(4,8),(5,5),(5,6),(6,4),(6,7),(9,3),(9,8),(10,2),(10,9)]$$

El grau d'immersió d'E respecte a $n = 3$ (amb $3 \mid 15$) és $k = 2$, ja que 2 és el menor enter tal que $n \mid p^k - 1$. Noteu que per a $k = 1$ la condició no es compleix, ja que $3 \nmid 11^1 - 1$. En canvi, per a $k = 2$ tenim que $3 \mid 11^2 - 1$.

Hi ha 9 punts a la 3-torsió:

$$\begin{aligned} E[3] =&\{P \in E/(\mathbb{Z}_{11}/z^2+1) \text{ tals que } 3P = \mathcal{O}\} = \\ =&[\mathcal{O},(1,2),(1,9),(8,3z),(8,8z),(2z+1,z+9),(2z+1,10z+2),(9z+1,z+2), \\ &(9z+1,10z+9)] \end{aligned}$$

Dels 9 punts de la 3-torsió, 3 es troben al cos base ($\mathcal{O},(1,2),(1,9) \in E/\mathbb{Z}_{11}$) i la resta al cos estès $E/(\mathbb{Z}_{11}/z^2+1)$.

A continuació comprovem que els punts compleixen la condició per pertànyer a la 3-torsió. Com a exemple, mostrem els càlculs per als punts $(1,2)$ i $(8,3z)$:

$$\begin{aligned} P &= (1,2) \\ 2P &= (1,9) \\ 3P &= \mathcal{O} \end{aligned}$$

$$\begin{aligned} P &= (8,3z) \\ 2P &= (8,8z) \\ 3P &= \mathcal{O} \end{aligned}$$

És interessant notar l'estructura dels subgrups de la 3-torsió: la 3-torsió té 4 subgrups cíclics, tots ells d'ordre 3:

Ordre	Subgrup
3	$\{\mathcal{O},\ (1,2),\ (1,9)\}$
3	$\{\mathcal{O},\ (8,3z),\ (8,8z)\}$
3	$\{\mathcal{O},\ (2z+1,z+9),\ (2z+1,10z+2)\}$
3	$\{\mathcal{O},\ (9z+1,z+2),\ (9z+1,10z+9)\}$

Exercici 9.1 Donada la corba de l'exemple anterior (Exemple 9.2), calculeu el grau d'immersió d'E respecte a $n = 5, 7$ i 15.

9.2.3 El divisor d'una funció

Per acabar la presentació de les eines matemàtiques que ens permetran definir els *pairings*, exposarem el concepte de divisor d'una funció.

Abans, però, definirem els conceptes de funció racional, i els zeros i pols d'aquestes.

Definició 9.4 Una **funció racional** $f(x)$ és una funció que pot ser expressada com a una divisió de polinomis en la qual el denominador no és 0 (és a dir, $f(x) = q(x)/p(x)$ amb $p(x) \neq 0$). Quan el numerador $q(x)$ i el denominador $p(x)$ no tenen arrels en comú, diem que la funció està en **forma reduïda**.

Definició 9.5 Els **zeros** d'una funció racional són els punts en què $f(x) = 0$ mentre que els **pols** són els punts en què $f(x) = \pm\infty$.

Donada una funció racional en forma reduïda expressada amb els polinomis factoritzats:

$$f(x) = \frac{a(x-\alpha_1)^{\mu_1}(x-\alpha_2)^{\mu_2}\cdots(x-\alpha_n)^{\mu_n}}{b(x-\beta_1)^{\nu_1}(x-\beta_2)^{\nu_2}\cdots(x-\beta_n)^{\nu_n}} \tag{9.1}$$

els zeros corresponen als valors α_i mentre que els pols són els β_j. Noteu que $\alpha_i \neq \beta_j$ per a qualsevol i i j, ja que la funció es troba en forma reduïda. Direm que els μ_i i els ν_i són la multiplicitat de cada zero i de cada pol, respectivament.

A més, si el grau del polinomi del numerador difereix del grau del polinomi del denominador ($deg(q(x)) \neq deg(p(x))$), hi haurà un zero o un pol a l'infinit. En concret, si $deg(q(x)) > deg(p(x))$ hi haurà un zero a l'infinit i si $deg(q(x)) < deg(p(x))$ hi haurà un pol a l'infinit. La multiplicitat del zero o del pol serà la diferència entre els graus dels polinomis ($|deg(q(x)) - deg(p(x))|$), de manera que l'ordre total de zeros i pols és igual.

Els **divisors** són una eina que es fa servir per descriure els zeros i els pols d'una funció. Donada una funció racional:

$$f(x) = c\prod_i (x-\alpha_i)^{\mu_i} \tag{9.2}$$

escriurem:

$$div(f) = \sum_i \mu_i \langle \alpha_i \rangle$$

En primer lloc, noteu com l'equació 9.2 és equivalent a l'equació 9.1. Només cal expressar els factors que es trobaven al denominador amb valors negatius als exponents. En segon lloc, és important interpretar el divisor com a tal, i evitar operar com si estiguéssim treballant amb números (o, més endavant, amb punts de la corba). Per aquest motiu, es fan servir les claus $\langle$ i $\rangle$ per denotar que no estem parlant d'un enter (o un punt) α_i sinó d'un zero o un pol en aquell punt (més endavant tornarem a fer incís en aquest detall, quan definim els divisors sobre corbes el·líptiques). Per últim, cal destacar que efectivament el divisor ens permet anotar els pols i zeros de la funció, ja que ens descriu a on són i quina multiplicitat tenen.

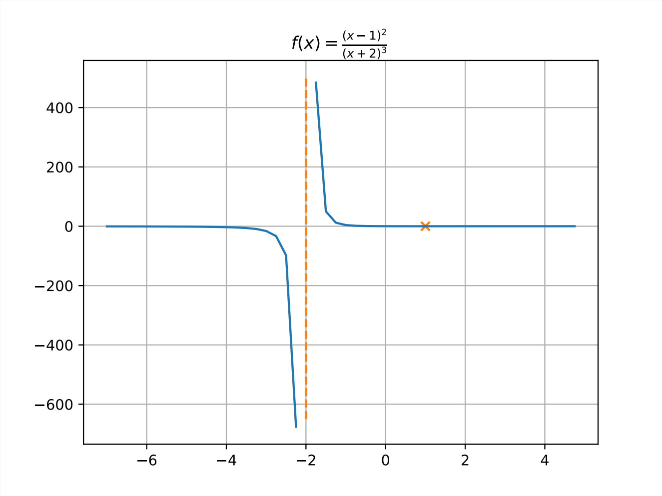
Exemple 9.3 Zeros i pols d'una funció

La funció:

$$f(x) = \frac{(x-1)^2}{(x+2)^3} = (x-1)^2(x+2)^{-3}$$

té un zero de multiplicitat 2 a $x = 1$ i un pol de multiplicitat 3 a $x = -2$. Addicionalment, la funció té un zero de multiplicitat 1 a l'infinit, ja que el grau del denominador és superior al del numerador en una unitat.

Observant la representació gràfica de la funció, podem veure que efectivament és zero en $x = 1$ i tendeix a infinit en $x = -2$:



Per tant, podem descriure els pols i zeros de la funció f amb el divisor:

$$div(f) = 2\langle 1\rangle - 3\langle -2\rangle + \langle\infty\rangle$$

Exercici 9.2 Indiqueu el divisor de la funció racional:

$$f(x) = \frac{(x-1)^3(x+5)^2}{(x-12)^4}$$

És interessant notar com es reflecteixen les operacions entre funcions en els divisors:

$$div(f \cdot g) = div(f) + div(g) \tag{9.3}$$

$$div(f/g) = div(f) - div(g) \tag{9.4}$$

Exemple 9.4 Operacions entre funcions

Donades les funcions:

$$f(x) = (x-1)^2$$

$$g(x) = \frac{1}{(x+2)^3}$$

amb divisors:

$$div(f) = 2\langle 1\rangle - 2\langle\infty\rangle$$

$$div(g) = -3\langle -2\rangle + 3\langle\infty\rangle$$

podem comprovar com el divisor del seu producte és la suma del divisor de cadascuna d'elles:

$$f(x)g(x) = (x-1)^2 \cdot \frac{1}{(x+2)^3} = \frac{(x-1)^2}{(x+2)^3} = (x-1)^2(x+2)^{-3}$$

$$div(f\cdot g) = 2\langle 1\rangle - 3\langle -2\rangle + \langle\infty\rangle = div(f) + div(g)$$

i que el divisor del seu quocient és la resta del divisor de cadascuna d'elles:

$$\frac{f(x)}{g(x)} = \frac{(x-1)^2}{\frac{1}{(x+2)^3}} = (x-1)^2(x+2)^3$$

$$div(f/g) = 2\langle 1\rangle + 3\langle -2\rangle - 5\langle\infty\rangle = div(f) - div(g)$$

A més, dues funcions que tenen el mateix divisor són iguals excepte per una constant i el divisor d'una funció és zero si i només si la funció és constant.

En la criptografia basada en corbes el·líptiques, es fan servir divisors per descriure els punts d'intersecció d'una corba E amb una funció $f(x)$, de manera que s'utilitzen per descriure els zeros i els pols de la funció $E - f(x)$.

Definició 9.6 Sigui E una corba el·líptica. Un **divisor sobre** E és una suma formal:

$$D = \sum_{P\in E} n_p\langle P\rangle$$

on els n_p són enters i on tots els n_p excepte un nombre finit són zero.

És a dir, ara els zeros i pols seran punts de la corba el·líptica ($P \in E$), i n'expressarem la seva multiplicitat amb els enters n_p, de la mateixa manera que ho fèiem anteriorment amb funcions racionals definides sobre els reals.

De nou, és important diferenciar la suma de punts d'una corba (que denotàvem amb el símbol $+$, per exemple, $P_1 + P_2$) i la multiplicació escalar d'un enter per un punt (que denotàvem per sP o bé $s\cdot P$) de la suma formal que conforma un divisor (que denotem fent servir també el símbol $+$ i de manera similar a la multiplicació escalar, però indicant els punts dins de les claus, $s_1\langle P_1\rangle + s_2\langle P_2\rangle$).

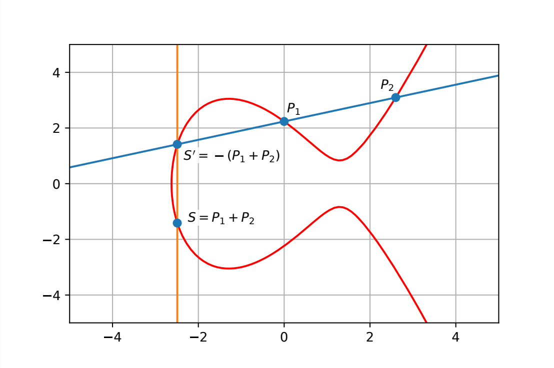
Exemple 9.5 Divisors de les rectes que defineixen la suma

Ja hem fet servir funcions sobre corbes el·líptiques a l'inici d'aquest capítol, quan descrivíem com sumar dos punts d'una corba el·líptica. Recapitulant, el procediment per calcular la suma entre dos punts requeria del càlcul de la recta que passava per aquests dos punts (o bé de la recta tangent a la corba en aquell punt, en l'operació de doblat) i de la recta vertical que passava pel tercer (o segon, en el cas del doblat) punt d'intersecció de la corba. A continuació descriurem els divisors d'aquestes funcions.

Anomenem l_{P_1,P_2} a la funció que representa la recta que passa pels punts P_1 i P_2 (recta que tracem per a calcular la suma $P_1 + P_2$, representada en blau a la figura següent) per a $P_1 \neq P_2$. Aleshores podem escriure els divisors de la funció l_{P_1,P_2}:

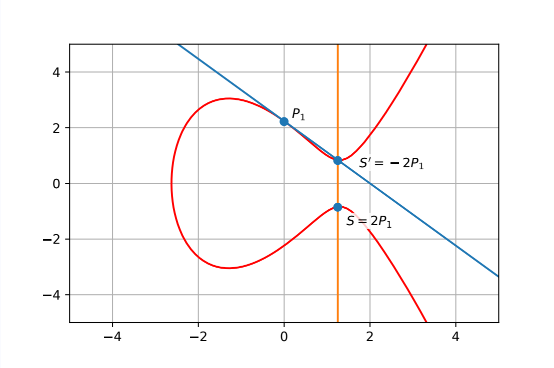
$$div(l_{P_1,P_2}) = \langle P_1\rangle + \langle P_2\rangle + \langle -(P_1+P_2)\rangle - 3\langle\mathscr{O}\rangle$$

ja que efectivament la funció l_{P_1,P_2} interseca la corba E en els punts P_1, P_2 i $-(P_1+P_2)$ (el punt simètric al resultat de la suma).



Per al cas en què els dos punts a sumar són iguals (línia blava a la figura següent), aleshores tenim que:

$$div(l_{P_1,P_1}) = 2\langle P_1\rangle + \langle -2P_1\rangle - 3\langle \mathscr{O}\rangle$$



Anomenem v_S a la funció que representa la recta vertical que passa pels punts S i $-S$ (línia taronja de les dues figures anteriors). Aleshores, podem dir que:

$$div(v_S) = \langle S\rangle + \langle -S\rangle - 2\langle \mathscr{O}\rangle$$

ja que la funció v_S interseca la corba E en els punts S i S'.

Tot i que ja es pot intuir de la descripció que s'ha fet dels divisors de funcions racionals, en els propers paràgrafs acabarem d'aprofundir en la multiplicitat del punt $\mathscr{O}$ de les funcions anteriors.

A continuació es defineixen algunes característiques dels divisors.

El **suport** d'un divisor D és el conjunt de tots els punts P que tenen el coeficient n_p diferent de zero. Dos divisors D_1 i D_2 tenen un **suport disjunt** si la intersecció entre el suport de tots dos és un conjunt buit (és a dir, no tenen punts en comú al suport).

El **grau** d'un divisor D és la suma dels coeficients n_p:

$$deg(D) = \sum_{P \in E} n_p$$

Un divisor D és un **divisor principal** si existeix una funció racional f tal que $D = div(f)$, és a dir, si representa els zeros i els pols d'una funció racional. Equivalentment podem afirmar que un divisor és principal si i només si té grau zero i $\sum_{P \in E} n_p P = \mathcal{O}$. Noteu que en aquesta última expressió no hi ha les claus $\langle$ i $\rangle$, de manera que aquí s'està calculant el sumatori de multiplicacions escalars (a diferència de l'expressió del divisor).

Exemple 9.6 Divisors de les rectes que defineixen la suma

A l'Exemple 9.5 hem vist els divisors de les rectes que es fan servir per a sumar punts (l_{P_1,P_2}, l_{P_1,P_1} i v_S), deduint-los a partir de la definició de les pròpies rectes:

$$div(l_{P_1,P_2}) = \langle P_1 \rangle + \langle P_2 \rangle + \langle -(P_1+P_2) \rangle - 3\langle \mathcal{O} \rangle$$
$$div(l_{P_1,P_1}) = 2\langle P_1 \rangle + \langle -2P_1 \rangle - 3\langle \mathcal{O} \rangle$$
$$div(v_S) = \langle S \rangle + \langle -S \rangle - 2\langle \mathcal{O} \rangle$$

No hem descrit amb detall, però, per què aquests divisors inclouen el punt a l'infinit $\mathcal{O}$ ni la seva multiplicitat. Doncs bé, com que les funcions que defineixen aquestes rectes són funcions racionals, els seus divisors són principals i, per tant, han de tenir grau zero. Per això la multiplicitat de $\mathcal{O}$ a $div(l)$ i $div(v)$ és -3 i -2, respectivament.

Exemple 9.7 Divisors principals

Sigui $P \in E$ un punt de la corba el·líptica E, tal que l'ordre de P és n. Aleshores, el divisor:

$$D = n\langle P \rangle - n\langle \mathcal{O} \rangle$$

és un divisor principal.

En efecte, el divisor compleix les dues propietats necessàries per ser principal:

$$deg(D) = \sum_{P \in E} n_p = n - n = 0$$
$$\sum_{P \in E} n_p P = nP - n\mathcal{O} = \mathcal{O}$$

Noteu que com que l'ordre del punt P és n, $nP = \mathcal{O}$.

Cal destacar també que en aquest cas sabem que existeix una funció racional que té com a divisor D (ja que el divisor és principal) però, a diferència de l'exemple anterior (Exemple 9.6), ara no la coneixem. Més endavant, a la propera subsecció, veurem com construir funcions que tinguin un divisor concret.

Dos divisors D_1 i D_2 són **equivalents** si difereixen en un divisor principal, és a dir, si $D = D_1 - D_2$ és un divisor principal. Com que un divisor principal té grau zero, dos divisors equivalents tenen el mateix grau. Per denotar que dos divisors són equivalents, escrivim $D_1 \sim D_2$.

Ja per acabar, només ens queda descriure l'avaluació d'una funció en un divisor, cosa que ens permetrà il·lustrar el teorema de la reciprocitat de Weil. L'**avaluació d'una funció racional f en un divisor** $D = \sum_{P \in E} n_p \langle P \rangle$ amb el suport de D i de $div(f)$ disjunts es defineix com:

$$f(D) = \prod_{P \in E} f(P)^{n_P}$$

Per tal de poder avaluar la funció f en un divisor D, els suports de D i de $div(f)$ han de ser disjunts, ja que si $P \in div(f)$ aleshores $f(P)$ seria zero o infinit i, per tant, $f(D)$ també ho seria.

Això ens permet descriure el teorema de la reciprocitat de Weil, que és la base de moltes de les propietats que es fan servir en criptografia basada en *pairings*:

Teorema 9.1 Siguin f i g dues funcions diferents de zero en una corba el·líptica tals que $div(f)$ i $div(g)$ tenen suports disjunts. Aleshores, $f(div(g)) = g(div(f))$.

Exemple 9.8 Reciprocitat de Weil

A continuació comprovarem la reciprocitat de Weil per a una corba i funcions concretes, tot aprofitant per exemplificar els diferents conceptes presentats en aquesta secció. Aquest exemple està basat en l'exemple 3.3.1 del manual *Pairings for beginners* de *Craig Costello*.

Donada una corba $E/\mathbb{Z}_{503} : y^2 = x^3 + 1$ i les funcions $f(x,y) = \frac{20y+9x+179}{199y+187x+359}$ i $g(x,y) = y + 251x^2 + 129x + 201$ sobre E, els divisors d'f i g són:

$$div(f) = 2\langle 433, 98\rangle + \langle 232, 113\rangle - \langle 432, 27\rangle - 2\langle 127, 258\rangle$$
$$div(g) = \langle 413, 369\rangle + \langle 339, 199\rangle + \langle 147, 443\rangle + \langle 124, 42\rangle - 4\langle \mathcal{O}\rangle$$

Pel que fa als divisors d'f, noteu com efectivament els punts $(433, 98)$ i $(232, 113)$ són punts on el numerador d'f és 0; i els punts $(432, 27)$ i $(127, 258)$ són punts on el denominador és 0.

$$\begin{aligned}
(433,98) &: 20y + 9x + 179 = 20 \cdot 98 + 9 \cdot 433 + 179 \mod 503 = 0 \\
(232,113) &: 20y + 9x + 179 = 20 \cdot 113 + 9 \cdot 232 + 179 \mod 503 = 0 \\
(432,27) &: 199y + 187x + 359 = 199 \cdot 27 + 187 \cdot 432 + 359 \mod 503 = 0 \\
(127,258) &: 199y + 187x + 359 = 199 \cdot 258 + 187 \cdot 127 + 359 \mod 503 = 0
\end{aligned}$$

A més, tots ells pertanyen a la corba E:

$$\begin{aligned}
98^2 &= 433^3 + 1 \mod 503 \\
113^2 &= 232^3 + 1 \mod 503 \\
27^2 &= 432^3 + 1 \mod 503 \\
256^2 &= 127^3 + 1 \mod 503
\end{aligned}$$

De manera similar, pel que fa als divisors de g, els punts $(413, 369)$, $(339, 199)$, $(147, 443)$, $(124, 42)$ són punts on el numerador és 0 i, a més, pertanyen a la corba E (els càlculs són anàlegs i es deixen com a exercici per al lector).

Pel que fa al grau i el suport dels divisors d'f i g, podem dir que:

$$deg(div(f)) = 2 + 1 - 1 - 2 = 0$$

$$deg(div(g)) = 1 + 1 + 1 + 1 - 4 = 0$$

$$sup(div(f)) = \{(433,98),(232,113),(432,27),(127,258)\}$$
$$sup(div(g)) = \{(413,369),(339,199),(147,443),(124,42),\mathscr{O}\}$$
$$sup(div(f)) \cap sup(div(g)) = \emptyset$$

Els dos divisors són principals, ja que el seu grau és 0 i, a més:

$$div(f) = 2(433,98)+(232,113)-(432,27)-2(127,258) = \mathscr{O}$$

$$div(g) = (413,369)+(339,199)+(147,443)+(124,42)-4\cdot\mathscr{O} = \mathscr{O}$$

Com que el suport dels dos divisors és disjunt, podem calcular l'avaluació de la funció f en el divisor $div(g)$:

$$\begin{aligned} f(div(g)) =& f(413,369)\cdot f(339,199)\cdot f(147,443)\cdot f(124,42)\cdot f(\mathscr{O})^{-4} = \\ =& \frac{20\cdot 369+9\cdot 413+179}{199\cdot 369+187\cdot 413+359}\cdot\frac{20\cdot 199+9\cdot 339+179}{199\cdot 199+187\cdot 339+359}\cdot\frac{20\cdot 443+9\cdot 147+179}{199\cdot 443+187\cdot 147+359}\cdot \\ &\cdot\frac{20\cdot 42+9\cdot 124+179}{199\cdot 42+187\cdot 124+359}\cdot\left(\frac{20\cdot 1+9\cdot 0+179}{199\cdot 1+187\cdot 0+359}\right)^{-4} = 321 \end{aligned}$$

Cal remarcar que per a la resta de punts de la corba, els enters n_p són 0 i, per tant, el resultat del seu factor és sempre 1.

Coordenades projectives

Noteu que per a calcular l'avaluació d'f en el punt $\mathscr{O}$ hem considerat que $x = 0$ i $y = 1$. Això és així ja que s'ha utilitzat la representació en coordenades projectives, de manera que $\mathscr{O} = (0:1:0)$. El lector interessat en aprendre aquesta representació pot consultar *The arithmetics of Elliptic Curves* de Joseph H. Silverman.

També podem calcular l'avaluació de la funció g en el divisor $div(f)$:

$$\begin{aligned} g(div(f)) =& g(433,98)^2\cdot g(232,113)\cdot g(432,27)^{-1}\cdot g(127,258)^{-2} = \\ =&(98+251\cdot 433^2+129\cdot 433+201)^2\cdot(113+251\cdot 232^2+129\cdot 232+201)\cdot \\ &\cdot(27+251\cdot 432^2+129\cdot 432+201)^{-1}\cdot(258+251\cdot 127^2+129\cdot 127+201)^{-2} = \\ =&321 \end{aligned}$$

Així doncs, efectivament, $f(div(g)) = g(div(f)) = 321$.

9.2.4 Construcció de funcions a partir del divisor

Com a últim apunt abans de presentar la construcció explícita dels *pairings* de Weil i de Tate, veurem com podem construir funcions amb un divisor donat.

Un divisor d'especial importància en la definició dels *pairings* és el divisor:

$$div(f_{m,P}) = m\langle P\rangle - \langle mP\rangle - (m-1)\langle\mathscr{O}\rangle$$

per a qualsevol enter m i qualsevol $P \in E$. El divisor és un divisor principal, ja que el grau és zero (en efecte, $m-1-(m-1) = 0$) i $\sum_{P\in E} n_p P = mP - mP - (m-1)\mathscr{O} = \mathscr{O}$. Per tant, sempre existeix una funció racional

$f_{m,P}$ per a qualsevol m i P.

A més, si el punt P pertany a la r-torsió, aleshores:

$$\begin{aligned} div(f_{r,P}) &= r\langle P\rangle - \langle rP\rangle - (r-1)\langle\mathcal{O}\rangle = \\ &= r\langle P\rangle - \langle\mathcal{O}\rangle - (r-1)\langle\mathcal{O}\rangle = \\ &= r\langle P\rangle - r\langle\mathcal{O}\rangle \end{aligned}$$

Sabem que existeix una funció racional $f_{m,P}$ per a qualsevol m i P ja que el divisor és principal, però necessitem saber com trobar aquesta funció. Doncs bé, podem construir $f_{m,P}$ iterativament, a partir d'una funció constant de divisor zero, de la manera següent:

$$f_{m+1,P} = f_{m,P} \cdot \frac{l_{mP,P}}{v_{(m+1)P}} \tag{9.5}$$

on $l_{mP,P}$ és la recta que passa pels punts mP i P, i $v_{(m+1)P}$ és la recta vertical que passa pel punt $(m+1)P$ (recordeu que hem descrit aquestes funcions a l'Exemple 9.5).

Noteu com, efectivament, $div(f_{m+1,P}) = div(f_{m,P} \cdot \frac{l_{mP,P}}{v_{(m+1)P}})$:

$$\begin{aligned} div(f_{m+1,P}) &= (m+1)\langle P\rangle - \langle(m+1)P\rangle - m\langle\mathcal{O}\rangle \\ div(f_{m,P}) &= m\langle P\rangle - \langle mP\rangle - (m-1)\langle\mathcal{O}\rangle \\ div(l_{mP,P}) &= \langle mP\rangle + \langle P\rangle + \langle -(m+1)P\rangle - 3\langle\mathcal{O}\rangle \\ div(v_{(m+1)P}) &= \langle(m+1)P\rangle + \langle -(m+1)P\rangle - 2\langle\mathcal{O}\rangle \end{aligned}$$

i, fent servir les Equacions 9.3 i 9.4, veiem que:

$$\begin{aligned} & div(f_{m,P} \cdot \frac{l_{mP,P}}{v_{(m+1)P}}) = \\ &= div(f_{m,P}) + div(l_{mP,P}) - div(v_{(m+1)P}) = \\ &= \Big(m\langle P\rangle - \langle mP\rangle - (m-1)\langle\mathcal{O}\rangle\Big) + \Big(\langle mP\rangle + \langle P\rangle + \langle -(m+1)P\rangle - 3\langle\mathcal{O}\rangle\Big) - \\ &\quad - \Big(\langle(m+1)P\rangle + \langle -(m+1)P\rangle - 2\langle\mathcal{O}\rangle\Big) = \\ &= m\langle P\rangle + \langle P\rangle + \langle mP\rangle - \langle mP\rangle + \langle -(m+1)P\rangle - \langle -(m+1)P\rangle - \langle(m+1)P\rangle - \\ &\quad - (m-1)\langle\mathcal{O}\rangle - 3\langle\mathcal{O}\rangle + 2\langle\mathcal{O}\rangle = \\ &= (m+1)\langle P\rangle - \langle(m+1)P\rangle - m\langle\mathcal{O}\rangle = \\ &= div(f_{m+1,P}) \end{aligned}$$

El divisor d'una funció la determina excepte múltiples escalars diferents de zero, de manera que podrem construir funcions amb uns divisors concrets fent servir l'Equació 9.5.

Exemple 9.9 Construcció de la funció $f_{r,P}$

Procedim a construir la funció $f_{3,P}$ per a $P = (2,11) \in E/\mathbb{Z}_{23} : y^2 = x^3 - x$.

La funció $f_{3,P}$ es construeix iterativament a partir d'$f_{1,P}$, la funció constant amb divisor zero:

$$f_{1,P} = 1$$

Per a construir $f_{2,P}$ cal calcular la funció $l_{P,P}$ i la funció v_{2P}.

La funció $l_{P,P}$ és la recta tangent a la corba que passa pel punt P:

$$m = \frac{3x_1^2 + a}{2y_1} \mod p = \frac{3 \cdot 2^2 - 1}{2 \cdot 11} \mod 23 = 12$$

$$y = mx + b;\ 11 = 12 \cdot 2 + b;\ b = 10$$

$$l_{P,P} : y = 12x + 10$$

La funció v_{2P} és la recta vertical que passa pel punt $2P$:

$$2P = 2(2,11) = (2,12)$$

$$v_{2P} : x = 2$$

Aleshores:

$$f_{2,P} = f_{1,P} \cdot \left(\frac{l_{P,P}}{v_{2P}}\right) = \frac{y - 12x - 10}{x - 2} = \frac{y + 11x + 13}{x + 21}$$

Per últim, es calcula $f_{3,P}$ a partir d'$f_{2,P}$:

$$f_{3,P} = f_{2,P} \cdot v_P = \frac{y + 11x + 13}{x + 21}(x - 2) = y + 11x + 13$$

9.3 Construcció explícita dels *pairings* de Weil i Tate

Arribats a aquest punt, ja estem en disposició de poder descriure els *pairings* de Weil i de Tate.

9.3.1 El *pairing* de Weil

Siguin $P, Q \in E/\mathbb{F}_{p^k}[r]$ dos punts de la r-torsió d'una corba el·líptica i D_P, D_Q dos divisors de grau zero amb suports disjunts tals que:

$$D_P \sim \langle P \rangle - \langle \mathscr{O} \rangle$$

$$D_Q \sim \langle Q \rangle - \langle \mathscr{O} \rangle$$

Existeixen funcions f i g tals que:

$$div(f) = rD_P$$

$$div(g) = rD_Q$$

El ***pairing* de Weil** és una aplicació que rep un parell de punts de la r-torsió i retorna una arrel r-èsima de la unitat, definida com:

$$w_r(P,Q) = \frac{f(D_Q)}{g(D_P)}$$

Arrels de la unitat	Una arrel de la unitat és un número que elevat a un enter positiu dona 1. Quan l'enter positiu és 2, parlem d'arrels quadrades; i quan és 3, d'arrels cúbiques.

Exemple 9.10 Càlcul del *pairing* de Weil

A continuació calcularem el *pairing* de Weil per a dos punts concrets d'una corba el·líptica. Aquest exemple està basat en l'exemple 5.1.1 del manual *Pairings for beginners* de *Craig Costello*.

La corba $E/\mathbb{Z}_{23} : y^2 = x^3 - x$ té $\#E/\mathbb{Z}_{23} = 24$ elements.

El punt $P = (2,11)$ té ordre $r = 3$.

El grau d'immersió de la corba respecte a $r = 3$ és $k = 2$, ja que $3 \nmid 23 - 1$ però en canvi $3 \mid 23^2 - 1$.

Hi ha 9 punts a la 3-torsió:

$$\begin{aligned} E/(\mathbb{Z}_{23}/(z^2+1))[3] =& [\mathcal{O}, (2,11), (2,12), (21,11z), (21,12z), (5z,2z+2), (5z,21z+21),\\ & (18z,2z+21), (18z,21z+2)] \end{aligned}$$

Calcularem el *pairing* de Weil $w_3(P,Q)$ per a $P = (2,11)$ i $Q = (21,12z)$, dos punts que pertanyen a la 3-torsió.

En primer lloc, hem de trobar els divisors de grau zero D_P i D_Q amb suports disjunts, i equivalents a $\langle P\rangle - \langle\mathcal{O}\rangle$ i $\langle Q\rangle - \langle\mathcal{O}\rangle$, respectivament. Per fer-ho, seleccionem dos punts addicionals aleatoris de la corba sobre el cos estès, $R = (17z, 2z+21)$ i $S = (10z+18, 13z+13)$, i fixem els divisors D_P i D_Q com:

$$\begin{aligned} D_P &= \langle P+R\rangle - \langle R\rangle \\ D_Q &= \langle Q+S\rangle - \langle S\rangle \end{aligned}$$

Noteu com efectivament els divisors tenen grau zero i suports disjunts:

$$\begin{aligned} deg(D_P) = deg(D_Q) &= 1-1 = 0 \\ sup(D_P) = \{P+R, R\} &= \{(z+16, 18z+20), (17z, 2z+21)\} \\ sup(D_Q) = \{Q+S, S\} &= \{(19z+22, 12z+10), (10z+18, 13z+13)\} \\ sup(D_P) \cap sup(D_Q) &= \emptyset \end{aligned}$$

A més, els divisors són efectivament equivalents a $\langle P\rangle - \langle\mathcal{O}\rangle$ i $\langle Q\rangle - \langle\mathcal{O}\rangle$. En efecte, el divisor D'_P resultant de restar $\langle P\rangle - \langle\mathcal{O}\rangle$ a D_P és un divisor principal:

$$\begin{aligned} D'_P = D_P - (\langle P\rangle - \langle\mathcal{O}\rangle) &= \langle P+R\rangle - \langle R\rangle - \langle P\rangle + \langle\mathcal{O}\rangle \\ deg(D'_P) &= 1-1-1+1 = 0 \\ (P+R) - R - P + \mathcal{O} &= \mathcal{O} \end{aligned}$$

Els càlculs per a D_Q són anàlegs.

En segon lloc, necessitem trobar les funcions f i g que tenen com a divisors $3D_P$ i $3D_Q$, respectivament. Podem trobar f i g com:

$$f = f_{3,P}\left(\frac{v_{P+R}}{l_{P,R}}\right)^3$$

$$g = f_{3,Q}\left(\frac{v_{Q+S}}{l_{Q,S}}\right)^3$$

on l i v són les funcions que descriuen la recta que passa per dos punts i la recta vertical que passa per un punt, respectivament.

Efectivament, el divisor de f és $3D_P$ (els càlculs per a g són equivalents i s'ometen per brevetat):

$$\begin{aligned} div(f) &= div(f_{3,P}) + 3(div(v_{P+R}) - div(l_{P,R})) = \\ &= \Big(3\langle P\rangle - 3\langle \mathscr{O}\rangle\Big) + 3\Big(\langle -(P+R)\rangle + \langle P+R\rangle - 2\langle \mathscr{O}\rangle - \langle P\rangle - \langle R\rangle - \langle -(P+R)\rangle + 3\langle \mathscr{O}\rangle\Big) = \\ &= 3\langle P+R\rangle - 3\langle R\rangle \end{aligned}$$

Procedim doncs a construir les funcions f i g. Per a construir f, cal trobar les funcions v_{P+R}, $l_{P,R}$ i $f_{3,P}$:

La funció v_{P+R} és la recta vertical que passa pel punt $P+R$:

$$P+R = (2,11) + (17z, 2z+21) = (z+16, 18z+20)$$

$$v_{P+R} : x = z+16$$

La funció $l_{P,R}$ és la recta que passa pels punts P i R:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \mod p = \frac{2z+21-11}{17z-2} = 6z+13$$

$$y = mx + c; c = y - mx = 11 - 2(6z+13) = 11z+8$$

$$l_{P,R} : y = (6z+13)x + (11z+8)$$

La funció $f_{3,P}$ es construeix iterativament a partir d'$f_{1,P}$, tal com hem vist a l'Exemple 9.9.

$$f_{3,P} = y + 11x + 13$$

Així doncs, la funció f és:

$$f(x,y) = f_{3,P}\left(\frac{v_{P+R}}{l_{P,R}}\right)^3 = (y+11x+13)\cdot\left(\frac{x-z-16}{y-(6z+13)x-(11z+8)}\right)^3$$

De la mateixa manera, per a construir g cal trobar les funcions v_{Q+S}, $l_{Q,S}$ i $f_{3,Q}$. A continuació es proporcionen aquestes funcions, i es deixa com a exercici per al lector els càlculs individuals per a trobar-les:

Exercici 9.3 Trobeu les funcions v_{Q+S}, $l_{Q,S}$ i $f_{3,Q}$.

$$v_{Q+S} : x = 19z+22$$

$$l_{Q,S} : y = (3z+1)x + (18z+2)$$

$$f_{3,Q} : y = -11zx - 10z$$

De manera que la funció g és:

$$g(x,y) = f_{3,Q}\left(\frac{v_{Q+S}}{l_{Q,S}}\right)^3 = (y + 11zx + 10z) \cdot \left(\frac{x - 19z - 22}{y - (3z+1)x - (18z+2)}\right)^3$$

En tercer lloc i ja per acabar, procedim a calcular el *pairing* de Weil:

$$w_3(P,Q) = \frac{f(D_Q)}{g(D_P)} = \frac{f(Q+S)g(R)}{f(S)g(P+R)} = \frac{(7z+22)(21z+22)}{(5z+4)(15z+1)} = 15z + 11$$

Noteu com, efectivament, $15z + 11$ és una arrel 3-èsima de la unitat, ja que $(15z+11)^3 = 1$.

A partir de l'exemple anterior, podem veure ara alguns dels efectes de la bilinealitat del *pairing* de Weil:

Exemple 9.11 Bilinealitat en el *pairing* de Weil

A l'exemple anterior hem calculat el *pairing* de Weil per a $P = (2,11)$ i $Q = (21,12z)$. A continuació veurem exemples de bilinealitat en el *pairing* de Weil per a aquests punts concrets.

Els punts resultants de doblar P i Q són $2P = (2,12)$ i $2Q = (21,11z)$.

Si calculem el *pairing* de Weil de $2P$ i Q o bé el de P i $2Q$, veurem que són iguals, i també que coincideixen amb el quadrat del de P i Q:

$$w_3(2P,Q) = w_3(P,2Q) = 8z + 11 = (15z+11)^2 = w_3(P,Q)^2$$

9.3.2 El *pairing* de Tate

En la definició bàsica del *pairing* de Tate, el resultat del *pairing* per un parell de punts concret no és únic. Com que en aplicacions criptogràfiques aquesta característica és problemàtica, habitualment s'utilitza el *pairing* de Tate reduït, que no és res més que el *pairing* de Tate elevat a $(q^k - 1)/r$. Així, s'aconsegueix que el resultat sigui una arrel r-èsima de la unitat, i que per cada parell de punts el valor del *pairing* sigui únic. En aquest text descrivim doncs directament el *pairing* de Tate reduït.

Siguin $P, Q \in E/\mathbb{F}_{q^k}[r]$ dos punts de la r-torsió d'una corba el·líptica. Existeix una funció f tal que:

$$div(f) = r\langle P\rangle - r\langle O\rangle$$

Sigui D_Q un divisor de grau zero amb suport disjunt de $div(f)$ i tal que:

$$D_Q \sim \langle Q\rangle - \langle \mathcal{O}\rangle$$

El ***pairing* de Tate reduït** és una aplicació que rep un parell de punts de la r-torsió i retorna una arrel r-èsima de la unitat, definida com:

$$t_r(P,Q) = f(D_Q)^{(q^k-1)/r}$$

A diferència del *pairing* de Weil, en el *pairing* de Tate només cal que un dels punts pertanyi a la r-torsió (el punt P que descriu el divisor d'f), i el segon punt pot no ser-hi (però ha de complir unes propietats concretes que no són trivials de definir). Com que els punts de la r-torsió les compleixen, per simplicitat en aquest document seleccionem sempre punts que hi pertanyin.

Exemple 9.12 Càlcul del *pairing* de Tate

A continuació calcularem el *pairing* de Tate reduït per als mateixos dos punts que en l'exemple anterior (Exemple 9.10).

Sigui $E/\mathbb{Z}_{23} : y^2 = x^3 - x$ la corba el·líptica, amb $P = (2,11), Q = (21,12z) \in E/(\mathbb{Z}_{23}/(z^2+1))[3]$. El grau d'immersió de la corba respecte a $r = 3$ és $k = 2$.

En primer lloc, trobem una funció f amb divisor $3\langle P\rangle - 3\langle O\rangle$. Tal com hem calculat a l'exemple anterior, la funció $f_{3,P} : y + 11x + 13$ té aquest divisor.

En segon lloc, trobem un divisor D_Q de grau zero amb suport disjunt al divisor d'f i equivalent a $\langle Q\rangle - \langle \mathcal{O}\rangle$. Per fer-ho, podem fer servir la mateixa estratègia que a l'exemple del *pairing* de Weil: seleccionem un punt S aleatori i fixem:

$$D_Q = \langle Q+S\rangle - \langle S\rangle$$

Per al punt $S = (10z+18, 13z+13)$, tenim doncs que:

$$D_Q = \langle (19z+22, 12z+10)\rangle - \langle (10z+18, 13z+13)\rangle$$

Noteu que en aquest cas no ens cal calcular la funció que té per divisor D_Q.

Finalment, calculem el *pairing* de Tate reduït:

$$t_r(P,Q) = f(D_Q)^{(p^k-1)/r} = \left(\frac{f(Q+S)}{f(S)}\right)^{(q^k-1)/r} = \left(\frac{f(19z+22, 12z+10)}{f(10z+18, 13z+13)}\right)^{(23^2-1)/3} =$$
$$= \left(\frac{14z+12}{8z+17}\right)^{176} = 15z+11$$

De nou, podem comprovar com el resultat del *pairing* és una arrel 3-èsima de la unitat: $(15z+11)^3 = 1$.

Exemple 9.13 Bilinealitat en el *pairing* de Tate

De la mateixa manera que amb el *pairing* de Weil, podem veure també un exemple concret de les propietats de bilinealitat del *pairing* de Tate a partir dels valors de l'exemple anterior.

Recordem que $P = (2,11)$, $Q = (21,12z)$, $2P = (2,12)$ i $2Q = (21,11z)$.

Si calculem el *pairing* de Tate de $2P$ i Q o bé el de P i $2Q$, veurem que són iguals, i també que coincideixen amb el quadrat del de P i Q:

$$t_r(2P,Q) = t_r(P,2Q) = 8z+11 = (15z+11)^2 = t_r(P,Q)^2$$

9.4 Algorismes criptogràfics basats en *pairings*

En aquesta secció es descriuen algorismes criptogràfics que fan servir *pairings*. En primer lloc, veurem l'esquema de signatura BLS, que permet fer signatures curtes i, a més, permet agregar-les. A continuació, descriurem la criptografia basada en la identitat i explicarem un dels algorismes d'aquesta família, l'algorisme de xifratge de Boneh-Franklin.

9.4.1 L'esquema de signatura BLS

L'esquema de signatura digital BLS (anomenat així per les inicials dels cognoms dels seus creadors, Dan Boneh, Ben Lynn i Hovav Shacham) va ser proposat l'any 2001. L'esquema fa servir un *pairing* bilineal per a la verificació de signatures.

La característica principal d'aquest esquema de signatura digital és que produeix signatures *curtes*: la mida de la signatura digital és la meitat de la tindria una signatura DSA amb el mateix nivell de seguretat. Això fa que l'esquema sigui idoni per a situacions amb poc ample de banda o quan és necessari que un humà transcrigui la signatura digital manualment.

L'esquema de signatura BLS fa servir un *pairing* i una funció hash. Sigui $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ un *pairing* on $\mathbb{G}_0, \mathbb{G}_1$ i $\mathbb{G}_T$ són grups cíclics d'ordre primer q amb $G_0 \in \mathbb{G}_0$ i $G_1 \in \mathbb{G}_1$ elements generadors dels grups. Sigui H una funció hash que relaciona els missatges de l'espai de missatges a elements de $\mathbb{G}_0$.

L'algorisme de **generació de claus BLS** consta dels passos següents:

1. Es tria un enter aleatori $\alpha \in_R \mathbb{Z}_q$.
2. Es calcula $u = \alpha \cdot G_1 \in \mathbb{G}_1$.
3. La clau pública és $k_{pub} = u$, mentre que la clau privada és $k_{priv} = \alpha$.

L'algorisme de generació de claus és doncs anàleg al de l'ECDSA: es tria un enter aleatori que serà la clau privada i es multiplica aquest enter pel generador d'un grup cíclic, que és un paràmetre públic del sistema. La clau pública és un element de $\mathbb{G}_1$, és a dir, un punt d'una corba el·líptica.

Les claus generades per l'algorisme anterior es poden fer servir per a generar i validar signatures digitals fent servir els algorismes següents:

A partir d'un missatge en clar m, la clau privada de l'emissor $k_{priv} = \alpha$, i els paràmetres de domini, es calcula la **signatura digital BLS** del missatge:

1. Es calcula $\sigma = \alpha \cdot H(m) \in \mathbb{G}_0$
2. La signatura és el valor σ.

L'algorisme de signatura requereix de l'ús d'una funció hash que s'aplica al missatge abans de signar-lo, i que el converteix en un element del grup $\mathbb{G}_0$, és a dir, en un punt de la corba el·líptica. Una vegada es té la representació del missatge m en el grup $\mathbb{G}_0$, només cal calcular una multiplicació escalar entre el punt de la corba que representa el missatge i la clau privada. Convé destacar que la signatura és doncs un punt d'una corba el·líptica (un element de $\mathbb{G}_0$), de manera que si la representació dels punts és curta, la signatura també ho serà.

Fins aquí només s'ha fet ús de corbes el·líptiques però encara no s'ha introduït l'ús del *pairing*. L'algorisme de verificació de la signatura és precisament el punt de l'esquema que requereix de l'ús de *pairings*:

A partir d'un missatge en clar m, la clau pública k_{pub}, els paràmetres de domini i una signatura del missatge σ, els passos següents permeten **verificar una signatura BLS**:

1. Es verifica que $e(H(m),u) = e(\sigma, G_1)$.
2. Si la igualtat es compleix, aleshores la signatura és vàlida i la verificació finalitza correctament. En cas contrari, la signatura es considera invàlida i la verificació fracassa.

Convé notar com una signatura digital correcta serà donada per vàlida per l'algorisme de verificació, ja que per les propietats de bilinealitat del *pairing*:

$$e(H(m),u) = e(H(m),\alpha G_1) = e(H(m),G_1)^{\alpha} = e(\alpha H(m),G_1) = e(\sigma,G_1)$$

L'esquema de signatura BLS és determinista, ja que donats uns paràmetres de domini, una clau privada i un missatge en clar, la signatura que es produeix és única. Això el diferencia de la variant clàssica de l'algorisme de signatura ECDSA, que és probabilístic.

Exemple 9.14 Exemple de signatura i validació amb BLS

Sigui $E/\mathbb{Z}_{43} : y^2 = x^3 + 7x$ una corba el·líptica d'ordre 44. Farem servir com a grups cíclics $\mathbb{G}_0$ i $\mathbb{G}_1$ dos subgrups cíclics de la 11-torsió. El grau d'immersió de la corba respecte a $r = 11$ és $k = 2$. L'exemple utilitzarà com a *pairing* e el *pairing* de Weil (w_{11}).

Sigui $G_0 = (4,7)$ el generador del subgrup cíclic $\mathbb{G}_0$ i $G_1 = (2,8z)$ el generador del subgrup cíclic $\mathbb{G}_1$ (tots dos d'ordre 11). $\mathbb{G}_0$ es troba en el cos base ($\mathbb{G}_0 < E/\mathbb{Z}_{43}$), mentre que $\mathbb{G}_1$ es troba al cos estès $E/(\mathbb{F}_{43^2}/z^2+1)$.

Abans de signar, l'usuari ha de disposar d'un parell de claus. Per aconseguir-les, l'usuari executa l'algorisme de generació de claus:

1. Tria un enter aleatori $\alpha = 5 \in_R \mathbb{Z}_{43}$.
2. Calcula $u = \alpha \cdot G_1 = 5(2,8z) = (39,7z) \in \mathbb{G}_1$.
3. La clau pública és $k_{pub} = (39,7z)$, mentre que la clau privada és $k_{priv} = 5$.

Ara, l'usuari pot signar executant l'algorisme de generació de la signatura. Suposem que la representació del missatge m al subgrup $\mathbb{G}_0$ és $H(m) = (41,35)$. Efectivament, $H(m) \in \mathbb{G}_0$ ja que $H(m) = 2G_0$. Recordeu que ja hem vist a la secció 8.4.3 com crear funcions hash que retornin punts d'una corba concreta.

1. Es calcula $\sigma = \alpha \cdot H(m) = 5(41,35) = (4,36) \in \mathbb{G}_0$
2. La signatura és el valor $\sigma = (4,36)$.

Un receptor pot verificar la signatura a partir del missatge m, la signatura σ, i la clau pública k_{pub} (i coneixent els paràmetres de domini que són públics):

1. El receptor verifica que $e(H(m),u) = e(\sigma,G_1)$.
   (a) $e(H(m),u) = w_{11}((41,35),(39,7z)) = 34z+7$.
   (b) $e(\sigma,G_1) = w_{11}((4,36),(2,8z)) = 34z+7$.
2. Com que la igualtat es compleix, la signatura és vàlida.

Exercici 9.4 Suposeu que es fa servir l'esquema de signatura BLS amb la construcció ingènua de la funció hash H següent:

$$H(m) = H'(m) \cdot G_1$$

on $H'(m) = SHA\text{-}1(m) \mod q$. Expliqueu perquè aquesta construcció no és segura.

Més enllà de produir signatures curtes, l'esquema de signatura BLS té algunes propietats addicionals que el fan especialment interessant. El BLS permet agregació de signatures i esquemes de signatures llindar. A continuació descriurem com construir un esquema de signatures agregables en base a l'algorisme de signatura BLS.

Agregació de signatures

Els esquemes de signatura digital que permeten **agregació de signatures** es caracteritzen per permetre comprimir diverses signatures (sobre diferents missatges i amb diferents claus) en una sola signatura agregada, que es pot fer servir per verificar totes les signatures de cop. Aquesta signatura agregada té una longitud similar a la d'una signatura individual, independentment del nombre de signatures que comprimeixi.

De la mateixa manera que un esquema de signatura queda definit per tres algorismes (generació de claus, signatura i verificació), els esquemes que permeten agregació de signatures incorporen dos algorismes addicionals: l'agregació de signatures i la verificació d'una signatura agregada. L'agregació de signatures rep un conjunt de signatures (i, en alguns esquemes, les claus públiques associades) i genera una única signatura agregada. Com que l'agregació de signatures no requereix de claus privades ni de la interacció dels signants, qualsevol persona (amb coneixement de les signatures i les claus públiques) pot executar l'algorisme i generar una signatura agregada. Això implica que l'agregació de signatures es pot fer amb posterioritat a la creació de les signatures. La verificació de signatures rep una signatura agregada i el conjunt de missatges que s'han signat (i, en alguns esquemes, les claus públiques associades), i valida que totes les signatures que resumeix la signatura agregada siguin correctes.

Els esquemes de signatura digital que permeten agregació de signatures són útils en diversos escenaris. Per exemple, en la verificació d'una cadena de certificats digitals, és habitual haver de validar diverses signatures, des del certificat a comprovar fins al certificat arrel de la CA en el qual es confia. Un altre escenari on l'agregació de signatures és especialment interessant és en criptomonedes basades en cadena de blocs, on té diverses aplicacions. D'una banda, en les transaccions amb múltiples entrades, permetria agregar les signatures de cada entrada en una sola signatura, cosa que redueix la mida d'aquestes transaccions. La mida de les transaccions és crítica en sistemes *blockchain*, ja que és un dels grans limitadors de l'escalabilitat del sistema. D'altra banda, l'agregació de signatures també permetria implementar sortides multisignatura de manera eficient. Les sortides multisignatura són sortides de transaccions que requereixen més d'una signatura per a ser gastades. Aquestes sortides especifiquen un conjunt de claus públiques i un llindar de signatures mínim necessàries per a autoritzar el pagament. Així, per tal de gastar aquestes sortides es requereix habitualment d'un conjunt de signatures. Si es disposa d'un esquema amb agregació de signatures, aquest conjunt de signatures a proporcionar es pot resumir en una única signatura, oferint de nou millores en la longitud de les transaccions. Addicionalment, alguns esquemes d'agregació de signatures també permeten agregar les claus públiques, cosa que redueix encara més la mida de les transaccions i ofereix privadesa afegida.

BLS i Ethereum

Les primeres versions de les criptomonedes Bitcoin i Ethereum feien servir ECDSA. La nova versió d'Ethereum (Eth2) que es troba actualment en desplegament (2021) incorpora signatures BLS, amb l'objectiu d'accelerar la verificació de signatures. Bitcoin incorpora des de novembre de 2021 l'ús de signatures Schnorr, que també permeten agregació.

A continuació es descriu l'algorisme d'agregació de signatures BLS ingenu, que permet agregar signatures però no és segur. Més endavant es descriu el problema de seguretat d'aquesta versió de l'algorisme i una modificació que permet solucionar-lo.

A partir d'un conjunt d'n claus públiques $K^{pub} = \{k_1^{pub}, \cdots, k_n^{pub}\}$ i d'un conjunt d'n signatures $\Sigma = \{\sigma_1, \cdots, \sigma_n\}$ es **calcula la signatura agregada** σ_{ag}:

1. Es calcula la signatura agregada $\sigma_{ag} = \sigma_1 + \cdots + \sigma_n$
2. La signatura agregada és el valor $\sigma_{ag} \in \mathbb{G}_0$.

L'agregació de signatures consisteix doncs en la suma de punts de la corba $\mathbb{G}_0$, que conformen les signatures individuals, i la signatura agregada és també un punt de la mateixa corba. Noteu com el procés d'agregació de signatures no requereix ni dels missatges signats ni de les claus privades que han generat les signatures.

A partir d'un conjunt d'n claus públiques $K^{pub} = \{k_1^{pub}, \cdots, k_n^{pub}\}$, d'un conjunt d'$n$ missatges $M = \{m_1, \cdots, m_n\}$ i d'una signatura agregada σ_{ag}, els passos següents permeten **verificar la signatura agregada**:

1. Es comprova si $e(\sigma_{ag}, G_1) \stackrel{?}{=} e(H(m_1), k_1^{pub}) \cdot \cdots \cdot e(H(m_n), k_n^{pub})$
2. La signatura agregada és vàlida (i, per tant, totes les signatures individuals que resumeix es consideren vàlides) si la comprovació del pas anterior es fa amb èxit. En cas contrari, es rebutja la signatura agregada.

L'algorisme de verificació dona per vàlida una signatura agregada que comprimeix un conjunt de signatures individuals vàlides per les propietats del *pairing*:

$$\begin{aligned}
e(\sigma_{ag}, G_1) &= e(\sigma_1 + \cdots + \sigma_n, G_1) = \\
&= e(\sigma_1, G_1) \cdot \cdots \cdot e(\sigma_n, G_1) = \\
&= e(\alpha_1 H(m_1), G_1) \cdot \cdots \cdot e(\alpha_n H(m_n), G_1) = \\
&= e(H(m_1), \alpha_1 G_1) \cdot \cdots \cdot e(H(m_n), \alpha_n G_1) = \\
&= e(H(m_1), k_1^{pub}) \cdot \cdots \cdot e(H(m_n), k_n^{pub})
\end{aligned}$$

La verificació de signatures agregades es pot calcular de manera especialment eficient quan el missatge signat és el mateix per a totes les signatures, és a dir, quan $m = m_1 = m_2 = \cdots = m_n$. En aquests casos, l'equació de verificació es pot simplificar:

$$e(\sigma_{ag}, G_1) \stackrel{?}{=} e(H(m_1), k_1^{pub}) \cdot \cdots \cdot e(H(m_n), k_n^{pub})$$

$$e(\sigma_{ag}, G_1) \stackrel{?}{=} e(H(m), k_1^{pub}) \cdot \cdots \cdot e(H(m), k_n^{pub})$$

$$e(\sigma_{ag}, G_1) \stackrel{?}{=} e(H(m), k_1^{pub} + \cdots + k_n^{pub})$$

de manera que es passa de necessitar calcular $n+1$ *pairings* a calcular-ne només 2.

Exercici 9.5 Aquest exercici fa servir els mateixos paràmetres de domini que l'Exemple 9.14. Sigui $E/\mathbb{Z}_{43} : y^2 = x^3 + 7x$ una corba el·líptica d'ordre 44; $\mathbb{G}_0$ i $\mathbb{G}_1$ dos subgrups cíclics de la 11-torsió generats per $G_0 = (4,7)$ i $G_1 = (2, 8z)$, respectivament; i e el *pairing* de Weil (w_r).

També aprofitem el parell de claus i la signatura generades a l'Exemple 9.14:

$$k_1^{priv} = 5, \; k_1^{pub} = (39, 7z)$$

$$m = (41, 35), \; \sigma_1 = (4, 36)$$

1. Sigui $k_2^{priv} = 7$ la clau privada d'un segon usuari. Genereu la seva clau pública.
2. Genereu una signatura σ_2 de l'usuari 2 per al missatge $m = (41, 35)$.
3. Agregueu les dues signatures (σ_1 i σ_2) en una sola signatura agregada σ_{ag}.
4. Verifiqueu la signatura agregada σ_{ag} fent servir l'algorisme de verificació de signatura agregada.
5. Verifiqueu de nou la signatura agregada σ_{ag}, aprofitant que els dos missatges signats són idèntics.

Per a facilitar la resolució de l'exercici, a continuació es proporcionen alguns valors precalculats:

$$
\begin{array}{rlrll}
G_0 &= (4,7) & G_1 &= (2,8z) & \\
2G_0 &= (41,35) & 2G_1 &= (26,42z) & \\
3G_0 &= (15,30) & 3G_1 &= (33,9z) & \\
4G_0 &= (17,1) & 4G_1 &= (28,30z) & \\
5G_0 &= (10,9) & 5G_1 &= (39,7z) & w_r((41,35),(2,8z)) = 40z+11 \\
6G_0 &= (10,34) & 6G_1 &= (39,36z) & w_r((41,35),(39,7z)) = 34z+7 \\
7G_0 &= (17,42) & 7G_1 &= (28,13z) & w_r((41,35),(28,13z)) = 35z+18 \\
8G_0 &= (15,13) & 8G_1 &= (33,34z) & \\
9G_0 &= (41,8) & 9G_1 &= (26,a) & \\
10G_0 &= (4,36) & 10G_1 &= (2,35z) & \\
11G_0 &= \mathcal{O} & 11G_1 &= \mathcal{O} &
\end{array}
$$

L'algorisme que acabem de presentar és, però, vulnerable a atacs de clau pública múrria (en anglès, es coneixen amb el nom de *rogue public key attacks*). En aquests atacs un atacant és capaç de generar una signatura agregada vàlida que inclou una signatura d'un missatge m per part d'una víctima V sense que la víctima hagi proporcionat tal signatura.

L'atac de clau pública múrria fa servir la clau pública d'una víctima $k_{pubV} = u_V$ i genera una signatura agregada σ_{ag} vàlida que inclou una signatura de la víctima del missatge m. L'atac consta dels passos següents:

1. L'atacant selecciona un enter aleatori $\alpha \in_R \mathbb{Z}_q$.
2. L'atacant calcula la clau pública auxiliar $k_{pubA} = u_A = \alpha \cdot G_1 \in \mathbb{G}_1$.
3. L'atacant calcula la clau pública múrria $k_{pubM} = u_A - u_V \in \mathbb{G}_1$.
4. L'atacant calcula la signatura agregada $\sigma_{ag} = \alpha \cdot H(m)$.
5. L'atacant presenta la signatura agregada σ_{ag} per al conjunt de missatges $M = \{m, m\}$ amb claus públiques $K^{pub} = \{k_{pubV}, k_{pubM}\}$

Noteu que l'adversari fa servir la clau pública k_{pubM} per a l'atac, però que en desconeix la clau privada corresponent (l'adversari ha creat aquesta clau pública combinant la clau pública de la víctima i la clau pública auxiliar que ha generat i per a la qual sí que en coneix la clau privada).

Exercici 9.6 Demostreu que l'atac de clau pública múrria aconsegueix generar una signatura agregada vàlida en la versió bàsica del BLS fent servir les propietats de bilinealitat del *pairing*.

Aquest atac trenca la seguretat de l'esquema de signatura agregada ingènua que hem descrit i motiva la creació de variants que en siguin resistents. Es coneixen diferents variants de l'esquema segures i, a continuació, en descriurem una d'elles.

L'algorisme d'agregació de signatures BLS segur parteix d'una variant modificada de l'algorisme de signatures BLS:

A partir d'un missatge en clar m, el parell de claus de l'emissor ($k_{priv} = \alpha$ i $k_{pub} = u = \alpha \cdot G_1 \in \mathbb{G}_1$), i els paràmetres de domini, es calcula la **signatura digital BLS modificada** del missatge:

1. Es calcula $\sigma = \alpha \cdot H(k_{pub}, m)$
2. La signatura és el valor σ.

És a dir, la signatura es fa no només sobre el missatge m sinó també sobre la clau pública del signant k_{pub}, impedint així l'atac de clau pública múrria que hem vist anteriorment. Això requereix de la utilització d'una funció hash H que rebi dos valors (el primer de $\mathbb{G}_1$ i el segon de l'espai de missatges) i retorni un valor de $\mathbb{G}_0$.

L'algorisme d'agregació de signatures es manté tal com l'hem definit anteriorment. L'algorisme de verificació de signatures agregades es modifica lleugerament de manera anàloga al procés de signatura:

A partir d'un conjunt d'n claus públiques $K^{pub} = \{k_1^{pub}, \cdots, k_n^{pub}\}$, d'un conjunt d'$n$ missatges $M = \{m_1, \cdots, m_n\}$ i d'una signatura agregada σ_{ag}, els passos següents permeten **verificar la signatura agregada**:

1. Es comprova si $e(\sigma_{ag}, G_1) \stackrel{?}{=} e(H(k_1^{pub}, m_1), k_1^{pub}) \cdots e(H(k_n^{pub}, m_n), k_n^{pub})$
2. La signatura agregada és vàlida si la comprovació del pas anterior es fa amb èxit. En cas contrari, es rebutja la signatura agregada.

Demostració La demostració de perquè aquesta variant és segura queda fora de l'abast d'aquest text. El lector interessat pot consultar l'article original *Compact Multi-Signatures for Smaller Blockchains* de Dan Boneh, Manu Drijvers i Gregory Neven per a llegir-ne els detalls.

9.4.2 Criptografia basada en la identitat

Els *pairings* permeten també construir esquemes de xifratge amb propietats addicionals als sistemes de xifratge tradicionals.

En els esquemes de xifratge de clau pública tradicionals, per tal d'enviar un missatge xifrat a una persona caldrà que en coneguem la seva clau pública. Ja hem vist com aquesta associació entre una clau pública i la identitat d'un usuari es fa habitualment amb un certificat digital. Per tant, per aconseguir la clau pública del receptor del missatge, l'emissor pot demanar-li el seu certificat digital o bé pot descarregar-lo d'un repositori públic de certificats. En qualsevol dels dos casos, l'emissor necessita aconseguir i validar la clau pública del receptor abans de poder iniciar el procés de xifratge del missatge. El xifratge basat en la identitat, proposat per Adi Shamir el 1984, permet evitar aquest procés previ i fer servir la identitat del receptor directament com a la seva clau pública.

Tot i que la idea del xifratge basat en la identitat va ser proposada el 1984, en aquell moment només es coneixia un esquema de signatura basat en la identitat (proposat pel mateix Shamir). No va ser fins al 2001 quan Dan Boneh i Matthew K. Franklin van proposar el primer esquema de xifratge basat en la identitat, que utilitzava el pairing de Weil sobre corbes el·líptiques.

El **xifratge basat en la identitat** (IBE, de l'anglès, *Identity Based Encryption*) és un tipus de xifratge de clau pública en què la clau pública d'un usuari es deriva directament d'una informació única sobre la identitat d'aquest usuari.

En els esquemes de xifratge basat en la identitat, qualsevol cadena de caràcters que identifiqui a l'usuari pot ser utilitzada per a calcular la clau pública. Cal, però, que aquest identificador sigui únic entre tots els usuaris de l'esquema. Alguns exemples d'identificadors habituals són el correu electrònic, el número de telèfon o el nom de domini.

Els esquemes de xifratge basats en la identitat requereixen d'una entitat de confiança, que és l'encarregada de generar les claus dels usuaris. L'entitat de confiança disposa d'un parell de claus mestra. La clau pública mestra és coneguda per totes les entitats del sistema i es fa servir en el procés de xifratge, mentre que la clau privada mestra és secreta (només coneguda per l'entitat de confiança). Aquesta clau privada mestra es fa servir per a derivar les claus privades dels usuaris.

Per tant, els usuaris necessiten interactuar amb l'entitat de confiança per tal d'obtenir les claus privades associades als seus identificadors, de manera que cal que l'entitat de confiança pugui autenticar els usuaris (per assegurar que obtenen la clau privada d'un identificador propi) i disposi d'un canal confidencial amb els usuaris (per transmetre'ls la clau privada).

Així, els usuaris d'un esquema IBE es poden intercanviar missatges xifrats sense necessitat d'haver tingut cap contacte previ entre ells per tal d'intercanviar-se les claus públiques però, en canvi, sí que caldrà que els usuaris puguin comunicar-se amb l'entitat de confiança.

El flux d'informació a l'hora de xifrar no és l'única diferència entre els esquemes IBE i els esquemes de clau pública tradicionals. Una altra diferència és la necessitat d'existència de les claus prèvia al procés de xifratge. En un esquema tradicional, l'usuari ha de generar un parell de claus abans de poder rebre un missatge xifrat. En canvi, amb IBE, l'usuari pot rebre un missatge xifrat per al qual encara no se n'ha generat la clau privada.

En una infraestructura de clau pública tradicional existeixen mecanismes de revocació dels certificats digitals, que permeten gestionar situacions com ara el compromís de les claus privades dels usuaris. En els esquemes basats en IBE, les claus públiques no es poden revocar, ja que no hi ha cap manera de comunicar a l'emissor que una clau ha estat revocada: l'emissor fa servir la identitat de l'usuari i la clau pública mestra per a xifrar un missatge, sense comunicar-se amb cap entitat que pugui informar-lo de la possible revocació d'una clau. Per a solucionar aquesta limitació, els esquemes IBE acostumen a combinar l'identificador de l'usuari amb una cadena que representi el període de temps en què es considera vàlida la clau. D'aquesta manera, una mateixa clau només és vàlida durant un període de temps concret, cosa que limita les possibles conseqüències d'una pèrdua o compromís.

Exemple 9.15 Identificadors per a IBE

Us sistema IBE per a correu electrònic pot fer servir com a identificador d'usuari el correu de l'usuari concatenat amb la data, de manera que les claus tindrien una vigència d'un dia.

Així, per exemple, l'identificador `2021-09-30:info@uoc.edu` seria la clau pública associada a l'adreça `info@uoc.edu` vàlida durant el dia 30 de setembre de 2021.

Fent servir aquest mateix format, es poden enviar correus que només es poden desxifrar en el futur, per exemple, xifrant un correu per a l'identificador `2221-09-30:info@uoc.edu`.

Una altra diferència notable dels esquemes IBE és que incorporen implícitament un sistema de recuperació de claus. Com que l'entitat de confiança pot calcular totes les claus privades de tots els usuaris, l'entitat de confiança pot recuperar qualsevol clau del sistema. A més, pot fer-ho sense necessitat de guardar claus individuals, ja que les claus privades dels usuaris es deriven directament de la clau privada mestra.

Els esquemes IBE consten dels tres algorismes habituals en els esquemes de xifratge (generació de claus, xifratge i desxifratge) més un algorisme addicional d'inicialització, que consisteix en la generació del parell de claus de l'entitat de confiança. Els dos algorismes de generació de claus (el de l'entitat de confiança i el dels usuaris del sistema) són executats per l'entitat de confiança. Els algorismes de xifratge i desxifratge són executats pels usuaris de l'esquema (emissors i receptors de missatges), com en els esquemes de xifratge tradicionals.

L'esquema bàsic de Boneh-Franklin

A continuació es presenta una de les construccions de xifratge basat en la identitat proposades per Dan Boneh i Matthew K. Franklin. La construcció fa servir *pairings* sobre corbes el·líptiques.

Sigui $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ un *pairing* on $\mathbb{G}_0, \mathbb{G}_1$ i $\mathbb{G}_T$ són grups cíclics d'ordre primer q amb $G_0 \in \mathbb{G}_0$ i $G_1 \in \mathbb{G}_1$ elements generadors dels grups.

L'esquema d'IBE fa ús d'un criptosistema de clau simètrica (tal que $m = D_k(E_k(m))$) i de dues funcions hash. Sigui H_0 una funció hash que relaciona els missatges de l'espai d'identificadors a elements de $\mathbb{G}_0$ i H_1 una funció hash que relaciona parells de $\mathbb{G}_1 \times \mathbb{G}_T$ amb les claus del criptosistema simètric.

L'algorisme d'inicialització és executat per l'entitat de confiança per tal de generar el parell de claus mestre:

L'algorisme d'inicialització consisteix en la **generació del parell de claus mestre** de l'entitat de confiança $\mathfrak{C}$ i consta dels passos següents:

1. Es tria un enter aleatori $\alpha \in_R \mathbb{Z}_q$.
2. Es calcula $u_1 = \alpha \cdot G_1 \in \mathbb{G}_1$.
3. La clau pública mestra és $k^{\mathfrak{C}}_{pub} = u_1$, mentre que la clau privada mestra és $k^{\mathfrak{C}}_{priv} = \alpha$.

L'algorisme de generació de claus d'un usuari és executat també per l'entitat de confiança, en el moment en què l'usuari li sol·licita la clau privada associada al seu identificador:

L'algorisme de **generació de claus** d'un usuari rep la identitat *id* de l'usuari i la clau privada mestra $k^{\mathfrak{C}}_{priv} = \alpha$ i executa els passos següents:

1. L'entitat de confiança $\mathfrak{C}$ calcula $sk_{id} = \alpha \cdot H_0(id) \in \mathbb{G}_0$.
2. La clau pública de l'usuari és $k_{pub} = id$, mentre que la clau privada és $k_{priv} = sk_{id}$.

L'algorisme de xifratge consisteix en la derivació d'una clau simètrica k a partir de l'identificador del receptor i la clau pública mestra. Aquesta clau simètrica es fa servir per a xifrar el missatge amb un algorisme de xifratge simètric.

A partir d'un missatge en clar m, la identitat del receptor *id*, i la clau pública mestra $k^{\mathfrak{C}}_{pub} = u_1$, es calcula el **missatge xifrat**:

1. Es tria un enter aleatori $\beta \in_R \mathbb{Z}_q$.
2. Es calcula $w_1 = \beta \cdot G_1 \in \mathbb{G}_1$.
3. Es calcula el *pairing* $z = e(H_0(id), \beta u_1) \in \mathbb{G}_T$.
4. Es calcula la clau simètrica $k = H_1(w_1, z)$.
5. Es calcula el missatge xifrat $c = E_k(m)$.
6. La sortida és la tupla (w_1, c).

Noteu com l'usuari pot xifrar sense obtenir la clau pública de l'usuari, ja que aquesta és directament l'identificador.

L'algorisme de desxifratge procedeix a derivar la clau simètrica k amb què s'ha xifrat el missatge, a partir de la informació que rep de l'emissor i la clau privada de l'usuari (que ha obtingut de l'entitat de confiança).

A partir d'un missatge xifrat (w_1, c) i la clau secreta d'un usuari $k_{priv} = sk_{id}$, es calcula el **missatge desxifrat**:

1. Es calcula el *pairing* $z = e(sk_{id}, w_1) \in \mathbb{G}_T$.
2. Es calcula la clau simètrica $k = H_1(w_1, z)$.
3. Es calcula el missatge en clar $m = D_k(c)$.
4. La sortida és el missatge en clar m.

L'algorisme serà correcte si la clau simètrica que es fa servir al desxifrar és exactament la mateixa que s'utilitza al xifrar. La clau simètrica k es deriva dels valors w_1 i z, i el valor w_1 es transmet com a part del text xifrat. Per tant, cal comprovar que les z que es calculen en els algorismes de xifratge i desxifratge són les mateixes. En efecte, per les propietats del *pairing*, podem veure com els valors z fets servir pels dos algorismes coincideixen:

$$
\begin{aligned}
e(sk_{id}, w_1) &= e(\alpha \cdot H_0(id), \beta \cdot G_1) = \\
&= e(H_0(id), G_1)^{\alpha\beta} = \\
&= e(H_0(id), \alpha \cdot G_1)^{\beta} = \\
&= e(H_0(id), u_1)^{\beta} = \\
&= e(H_0(id), \beta \cdot u_1)
\end{aligned}
$$

9.5 Resum

En aquest capítol s'han presentat els *pairings* sobre corbes el·líptiques, tot descrivint-ne les seves propietats. Després, d'una banda, s'han descrit les eines matemàtiques necessàries per entendre la seva formulació explícita i s'ha explicat com construir-los. D'altra banda, s'han presentat els algorismes criptogràfics més populars que els fan servir: un esquema de signatures que permet agregació (l'esquema BLS) i un esquema de criptografia basada en la identitat (l'esquema de Boneh-Franklin).

9.6 Solucions dels exercicis

Exercici 9.1:

El grau d'immersió d'E respecte a $n = 5$ és $k = 1$ ja que $5 \mid 11^2 - 1$.

El grau d'immersió d'E respecte a $n = 7$ no està definit, ja que $7 \nmid 15$.

El grau d'immersió d'E respecte a $n = 15$ no està definit, ja que 15 no és primer.

Exercici 9.2:

La funció pot expressar-se com:

$$f(x) = \frac{(x-1)^3(x+5)^2}{(x-12)^4} = (x-1)^3(x+5)^2(x-12)^{-4}$$

i, per tant:

$$div(f) = 3\langle 1\rangle + 2\langle -5\rangle - 4\langle 12\rangle - \langle \infty\rangle$$

Exercici 9.3:

La funció v_{Q+S} és la recta vertical que passa pel punt $Q+S$:

$$Q+S = (21, 12z) + (10z+18, 13z+13) = (19z+22, 12z+10)$$
$$v_{Q+S} : x = 19z+22$$

La funció $l_{Q,S}$ és la recta que passa pels punts Q i S:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{13z+13-12z}{10z+18-21} = 3z+1$$
$$y = mx + c; c = y - mx = 12z - 21(3z+1) = 18z+2$$
$$l_{Q,S} : y = (3z+1)x + (18z+2)$$

La funció $f_{r,Q}$ es construeix iterativament a partir d'$f_{1,Q}$:

$$f_{1,Q} = 1$$
$$f_{2,Q} = f_{1,Q}\left(\frac{l_{Q,Q}}{v_{2Q}}\right) = \frac{y+11zx+10z}{x-21}$$
$$f_{3,Q} = f_{2,Q}\, v_Q = \frac{y+11zx+10z}{x-21}(x-21) = y+11zx+10z$$

Exercici 9.4:

La signatura d'un missatge m seria:

$$\begin{aligned}\sigma &= \alpha \cdot H(m) = \\ &= \alpha \cdot H'(m) \cdot G_1 = \\ &= H'(m) \cdot \alpha \cdot G_1 = \\ &= H'(m) \cdot u\end{aligned}$$

i, per tant, la signatura dependria únicament del missatge m i la clau pública u. Així doncs, un atacant podria crear signatures vàlides només amb informació pública.

Exercici 9.5:

1. Calculem la clau pública:

$$k_2^{pub} = \alpha \cdot G_1 = 7(2, 8z) = (28, 13z) \in \mathbb{G}_1$$

2. Calculem la signatura:

$$\sigma_2 = \alpha_2 \cdot H(m) = 7(41, 35) = (15, 30) \in \mathbb{G}_0$$

3. Calculem la signatura agregada:

$$\sigma_{ag} = \sigma_1 + \cdots + \sigma_n = \sigma_1 + \sigma_2 = (4, 36) + (15, 30) = (41, 35)$$

4. Per validar la signatura agregada es comprova si $e(\sigma_{ag}, G_1) \stackrel{?}{=} e(H(m_1), k_1^{pub}) \cdot e(H(m_2), k_2^{pub})$:

$$e(\sigma_{ag}, G_1) = e((41, 35), (2, 8z)) = 40z + 11$$
$$e(H(m_1), k_1^{pub}) \cdot e(H(m_2), k_2^{pub}) = e((41, 35), (39, 7z)) \cdot e((41, 35), (28, 13z)) = 40z + 11$$

La igualtat es compleix, de manera que la signatura agregada és vàlida.

5. Per validar la signatura agregada aprofitant que les dues signatures corresponen al mateix missatge comprovem si $e(\sigma_{ag}, G_1) \stackrel{?}{=} e(H(m), k_1^{pub} + \cdots + k_n^{pub})$:

$$e(\sigma_{ag}, G_1) = e((41, 35), (2, 8z)) = 40z + 11$$
$$e(H(m), k_1^{pub} + \cdots + k_n^{pub}) = e((41, 35), (39, 7z) + (28, 13z)) = 40z + 11$$

De nou, la igualtat es compleix de manera que la signatura agregada és vàlida.

Exercici 9.6:

Per validar la signatura agregada, el verificador comprovarà si

$$e(\sigma_{ag}, G_1) \stackrel{?}{=} e(H(m), k_{pubV}) \cdot e(H(m), k_{pubM})$$

Com que:

$$\begin{aligned} e(\sigma_{ag}, G_1) &= e(\alpha H(m), G_1) = \\ &= e(H(m), \alpha G_1) = \\ &= e(H(m), k_{pubA}) = \\ &= e(H(m), k_{pubV} + k_{pubM}) = \\ &= e(H(m), k_{pubV}) \cdot e(H(m), k_{pubV}) \end{aligned}$$

la verificació serà correcta i la signatura agregada serà donada per vàlida.

9.7 Bibliografia

Boneh, Dan; Lynn, Ben; i Shacham, Hovav (2001). *Short signatures from the Weil pairing*. International conference on the theory and application of cryptology and information security. Springer, Berlin, Heidelberg.

Boneh, Dan; Drijvers, Manu; i Neven, Gregory (2018). *Compact Multi-Signatures for Smaller Blockchains*. International Conference on the Theory and Application of Cryptology and Information Security.

Boneh, Dan; i Victor Shoup (2020). *A graduate course in applied cryptography*.

Costello, Craig (2012). *Pairings for beginners*.

Martin, Luther (2008). *Introduction to identity-based encryption*. Artech house.

Kerry, Cameron F.; i Charles Romine (2013). *NIST FIPS PUB 186-4: Digital Signature Standard (DSS)*.

Open University, The (2016). *Further pure mathematics: Group theory*.

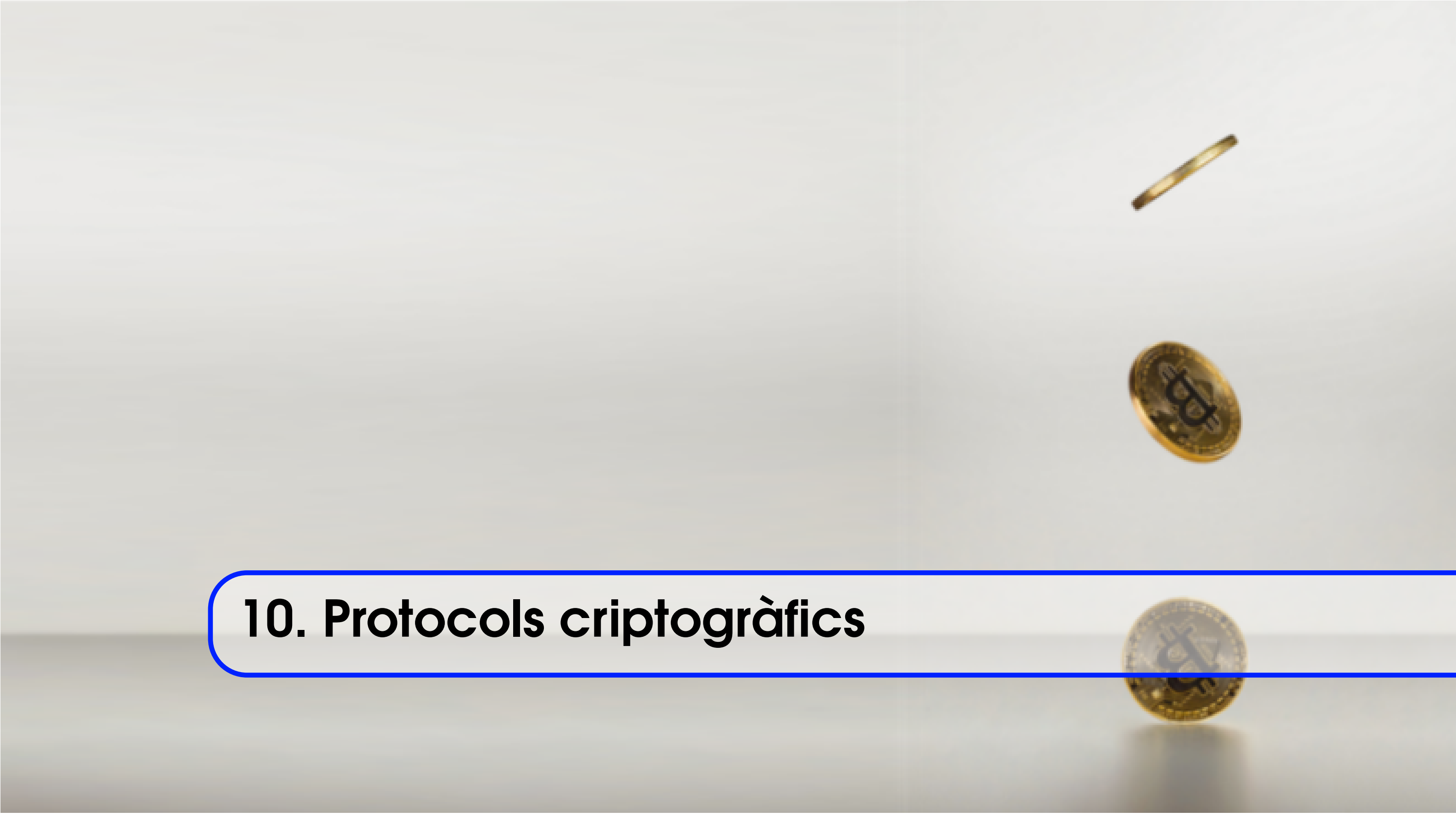
Paar, Christof, and Jan Pelzl (2009). *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media.

Perlroth, Nicole (2013). *Government announces steps to restore confidence on encryption standards*. The New York Times.

Schwennesen, Ben; i Hubert Bray (2016). *Elliptic curve cryptography and government backdoors*.

Warburton, David (2019). *The 2019 TLS Telemetry Report*. F5.

IV Protocols criptogràfics



10. Protocols criptogràfics

Més enllà dels mecanismes per xifrar i desxifrar missatges el cert és que la criptografia permet construccions més elaborades que continuen tenint el mateix objectiu que els criptosistemes: protegir la informació. Així, ens podem trobar diferents situacions on calguin protocols que ens garanteixin un seguit de propietats de seguretat que els criptosistemes per si sols no poden proporcionar. És en aquest punt on intervenen els protocols criptogràfics, protocols entre dos o més usuaris que utilitzen mecanismes criptogràfics per protegir la informació.

En aquest capítol estudiarem diversos protocols criptogràfics cada un d'ells amb un propòsit diferent. Llevat de l'esquema de compartició de secrets, els protocols descrits en aquest capítol són protocols en els que hi intervenen dos usuaris i no es contempla l'existència de cap tercera part de confiança. Així, les operacions es realitzen, sovint de forma conjunta, entre els dos usuaris per aconseguir l'objectiu del protocol. La suposició que es fa en tot moment és que els usuaris poden actuar de forma deshonesta de manera que és important que els propis protocols incorporin els mecanismes de seguretat necessaris per tal que, en cas que una part actuï de forma maliciosa, l'altra part no se'n vegi afectada o, com a mínim, pugui detectar l'engany.

10.1 El protocol de tres passos de Shamir

El protocol de tres passos de Shamir, va ser proposat per A. Shamir tot i que no el va publicar mai. El protocol permet establir una comunicació secreta entre dues parts sense cap intercanvi previ de claus. La base del protocol és una funció de xifratge commutativa respecte a les claus. És a dir, serà el mateix xifrar un missatge m amb una clau k_1 i el resultat tornar-lo a xifrar amb una clau k_2, que xifrar-lo primer amb la clau k_2 i el resultat xifrar-lo amb la k_1, és a dir:

$$E_{k_1}(E_{k_2}(m)) = E_{k_2}(E_{k_1}(m))$$

Passem a descriure el **protocol de tres passos de Shamir** en el qual l'Alice vol fer arribar el missatge m al Bob. Per fer-ho, l'Alice disposarà d'una clau per xifrar, k_A^e i una clau per desxifrar k_A^d i el Bob també tindrà una clau per xifrar k_B^e i una per desxifrar k_B^d. Denotarem per $E_{k_A^e}(m)$ l'acció de xifrar el missatge m amb la clau de xifrat k_A^e de l'Alice. Igualment, denotarem per $D_{k_A^d}(c)$ el desxifrat del missatge c amb la clau de desxifrat k_A^d de l'Alice.

En l'esquema de la Taula 10.1 es poden veure els diferents passos del protocol i la informació que s'intercanvien els usuaris que hi participen.

Taula 10.1: Esquema de 3 passos de Shamir

Pas	Alice		Bob
1.	Calcula $c_1 = E_{k_A^e}(m)$	$\xrightarrow{c_1}$	
2.		$\xleftarrow{c_2}$	Calcula $c_2 = E_{k_B^e}(c_1) = E_{k_B^e}(E_{k_A^e}(m))$
3.	Calcula $c_3 = D_{k_A^d}(E_{k_B^e}(E_{k_A^e}(m))) =$ $= D_{k_A^d}(E_{k_A^e}(E_{k_B^e}(m))) = E_{k_B^e}(m)$	$\xrightarrow{c_3}$	
4.			Calcula $m = D_{k_B^d}(c_3) = D_{k_B^d}(E_{k_B^e}(m))$

Com podem veure, al final del protocol l'Alice ha fet arribar a Bob el missatge m de forma segura ja que en cap dels missatges que s'han intercanviat en cada un dels tres passos el missatge m no ha viatjat en clar. Així, un atacant que estigui analitzant les comunicacions entre A i B no podrà extreure cap informació de m. Noteu, a més, que en cap moment s'ha produït un intercanvi de claus. L'Alice només coneix k_A^e i k_A^d i en Bob només k_B^e i k_B^d.

En els següents apartats veurem alguns criptosistemes que tenen la propietat de commutativitat de claus i quins resultats presenten quan s'utilitzen com a esquema de xifrat en el protocol de tres passos de Shamir.

10.1.1 El xifrat de Vernam i el protocol de tres passos de Shamir

Un dels criptosistemes que hem presentat en aquest llibre és el criptosistema de Vernam, que seria el criptosistema més segur que existeix ja que, utilitzant una bona clau, ens aporta seguretat incondicional. Si recordem, el mecanisme tant de xifrat com de desxifrat d'aquest criptosistema és molt simple. Donat un missatge m expressat en bits, i una clau k de la mateixa mida que el missatge també expressada en bits, la funció de xifrat consisteix en fer una XOR entre el missatge i la clau, és a dir $E_k(m) = m \oplus k = c$. D'altra banda, per desxifrar el missatge c simplement haurem de fer de nou una XOR amb la mateixa clau k amb la que hem xifrat $D_k(c) = c \oplus k = m$.

Si ens hi fixem, aquest criptosistema presenta commutativitat de claus, ja que si tenim dues claus k_1 i k_2 es compleix que:

$$E_{k_1}(E_{k_2}(m)) = (m \oplus k_2) \oplus k_1 = (m \oplus k_1) \oplus k_2 = E_{k_2}(E_{k_1}(m))$$

ja que l'operació XOR és commutativa.

Així, si utilitzem el criptosistema de Vernam per al protocol de tres passos de Shamir entre A i B tenim que les claus de xifrar i desxifrar per a cada usuari són la mateixa, és a dir, $k_A^e = k_A^d = k_A$ i $k_B^e = k_B^d = k_B$ i en els tres intercanvis d'informació del protocol es generaran els missatges mostrats en l'esquema de la Taula 10.2.

Taula 10.2: Esquema de tres passos de Shamir amb el xifrat de Vernam

Pas	Alice		Bob
1.	Calcula $c_1 = m \oplus k_A$	$\xrightarrow{c_1}$	
2.		$\xleftarrow{c_2}$	Calcula $c_2 = c_1 \oplus k_B$
3.	Calcula $c_3 = c_2 \oplus k_A = m \oplus k_B$	$\xrightarrow{c_3}$	
4.			Calcula $m = c_3 \oplus k_B$

Tot i que aparentment hem aconseguit desenvolupar el protocol correctament utilitzant un dels criptosistemes més segurs que hi ha, el problema està en que un atacant que pugui veure la comunicació en té prou en prendre nota dels tres missatges xifrats que s'intercanvien l'Alice i en Bob, ja que un cop intercepta c_1, c_2 i c_3 per obtenir el missatge xifrat m només cal que faci una suma XOR dels tres:

$$c_1 \oplus c_2 \oplus c_3 = (m \oplus k_A) \oplus (m \oplus k_A \oplus k_B) \oplus (m \oplus k_B) = m$$

Per tant, podem concloure que a l'hora d'utilitzar un criptosistema per al protocol de tres passos de Shamir no en tindrem prou en assegurar-nos que compleixi la commutativitat de les claus sinó que caldrà anar en compte sobre la relació que tenen els missatges una vegada han estat xifrats.

Aquest fet ens fa veure que, més enllà d'aquest exemple concret, en la creació de protocols criptogràfics és important no només que cada una de les eines criptogràfiques que s'utilitza sigui segura sinó que a més, la seva combinació ho continuï essent, fet que com hem vist, no sempre succeeix.

10.1.2 El criptosistema d'exponenciació

Un altre esquema amb commutativitat de claus el va proposar el mateix A. Shamir. Aquest sistema es basa amb l'exponenciació i la seva seguretat recau en la dificultat del càlcul del logaritme discret. És un criptosistema semblant amb l'RSA però no s'ha de confondre amb l'RSA ja que en aquest cas les dues claus que s'utilitzen, una per xifrar i l'altra per dexifrar, són dues claus secretes que únicament estan en possessió d'un sol usuari.

En primer lloc es tria un paràmetre per a l'intercanvi, un primer p gran. Totes les operacions es realitzaran al cos $\mathbb{Z}_p$. L'Alice genera les seves claus de la següent manera. Tria com a clau de xifrat k_A^e un valor aleatori i com a clau de desxifrat calcula el valor k_A^d tal que $k_A^e \cdot k_A^d = 1 \pmod{p-1}$. La funció de xifrat per a un missatge m serà $E_{k_A^e}(m) = m^{k_A^e} \pmod{p}$. La funció de desxifrat d'un missatge c serà $D_{k_A^d}(c) = c^{k_A^d} \pmod{p}$. De la mateixa manera, en Bob generarà les seves claus k_B^e i k_B^d i utilitzarà les mateixes funcions de xifrat i desxifrat. Amb aquestes condicions el protocol queda descrit en l'esquema de la Taula 10.3.

Taula 10.3: Esquema de tres passos de Shamir amb el xifrat d'exponenciació

Pas	Alice		Bob
1.	Calcula $c_1 = m^{k_A^e} \pmod{p}$	$\xrightarrow{c_1}$	
2.		$\xleftarrow{c_2}$	Calcula $c_2 = (c_1)^{k_B^e} \bmod p$
3.	Calcula $c_3 = (c_2)^{k_A^d} \bmod p = m^{k_B^e} \bmod p$	$\xrightarrow{c_3}$	
4.			Calcula $m = (c_3)^{k_B^d} \bmod p$

Fixeu-vos que, en aquest cas, un atacant que intercepti els tres missatges de la comunicació, c_1, c_2 i c_3 no podrà obtenir cap informació sobre el missatge transmès ja que les claus per xifrar només les coneixen A i B.

Exemple 10.1 Exemple de protocol de tres passos de Shamir amb el criptosistema d'exponenciació. En aquest exemple suposarem que els dos usuaris treballen amb el paràmetre $p = 131$. A més, l'usuari A disposarà de la clau de xifrat $k_A^e = 21$ i de la clau de desxifrat $k_A^d = (k_A^e)^{-1} \pmod{p-1} = 31$. D'altra banda, l'usuari B també tindrà el seu parell de claus. La de xifrat serà $k_B^e = 27$ i la de desxifrat $k_B^d = (k_B^e)^{-1} \pmod{p-1} = 53$.

Amb aquests paràmetres, l'usuari A vol enviar de forma secreta el missatge $m = 15$ a B i per fer-ho els passos del protocol seran els següents:

Pas	Alice		Bob
1.	$c_1 = 15^{21} \pmod{131} = 125$	$\xrightarrow{125}$	
2.		$\xleftarrow{27}$	$c_2 = (125)^{27} \bmod 131 = 27$
3.	$c_3 = (27)^{31} \bmod 131 = 129$	$\xrightarrow{129}$	
4.			$m = (129)^{53} \bmod 131 = 15$

Exercici 10.1 Reproduïu el protocol de tres passos de Shamir per tal que A enviï el missatge $m = 20$ a B utilitzant el criptosistema d'exponenciació on la clau de xifrat d'A val $k_A^e = 19$ la clau de desxifrat d'A val $k_A^d = 79$ i les corresponents claus de xifrat i desxifrat de B valen $k_B^e = 13$ i $k_B^d = 77$ respectivament. Suposarem, també, que $p = 101$.

10.2 Esquemes de compartició de secrets

Quan volem emmagatzemar un secret cal tenir en compte que hi ha situacions en les que el secret no pot ser guardat de forma centralitzada perquè hi ha el perill que aquesta centralització esdevingui un punt feble en la seguretat. En aquestes situacions el concepte de centralització pot tenir diferents vessants. Per exemple, imaginem-nos que tenim el codi d'obertura d'una caixa forta però no volem que estigui custodiat per una sola persona perquè té el perill que aquesta persona pugui marxar amb tots els diners. Voldríem poder distribuir aquest codi de manera que més d'una persona fos necessària per a l'obertura de la caixa forta.

Una altra situació, potser més quotidiana, és l'emmagatzemament de contrasenyes. Si emmagatzemem la contrasenya en un únic lloc, si aquest lloc sofrís algun incident perdríem la clau. Podríem solucionar aquest problema guardant la mateixa clau en diferents llocs, però això implicaria una reducció de la seguretat ja que les probabilitats que algú la trobi són més grans. Al igual que el que hem fet amb la caixa forta, podríem repartir el valor de la clau en diferents fragments. Fixeu-vos que en aquest cas, la possibilitat de poder recuperar la clau només amb alguns fragments (i no necessàriament amb tots) és important ja que si els necessitem tots per recuperar-la, tornem a estar en el punt de partida: si un dels llocs on hi ha un dels fragments de la clau sofrís algun incident no podríem recuperar la clau i també l'hauríem perduda.

Per resoldre aquests tipus de situacions tenim els esquemes de compartició de secrets. Aquests esquemes van ser proposats de forma independent l'any 1979 per Adi Shamir i George Blakley.

Un **esquema de compartició de secrets llindar** (m,n) (en anglès (m,n)-*threshold secret sharing scheme*), és un esquema que permet distribuir un secret en n fragments diferents de manera que si s'ajunten m o més fragments es pot recuperar el secret, però no és possible obtenir cap informació del secret si es disposen de menys d'm fragments.

Si assumim l'escenari en el qual volem repartir un secret S entre diferents usuaris, un esquema de compartició de secrets llindar (m,n) està format per n usuaris, $u_1, \cdots, u_n$. Cada usuari té el seu corresponent fragment s_i del secret S. A més cal que es compleixin les següents propietats:

1. Per a tot $i = 1, \cdots, n$, l'usuari u_i només coneix el seu fragment s_i.
2. El secret S es pot obtenir a partir d'm valors diferents s_i per a qualsevol $i \in \{1, \cdots, n\}$.
3. Donats $m-1$ valors diferents, s_i, no es pot obtenir cap informació d'S.

10.2.1 Esquema de compartició de secrets polinòmic

Un esquema per compartir secrets llindar (m,n) força utilitzat és el proposat per A. Shamir basat en la interpolació polinòmica.

Suposem que volem compartir el secret S utilitzant un esquema de llindar (m,n). Això vol dir que hem de crear n fragments i que en tenim prou en tenir-ne m per reconstruir-lo, però que menys d'aquesta quantitat no ens serà suficient. En aquest tipus d'esquema hi haurà un superusuari, el gestor, que serà l'encarregat de, partint del secret S, generar els n fragments. Com a paràmetres públics tindrem un nombre primer p tal que $p > n$ i $p > S$.

Per construir els fragments, el gestor construeix un polinomi $a(x)$ de grau $m-1$ amb coeficients a_i a $\mathbb{Z}_p$, és a dir $a(x) \in \mathbb{Z}_p[x]$. Aquest polinomi tindrà com a coeficients valors aleatoris, llevat del terme independent que serà exactament el valor secret S, és a dir, podem expressar el polinomi de la següent manera:

$$a(x) = S + a_1 x + a_2 x^2 + \cdots + a_{m-1} x^{m-1} \pmod{p}$$

La generació dels fragments es realitzarà de la següent manera. El gestor tria n valors aleatoris de $\mathbb{Z}_p$, $\{x_1, \cdots, x_n\}$, i per a cada valor en calcula la seva avaluació pel polinomi, és a dir, $a(x_i) = S + a_1 x_i + a_2 x_i^2 + \cdots + a_{m-1} x_i^{m-1} \pmod{p}$.

El polinomi $a(x)$ es manté en secret i només el coneix el gestor, però es pot eliminar una vegada s'han generats els fragments.

Cada participant rep com a fragment del secret el parell $\{x_i, a(x_i)\}$, és a dir, un valor x_i i la seva avaluació en el polinomi, $a(x_i)$.

Podrem recuperar el secret si tenim m fragments plantejant el següent sistema d'equacions:

$$\begin{aligned}
a(x_1) &= S + a_1 x_1 + a_2 x_1^2 + \cdots + a_{m-1} x_1^{m-1} \pmod{p} \\
a(x_2) &= S + a_1 x_2 + a_2 x_2^2 + \cdots + a_{m-1} x_2^{m-1} \pmod{p} \\
\vdots \quad & \quad \vdots \\
a(x_m) &= S + a_1 x_m + a_2 x_m^2 + \cdots + a_{m-1} x_m^{m-1} \pmod{p}
\end{aligned}$$

Si ens fixem, en aquest sistema hi tenim m incògnites corresponents als m coeficients dels polinomis $S, a_1, a_2, \cdots, a_{m-1}$ i també hi ha m equacions, per la qual cosa al resoldre'l obtindrem el valors de les incògnites i en particular la que ens interessa, que és el valor secret S. A més, aquest sistema sempre tindrà solució i serà única perquè hi intervé el determinant de Vandermonde.

Exemple 10.2 Exemple de protocol de compartició de secrets llindar (3,5)

Suposem que tenim cinc usuaris u_1, u_2, u_3, u_4, u_5 que volen repartir-se el valor secret $S = 673$. Per fer-ho utilitzaran l'esquema de compartició de secrets polinòmic de Shamir i treballaran amb el primer $p = 1931$.

Passem a descriure els dos processos d'un esquema de compartició de secrets: la generació dels fragments i la recuperació del secret.

Generació dels fragments:

Donat que amb 3 usuaris n'hi haurà prou per recuperar el secret, el gestor construirà un polinomi de grau 2 amb coeficients a $\mathbb{Z}_{1931}$ on el terme independent sigui el secret $S = 673$. Així, el gestor triarà dos valors aleatoris per crear el polinomi, per exemple 436 i 806 i construirà el polinomi $a(x) = 673 + 806x + 436x^2$. Amb aquest polinomi, procedirà a construir els fragments de cada usuari avaluant el polinomi en una component x per a cada participant. Si suposem que u_1 té la component $x = 1$, u_2 la component $x_2 = 2$ i així per a cada usuari, tindrem les següents avaluacions:

$a(1) = 673 + 806 \cdot 1 + 436 \cdot 1^2 = 1915 \pmod{1931}$
$a(2) = 673 + 806 \cdot 2 + 436 \cdot 2^2 = 167 \pmod{1931}$
$a(3) = 673 + 806 \cdot 3 + 436 \cdot 3^2 = 1222 \pmod{1931}$
$a(4) = 673 + 806 \cdot 4 + 436 \cdot 4^2 = 1218 \pmod{1931}$
$a(5) = 673 + 806 \cdot 5 + 436 \cdot 5^2 = 155 \pmod{1931}$

Per tant l'usuari u_1 rebrà el fragment $[1, 1915]$, l'usuari u_2 el fragment $[2, 167]$, l'usuari u_3 el fragment $[3, 1222]$, l'usuari u_4 el fragment $[4, 1218]$ i l'usuari u_5 el fragment $[5, 155]$.

Recuperació del secret:

Suposem ara que tres dels cinc usuaris es reuneixen per recuperar el secret. Suposem que són els usuaris u_1, u_4 i u_5 (però haguéssim pogut triar qualssevol tres altres). Els fragments d'aquests usuaris són $[1, 1915]$,

$[4,1218]$ i $[5,155]$ respectivament. Com que aquests valors són punts del polinomi utilitzat per generar els fragments, podem plantejar el següent sistema d'equacions:

$$\begin{array}{rcl} S + a_1 \cdot 1 + a_2 \cdot 1^2 & = & 1915 \pmod{1931} \\ S + a_1 \cdot 4 + a_2 \cdot 4^2 & = & 1218 \pmod{1931} \\ S + a_1 \cdot 5 + a_2 \cdot 5^2 & = & 155 \pmod{1931} \end{array}$$

Com que només ens interessa resoldre el sistema per la variable S, que és el secret, podem aplicar el mètode de Cramer i obtenim:

$$\frac{\begin{vmatrix} 1915 & 1 & 1 \\ 1218 & 4 & 16 \\ 155 & 5 & 25 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \end{vmatrix}} = \frac{352}{12} = 352 \cdot 161 = 673 \pmod{1931}$$

Exercici 10.2 Utilitzeu un esquema de compartició de secrets de Shamir per generar els fragments d'un sistema $(3,5)$-llindar per compartir el nombre secret 11. Preneu com a primer $p = 13$.

Exercici 10.3 En un esquema de compartició de secrets polinòmic de Shamir amb llindar $(3,6)$ els participants reben els següents fragments $(58,137), (11,48), (50,99), (80,50), (104,33), (39,114)$. Tenint en compte que treballen a $\mathbb{Z}_{149}$ recupereu el secret.

Exercici 10.4 En un esquema de compartició de secrets polinòmic de Shamir amb llindar $m = 3$, construït sobre $\mathbb{Z}_{13}$, l'usuari A té l'avaluació del polinomi per a $x = 1$, l'usuari B, $x = 2$ i l'usuari C, $x = 3$. Els tres usuaris es reuneixen per poder trobar la clau del sistema. Tots tres usuaris fan trampa; els usuaris A i C li sumen 2 a l'avaluació del polinomi en el seu punt però la clau que recuperen és la correcta. Quina és la trampa que ha fet l'usuari B?

10.2.2 Problemàtiques dels esquemes de compartició de secrets

A la pràctica, els esquemes de compartició de secrets tenen un seguit de restriccions que fan que el seu ús requereixi construccions molt més complexes que les que hem presentat aquí.

El primer punt a tenir en compte en un esquema de compartició de secrets és la confiança que es diposita en el gestor del sistema. Fixeu-vos que el gestor és el que s'encarrega de generar el polinomi que permetrà crear els fragments de cada participant i, per fer-ho, necessita el valor del secret. Per tant, cal que el gestor sigui una tercera part de confiança o bé que aquest procés es realitzi amb les garanties de seguretat necessàries.

D'altra banda, també ens podríem preguntar què passaria si un dels participants donés un valor aleatori en comptes del seu fragment. El cert és que el secret no es recuperaria i encara més, no sabríem qui ha estat el culpable. I encara pitjor, l'atacant podria utilitzar el secret recuperat erròniament, el seu fragment fals i el seu fragment correcte per recuperar el secret real sense l'ajut de la resta de participants mentre que la resta de participants continuarien sense poder recuperar el secret.

Per tal de resoldre aquests problemes hi ha els esquemes de compartició de secrets verificables, esquemes més elaborats que utilitzen mecanismes de compromís de bit que veurem més endavant.

Fixeu-vos, però, que totes aquestes problemàtiques no ens afecten quan només volem utilitzar l'esquema de compartició de secrets per emmagatzemar una contrasenya de forma distribuïda i segura ja que en aquest cas, tant el gestor com els usuaris que proporcionaran els fragments són tots el mateix.

10.3 Esquemes de compromís de bit

Hi ha situacions quotidianes en les que estem acostumats a fer servir alguns mecanismes molt simples que funcionen sense cap dificultat d'execució. Un d'aquests casos és el de 'tirar una moneda a l'aire' per, per exemple, decidir quin dels dos jugadors d'una partida d'escacs tindrà les fitxes blanques. Ara bé, quan les dues parts que duen a terme aquest petit protocol no es troben físicament al mateix lloc, la simplicitat de tirar una moneda a l'aire no ens serveix si hi ha certa desconfiança entre els dos participants.

Si analitzem el procés de tirar una moneda a l'aire veiem que, normalment, un dels dos usuaris tria cara o creu i l'altre, una vegada s'ha decidit qui guanyarà segons el revers de la moneda, tira la moneda a l'aire. En aquest simple esquema, l'usuari que tria cara o creu ho fa de forma pública, de manera que després (quan cau la moneda) no pot dir que ha triat una altra cosa. I l'usuari que tira la moneda no pot fer trampa (assumint que la moneda no està trucada!) perquè tira la moneda davant de l'altre usuari i els dos veuen el resultat que en surt, de manera que qui tira la moneda no pot canviar-ne el resultat.

Per emular aquest protocol de forma remota (o digital) es fa servir un esquema de compromís de bit.

> Un **esquema de compromís de bit** (en anglès, *bit commitment*) és una tècnica per la qual un usuari A es compromet, davant d'un usuari B, a un valor m per mitjà d'un valor $C(m)$, que serà el compromís. Aquest compromís ha de tenir les següents propietats:
> 1. Donat el compromís $C(m)$, B no pot obtenir informació del valor compromès m.
> 2. A ha de poder obrir el compromís $C(m)$ mostrant el valor compromès m.
> 3. A no pot obrir el compromís $C(m)$ mostrant un valor diferent al valor m compromès inicialment.

Amb un esquema de compromís de bit com el que acabem de descriure, el protocol de tirar una moneda a l'aire es pot definir amb els següents passos.

1. L'usuari A tria cara o creu i codifica la seva tria en el missatge m. Posteriorment, calcula el compromís d'm, $C(m)$, i l'envia a B.
2. B genera aleatòriament un bit, on 1 correspondrà al valor cara i 0 correspondrà a creu. B enviarà a A el valor aleatori generat.
3. A obrirà el compromís $C(m)$ mostrant a B quin valor (cara o creu) havia triat, de manera que es veurà qui ha guanyat en el protocol de tirar una moneda a l'aire.

Fixeu-vos que en el pas 2 del protocol, l'usuari A ja ha triat cara o creu però l'usuari B, tot i tenir el compromís $C(m)$, no pot saber quin valor ha triat (gràcies a la primera propietat de l'esquema de compromís de bit). En el pas 2, tot i que l'usuari B no generés el bit de forma aleatòria (per intentar alterar el protocol) el fet que no coneix si A ha triat cara o creu fa que la tria d'aquest valor aleatori sigui intrascendent. D'altra banda, en el pas 3, A ja sap quin valor ha obtingut B i per tant B no pot desdir-se'n. A més, A obre el seu compromís i, tot i conèixer el valor obtingut per B, no pot obrir-lo mostrant un altre valor diferent al que s'ha compromès, gràcies a la tercera propietat de l'esquema de compromís de bit.

Els protocols de compromís de bit es descriuen per mitjà de dues fases: fase de generació del compromís i fase d'obertura del compromís i en els següents apartats veurem dues tècniques diferents que implementen un esquema de compromís de bit.

10.3.1 Compromís de bit utilitzant funcions hash

Una de les tècniques més utilitzades per implementar un esquema de compromís de bit és mitjançant una funció hash, funcions que hem definit en el Capítol 5. Una funció hash h és una funció que parteix d'una informació de mida qualsevol x i en retorna un resum de mida fixa $h(x)$, un valor petit d'alguns centenars de bits. Perquè aquesta funció hash sigui considerada criptogràficament segura cal que compleixi tres propietats essencials. En primer lloc, donat un valor y tal que $h(x) = y$, no és possible trobar la seva antiimatge, x, és a dir, la funció hash no es pot invertir. D'altra banda, donats els valors x i y tals que $h(x) = y$ no és possible trobar un valor $x' \neq x$ tal que $h(x) = h(x') = y$. Finalment, tampoc és possible trobar dos valors x_1 i x_2 tals que $x_1 \neq x_2$ i que $h(x_1) = h(x_2)$. Amb una funció amb aquestes propietats podem definir un compromís de bit de la següent manera.

Sigui m el missatge al qual l'usuari es vol comprometre, en la **fase de generació del compromís** l'usuari A selecciona un valor aleatori r i calcula $C(m) = h(r \parallel m)$ on h és una funció hash criptogràfica.

En la **fase d'obertura del compromís** $C(m)$, l'usuari A revela els valors r i m. A partir d'aquests valors, l'usuari B pot calcular $h(r \parallel m)$ i comprovar que efectivament coincideix amb el valor $C(m)$ al qual A s'havia compromès.

Comprovem que aquest esquema compleix amb les tres propietats d'un esquema de compromís de bit.

1. B no pot obtenir el valor compromès m a partir el compromís $C(m)$ ja que $h(\cdot)$ és una funció hash criptogràfica i per tant no es pot invertir. Fixeu-vos que el valor aleatori r s'utilitza en cas que el missatge m se seleccioni d'un conjunt petit de missatges, per tal d'evitar que B pugui calcular totes les imatges de la funció hash per a tots els possibles valors diferents d'm i descobrir-ne el valor compromès.
2. A pot obrir el compromís $C(m)$ fent públics els valors r i m.
3. A no pot obrir el compromís, $C(m)$, obtenint un valor $m' \neq m$ perquè això voldria dir que A pot trobar $(r \parallel m) \neq (r' \parallel m')$ tal que $h(r \parallel m) = h(r' \parallel m')$ i això no és possible per les propietats que hem enumerat de la funció hash criptogràfica que s'utilitza.

10.3.2 Compromís de Pedersen

Un altre algorisme de compromís de bit és el Compromís de Pedersen, presentat per Torben Pryds Pedersen l'any 1991 com a part d'un esquema de compartició de secrets verificable. Les dues fases d'aquest tipus de compromís de bit es descriuen a continuació.

Sigui m el missatge al qual l'usuari es vol comprometre, en la **fase de generació del compromís** l'usuari A selecciona un grup multiplicatiu $\mathbb{G}$ d'ordre q i tria aleatòriament dos generadors d'aquest grup, g i h, tal que no es conegui el logaritme discret d'h en base g. Aquests valors seran valors públics de l'esquema. Aleshores A calcula el valor del compromís com $C(m) = g^m \cdot h^r \pmod q$, on r és un valor aleatori.

En la **fase d'obertura del compromís** $C(m)$, l'usuari A revela els valors r i m. A partir d'aquests valors, l'usuari B pot calcular $g^m \cdot h^r \pmod q$ i comprovar que efectivament coincideix amb el valor $C(m)$ al qual A s'havia compromès.

En aquest cas també es pot verificar que aquest esquema compleix amb les tres propietats d'un esquema de compromís de bit.

1. B no pot obtenir cap informació del valor compromès m a partir del compromís $C(m)$ ja que donat el missatge m, com que r és triat de forma uniformement aleatoria en el conjunt $\mathbb{G}$, el resultat del compromís $C(m) = g^m \cdot h^r \pmod q$ també és un valor uniformement distribuït en $\mathbb{G}$ i per tant B no en pot obtenir cap informació.
2. A pot obrir el compromís $C(m)$ fent públics els valors r i m.
3. A no pot obrir el compromís, $C(m)$, obtenint un valor $m' \neq m$ perquè això voldria dir que A pot trobar $(r, m) \neq (r', m')$ tal que $g^m \cdot h^r = g^{m'} \cdot h^{r'} \pmod q$. Però això no és possible perquè aleshores es podria calcular el logaritme discret d'h en base g com $\frac{m-m'}{r'-r}$ i això no pot succeir perquè el càlcul

del logaritme discret és un problema amb una complexitat massa elevada.

Exemple 10.3 Exemple d'esquema de compromís de bit de Pedersen

Suposem que l'usuari A vol comprometre's al valor $m = 11$. Per fer-ho, treballarà en el grup multiplicatiu $\mathbb{G} = \mathbb{Z}_{103}$ i triarà els valors $g = 6$ i $h = 88$ que són generadors de $\mathbb{Z}_{103}$. Aleshores, per a la generació del compromís, A tria com a valor aleatori $r = 17 \in \mathbb{Z}_{103}$ i calcula el compromís com $C(m) = 6^{11} \cdot 88^{17} \pmod{103} = 34$.

En la fase d'obertura, A envia a B els valors $(m,r) = (11,17)$ i B comprova que efectivament $6^{11} = 53 \pmod{103}$, que $88^{17} = 57 \pmod{103}$ i que per tant el seu producte és efectivament el valor $34 = 53 \cdot 57 \pmod{103}$.

Una de les característiques interessants que presenta el compromís de Pedersen és la seva propietat homomòrfica. Aquesta propietat ens permet generar el compromís de la suma de dos valors sense conèixer els valors i només a partir dels compromisos de cada un d'ells. Així, si tenim dos missatges m_1 i m_2 i els seus respectius compromisos de Pedersen, $C(m_1)$ i $C(m_2)$, tenim que el compromís del valor suma $m = m_1 + m_2$ serà igual al producte dels seus compromisos $C(m) = C(m_1) \cdot C(m_2)$.

En efecte, si escrivim la formulació per a cada un dels compromisos:

$$C(m_1) = g^{m_1} \cdot h^{r_1} \pmod{q} \quad \text{i} \quad C(m_2) = g^{m_2} \cdot h^{r_2} \pmod{q}$$

i en fem el producte, tenim

$$C(m_1) \cdot C(m_2) = (g^{m_1} \cdot h^{r_1}) \cdot (g^{m_2} \cdot h^{r_2}) \pmod{q} = g^{m_1+m_2} \cdot h^{r_1+r_2} \pmod{q} = C(m_1 + m_2)$$

Exercici 10.5 En un esquema de compromís de Pedersen s'utilitzen com a valors de l'esquema $\mathbb{G} = \mathbb{Z}_{113}$ i els generadors $g = 27$ i $h = 94$. L'usuari A s'ha compromès al valor $m_1 = 29$ amb el compromís $C(m_1) = 24$ i a més del missatge, el valor necessari per a obrir el compromís és $r_1 = 90$. Comproveu que efectivament, amb els valors $m_1 = 29$ i $r_1 = 90$ es pot obrir el compromís $C(m_1) = 24$. L'usuari A també s'ha compromès al valor $m_2 = 20$ per mitjà del compromís $C(m_2) = 91$. Calculeu el compromís per al valor $m_1 + m_2$, és a dir $C(m_1 + m_2)$. Podeu obrir el valor d'aquest compromís $C(m_1 + m_2)$?

10.3.3 Aplicacions dels esquemes de compromís de bit

Els esquemes de compromís de bit tenen múltiples aplicacions en protocols criptogràfics on hi ha una desconfiança mútua entre els usuaris que hi participen. Una de les aplicacions és en l'esquema de llançament d'una moneda, que ja hem detallat a l'inici d'aquest apartat.

Una altra aplicació d'aquests esquemes és en l'àmbit dels esquemes de compartició de secrets. Com ja hem comentat en l'apartat corresponent, quan en un esquema de compartició de secrets els usuaris mostren els seus fragments, en cas que alguns usuaris no proporcionin el fragment correcte, la resta d'usuaris no sap si el fragment proporcionat és correcte o no i poden recuperar un secret incorrecte. A més, en els esquemes de compartició de secrets, el gestor que reparteix el secret cal que sigui honest. Els esquemes de compartició de secrets verificables solucionen aquests problemes fent que el gestor que reparteix els secrets també reparteixi un compromís per a cada coeficient del polinomi que fragmenta el secret. D'aquesta manera, tot i no conèixer el polinomi, gràcies a les propietats homomòrfiques del compromís es pot comprovar si un fragment és o no correcte.

Els esquemes de compromís de bit també s'utilitzen en proves de coneixement nul. Com veurem més endavant, les proves de coneixement nul es basen en processos iteratius. Per tal de paral·lelitzar aquests processos sense que en la primera ronda es mostrin tots els valors, es pot utilitzar un esquema de compromís de bit per tal que una de les parts del protocol pugui seleccionar certs valors per endavant però sense necessitat de revelar-los, de manera que a posteriori, quan s'hagin d'utilitzar en el protocol, es puguin obrir els compromisos per revelar-ne el valor.

10.4 Signatures cegues

Un altre dels protocols interessants en criptografia són les signatures cegues, un protocol que s'utilitza per signar digitalment missatges de forma especial.

> En un **protocol de signatura cega** (en anglès *blind signature*) l'usuari A aconsegueix la signatura d'un missatge m per part de l'usuari B sense que B sàpiga quin missatge ha signat.

El concepte de signatura cega el va proposar David Chaum l'any 1982 per al seu ús en esquemes de pagament anònim.

Per imaginar-se com funciona un protocol de signatura cega és interessant utilitzar una analogia en termes de papers i signatures manuscrites. La idea és que l'usuari A té el document que ha de signar B i en comptes de proporcionar-li directament (fet que faria que B en pogués veure el contingut), A el posa dins d'un sobre. La peculiaritat d'aquest sobre és que està fet de paper carbó, és a dir, si escrivim alguna cosa fora del sobre es calcarà a l'interior. En particular, si B fa una signatura manuscrita fora del sobre, quan posteriorment traiem el document de dins del sobre tindrem el document signat per les propietats de calca del sobre de paper carbó. A més, B no haurà pogut veure el contingut del document que ha signat.

Com amb altres protocols criptogràfics, no és òbvia quina utilitat pot tenir que un usuari pugui signar un document sense saber el que signa. Tot i això, al llarg d'aquest apartat veurem en quines situacions tenen aplicabilitat les signatures cegues.

10.4.1 Signatura cega amb RSA

Donat que un protocol de signatura cega pretén obtenir la signatura digital d'un missatge, aquest protocol sempre inclourà un esquema de signatura digital en concret. A continuació veurem un protocol de signatura cega basat en RSA. Aquest mateix protocol és el que va idear D. Chaum quan va proposar el concepte de signatura cega.

Denotarem per m el missatge que A vol tenir signat per B. B signarà digitalment els seus missatges amb un esquema RSA. Per fer-ho utilitzarà la seva clau privada d. La clau pública corresponent a aquesta clau privada la denotarem per (e,n). El protocol es desenvoluparà en els següents passos:

1. A tria un valor aleatori r a $\mathbb{Z}_n$ tal que $gcd(r,n) = 1$ i el xifra amb la clau pública de B, és a dir, calcula $t = r^e \pmod n$. El valor t és el valor que utilitzarà per tapar el missatge m que B ha de signar. Per fer-ho, A calcularà $m' = m \cdot t \pmod n$ i enviarà el valor m' a B.
2. En rebre m', B simplement realitzarà la signatura sobre aquest valor de forma estàndard, utilitzant la seva clau privada d. Així obtindrà $s' = (m')^d \pmod n$ i enviarà el valor s' a A.
3. A destaparà la signatura feta per B simplement dividint la signatura que ha rebut de B, s', pel valor aleatori r generat en el primer pas, $s = \frac{s'}{r}$.

En la Taula 10.4 es mostra el protocol de forma esquemàtica.

Fixeu-vos que el valor s destapat per A en el pas 3 efectivament correspon a la signatura del missatge original m. Això és així perquè:

$$s = \frac{s'}{r} = \frac{(m')^d}{r} = \frac{(m \cdot t)^d}{r} = \frac{m^d \cdot t^d}{r} = \frac{m^d \cdot (r^e)^d}{r} = \frac{m^d \cdot r}{r} = m^d \pmod n$$

Taula 10.4: Protocol de signatura cega

Pas	Alice		Bob
1.	Tria $r \in_R \mathbb{Z}_n$ t.q $gcd(r,n) = 1$ Calcula $t = r^e \bmod n$ Tapat : calcula $m' = m \cdot t \bmod n$	$\xrightarrow{m'}$	
2.		$\xleftarrow{s'}$	Signa el valor m' calculant: $s' = (m')^d \bmod n$
3.	Obté la signatura de m calculant $s = \frac{s'}{r}$ (destapat)		

Exemple 10.4 Exemple de protocol de signatura cega amb RSA

Suposem que l'usuari A vol que l'usuari B li signi el missatge $m = 15$. L'usuari B utilitza per a realitzar signatures digitals el criptosistema RSA. La clau pública de B és $(e,n) = (19,551)$ i la corresponent clau privada $d = 451$. Amb aquests paràmetres, el protocol de signatura cega entre A i B serà el següent:

Pas	Alice		Bob
1.	Tria $25 \in_R \mathbb{Z}_{551}$ t.q. $gcd(15,551) = 1$ Calcula $t = 25^{19} = 310 \bmod 551$ Tapat : calcula $m' = 15 \cdot 310 = 242 \bmod 551$	$\xrightarrow{m'=242}$	
2.		$\xleftarrow{s'=14}$	Signa el valor $m' = 242$ calculant: $s' = 242^{451} = 14 \bmod 551$
3.	Obté la signatura de m calculant $s = \frac{14}{25} = 14 \cdot 529 = 243 \pmod{551}$		

Fixeu-vos que el valor $s = 243$ és efectivament la signatura del missatge original $m = 15$ ja que $s = 15^{451} = 243 \bmod 551$

Exercici 10.6 En un sistema d'autentificació anònima, l'usuari A té accés a un recurs S. Per poder-hi accedir, l'autoritat de certificació CA li generarà una credencial que consistirà en la signatura d'un missatge m que contindrà una clau pública generada per l'usuari A i l'identificador del recurs S. Per tal que la credencial sigui anònima, la CA realitzarà una signatura cega de manera que no tindrà manera de saber quina és la clau pública que certifica i per tant quan A accedeixi al recurs la CA no podrà saber-ho. Ara bé, per assegurar-se que A no accedeix a un recurs diferent, la signatura cega la realitzaran amb un protocol de triar i remenar. Així, A prepararà 5 missatges diferents m_i tals que $m_i = (PK_i||S)$, on PK_i serà una clau pública de la qual A en coneix la corresponent clau privada i el símbol $||$ denota la concatenació. Expliciteu tots els missatges que s'intercanviaran A i la CA en aquest protocol. Suposeu que treballen a $\mathbb{Z}_{899}$ i que els criptosistemes de clau pública que fem servir són l'RSA. Suposeu que el valor $S = 5$ i que el parell de claus (pública i privada) de la CA són $PK_{CA} = 19$, $SK_{CA} = 619$. Per simplificar, no cal indicar les corresponents claus privades de les 5 clau públiques triades.

10.4.2 Aplicacions de les signatures cegues

Hi ha múltiples escenaris on les signatures cegues són interessants d'utilitzar i la majoria d'ells tenen a veure amb la protecció de l'anonimat. Vegem com es poden fer servir en el següent escenari per tenir identificadors anònims.

Suposeu un sistema amb una autoritat central que té identificats als seus usuaris. Per tal de permetre utilitzar els recursos del sistema de forma anònima, els usuaris poden obtenir uns pseudònims per part de l'autoritat central. Aquests pseudònims estan signats digitalment per l'autoritat central una vegada ha comprovat que l'usuari té suficients privilegis per a utilitzar els corresponents recursos. La signatura de l'autoritat central sobre el pseudònim ha de permetre la seva validació per una tercera part quan l'usuari vol utilitzar el pseudònim davant d'algun dels recursos del sistema on es vol autenticar.

Amb aquest escenari, si l'autoritat central signa els pseudònims dels usuaris de forma estàndard, els usuaris obtindran anonimat davant dels tercers amb qui s'autentifiquin utilitzant el pseudònim. Ara bé, no aconseguiran anonimat davant de l'autoritat central ja que l'autoritat central, quan signa el pseudònim, sap la identitat real de l'usuari i, per tant, la correspondència entre la identitat real i el pseudònim, trencant així l'anonimat.

Una opció per resoldre aquest problema és que l'autoritat central signi el pseudònim però utilitzant un protocol de signatura cega. D'aquesta manera, l'autoritat central donaria validesa al pseudònim però no sabria a qui correspon el pseudònim.

10.4.3 Protecció contra abusos en les signatures cegues

Malgrat que les signatures cegues són interessants d'utilitzar en alguns escenaris, el cert és que la possibilitat que un usuari signi un valor sense saber exactament el que signa pot comportar també alguns problemes de seguretat. Per exemple, com ja hem estudiat anteriorment, la realització d'una signatura digital és equivalent al desxifrat d'un missatge. Per tant, un usuari A que hagués interceptat un missatge xifrat c dirigit a B, podria utilitzar un protocol de signatura cega per tapar c, fer-lo signar per B i d'aquesta manera obtenir el missatge desxifrat. D'altra banda, en escenaris més complexos, el contingut del que signa B pot ser rellevant i A pot voler-lo modificar per treure'n profit. Per exemple, imaginem-nos el cas descrit en l'apartat anterior en el que l'usuari A vol obtenir un pseudònim per autenticar-se. B només li proporcionarà el pseudònim en funció dels privilegis que tingui A en el sistema. A més, el pseudònim ha d'incloure aquesta informació per tal que A el pugui fer servir. Un possible atac d'A seria presentar un pseudònim amb unes atribucions diferents de les que el sistema li permet. Si B ha de realitzar una signatura cega, no podrà verificar aquestes condicions i podria arribar a signar condicions no desitjades.

Per evitar aquest tipus d'accions hi ha diferents estratègiques. La primera és utilitzar una clau específica per a les signatures cegues. És a dir, una clau pública que incorporés la pròpia semàntica de l'autorització. Per exemple, qualsevol pseudònim signat amb la clau pública que tingués el valor concret $PK_{CA}^{S_1,S_2}$ només serviria per autenticar-se davant dels recursos S_1 i S_2. Per a autentica-se davant del recurs S_3, per exemple, caldria tenir el pseudònim signat amb la clau pública $PK_{CA}^{S_3}$. A més, aquestes claus públiques de signatures cegues només es farien servir en aquest context i mai s'utilitzarien per xifrar missatges, de manera que l'atac per al desxifrat no seria possible.

Tot i que aquesta protecció que associa una semàntica a una clau és factible, a la pràctica pot comportar la gestió d'un volum de claus molt gran. Per evitar-ho una altra opció és utilitzar el procediment de "remenar i triar" per assegurar que B no signa res fraudulent. El procés funciona tal i com es mostra en la Figura 10.1.

L'usuari A, en comptes d'enviar un únic valor tapat m' a B, calcula múltiples valors tapats $m'_1, m'_2, \cdots, m'_n$. És important que cada valor s'hagi tapat amb un element diferent, és a dir, per a cada m'_i tindrem un valor t_i diferent, seguint la nomenclatura que hem utilitzat en l'esquema de signatura cega. Cada un d'aquests valors tapats $m'_1, m'_2, \cdots, m'_n$ conté certa informació que B ha de poder validar abans de signar i una altra informació diferent per a cada un dels valors m'_i. Per exemple, en el cas dels pseudònims per a l'autenticació, la part que ha de poder validar B és la part que indica a quins recursos permet accedir el pseudònim. Aquesta part ha de ser la mateixa per a tots els valors. La part que és diferent per a cada valor m'_i és la que indicarà el pseudònim que A farà servir.

Una vegada A ha enviat els n valors tapats $m'_1, m'_2, \cdots, m'_n$ a B, B demana a A que destapi $n-1$ valors, és a dir, A proporcionarà els corresponents t_i per a $n-1$ valors que B haurà triat aleatòriament. Una vegada

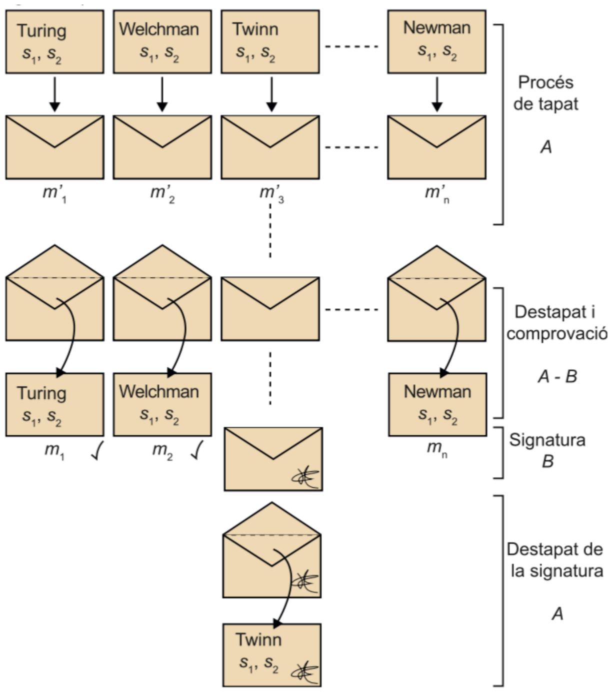


Figura 10.1: Esquema del mecanisme de remenar i triar

destapats, B podrà comprovar que la part que ha de validar coincideix en tots i cada un dels $n-1$ valors que ha destapat. Si és així, assumirà que el valor que li resta per destapar (el qual no pot destapar perquè A manté el corresponent t_i per fer-ho) també compleix les condicions estipulades. Per tant pot procedir a signar de forma cega aquest valor.

Fixeu-vos que cada un dels valors que ha destapat A pot contenir un pseudònim diferent, de manera que B no sap quin pseudònim hi haurà en el valor que ha signat. D'altra banda, la probabilitat que A pugui enganyar a B aconseguint que signi algun contingut que no vulgui es pot fer tant petita com es vulgui ja que el seu valor és de $\frac{1}{n}$.

10.5 Signatures d'anell

Un altre dels protocols relacionats amb les signatures digitals són les signatures d'anell, que van ser formalitzades per Ron Rivest, Adi Shamir i Yael Tauman al 2001.

En un **protocol de signatura d'anell** (en anglès *ring signature*) un usuari u_s que pertany a un grup d'usuaris $\mathscr{R} = \{u_1, \cdots, u_r\}$ (amb $s \in [1, r]$) signa un missatge m, de manera que un validador pot comprovar que la signatura ha estat realitzada per algun membre del grup $\mathscr{R}$ però, alhora, és computacionalment impossible saber quin usuari individual del grup ha realitzat la signatura.

La particularitat dels protocols de signatura en anell és que no necessiten cap mena de coordinador entre els membres del grup, ni tampoc cap procediment d'inicialització dels grups. A partir d'un conjunt d'usuaris cadascun dels quals té un parell de claus pública-privada, qualsevol usuari pot seleccionar un conjunt de possibles signants $\mathscr{R}$ (entre els quals hi és el propi usuari) i crear una signatura, sense necessitar l'ajuda o aprovació dels altres signants, ni la col·laboració de cap tercera part de confiança.

Aquest tipus d'esquemes poden ser útils, per exemple, per a permetre filtrar informació sense revel·lar la identitat de qui ha fet la filtració, però oferint garanties sobre la font. Així, un metge d'un hospital pot voler donar la seva opinió sobre la gestió d'una crisi sanitària a un periodista sense revel·lar la seva identitat (per por a represàlies), però assegurant al periodista que és un metge col·legiat. Així, el metge podria crear una signatura d'anell sobre el missatge on dona la seva opinió, seleccionant com a possibles signants un conjunt de metges col·legiats a la seva elecció. D'aquesta manera, el periodista podria verificar la signatura, comprovant que és vàlida per al grup de signants i que tots ells són metges col·legiats, i per tant podria estar segur que qui ha signat el missatge és efectivament un metge col·legiat. Alhora, la identitat individual del metge que ha signat el missatge quedaria oculta, i el periodista només sabria qui és amb probabilitat $1/r$, amb r el número de possibles signants.

10.5.1 Les signatures d'anell basades en RSA

En aquesta secció presentarem un dels esquemes de signatures d'anell fent servir claus RSA. Els usuaris del sistema disposaran doncs d'un parell de claus pública-privada del criptosistema RSA. Per tal de realitzar una signatura d'anell, un usuari seleccionarà un conjunt de claus públiques (que formaran l'anell, el grup de possibles signants del missatge entre els quals hi serà el propi usuari) i generarà una signatura fent servir les claus públiques dels altres membres de l'anell i la seva clau privada. Aquesta signatura podrà ser després validada per un receptor, coneixent el missatge original i les claus públiques dels usuaris de l'anell.

L'esquema fa servir quatre primitives criptogràfiques bàsiques:

- L'RSA.
- Un criptosistema de clau simètrica.
- Una funció hash.
- Una funció de combinació.

RSA

Farem servir $\mathscr{R} = \{u_1, \cdots, u_r\}$ per descriure al conjunt de possibles signants de l'anell, on $u_s \in \mathscr{R}$ és l'usuari que generarà la signatura. Cada usuari $u_i \in \mathscr{R}$ disposa d'un parell de claus RSA: una de pública $PK_i = (n_i, e_i)$ que és de domini públic, i una de privada $SK_i = (n_i, d_i)$ que només el propi usuari coneix.

Com ja hem vist en el Capítol 6, l'RSA es basa en el fet que la funció $f_i(x) = x^{e_i} \pmod{n_i}$ és computacionalment impossible d'invertir si no es coneix la factorització del mòdul n_i (el que permet calcular l'exponent privat d_i). Per tant, només els usuaris que són coneixedors de la clau privada SK_i, poden calcular $f_i^{-1}(y) = y^{d_i} \pmod{n_i}$.

Criptosistema de clau simètrica i funció hash

El protocol fa servir també un criptosistema de clau simètrica. Denotarem amb $E_k(m)$ la funció de xifrat del missatge m amb la clau k, i amb $E_k^{-1}(y)$ la funció de desxifrat. Els missatges a xifrar i desxifrar seran cadenes de b bits, i la mida de la clau del criptosistema simètric serà l.

Addicionalment, el protocol fa servir una funció hash h, que pot rebre entrades de qualsevol mida i retornarà cadenes d'l bits (que es faran servir com a clau del criptosistema de clau simètrica).

Funció de combinació

Per últim, el protocol fa servir una funció de combinació amb clau, que incorpora totes les primitives anteriors i que és la base del protocol de signatura d'anell. La funció de combinació rep una clau k d'l bits, un valor d'inicialització v de b bits, i un número arbitrari d'entrades y_i també de b bits, i retorna una cadena de b bits:

$$C_{k,v}(y_1, y_2, \cdots, y_r) = E_k(y_r \oplus E_k(y_{r-1} \oplus E_k(y_{r-2} \oplus E_k(\cdots \oplus E_k(y_1 \oplus v) \cdots)))) = z$$

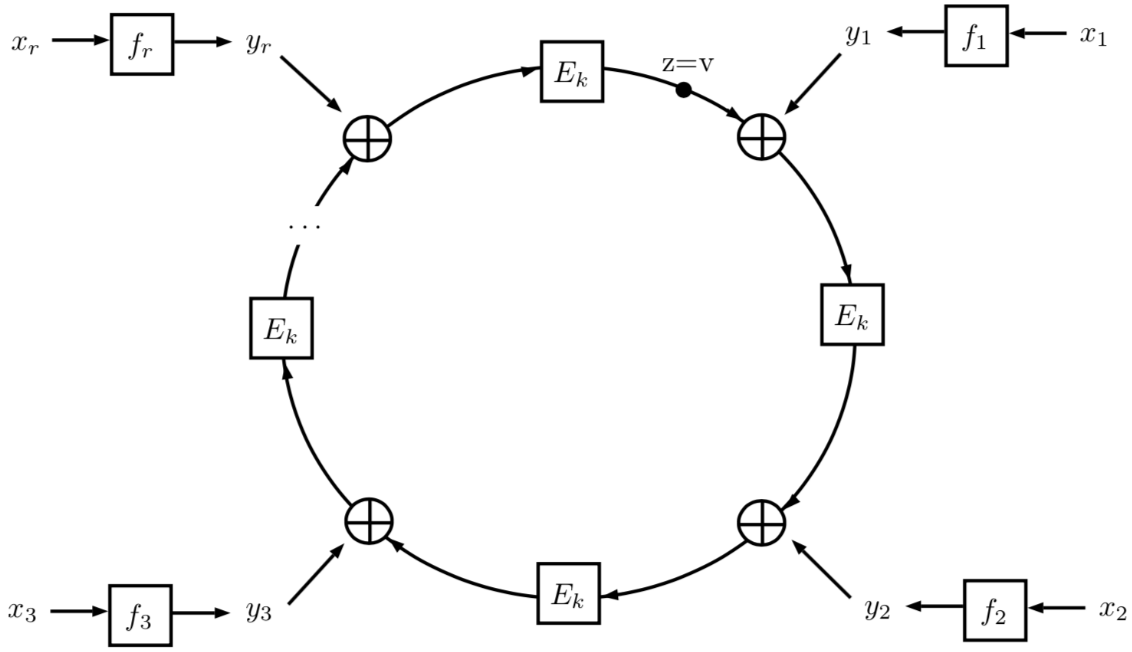


Figura 10.2: Esquema de la signatura d'anell amb RSA

Tot i que a primer cop d'ull la funció de combinació pugui semblar complexa, fixeu-vos que no fa res més que calcular, reiteradament, una xor entre dos valor, xifrant-ne després el resultat amb el criptosistema de clau simètrica.

El punt clau de l'esquema recau en com s'aplica la funció de combinació en la creació de les signatures d'anell. Doncs bé, d'una banda, la clau k que es fa servir en el criptosistema simètric correspon al hash del missatge a signar, és a dir, $k = h(m)$. D'altra banda, la funció s'aplica a la seqüència d'entrada $(y_1, y_2, \cdots, y_r)$, amb $y_i = f_i(x_i)$ per al conjunt de signants de l'anell. Per últim, es força que la sortida z de la funció de combinació hagi de ser igual al valor d'inicialització v, és a dir:

$$C_{k,v}(y_1, y_2, \cdots, y_r) = v$$

Aquest últim punt fa que sigui necessari conèixer com a mínim una de les funcións f_i^{-1} per tal de poder calcular tots els y_i de la seqüència d'entrada, de manera que podrem assegurar que només els membres de l'anell poden generar signatures vàlides per a aquell anell.

Addicionalment, aquest punt també és el que dona el nom de signatures d'anell a la construcció que estem presentant: al forçar que el valor de sortida de la funció de combinació hagi de ser igual al valor d'inicialització, es crea una estructura de dependències circular entre els valors, que té forma d'anell. La Figura 10.2 mostra com es combinen les diferents primitives criptogràfiques en la funció de combinació.

El protocol de signatura d'anell basat en RSA

Una vegada presentades les diferents primitives criptogràfiques que intervenen en el protocol de signatura d'anell basat en RSA, veiem ara com s'executa el protocol.

En primer lloc, descriurem a grans trets en què consisteix la creació i validació d'una signatura amb el protocol de signatura en anell.

En el procés de realització de la signatura, l'usuari que la genera calcularà els y_i corresponents a tots els altres possibles signants fent servir les seves claus públiques i calculant $y_i = f_i(x_i)$ per a un conjunt x_i de valors aleatoris. Tenint en compte la construcció de la funció de combinació, per a un missatge i valor d'inicialització concrets, això determinarà el valor y_s corresponent al signant. Coneixent la funció f_s^{-1}, el signant podrà invertir el valor y_s i trobar el valor x_s tal que $f_s(x_s) = y_s$. La signatura serà aleshores el conjunt de tots els valors x_i (incloent el del signant) amb les corresponents claus públiques, i el valor d'inicialització. Fixeu-vos que, a diferència d'una signatura convencional on no cal incloure la clau pública del signant, en aquest cas cal incloure totes les claus públiques utilitzades perquè qui posteriorment validi la signatura no té coneixement ni de qui realment ha realitzat la signatura ni de quines claus ha utilitzat en l'anell.

En el procés de validació, el validador podrà comprovar que la signatura és vàlida, ja que podrà recrear tots els y_i a partir dels x_i i les claus públiques, i comprovar després que efectivament l'equació de la funció de combinació es compleix per al valor v rebut. A més, el verificador no podrà saber quin usuari ha signat el missatge, ja que per a tots els usuaris de l'anell de possibles signants, en rep exactament la mateixa informació (el parell x_i, y_i).

En segon lloc, prosseguim a presentar el detall de l'execució del protocol.

El procés de realització de signatura s'inicia quan l'usuari que vol realitzar la signatura del missatge m, selecciona un conjunt d'usuaris, dels quals en coneix la clau pública, per formar part de l'anell de possibles signants $\mathscr{R}$. És a dir, el signant obté r claus públiques $\{PK_i = (n_i, e_i), i \in \{1, \cdots r\}\}$. Amb aquesta informació i els seu propi parell de claus (SK_s, PK_s) (fem servir l'índex s per referir-nos al signant) executa els següents passos:

1. El signant calcula els següent valors:
   - $k = h(m)$
   - b tal que $2^b > n_i$ per a $1 \leq i \leq r$
   - $v \in_R \{0,1\}^b$
   - $x_i \in_R \{0,1\}^b$ per a $1 \leq i \leq r$, per $i \neq s$
   - $y_i = f_i(x_i)$ per a $1 \leq i \leq r$, per $i \neq s$
2. Troba el valor y_s que soluciona l'equació: $C_{k,v}(y_1, y_2, \cdots, y_r) = v$
3. Calcula: $x_s = f_s^{-1}(y_s)$

Per tant, el valor de la signatura del missatge m serà

$$\sigma = \{PK_1, \cdots, PK_r, v, x_1, \cdots, x_r\}$$

Fixeu-vos que al Pas 1, l'usuari signant calcula un conjunt de valors necessaris per a la resolució de l'equació que descriu la funció de combinació. D'una banda, calcula la clau del criptosistema simètric k a partir del hash del missatge a signar. També tria un valor b tal que 2^b sigui major que tots els mòduls de les claus públiques dels usuaris de l'anell. Després, selecciona un valor d'inicialització v aleatori de b bits, així com $r-1$ valors aleatoris x_i (també de b bits). Finalment, per a cada valor x_i, calcula l'y_i corresponent fent servir la clau pública de cadascun dels altres usuaris de l'anell de possibles signants.

A continuació, al Pas 2, el signant resol l'equació plantejada per la funció de combinació, fent servir els valors calculats al pas anterior, per tal de trobar el valor y_s.

Una vegada calculat el valor y_s, al Pas 3 el signant troba el valor x_s tal que $f_s(x_s) = y_s$. Aquesta operació la pot fer ja que el signant coneix el valor de la clau privada, SK_s i, per tant, pot invertir la funció i calcular $f_s^{-1}(y_s) = x_s$. Noteu que aquest procés només el pot fer per al valor y_s, però no per cap altra valor y_i (amb $i \neq s$).

Finalment, al Pas 4 el signant genera la signatura σ, que no és res més que el conjunt de tots els valors x_i (un dels quals haurà calculat a partir de la seva clau privada, i els altres correspondran als valors aleatoris obtinguts al Pas 1), el conjunt de totes les claus públiques dels usuaris de l'anell (entre les quals hi haurà la del signant, que serà indistingible de les altres), i el valor d'inicialització v.

Per tal de validar la signatura σ, el verificador necessita tant el valor de la signatura $\sigma = \{PK_1, \cdots, PK_r, v, x_1, \cdots, x_r\}$ com el missatge m sobre el que s'ha realitzat la signatura. Amb aquestes dades, el verificador realitza els següents passos:

1. Calcula:

- $y_i = f_i(x_i)$ per a $1 \leq i \leq r$
- $k = h(m)$

2. Verifica que es compleixi la igualtat $C_{k,v}(y_1, y_2, \cdots, y_r) = v$

Fixeu-vos que al Pas 1 de la validació, el verificador procedirà a recalcular tots els y_i, aplicant les funcions f_i (que coneix ja que ha rebut també les claus públiques dels possibles signants) a cadascun dels valors x_i rebuts. També calcularà la clau k a partir del hash del missatge rebut.

Per acabar la validació, al Pas 2 el verificador comprovarà que el resultat de la funció de combinació per als valors y_i calculats, la clau k corresponent al missatge i el valor d'inicialització v rebut és també el valor v. Fixeu-vos que, efectivament, el verificador no pot saber quin dels usuaris ha signat el missatge, ja que no pot distinguir de cap manera l'usuari que ha signat de la resta d'usuaris de l'anell de possibles signants.

Exemple 10.5 Exemple de signatura d'anell basada en RSA

Per tal d'executar el protocol de signatura d'anell basada en RSA, caldrà primer triar una funció hash h i una funció de xifrat simètric E. Amb l'objectiu de simplificar al màxim l'exemple i centrar-nos en el càlcul de la signatura d'anell, seleccionem dues funcions senzilles per a aquestes dues primitives, que no oferiran la seguretat desitjada però que ens permetran exemplificar el protocol. Així, d'una banda, farem servir la funció identitat com a funció hash, de manera que $h(x) = x$ per a qualsevol valor d'entrada x. D'altra banda, farem servir una xor entre el missatge i la clau com a funció de xifrat simètric, de manera que $E_k(x) = x \oplus k$.

Suposarem també que el conjunt d'usuaris $\mathscr{U}$ amb claus públiques RSA conegudes que formaran part de l'anell serà $\mathscr{R} = \{u_1, u_2, u_3, u_4\}$ i les claus de cada un:
$PK_1 = (28907, 18541)$
$PK_2 = (41917, 22491)$
$PK_3 = (39407, 26077)$
$PK_4 = (32743, 17539)$
Per a aquest exemple, suposarem que l'usuari que fa la signatura és $s = 3$. La seva corresponent clau privada és $SK_3 = (39407, 27013)$. Suposarem també que el missatge sobre el qual vol realitzar la signatura és $m = 16962$.

El procés de signatura tindrà els següents passos:

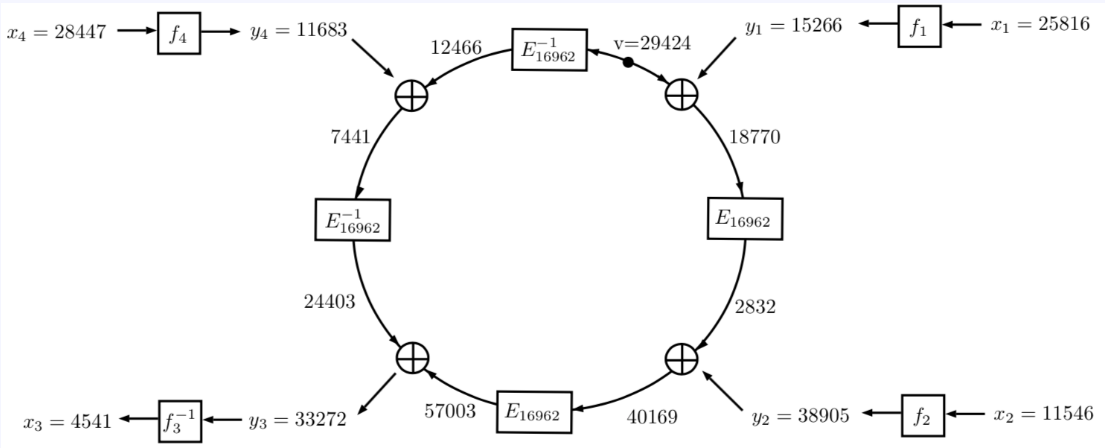
1. Calcula:
   - $k = h(16962) = 16962$
   - $b = 16$: ja que $2^{16} = 65536 > n_i$ per a $1 \leq i \leq 4$
   - $v = 29424 \in_R \{0,1\}^{16}$
   - $x = \{25816, 11546, \emptyset, 28447\}$ amb $x_i \in_R \{0,1\}^{16}$
   - $y_1 = f_1(25816) = 25816^{18541} \pmod{28907} = 15266$
   - $y_2 = f_2(11546) = 11546^{22491} \pmod{41917} = 38905$
   - $y_4 = f_4(28447) = 28447^{17539} \pmod{32743} = 11683$
2. Troba el valor y_3 que soluciona l'equació: $C_{16962,29424}(15266, 38905, y_3, 11683) = 29424$ obtenint com a solució $y_3 = 33272$
3. Calcula: $x_3 = f_3^{-1}(33272) = 33272^{27013} \pmod{39407} = 4541$

Per tant, el valor de la signatura serà:

$$\begin{aligned}\sigma &= \{PK_1, PK_2, PK_3, PK_4, v, x_1, x_2, x_3, x_4\} = \\ &= \{(28907, 18541), (41917, 22491), (39407, 26077), (32743, 17539), 29424, 25816, \\ &\quad 11546, 4541, 28447\}\end{aligned}$$

Al Pas 1, el signant realitza tots els càlculs per obtenir els valors necessaris per plantejar l'equació de la funció de combinació. Això inclou generar alguns valors aleatòriament (el valor v i també x_1, x_2 i x_4), i calcular els valors y_i corresponents als altres signants (y_1, y_2 i y_4). Noteu com el valor y_3, corresponent al signant, no s'ha calculat encara en aquest pas, ja que precisament serà la incògnita de l'equació de la funció de combinació, i es calcularà per tal d'assegurar que el resultat de la funció de combinació és exactament el valor d'inicialització.

Al Pas 2 el signant calcula el valor y_3 resolent l'equació donada per la funció de combinació. En la figura següent es mostra, gràficament, el procés que pot seguir el signant per fer aquest càlcul. El signant parteix del valor d'inicialització seleccionat $v = 29424$ i va calculant tots els valors que se'n deriven reseguint l'esquema de la figura per dos camins diferents: d'una banda, en sentit horari i, d'altra banda, en sentit antihorari. Això li permet anar fent tots els càlculs fins a arribar a trobar el valor y_3.



Una vegada sap que $y_3 = 33272$, al Pas 4 el signant calcula el valor x_3 amb la seva clau privada, i al Pas 4 genera la signatura.

El verificador, per verificar la signatura realitzarà els següents passos:

1. Calcula:
   - $y_1 = f_1(25816) = 25816^{18541} \pmod{28907} = 15266$
   - $y_2 = f_2(11546) = 11546^{22491} \pmod{41917} = 38905$
   - $y_3 = f_3(4541) = 4541^{26077} \pmod{39407} = 33272$
   - $y_4 = f_4(28447) = 28447^{17539} \pmod{32743} = 11683$
   - $k = h(16962) = 16962$
2. Verifica:
$$C_{16962,29424}(15266, 38905, 33272, 11683) = 29424$$

Fixeu-vos, que la verificació de la signatura és molt més immediata, i passa per calcular tots els y_i a partir dels x_i rebuts, i comprovar que l'equació de la funció de combinació es compleix per als valors y_i calculats. En aquest cas, efectivament es compleix, i el verificador dona per vàlida la signatura.

Detalls sobre la combinació de claus públiques

Per tal de simplificar la presentació del protocol, la secció anterior ha passat per alt un detall pel que fa als càlculs realitzats en el protocol. En aquesta secció, presentarem doncs una petita modificació en el protocol, que permetrà que operi correctament.

A l'hora de calcular una signatura, s'utilitzen conjuntament diferents claus RSA que pertanyen a diferents usuaris, i que habitualment tindran mòduls també diferents. A més, aquestes claus podrien tenir, fins i tot, mides diferents. Per tal de poder realitzar la signatura considerant les diferències entre els mòduls de les diferents claus, al Pas 2 del protocol es tria un valor b tal que $2^b > n_i$ per a tots els mòduls de les claus de l'anell, i aleshores es treballa sempre amb valors de b bits.

Si apliquem el protocol tal com s'ha descrit a la secció anterior, cada vegada que s'aplica una funció f_i (o f_i^{-1})

es genera una sortida menor a n_i. Per tant, mai s'obté com a resultat un valor entre n_i i 2^b. Aquest interval serà major o menor en funció del mòdul n_i, però en tot cas fa que la funció f_i no generi una permutació entre tots els elements de b bits, ja que hi ha elements vàlids com a entrada que mai es generen com a sortida. Per a evitar-ho, en comptes de fer servir la funció f_i, el protocol de signatura en anell basat en RSA fa servir la funció g_i definida de la següent manera:

$$g_i(m) = \begin{cases} q_i n_i + f_i(r_i) & \text{si } (q_i+1)n_i \leq 2^b \\ m & \text{altrament} \end{cases}$$

on q_i i r_i són enters no negatius tals que $m = q_i n_i + r_i$ i $0 \leq r_i < n_i$.

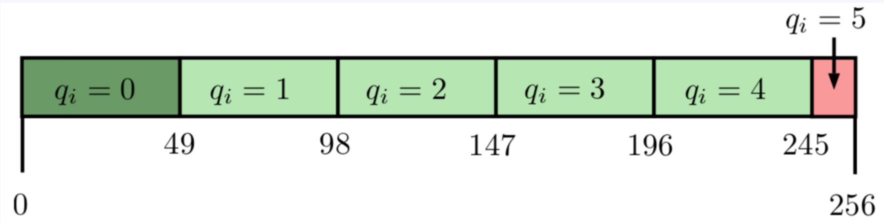
Així, per a valors $m < n_i$, la funció g_i retornarà el mateix que la funció f_i (noteu com aquests casos corresponen a $q_i = 0$). Per a valors d'm superiors al mòdul, intuïtivament podem dir que la funció g_i aplicarà la funció f_i al residu (als bits menys significatius) però mantindrà els bits més significatius. Això evitarà reduir la mida de la sortida. Finalment, per als casos excepcionals en els quals la primera expressió podria generar un valor de més de b bits, la funció g_i simplement retorna el valor que rep a l'entrada.

Fixeu-vos que amb l'ús de g_i, s'aconsegueix generar totes les possibles sortides de b bits i, alhora, es manté la propietat de l'RSA que es necessita per al protocol, ja que només l'usuari que sap com invertir f_i podrà invertir també g_i.

Exemple 10.6 Exemple de càlcul de g_i

Suposem un usuari $u_i \in \mathscr{R}$ amb una clau pública RSA $PK_i = (n_i, e_i) = (49, 11)$. A l'hora de fer la signatura d'anell, s'ha triat el valor $b = 8$, que compleix que $2^8 = 256 > 49$.

La imatge següent mostra gràficament els diferents intervals definits per a la funció g_i d'aquest usuari.



Els possibles valors d'entrada m de la funció g_i es troben a l'interval $[0, 256)$ (corresponen als valors que es poden representar amb $b = 8$ bits). La zona colorejada en verd correspon a la primera part de la definició de g_i, en la qual $(q_i + 1)n_i \leq 2^b$; la zona colorejada en vermell correspon a la segona part de la definió de g_i, que denota la resta de possibles valors d'entrada.

- El primer interval de la imatge, mostrat de color verd fosc, correspon als valors d'entrada inferiors al mòdul ($m < 49$): el valor q_i és 0, i la funció g_i retorna exactament el mateix que la funció f_i. Aquests són els valors que hem fet servir a l'exemple de la secció anterior, per tal de poder explicar el protocol sense haver de definir g_i. Així, per exemple, per a $m = 34$, $g_i(34) = f_i(34) = 41$.
- La resta d'intervals mostrats en color verd més clar corresponen a valors d'entrada inferiors a 245. En aquests casos, $0 < q_i < 5$, i la funció g_i aplica f_i sobre el residu r_i, però manté la mida de l'entrada. Així, per exemple, per al valor $m = 65$, $q_i = 1$ i $r_i = 16$. Aleshores, $f_i(16) = 4$, i $g_i(65) = 1 \cdot 49 + 4 = 53$. Fixeu-vos que, en aquest cas, la funció g_i retorna un valor superior al mòdul.
- Per últim, l'interval mostrat en vermell a la imatge correspon als valors d'entrada per als quals g_i retorna la mateixa entrada, i que corresponen als valors pels quals l'expressió $q_i n_i + f_i(r_i)$ podria generar una sortida de més de b bits. Així, per exemple, per a $m = 254$, tindríem que $q_i = 5$ i $r_i = 9$.

Aleshores, si calculéssim el resultat de l'expressió anterior:

$$q_i n_i + f_i(r_i) = 5 \cdot 49 + f_i(9) = 5 \cdot 49 + 46 = 291$$

Però $291 \geq 256$, de manera que es produiria un *overflow*. Per a evitar-ho, la funció g_i retorna el valor d'entrada, de manera que $g_i(254) = 254$.
Noteu com aquest aquest interval mostrat en vermell conté valors per als quals l'aplicació de l'expressió $q_i n_i + f_i(r_i)$ genera un valor de més de b bits, però no necessàriament tots els valors de l'interval generen aquest *overflow*. Per exemple, si avaluem l'entrada $m = 249$:

$$q_i n_i + f_i(r_i) = 5 \cdot 49 + f_i(4) = 5 \cdot 49 + 2 = 247$$

veiem que efectivament no sobrepassa el valor màxim de 256. Ara bé, com que l'entrada $m = 249$ és superior a 245, l'expressió que cal aplicar és: $g_i(m) = m$ i per tant $g_i(249) = 249$.

10.6 Proves de coneixement nul

Un dels usos de la criptografia és la gestió de la informació secreta. En ocasions la gestió d'aquesta informació pot comportar que ens interessi convèncer a algú que coneixem certa informació secreta però sense revelar aquesta informació. Dit d'una altra manera, ens interessa un mecanisme per poder demostrar que sabem un secret sense revelar-lo. Aquest concepte, batejat amb el nom de proves de coneixement nul, el van introduir S. Goldwasser, S. Micali i C. Rackoff l'any 1985.

Una **prova de coneixement nul** (en anglès *zero-knowledge proof*) és un protocol entre dos usuaris pel qual l'usuari que actua de provador, P, permet demostrar que coneix un cert valor secret s davant d'un usuari verificador, V, sense proporcionar el valor s. Al final del protocol, V estarà convençut que P coneix el valor s i alhora V no haurà obtingut cap informació sobre aquest valor.

Per tant, una prova de coneixement nul ha de complir les següents propietats:

1. **Correcció:** Si el provador coneix el valor s ha de poder convèncer al verificador que efectivament el coneix.
2. **Robustesa:** La probabilitat que el provador enganyi al verificador ha de ser molt petita. És a dir, si el provador no coneix el valor secret s, la probabilitat que la prova de coneixement nul s'executi correctament és molt petita.
3. **Coneixement nul:** Un cop realitzada la prova de coneixement nul, el verificador no té cap informació sobre el valor secret s que el provador coneix. En particular, el verificador no pot provar a una tercera persona, ni per mitjà d'una prova de coneixement nul, que coneix el secret.

Un exemple gràfic per entendre la mecànica de la majoria de les proves de coneixement nul és el que van proposar J.J. Quisquater i L. Guillou. En aquest exemple tenim una cova, com la que es mostra a la Figura 10.3. La cova té una entrada amb un únic camí. En un punt de la cova, el camí es bifurca i fa una volta fins a tornar-se a unir amb l'altra part del camí. Ara bé, el camí està tancat per una porta que s'obre per mitjà d'una paraula secreta. En Pep (P) coneix aquesta paraula secreta i vol convèncer a en Vicenç (V) que la coneix però no vol donar-li aquesta clau. Per fer-ho executen la següent prova de coneixement nul:

1. En Vicenç és queda a l'entrada de la cova (punt A del gràfic) mentre que en Pep entra dins i tria un dels dos camins fins a arribar a la porta. Per tant, pot estar en el punt C o bé en el punt D depenent de la tria que hagi fet.
2. Un cop en Pep ha arribat davant de la porta, en Vicenç avança fins a la bifurcació (punt B). Des d'allí tria un dels dos camins, el de la dreta o el de l'esquerra i li fa un crit a en Pep perquè surti pel camí

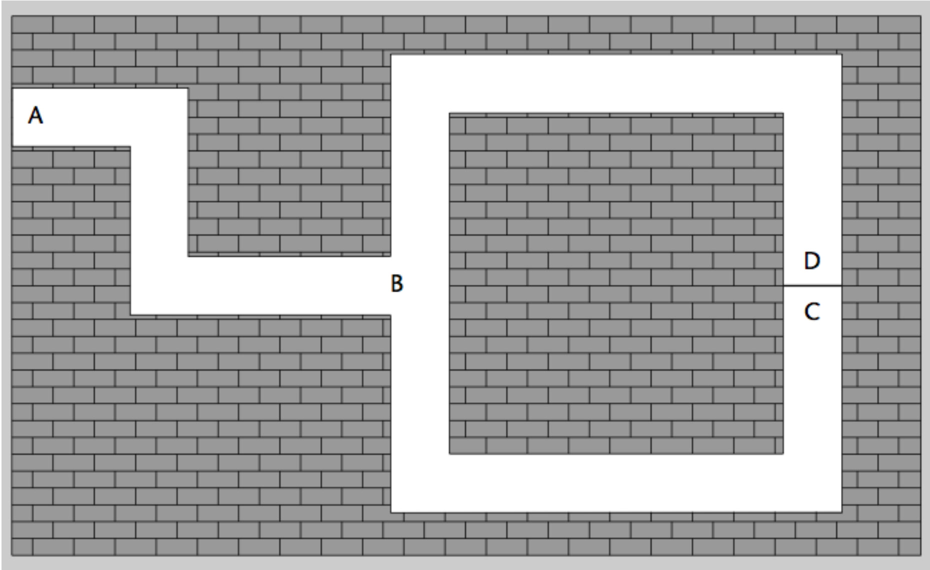


Figura 10.3: Gràfic de la cova de l'exemple

que ha triat.

3. Com que en Pep coneix la clau que obra la porta no tindrà cap problema per sortir pel costat que en Vicenç li ha demanat.

Comprovem ara si amb aquest exemple es compleixen les tres propietats que hem indicat anteriorment.

1. Si en Pep coneix la clau sempre podrà sortir per costat que en Vicenç li demana i per tant podrà demostrar que té el coneixement que vol provar.
   Si tornéssim a fer l'experiment i el repetíssim tantes vegades com volguéssim, en Pep sortiria sempre pel costat que en Vicenç li demanés, ja que coneix la clau que obre la porta i per tant no tindria cap problema.
2. Si en Pep no coneix la clau de la porta no hauria de poder convèncer al Vicenç que sí que la coneix. Fixeu-vos que si no coneix la clau, al fer la prova el Pep tindria una probabilitat d'$1/2$ d'encertar el camí que li demanarà més tard en Vicenç, ja que si en endinsar-se en la cova l'encerta, després podrà sortir pel mateix costat i no li caldrà utilitzar la clau de la porta que de fet no sap. Ara bé, si el procés el repetim un altre cop, en Pep només té $1/4$ de probabilitat d'enganyar-lo. Ja es veu que si repetim la prova n vegades la probabilitat que en Pep enganyi al Vicenç és d'$1/2^n$. Així doncs, si en Vicenç vol estar segur amb probabilitat 0,999023 que en Pep sap la paraula secreta que obre la porta només cal que realitzin la prova 10 vegades.
3. Un cop en Vicenç ha pogut validar que en Pep coneix la clau, en Vicenç no ha obtingut cap informació de la clau i tampoc pot utilitzar informació de la prova que ha fet amb en Pep, malgrat l'hagi repetida 10 vegades, per poder demostrar ell davant d'un tercer que coneix la clau.

L'exemple rebuscat

Com veurem més endavant, aquest exemple il·lustra com funciona una prova de coneixement nul, però òbviament, pel nostre propòsit, n'hi hauria prou en fer entrar en Pep per la dreta i fer-lo sortir per l'esquerra.

En general, les proves de coneixement nul funcionen d'aquesta manera, és a dir, són iteratives de manera que en cada iteració hi ha una probabilitat del 50% d'encertar. A més, en aquests tipus de protocols utilitzen la tècnica anomenada *challenge & response* on el verificador dóna al provador una informació que ell ha generat aleatòriament per tal que el provador la completi utilitzant el secret que coneix. Aquesta tècnica també s'anomena sovint *cut & choose* ja que fa referència al típic protocol de repartir un pastís entre dues persones, en el que una fa les parts (talla) i l'altre tria.

10.6.1 Prova del coneixement del logaritme discret

A continuació veurem un exemple concret d'una prova de coneixement nul aplicada al coneixement del logaritme discret d'un valor, prova que va ser proposada per D. Chaum, J. Evertse i J. Van de Graaf al 1987. Ja hem comentat anteriorment que el càlcul del logaritme discret té una complexitat elevada, és a dir donats uns valors y, g i p és difícil trobar per a quin valor x es compleix que $y = g^x \pmod{p}$. Per tant, aquesta prova de coneixement nul permet al provador demostrar que coneix el valor x que compleix l'equació $y = g^x \pmod{p}$ sense necessitat de revelar aquest valor.

El protocol funciona de la següent manera. En primer lloc el protocol estableix tres paràmetres públics (p, g, y), on p és un nombre primer gran, y és un nombre enter tal que $y < p$ i g és un generador del grup multiplicatiu Z_p. El provador ha de demostrar al verificador que coneix el valor x que satisfà l'equació $y = g^x \pmod{p}$. Per fer-ho el protocol realitza els següents passos:

Pas	Provador (P)		Verificador (V)
1.	Tria $r \in_R \mathbb{Z}_p \setminus \{0,1\}$ Calcula $c = g^r \pmod{p}$	$\xrightarrow{c}$	
2.		$\xleftarrow{b}$	Tria un bit aleatori $b \in_R \{0,1\}$
3.	Calcula $h = r + b \cdot x \pmod{p-1}$	$\xrightarrow{h}$	
4.			Verifica que $c \cdot y^b = g^h \pmod{p}$

El protocol consisteix en repetir n vegades els 4 passos descrits anteriorment.

Comprovem com es compleixen les propietats d'una prova de coneixement nul.

1. **Correcció:** en el cas que P conegui el valor x sempre podrà calcular el valor h en el tercer pas del protocol de manera que la validació que farà V en el pas quatre serà correcta.
2. **Robustesa:** per vertificar la propietat de robustesa, analitzarem com s'ho faria P per intentar fer creure a V que coneix x sense realment saber-ho. Per fer-ho, P ha de poder calcular el valor h del pas 3 sense conèixer r. Fixeu-vos que en cas que V li enviï a P el valor $b = 0$ en el pas 2, P calcularà $h = r + b \cdot x \pmod{p-1} = r \pmod{p-1}$ sense necessitat de saber x i aquest valor serà correcte i per tant superarà la validació del pas 4. Ara bé, si V tria $b = 1$ en el pas 2, aleshores P no pot calcular el valor h correcte (li falta el coneixement de x) de manera que no podrà concloure el protocol correctament. Fixeu-vos que la probabilitat que això passi és de $1/2$, ja que és la probabilitat que té V en el pas 2 de triar un 0 ó un 1. Per tant, si repetim el protocol n vegades, la probabilitat que P enganyi a V és d'$\frac{1}{2^n}$.
   Arribats a aquest punt, podríem pensar que no sembla que tingui sentit que V en el pas 2 enviï un 0, ja que en aquest cas, P no necessita conèixer x. Per tant, podríem concloure, erròniament, que en el pas 2, V podria enviar sempre un 1, forçant a P a conèixer x. Ara bé, aquesta estratègia no és correcta. Fixem-nos que si V sempre tria $b = 1$, P pot generar un valor r en el pas 1, però en comptes d'enviar $c = g^r \pmod{p}$ a V pot enviar $c' = \frac{g^r}{y} \pmod{p}$. Aleshores, en el pas 3, P envia r en comptes de $r + x$, però la verificació del pas 4 serà correcta perquè $c' \cdot y = \frac{g^r}{y} \cdot y = g^r = g^h$.
   Per tant, fixeu-vos que si P no sap si li arribarà un 0 ó un 1 en el pas 2 (i P no coneix el secret) no sap quina estratègia d'engany ha de seguir en el pas 1, és a dir si ha d'enviar $c = g^r \pmod{p}$ o bé $c' = \frac{g^r}{y} \pmod{p}$. Per tant, d'una manera o d'una altra té una probabilitat d'$1/2$ d'enganyar.
3. **Coneixement nul:** el protocol també té la propietat de coneixement nul ja que després d'executar-lo, V només coneix el valor g^r rebut en el pas 1, valor que no té cap relació amb el secret x. A més, el valor h rebut en el pas 3, tan pot correspondre al valor r com al valor $r + x$ i ambdós es presenten com a valors aleatoris per a V i per tant no poden proporcionar cap informació d'r.

Exemple 10.7 **Exemple de protocol de prova de coneixement nul del logaritme discret** Suposem que els paràmetres del protocol seran $p = 89$ i $g = 3$. El provador P coneix el logaritme discret de $y = 14 \pmod{89}$ que és $x = 9$. Suposarem que V tria el valor $b = 1$ en el pas 2. D'aquesta manera el protocol tindria els següents valors.

Pas	Provador (P)		Verificador (V)
1.	Tria $r = 20 \in_R \mathbb{Z}_{89} \setminus \{0,1\}$ Calcula $c = 3^{20} = 73 \pmod{89}$	$\xrightarrow{c=73}$	
2.		$\xleftarrow{b=1}$	Tria un bit aleatori $b = 1$
3.	Calcula $h = 20 + 1 \cdot 9 = 29 \pmod{88}$	$\xrightarrow{h=29}$	
4.			Verifica que $c \cdot y^b = 73 \cdot 14^1 = 43 \pmod{89}$ $g^h \pmod{p} = 3^{29} = 43 \pmod{89}$

Exercici 10.7 Voleu realitzar una prova de coneixement nul per demostrar que coneixeu el logaritme discret en base 7 de $y = 94$ a $\mathbb{Z}_{97}$, és a dir el valor x tal que $y = 7^x \pmod{97}$. El problema és que realment no coneixeu el valor x però voleu enganyar a un usuari fent una prova de coneixement nul i que es pugui convèncer que sí que el coneixeu. Afortunadament per vosaltres, el generador pseudoaleatori que fa servir el provador té una vulnerabilitat i vosaltres podeu saber el valor dels bits que genera en el pas 2 del protocol. L'usuari en qüestió vol fer una prova de coneixement nul que li asseguri que coneixeu el valor amb probabilitat superior a 0,75. Desenvolupeu tot el protocol de prova de coneixement nul assumint que el generador aleatori de V produeix els següents bits: 010011100.... Doneu el detall de les operacions i valors que s'intercanvien els usuaris en cada pas del protocol.

10.6.2 Aplicacions de les proves de coneixement nul

Les proves de coneixement nul tenen diferents camps d'aplicació. El primer camp és en els sistemes d'autenticació. El tradicional mètode de contrasenya comença a ser insuficient per a certes aplicacions ja que tant si aquesta, de forma incorrecta, es guarda en clar com si es guarda com a imatge d'una funció hash en algun moment l'usuari ha d'introduir-la en clar i és aleshores quan pot ser interceptada. A més, la utilització de la mateixa informació per a diferents processos d'autenticació pot donar lloc a atacs de repetició en el que un atacant utilitza informació d'una autenticació anterior per autenticar-se posteriorment. Utilitzant proves de coneixement nul, donat que el verificador no pot obtenir cap informació sobre el valor secret que té el provador, la possibilitat d'atacs de repetició desapareix.

Un altre camp on les proves de coneixement nul són importants és en la verificació de paràmetres en protocols criptogràfics més complexos. Per exemple, en protocols de votació electrònica, els votants han de proporcionar certs paràmetres per poder realitzar la votació. Alguns d'aquests paràmetres han de ser secrets, per preservar l'anonimat del vot, però a la vegada han de tenir certes característiques per tal que el protocol funcioni de forma correcta. Les proves de coneixement nul s'utilitzen per provar que un usuari coneix un paràmetre del protocol amb certes característiques sense haver de revelar cap informació del paràmetre en qüestió.

10.7 Protocol de transferència inconscient

Els protocols de transferència inconscient permeten que un usuari emissor "transmeti" informació a un altre usuari receptor de manera que al final de la transmissió, l'usuari receptor només obté una part de la informació "transmesa". A més, la particularitat d'aquests esquemes és que, d'una banda, l'emissor no sap quina informació finalment ha rebut el receptor i, d'altra banda, el receptor no obté cap informació de la informació que no li ha arribat.

0-1 OT	Aquest protocol es basa en la dificultat de calcular arrels quadrades modulars i amb la relació d'aquesta operació i la factorització d'enters.

El concepte de protocol de transferència inconscient va ser presentat per M. O. Rabin l'any 1981. La proposta de Rabin era un protocol en el qual l'emissor té un secret i, amb probabilitat 1/2 l'envia al receptor. Al final del protocol, el receptor pot tenir el secret o no tenir-lo (amb probabilitat 1/2) però l'emissor no pot saber si l'ha rebut o no. Aquest seria el que es coneix com a protocol de transferència inconscient 0-1. En aquest apartat, però, ens centrarem en els protocols de transferència inconscient 1-2.

En un **protocol de transferència inconscient 1-2** (en anglès 1-2 *Oblivious Transfer*) l'usuari A té dos secrets s_0 i s_1. Al final de l'execució del protocol entre A i B, B obté un dels dos secrets amb igual probabilitat. A més, A no pots saber quin secret ha rebut B i B no obtindrà cap informació sobre el secret que no ha rebut.

A continuació veurem un exemple concret d'aquest tipus de protocol.

10.7.1 Protocol d'Even, Goldreich i Lempel

Aquest protocol va ser proposat el 1985 pels criptogràfs Shimon Even, Oded Goldreich, i Abraham Lempel. La proposta utilitza el criptosistema RSA per tal de xifrar els valors secrets que hi intervenen. El protocol permet l'intercanvi inconscient 1-2 dels secrets s_0 i s_1 entre l'usuari A que és qui coneix els dos valors i l'usuari B que és qui en rebrà un dels dos. El funcionament del protocol així com les accions i els missatges que s'intercanvien en el protocol es mostra gràficament en el següent esquema:

Pas	Alice		Bob
1.	Secrets s_0 i s_1. Generació de la clau: $n = p \cdot q$ amb p, q primers $e \cdot d = 1 \pmod{\phi(n)}$ Genera $x_0, x_1 \in_R \mathbb{Z}_n$	$\xrightarrow{(e,n,x_0,x_1)}$	
2.		$\xleftarrow{v}$	Tria un bit aleatori b Genera $k \in_R \mathbb{Z}_n$ Calcula $v = x_b + k^e \pmod{n}$
3.	Calcula $k_0 = (v - x_0)^d \pmod{n}$ Calcula $k_1 = (v - x_1)^d \pmod{n}$ Calcula $s'_0 = s_0 + k_0 \pmod{n}$ Calcula $s'_1 = s_1 + k_1 \pmod{n}$	$\xrightarrow{(s'_0,s'_1)}$	
4.			Coneixent el valor b calcula $s_b = s'_b - k \pmod{n}$

En el Pas 1, A genera el parell de claus pública-privada i dos valors aleatoris. Envia la clau pública i els valors aleatoris a B. En el Pas 2, B triarà un dels dos valors aleatoris i l'amagarà utilitzant una clau. Fixeu-vos que la clau que utilitza per amagar el valor aleatori triat és k^e. Com que k ha estat triat aleatòriament, k^e també és un valor aleatori i el resultat v també aparenta un valor aleatori per a A ja que no coneix ni k ni k^e. En el Pas 3, A calcula dues claus k_0 i k_1 que utilitzarà per amagar els secrets s_0 i s_1 de la transferència inconscient obtenint els valors s'_0 i s'_1. El punt important està en com es calculen aquestes claus k_0 i k_1. Si ens fixem per exemple en k_0, en el cas que B hagi triat el calor x_0 en el Pas 2 tenim que:

$$k_0 = (v - x_0)^d = (x_0 + k^e - x_0)^d = (k^e)^d = k$$

És a dir que A haurà amagat el valor s_0 amb la clau k que B ha triat en el Pas 2. Per tant, B podrà descobrir el valor s_0 en el Pas 3 simplement restant-ne el valor k. En el cas que B hagi triat x_1 en comptes de x_0 en el Pas

2, podrà recuperar el secret s_1 ja que k_1 serà igual a k. Fixeu-vos que en aquest segon cas (B ha triat x_1) B no pot fer res amb el valor s'_0 per intentar esbrinar s_0 ja que no té cap informació de k_0 ja que:

$$k_0 = (v - x_0)^d = (x_1 - k - x_0)^d \neq k$$

Exemple 10.8 **Exemple de protocol de transferència inconscient 1-2** Suposem que Alice vol fer una transferència inconscient 1-2 a Bob dels secrets $s_0 = 22$ i $s_1 = 34$.

Pas	Alice		Bob
1.	Secrets $s_0 = 22$ i $s_1 = 34$. Generació de la clau: $n = 19 \cdot 29 = 551$ $e = 19$ i $d = 451$ Genera: $x_0 = 130, x_1 = 525$ aleatoris.	$\xrightarrow[x_0=130,x_1=525)]{(e=19,n=551,}$	
2.		$\xleftarrow{v}$	Tria un bit aleatori $b = 0$ Genera $k = 174$ Calcula: $v = 130 + 174^{19} = 304 \pmod{551}$
3.	Calcula: $k_0 = (304 - 130)^{451} = 174 \pmod{551}$ Calcula: $k_1 = (304 - 525)^{451} = 26 \pmod{551}$ Calcula: $s'_0 = 22 + 174 = 196 \pmod{551}$ Calcula: $s'_1 = 34 + 26 = 60 \pmod{551}$	$\xrightarrow{(s'_0=196,s'_1=60)}$	
4.			Coneixent el valor $b = 0$ calcula $s_0 = 196 - 174 = 22 \pmod{551}$

10.7.2 Aplicacions de la transferència inconscient

Com ja hem dit abans aquest protocol per si sol pot no tenir gaire interès però és la base d'altres esquemes com poden ser la signatura de contractes. Suposem l'escenari en el qual dos usuaris A i B volen signar digitalment un contracte però cap d'ells vol enviar primer la signatura a l'altre per no estar en desavantatge. Vegem com es pot aplicar la transferència inconscient 1-2 per solucionar aquesta situació.

L'usuari A descompon la seva signatura en $2n$ trossos d'm bits cada un, que denotarem per $\{a_i, 1 \leq i \leq 2n\}$. L'usuari B fa el mateix amb la seva signatura i obté els trossos $\{b_i, 1 \leq i \leq 2n\}$. Aleshores:

1. A divideix els seus $2n$ trossos de la seva signatura en n parells, per exemple (a_{2j-1}, a_{2j}) per $j = 1, \cdots, n$ i envia a B un element de cada parell utilitzant una transferència inconscient 1-2, per la qual cosa B rep a_{2j-1} o bé a_{2j}, per $j = 1, \cdots, n$, però A no sap quin dels elements ha rebut B (recordem que cada element del parell té un 50% de probabilitat de ser enviat).
2. Simultàniament al pas 1, B fa exactament el mateix amb els seus $2n$ trossos de la seva signatura: els divideix en parells i envia un element de cada parell a A utilitzant una transferència inconscient 1-2.
3. A i B s'envien l'un a l'altre el primer bit de tots els seus trossos a_i i b_i per $i = 1, \cdots, 2n$, després el segon bit, i així fins al final. Si A vol enganyar B, només té la probabilitat d'$1/2^n$ d'aconseguir-ho ja que B ja té n dels $2n$ nombres secrets del pas 1 i A no sap quins són. Simètricament, es pot aplicar el mateix si B vol enganyar a A.

Fixeu-vos que d'aquesta manera, A i B poden intercanviar la signatura del contracte i cap d'ells no està mai en avantatge de més d'un únic bit.

10.8 Protocols de recuperació privada d'informació

Els protocols de recuperació privada d'informació (coneguts també com a PIR per les seves sigles en anglès, *Private Information Retrival*) són una versió més laxa dels protocols de transferència inconscient presentats al capítol anterior. En ambdós tipus de protocols, un emissor envia dades a un receptor, amb la particularitat que l'emissor no sap quina informació finalment ha rebut el receptor. Els protocols es diferencien doncs en la informació que rep el receptor: mentre que en els protocols de transferència inconscient el receptor no obté cap informació de la dada que no li ha arribat, en els protocols de recuperació privada d'informació el receptor pot rebre informació addicional.

En els protocols de PIR hi intervenen, com a mínim, dues parts: un servidor que emmagatzema una base de dades i un usuari que vol fer una consulta sobre la base de dades. L'objectiu del protocol de recuperació privada d'informació és permetre a l'usuari recuperar un ítem de la base de dades sense que el servidor sàpiga quin ítem s'ha recuperat. Com comentàvem al paràgraf anterior, els protocols accepten que l'usuari recuperi més informació de la solicitada: l'objectiu del protocol és protegir la privadesa de la consulta de l'usuari envers del servidor que emmgatzema les dades.

Un **protocol de recuperació privada d'informació sobre una sola base de dades** (en anglès, *single-database Private Information Retrival protocol*) és un protocol d'intercanvi d'informació entre dos usuaris: una base de dades i un client. La base de dades té un conjunt d'n bits $D = b_1 b_2 \cdots b_n$ i l'usuari vol recuperar el bit en la posició i de la base de dades D sense revel·lar a la base de dades l'índex del bit que s'ha recuperat i.

Així doncs, un exemple de protocol de recuperació privada trivial consistiria en que l'emissor enviés la totalitat de les dades al receptor cada vegada que aquest fes una consulta. D'aquesta manera, efectivament, l'usuari rebria el bit d'interès i, alhora, l'emissor no sabria quin bit volia recuperar el receptor. Òbviament, aquest protocol és molt ineficient (la complexitat de la comunicació és de l'ordre de la mida de la base de dades), i se'n coneixen d'altres que milloren la complexitat d'aquesta versió trivial.

D'altra banda, hi ha protocols de PIR que es basen en la replicació de la base de dades. En aquests protocols, es creen k còpies de la base de dades, que s'emmagatzemen en servidors diferents. Aleshores, s'assumeix que els diferents servidors no poden comunicar-se entre ells (és a dir, no poden col·laborar en la seva tasca d'intentar esbrinar quina consulta ha fet l'usuari). L'usuari extreu informació parcial de cadascuna de les còpies, i la combina per tal de recuperar la informació que volia consultar. Els diferents servidors de bases de dades no aprenen res, individualment, sobre la consulta que ha fet l'usuari.

Un **protocol de recuperació privada d'informació amb k còpies de la base de dades** (en anglès, *k-database Private Information Retrival protocol*) és un protocol d'intercanvi d'informació entre un usuari i k servidors de base de dades. Cada servidor de base de dades té una còpia completa de la base de dades, un conjunt d'n bits $D = b_1 b_2 \cdots b_n$, i l'usuari vol recuperar el bit en la posició i de la base de dades D interactuant individualment amb cadascun dels servidors i sense revel·lar a cap servidor l'índex del bit que vol recuperar i, assumint que els servidors no poden comunicar-se entre ells i, per tant, no poden compartir la informació que coneixen de la consulta de l'usuari.

A continuació veurem el protocol de Kushilevitz i Ostrovsky, un exemple de protocol de PIR sobre una única base de dades i, després, el protocol de Chor et al., un exemple de protocol de PIR basat en la replicació de la base de dades.

10.8.1 Protocol de Kushilevitz i Ostrovsky

Notació Direm que $a \in QR(n)$ si $a \in \mathbb{Z}_n$ és un residu quadràtic. Per contra, direm que $a \in QNR(n)$ si $a \in \mathbb{Z}_n$ no és un residu quadràtic.

Aquest protocol va ser proposat el 1997 per Eyal Kushilevitz i Rafail Ostrovsky. La proposta utilitza el criptosistema probabilístic de Golwasser i Micali, per a xifrar la consulta que l’usuari realitza, de manera que la base de dades calcula el resultat sense conèixer els índexs que l’usuari està consultant.

En aquest protocol la base de dades és una matriu de bits, que té s files i t columnes, i que denotarem per $M_{s\times t} = (m_{ij})$. Així, el bit que es troba a la fila i columna j correspon al bit m_{ij}, i la base de dades té $s\times t$ bits emmagatzemats. L’usuari farà una consulta per recuperar un bit específic de la base de dades, que correspondrà al bit que es troba a la fila i' columna j', $m_{i'j'}$.

El funcionament del protocol així com les accions i els missatges que s’intercanvien en el protocol es mostra gràficament a l’esquema següent:

Pas	Usuari		Base de dades
	Índexs del bit a recuperar $i'\,j'$.		Matriu de bits $M_{s\times t} = (m_{ij})$
1.	Generació de la clau: $n = p \cdot q$ amb p, q primers aleatoris $a \in_R QNR(n)$		
2.	Calcula: $r_j \in_R \mathbb{Z}_n$ per tot $1 \le j \le t$ $x_j = \begin{cases} ar_j^2 \pmod{n}, & j = j' \\ r_j^2 \pmod{n}, & j \neq j' \end{cases}$	$\xrightarrow{(n,\{x_1,\cdots,x_t\})}$	
3.		$\xleftarrow{\{z_1,\cdots,z_s\}}$	Calcula: $z_i = \prod_{j=1}^{t} y_{ij}\ \forall\ 1 \le i \le s$ amb $y_{ij} = \begin{cases} x_j^2 \pmod{n} & \text{si } m_{ij} = 0 \\ x_j \pmod{n} & \text{si } m_{ij} = 1 \end{cases}$
4.	Recupera el bit $m_{i'j'}$ comprovant si z_i' és un QR: Si $z_{i'} \in QR(n)$, llavors $m_{i'j'} = 0$ Si $z_{i'} \in QNR(n)$, llavors $m_{i'j'} = 1$		

En el Pas 1, l’usuari que vol consultar la base de dades genera aleatòriament un parell de claus pública i privada del criptosistema de Goldwasser-Micali. La clau pública correspon als valors (n, a), mentre que la clau privada són els primers que factoritzen n: (p, q). El valor a es tria aleatòriament entre els valors que no són residus quadràtics mòdul n, de manera que no existeix cap x tal que $x^2 = a \pmod{n}$.

Una vegada l’usuari ha generat el parell de claus, procedeix a calcular els valors que codifiquen la consulta que vol fer a la base de dades. Així, en el Pas 2 l’usuari genera t valors aleatoris r_j, un per cada columna de la matriu que conforma la base de dades, i procedeix a calcular els t valors x_j, cadascun dels quals fa servir el corresponent valor aleatori r_j. Els valors x_j es calculen elevant al quadrat els r_j, a excepció de l’$x_{j'}$ (el valor que correspon a la columna que conté el bit d’interès), que es calcula elevant el valor aleatòri al quadrat i multiplicant-lo per a. D’aquesta manera, els valors x_j són residus quadràtics a excepció de l’$x_{j'}$, que no ho és. L’usuari envia aleshores els t valors x_j i el valor n a la base de dades.

Xifrat amb Goldwasser-Micali	Donada una clau pública (n,a) i un bit en clar m, la funció de xifrat del criptosistema de Goldwasser-Micali és: $E(m) = a^m r^2 \pmod{n}$, amb r un valor aleatori de $\mathbb{Z}_n^*$.

Fixeu-vos, d'una banda, que el Pas 2 és equivalent a xifrar una tira de t bits amb el criptosistema de Goldwasser-Micali, on tots els bits són 0 a excepció del que es troba a la columna j' (que és la columna on hi ha el bit que l'usuari vol recuperar). D'altra banda, noteu com al rebre els valors x_j i el valor del mòdul n, la base de dades no aprén res sobre els índexs de la consulta que l'usuari està realitzant: com que la base de dades no coneix la factorització del mòdul n, no pot distingir els x_j que són residus quadràtics del que no ho és i, per tant, desconeix quina és la columna d'interès per a l'usuari.

Al Pas 3 la base de dades procedeix a calcular el resultat de la consulta. Per fer-ho, calcula un valor z_i per a cada fila de la matriu, que correspon al producte dels x_j que ha rebut de l'usuari, elevant-los al quadrat si el bit m_{ij} de la matriu és un 0. Després, la base de dades envia el resultat de la consulta, és a dir, el conjunt d's valors z_i, a l'usuari.

Finalment, l'usuari recupera el resultat de la consulta a partir de les dades enviades per la base de dades al Pas 4, comprovant si el valor $z_{i'}$ (corresponent a la fila d'interès) és o no un residu quadràtic: si ho és, aleshores el bit recuperat de la base de dades, $m_{i'j'}$, és 0; per contra, si $z_{i'}$ no és un residu quadràtic, aleshores el bit recuperat és 1. En efecte, el valor $z_{i'}$ conté el producte dels x_j elevats al quadrat si corresponen a un bit 0 de la matriu i sense elevar al quadrat si corresponen a un bit 1 de la matriu. Si recordem com s'havien construït els x_j, veurem com aquests són residus quadràtics a excepció de l'$x_{j'}$, que no ho és. Per tant, el valor $z_{i'}$ serà un producte de residus quadràtics si $m_{i'j'}$ és 0 (l'únic valor que no ho era s'ha elevat al quadrat al Pas 3). En canvi, si $m_{i'j'}$ és 1, el producte $z_{i'}$ contindrà un factor que no és un residu quadràtic (precisament el que es troba en la posició j'), fent que el resultat $z_{i'}$ no sigui un residu quadràtic.

Fixeu-vos que el procés que permet recuperar els bits de la base de dades consisteix a calcular si un valor és o no un residu quadràtic mòdul n. Aquest és un problema computacionalment intractable quan n és un valor compost però, en canvi, és molt simple de calcular si el mòdul és un valor primer, o bé si es coneix la factorització d'n. Així doncs, l'usuari que fa la consulta coneix els dos primers p i q tals que $n = pq$, i és aquest coneixement el que li permetrà calcular si el valor $z_{i'}$ és o no un residu quadràtic.

És interessant notar que aquest protocol és un protocol de recuperació privada d'informació, ja que com a resultat de la consulta l'usuari obté més informació de la que ha demanat. Així, mentre que l'usuari únicament estava interessat a recuperar un únic bit, $m_{i'j'}$, l'usuari rep de la base de dades els z_i corresponents a totes les files. Per tant, l'usuari podria seguir el mateix procediment de recuperació per a tots els z_i, i obtindria així tota la columna j' de la base de dades, és a dir, tots els $m_{ij'}$ per a $1 \leq i \leq s$.

Exemple 10.9 **Exemple de protocol Kushilevitz i Ostrovsky (recuperació d'un** 1) Suposem que un usuari vol consultar una base de dades que està formada per una matriu de bits amb tres files i cinc columnes. L'usuari està interessat a recuperar el valor situat a la segona fila, tercera columna, fent servir el protocol de Kushilevitz i Ostrovsky.

Pas	Usuari		Base de dades
	Índexs del bit a recuperar $2,3$.		Matriu de bits $M_{3\times 5} = (m_{ij})$ $M_{3\times 5} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$
1.	Generació de la clau: $p = 59, q = 19$ (aleatòriament) $n = 59 \cdot 19 = 1121$ $325 \in_R QNR(n)$		
2.	Calcula: $r = \{25, 711, 77, 47, 779\}$ (aleatoris) $x = \{625, 1071, 1047, 1088, 380\}$ ja que $x_1 = 25^2 \pmod{1121} = 625$ $x_2 = 711^2 \pmod{1121} = 1071$ $x_3 = 325 \cdot 77^2 \pmod{1121} = 1047$ $x_4 = 47^2 \pmod{1121} = 1088$ $x_5 = 779^2 \pmod{1121} = 380$	$\xrightarrow{(1121,\{625,1071,1047,1088,380\})}$	
3.			Calcula: $z = \{1083, 437, 171\}$ ja que $z_1 = 625 \cdot 1071^2 \cdot 1047^2 \cdot 1088^2 \cdot 380 =$ $= 1083 \pmod{1121}$ $z_2 = 625^2 \cdot 1071^2 \cdot 1047 \cdot 1088^2 \cdot 380^2 =$ $= 437 \pmod{1121}$ $z_3 = 625 \cdot 1071^2 \cdot 1047^2 \cdot 1088 \cdot 380 =$ $= 171$
		$\xleftarrow{\{1083,437,171\}}$	
4.	Recupera el bit m_{23} comprovant si z_2 és un QR: $437 \in QNR(n)$ Per tant, $m_{23} = 1$.		

Exemple 10.10 Exemple de protocol Kushilevitz i Ostrovsky (recuperació d'un 0)
Per tal de veure una execució del protocol on es recuperi un bit que sigui 0 en comptes d'1, podem refer l'exemple anterior suposant que l'usuari volia recuperar el bit en la posició $(1,3)$. En aquest cas, fixeu-vos que els passos 1, 2 i 3 serien exactament els mateixos que a l'exemple anterior, ja que el bit a recuperar es troba també en la columna 3, de manera que l'única diferència estaria en l'últim pas. Ara, al Pas 4, l'usuari recuperaria el bit m_{13} comprovant si $z_1 = 1083$ és un residu quadràtic: $1083 \in QR(n)$ (ja que $209^2 = 1083 \pmod{1121}$) i, per tant, l'usuari aprendria que $m_{13} = 0$.

De manera anàloga, l'usuari podria recuperar també el bit a la posició $(3,3)$, ja que amb la informació rebuda, pot recuperar qualsevol valor de la tercera columna. Al Pas 4, l'usuari recuperaria el bit m_{33} comprovant si $z_3 = 171$ és un residu quadràtic: $171 \in QR(n)$ (ja que $76^2 = 171 \pmod{1121}$) i, per tant, l'usuari aprendria que $m_{33} = 0$.

Els dos exemples anteriors demostren la propietat diferenciadora dels protocols de recuperació privada d'informació envers dels protocols de trasferència inconscient: tot i que l'usuari només volia recuperar el bit a la posició $(2,3)$ de la base de dades (que corresponia a la consulta del primer exemple), l'usuari ha pogut recuperar informació addicional de la resposta de la base de dades. En efecte, l'usuari ha pogut obtenir també el bit en la posició $(1,3)$, sense necessitat de realitzar cap nova consulta a la base de dades, a partir de la informació obtinguda a la primera execució del protocol.

10.8.2 Protocol de Chor et al.

Aquest protocol va ser proposat el 1998 per Chor, Goldreich, Kushilevitz i Sudan, i es basa en replicar la base de dades entre diversos servidors incomunicats entre ells per assegurar que la consulta de l'usuari no es revela a cap dels servidors individuals. L'usuari interactuarà amb cadascun dels servidors, i reconstruirà el resultat de la consulta a partir de la informació parcial que li arriba de cadascuna de les còpies de la base de dades.

En aquest protocol la base de dades és una matriu de bits quadrada, que té s files i s columnes, i que denotarem per $M_{s\times s} = (m_{ij})$. Com al protocol anterior, l'usuari farà una consulta per recuperar un bit específic de la base de dades, que correspondrà al bit que es troba a la fila i' columna j', $m_{i'j'}$.

Per tal de descriure el protocol, definirem la funció de canviar bits (*bit flipping*), $f(x,i)$, com una funció que rep una seqüència x d'n bits i un enter $i \leq n$, i retorna una seqüència també d'n bits que resulta de canviar el bit en la posició i de la seqüència x (deixant la resta de bits iguals).

Exemple de *bit flipping*	Per a la seqüència de bits $x = 01010$, $f(x,1) = 11010$ i $f(x,3) = 01110$.

El funcionament del protocol així com les accions i els missatges que s'intercanvien en el protocol es mostra gràficament a l'esquema següent, per a una execució on la base de dades es troba replicada en quatre servidors diferents, que denotarem per DB^{uv} amb $u,v \in \{0,1\}$ (és a dir, DB^{00}, DB^{01}, DB^{10} i DB^{11}):

Pas	Usuari		Base de dades
	Índexs del bit a recuperar $i'j'$.		Matriu de bits $M_{s\times s} = (m_{ij})$
1.	Calcula: $x^0 = \{x_i \in_R \{0,1\}\}$ per tot $1 \leq i \leq s$ $y^0 = \{y_j \in_R \{0,1\}\}$ per tot $1 \leq j \leq s$ $x^1 = f(x^0, i')$ $y^1 = f(y^0, j')$	$\xrightarrow{x^0,y^0} DB^{00}$ $\xrightarrow{x^0,y^1} DB^{01}$ $\xrightarrow{x^1,y^0} DB^{10}$ $\xrightarrow{x^1,y^1} DB^{11}$	
2.		$\xleftarrow{z^{00}} DB^{00}$ $\xleftarrow{z^{01}} DB^{01}$ $\xleftarrow{z^{10}} DB^{10}$ $\xleftarrow{z^{11}} DB^{11}$	Cada BD^{uv} calcula: $z^{uv} = \bigoplus_{\substack{\forall i \mid x^u_i=1 \\ \forall j \mid y^v_j=1}} m_{ij}$
3.	Recupera el bit $m_{i'j'}$ calculant: $m_{i'j'} = z^{00} \oplus z^{01} \oplus z^{10} \oplus z^{11}$		

En el Pas 1, l'usuari que vol consultar la base de dades genera dues seqüències d's bits aleatòries (x^0 i y^0). Després, genera dues seqüències addicionals (x^1 i y^1), també d's bits, que correspondran a una còpia de les seqüències aleatòries generades en les quals s'ha modificat únicament un bit. Així, la seqüència x^1 serà una còpia d'x^0 amb el bit i' canviat (és a dir, $x^1 = f(x^0, i')$); i la seqüència y^1 serà una còpia d'y^0 amb el bit j' canviat (és a dir, $y^1 = f(y^0, j')$). L'usuari enviarà ara a cada base de dades dues de les seqüències calculades (una de les que codifiquen l'índex i' i una de les que codifiquen l'índex j'). Així, l'usuari enviarà a DB^{00} les seqüències x^0, y^0; a DB^{01} les seqüències x^0, y^1; a DB^{10} les seqüències x^1, y^0; i finalment a DB^{11} les seqüències x^1, y^1. Fixeu-vos que cada base de dades rep un parell de seqüències diferent, i que cada seqüència individual és enviada a dues bases de dades.

Al Pas 2, cada servidor de bases de dades calcularà la seva resposta. Per fer-ho, calcularà una xor de tots els bits de la matriu de la base de dades m_{ij} indicats per les dues seqüències de bits que ha rebut, x^u i y^v. En concret, calcularà la xor de tots els bits m_{ij} per a tots els índex i tals que x_i^u és un 1; i per a tots els índex j tals que y_j^v és un 1. Així, per exemple, la base de dades BD^{01} calcularà el valor z^{01} resultant de la xor entre els valors de la matriu m_{ij} amb els i tals que x_i^0 és 1 i els j tals que y_j^1 és 1. Cadascuna de les bases de dades retornarà el bit z^{uv} calculat a l'usuari.

Finalment, al Pas 3, l'usuari recupera el bit $m_{i'j'}$ calculant una xor dels quatre bits individuals que ha rebut (un de cada base de dades). Noteu com aquesta operació permet recuperar el valor del bit $m_{i'j'}$: degut a la construcció de les seqüències de bits, tots els bits que es tenen en compte a les xors que fa cada base de dades ho fan un número parell de vegades, a excepció del bit seleccionat, que només es té en compte una vegada. Així, quan l'usuari fa una xor de tots els bits z^{uv} rebuts, els que han aparegut un número parell de vegades es cancel·len, i el resultat és per tant el bit consultat $m_{i'j'}$.

Exemple 10.11 **Exemple d'execució del protocol de Chor et al.** Suposem que un usuari vol consultar una base de dades que està formada per una matriu de bits amb quatre files i quatre columnes. L'usuari està interessat a recuperar el valor situat a la segona fila, tercera columna, fent servir el protocol de Chor et al.

Pas	Usuari		Base de dades
	Índexs del bit a recuperar $2,3$.		Matriu de bits $M_{4\times4} = (m_{ij})$ $M_{4\times4} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$
1.	Calcula: $x^0 = \{0,0,1,0\}$ (aleatòriament) $y^0 = \{0,1,0,0\}$ (aleatòriament) $x^1 = f(\{0,0,1,0\},2) = \{0,1,1,0\}$ $y^1 = f(\{0,1,0,0\},3) = \{0,1,1,0\}$	$\xrightarrow{x^0,y^0} DB^{00}$ $\xrightarrow{x^0,y^1} DB^{01}$ $\xrightarrow{x^1,y^0} DB^{10}$ $\xrightarrow{x^1,y^1} DB^{11}$	
2.		$\xleftarrow{z^{00}=1} DB^{00}$ $\xleftarrow{z^{01}=1} DB^{01}$ $\xleftarrow{z^{10}=1} DB^{10}$ $\xleftarrow{z^{11}=1} DB^{11}$	BD^{00} calcula: $z^{00} = \bigoplus_{\substack{\forall i \mid x_i^0=1 \\ \forall j \mid y_j^0=1}} m_{ij} = m_{32} = 1$ BD^{01} calcula: $z^{01} = \bigoplus_{\substack{\forall i \mid x_i^0=1 \\ \forall j \mid y_j^1=1}} m_{ij} = m_{32} \oplus m_{33} = 1 \oplus 0 = 1$ BD^{10} calcula: $z^{10} = \bigoplus_{\substack{\forall i \mid x_i^1=1 \\ \forall j \mid y_j^0=1}} m_{ij} = m_{22} \oplus m_{32} = 0 \oplus 1 = 1$ BD^{11} calcula: $z^{11} = \bigoplus_{\substack{\forall i \mid x_i^1=1 \\ \forall j \mid y_j^1=1}} m_{ij} = m_{22} \oplus m_{23} \oplus m_{32} \oplus m_{33} = 0 \oplus 0 \oplus 1 \oplus 0 = 1$
3.	Recupera el bit m_{23} calculant: $m_{23} = 1 \oplus 1 \oplus 1 \oplus 1 = 0$		

Fixeu-vos en els càlculs que realitzen els diferents servidors de base de dades al Pas 2.

- BD^{00} rep $x^0 = \{0,0,1,0\}$ i $y^0 = \{0,1,0,0\}$, de manera que calcula la xor dels bits de la matriu que es troben a la fila $i \in \{3\}$ (ja que el bit a la posició 3 d'x^0 és l'únic bit que és 1) i a la columna $j \in \{2\}$ (ja que el bit a la posició 2 d'y^0 és l'únic bit que és un 1 d'y^0). Per tant, $z^{00} = m_{32}$.
- BD^{01} rep $x^0 = \{0,0,1,0\}$ i $y^1 = \{0,1,1,0\}$, de manera que calcula la xor dels bits de la matriu que es troben a la fila $i \in \{3\}$ (ja que el bit a la posició 3 d'x^0 és l'únic bit que és 1) i a la columna $j \in \{2,3\}$ (ja que els bits a les posicions 2 i 3 d'y^1 són els bits que tenen com a valor 1). Per tant, $z^{01} = m_{32} \oplus m_{33}$.
- BD^{10} rep $x^1 = \{0,1,1,0\}$ i $y^0 = \{0,1,0,0\}$, de manera que calcula la xor dels bits de la matriu que es troben a les files $i \in \{2,3\}$ i a la columna $j \in \{2\}$. Per tant, $z^{01} = m_{22} \oplus m_{32}$.
- BD^{11} rep $x^1 = \{0,1,1,0\}$ i $y^1 = \{0,1,1,0\}$, de manera que calcula la xor dels bits de la matriu que es troben a les files $i \in \{2,3\}$ i a les columnes $j \in \{2,3\}$. Per tant, $z^{11} = m_{22} \oplus m_{23} \oplus m_{32} \oplus m_{33}$.

D'altra banda, noteu perquè l'usuari pot recuperar el valor m_{23} quan fa una xor dels valors rebuts de les quatre bases de dades:

$$\begin{aligned} m_{23} &= z^{00} \oplus z^{01} \oplus z^{10} \oplus z^{11} = \\ &= (m_{32}) \oplus (m_{32} \oplus m_{33}) \oplus (m_{22} \oplus m_{32}) \oplus (m_{22} \oplus m_{23} \oplus m_{32} \oplus m_{33}) = \\ &= m_{22} \oplus m_{22} \oplus m_{23} \oplus m_{32} \oplus m_{32} \oplus m_{32} \oplus m_{32} \oplus m_{33} \oplus m_{33} = \\ &= m_{23} \end{aligned}$$

Efectivament, m_{23} és l'únic bit que no apareix un número parell de vegades, de manera que és l'únic bit que no s'anul·la, fent que el resultat de l'operació resulti en el bit de la base de dades que l'usuari volia recuperar.

Finalment, és interessant destacar per què és necessari que els servidors de base de dades no col·laborin entre ells per intentar revel·lar la consulta que l'usuari realitza. Per exemple, si els servidors BD^{00} i BD^{11} col·laboressin i compartissin la informació que tenen de la consulta, podrien calcular:

$$x^1 = f(x^0, i'); \{0,1,1,0\} = f(\{0,0,1,0\}, i') \Rightarrow i' = 2$$
$$y^1 = f(y^0, j'); \{0,1,1,0\} = f(\{0,1,0,0\}, j') \Rightarrow j' = 3$$

recuperant així els índexs de la consulta de l'usuari, $(2,3)$. En canvi, cada base de dades individual, només coneix un x^u i un y^v, que són seqüències aleatòries de bits que no aporten cap mena d'informació sobre la consulta que fa l'usuari.

10.9 Protocol multipart segur

En algunes aplicacions ens pot interessar que un conjunt d'usuaris realitzi un cert càlcul de forma que, tot i que cada usuari aporta una entrada per a la realització del càcul, al final del procés cada usuari només podrà obtenir el resultat del càlcul però no podrà obtenir els valors d'entrada d'altres usuaris. Aquests tipus de protocols es coneixen com a protocols multipart segurs.

En un **protocol multipart segur** (en anglès *multiparty computation*) un conjunt d'n participants cooperen per a avaluar el valor d'una funció f sobre un conjunt de valors $(v_1, \cdots, v_n)$ aportats pels participants. Com a sortida del protocol, cada usuari u_i obté l'avaluació de la funció $f(v_1, \cdots, v_n)$ però no obté cap informació sobre el contingut dels valors v_j per a $j \in [1,n]$ i $j \neq i$.

Com en la majoria de protocols criptogràfics, una solució simple en aquest escenari és la utilització d'una tercera part de confiança en la qual tothom confia. Aquesta tercera part és la que pot realitzar l'avaluació de la funció f i, com que tothom hi confia, tothom està segur que un cop cada usuari li ha lliurat el seu tros d'informació, no el mostrarà a cap altra part.

Justament, el que proporcionen els protocols multipart segurs és un mecanisme per poder prescindir de la tercera part de confiança. En els següents apartats veurem dos exemples concrets de protocol multipart segur.

10.9.1 El problema del milionari

Un exemple d'un protocol de càlcul segur a múltiples bandes, en aquest cas a dues bandes, és el protocol proposat per C. Yao l'any 1982, conegut com el problema del milionari. En aquest escenari, dos milionaris A i B volen saber qui és el més ric però no volen revelar el valor de la seva fortuna. És a dir, la funció que volem avaluar de forma segura és una comparació de la mida de dos valors en que cada un dels dos participants aporta un valor.

Per tal de simplificar una mica el problema (de fet, seria equivalent a fitar la fortuna que tenen els participants) transformarem aquest problema en un problema equivalent que consistirà en que l'Alice i en Bob volen saber qui és més gran sense dir quina edat té cada un. Suposarem que tots dos són honestos i que utilitzen les seves edats reals.

Suposarem que l'Alice té x anys, en Bob y i cap dels dos no en té més de 100, és a dir $1 \leq x, y \leq 100$. Per a realitzar aquest protocol utilitzarem un criptosistema de clau pública. Així, tant A com B tindran cada un d'ells un parell de claus pública i privada que seran (E_A, D_A) i (E_B, D_B) respectivament. D'altra banda també assumirem que ambdós usuaris coneixen la clau pública de l'altre participant. A més, A i B també es posen d'acord en la mida màxima que tindran dos dels valors utilitzats en el protocol, t_a i t_b. Així poden assegurar que els valors p_a i p_b triats en el pas 6 són més petits que aquests dos valors.

El protocol funciona tal i com es descriu en l'esquema de la Taula 10.5.

Taula 10.5: Protocol del milionari

Pas	Alice		Bob
1.	Tria $t_a \in_R \mathbb{Z}$		Tria $t_b \in_R \mathbb{Z}$
2.	Calcula $k_a = E_B(t_a)$ $K_a = k_a - x$		Calcula $k_b = E_A(t_b)$ $K_b = k_b - y$
3.		$\xrightarrow{K_a}$	
4.		$\xleftarrow{K_b}$	
5.	Calcula: $f_i = D_A(K_b + i)$ per $1 \leq i \leq 100$		Calcula: $f'_i = D_B(K_a + i)$ per $1 \leq i \leq 100$
6.	Tria $p_a < t_b$ Calcula: $g_i = f_i \pmod{p_a}$ per $1 \leq i \leq 100$ assegurant que $\lvert g_i - g_j \rvert \geq 2$ per a $i \neq j, 1 \leq i, j \leq 100$ Crea la seqüència: $G = \{g_1, \cdots, g_x, g_{x+1}+1, g_{x+2}+1, \cdots$ $g_{100}+1, p_a\}$		Tria $p_b < t_a$ Calcula: $g'_i = f'_i \pmod{p_b}$ per $1 \leq i \leq 100$ assegurant que $\lvert g'_i - g'_j \rvert \geq 2$ per a $i \neq j, 1 \leq i, j \leq 100$ Crea la seqüència: $G' = \{g'_1, \cdots, g'_y, g'_{y+1}+1, g'_{y+2}+1, \cdots,$ $g'_{100}+1, p_b\}$
7.		$\xrightarrow{G}$	
8.		$\xleftarrow{G'}$	
9.	Comprova: Si $G'_x = t_a \pmod{p_b}$, aleshores $y \geq x$ sinó $y < x$		Comprova: Si $G_y = t_b \pmod{p_a}$, aleshores $x \geq y$ sinó $x < y$

Com es pot veure en el protocol, la idea és que tant A com B creen una seqüència de valors, en aquest cas 100 que és el màxim de l'edat dels participants. La particularitat d'aquestes seqüències és que, per exemple, prenent la seqüència G que genera l'usuari A, per a índex inferiors o iguals a l'índex que determina l'edat de l'usuari A, el valors són congruents amb el valor aleatori que ha triat B, si l'edat d'A és major que la d'B.

Les conclusions sobre qui té més edat que cada usuari obté en el pas 9 són correctes. Per exemple, el raonament respecte a la comprovació de l'usuari B seria la següent: Si A té més edat que B, és a dir $x \geq y$ aleshores el valor de la posició y de la seqüència que A envia a B en el pas 7 és $G_y = g_y$. Per tant, $G_y = f_y \pmod{p_a}$. Com que $f_y = D_A(K_b + y) = D_A(k_b - y + y) = D_A(k_b)$ i $k_b = E_A(t_b)$ ens queda que $f_y = D_A(E_A(t_b)) = t_b$ i per tant $G_y = t_b \pmod{p_a}$.

D'altra banda, si $x < y$ aleshores el valor G_y verifica que:

$$G_y = g_y + 1 \neq g_y = f_y = t_b \pmod{p_a}$$

Exercici 10.8 L'Alice i el Bob han estat de sort i els ha tocat la loteria, que reparteix com a màxim 5 milions. L'Alice ha tingut més sort que en Bob i li han tocat 4 milions, mentre que al Bob n'hi han tocat 2. Com que cap d'ells vol dir quina quantitat li ha tocat, decideixen saber qui és més ric utilitzant el protocol del milionari. Desenvolupeu el protocol per tal que els dos puguin saber qui ha guanyat més diners sense saber quants diners li han tocat a l'altre. Suposarem que utilitzem com a sistema de clau pública l'RSA i el parell de claus pública-privada d'A val $[(e_A = 2573, n_A 5911), (d_A = 197, n_A = 5911)]$, mentre que el parell de B val $[(e = 3109, n_B = 5191), (d_B = 1795, n_B = 5191)]$.

10.9.2 El problema del milionari socialista

En aquest segon protocol, A i B tenen cada un la seva fortuna, representada pels valors x i y respectivament, però en comptes de saber qui és més ric el que volen saber és si la seva fortuna és igual o no. L'execució d'aquest protocol és més elaborada que la de l'exercici anterior ja que el nombre de missatges que s'intercanvien és més elevat, a causa que el protocol utilitza com a subprotocol, en diverses etapes, el protocol d'intercanvi de claus de Diffie i Hellman.

El protocol defineix dos paràmetres generals: un nombre primer p i un valor $h \in \mathbb{Z}_p$ tal que $h \neq 1$. El valor de p ha de ser més gran que la fortuna tant d'Alice com del Bob, és a dir $x < p$ i $y < p$.

El funcionament del protocol es mostra en l'esquema de la Taula 10.6.

Com es pot veure en el protocol, els primers 6 passos corresponen a un intercanvi de claus de Diffie-Hellman que permeten que A i B comparteixin dos valors f i g. Donat que la verificació final podria ser errònia en cas que els valors a_1, a_2, b_1, b_2 fossin 0, per aquest motiu es realitza la validació del pas 5.

Al final del protocol, en cas que la darrera comprovació del valor sigui correcta, tant A com B poden estar convençuts que els dos tenen la mateixa fortuna, ja que:

$$P_a P_b^{-1} = f^r (f^s)^{-1} = f^{r-s} = h^{a_2 b_2 (r-s)} \pmod{p}$$

però d'altra banda,

$$\begin{aligned}
c &= ((Q_a Q_b^{-1})^{b_2})^{a_2} \\
&= ((h^x g^x)(h^s g^s)^{-1})^{a_2 b_2} \\
&= (h^{(r-s)} g^{(x-y)})^{a_2 b_2} \\
&= (h^{(r-s)} (h^{a_1 b_1})^{(x-y)})^{a_2 b_2} \\
&= h^{a_2 b_2 (r-s)} h^{a_1 b_1 a_2 b_2 (x-y)} \\
&= P_a P_b^{-1} (h^{a_1 b_1 a_2 b_2 (x-y)}) \pmod{p}
\end{aligned}$$

i com que els valors a_1, a_2, b_1, b_2 han estat triats aleatòriament per A i B, l'única possibilitat que $c = P_a P_b^{-1} \pmod{p}$ és en el cas que $x = y$, és a dir, que A i B tinguin la mateixa fortuna.

Taula 10.6: Protocol del milionari socialista

Pas	Alice		Bob
1.	Tria $a_1, a_2 \in_R \mathbb{Z}_p$		Tria $b_1, b_2 \in_R \mathbb{Z}_p$
2.	Calcula $h^{a_1} \pmod p$ $h^{a_2} \pmod p$		Calcula $h^{b_1} \pmod p$ $h^{b_2} \pmod p$
3		$\xrightarrow{(h^{a_1}, h^{a_2})}$	
4.		$\xleftarrow{(h^{b_1}, h^{b_2})}$	
5.	Verifica que: $h^{b_1} \neq 1 \pmod p$ $h^{b_2} \neq 1 \pmod p$		Verifica que: $h^{a_1} \neq 1 \pmod p$ $h^{a_2} \neq 1 \pmod p$
6.	Calcula: $g = (h^{b_1})^{a_1} \pmod p$ $f = (h^{b_2})^{a_2} \pmod p$		Calcula: $g = (h^{a_1})^{b_1} \pmod p$ $f = (h^{a_2})^{b_2} \pmod p$
7.	Tria $r \in_R \mathbb{Z}_p$		Tria $s \in_R \mathbb{Z}_p$
8.	Calcula: $P_a = f^r \pmod p$ $Q_a = h^r g^x \pmod p$		Calcula: $P_b = f^s \pmod p$ $Q_b = h^s g^y \pmod p$
9		$\xrightarrow{(P_a, Q_a)}$	
10.		$\xleftarrow{(P_b, Q_b)}$	
11.	Comprova que: $P_a \neq P_b \pmod p$ $Q_a \neq Q_b \pmod p$		Comprova que: $P_a \neq P_b \pmod p$ $Q_a \neq Q_b \pmod p$
12.	Calcula: $(Q_a Q_b^{-1})^{a_2}$		Calcula: $(Q_a Q_b^{-1})^{b_2}$
13.		$\xrightarrow{(Q_a Q_b^{-1})^{a_2}}$	
14.		$\xleftarrow{(Q_a Q_b^{-1})^{b_2}}$	
15.	Calcula: $c = ((Q_a Q_b^{-1})^{b_2})^{a_2} \pmod p$		Calcula: $c = ((Q_a Q_b^{-1})^{a_2})^{b_2} \pmod p$
16.	Comprova que: $c = P_a P_b^{-1} \pmod p$		Comprova que:: $c = P_a P_b^{-1} \pmod p$

En cas que la comprovació no sigui correcta voldrà dir que les fortunes no són iguals però cap dels dos sabrà qui té una fortuna més gran.

10.10 Resum

En aquest capítol hem estudiant diferents protocols criptogràfics que permeten assolir diferents objectius, tots ells relacionats amb la seguretat de la informació. En primer lloc, hem vist com dos usuaris es poden intercanviar un missatge de forma secreta sense haver intercanviat prèviament cap clau, utilitzant el protocol de tres passos de Shamir. També hem vist com funcionen els esquemes de compartició de secrets, que permeten que un secret es descompongui en diferents fragments de manera que amb la unió d'un nombre fixat de fragments es pot recuperar el secret però amb menys sigui impossible.

D'altra banda, hem estudiat també altres protocols en que la seva aplicació directa pot no ser del tot òbvia. Un exemple són les signatures cegues, on el signatari no coneix el missatge que està signant i aquest fet es pot aprofitar per a protocols d'autenticació anònima. Un altre exemple estudiat són les proves de coneixement nul on un usuari pot demostrar davant d'un altre que coneix un secret sense revelar-ne cap informació del mateix. També hem vist com funciona un protocol de transferència inconscient on la comunicació entre dos usuaris es fa de forma probabilística de manera que l'emissor envia dos missatges i el receptor només en rep un. Ara bé, ni l'emissor sap quin missatge ha rebut el receptor ni el receptor pot triar quin dels dos rebre ja que té una probabilitat del 50 % de rebre'n un dels dos.

Finalment, hem analitzat dos exemples de protocols multipart segurs. En els protocols multipart segurs, n usuaris volen obtenir l'avaluació d'una funció $f(x_1, x_2, \cdots, x_n)$ proporcionant cada un d'ells una entrada de la funció x_i. El punt clau del protocol és que tots els usuaris han d'obtenir el resultat de l'avaluació de la funció però no poden obtenir cap informació sobre les entrades que han proporcionat la resta d'usuaris. Els exemples estudiats han mostrat protocols en els que hi intervenien dos usuaris, un d'ells permet avaluar la funció "menor o igual" i l'altre permet avaluar la funció d'igualtat.

10.11 Solucions dels exercicis

Exercici 10.1:

Amb aquests paràmetres, l'usuari A enviarà de forma secreta el missatge $m = 20$ a B amb el protocol de la Taula 10.7:

Taula 10.7: Exercici 1

Pas	Alice		Bob
1.	$c_1 = 20^{19} \pmod{101} = 30$	$\xrightarrow{30}$	
2.		$\xleftarrow{77}$	$c_2 = (30)^{13} \bmod 101 = 77$
3.	$c_3 = (77)^{79} \bmod 101 = 9$	$\xrightarrow{9}$	
4.			$m = (9)^{77} \bmod 101 = 20$

Exercici 10.2:

El polinomi per generar els fragments estarà compost pel terme independent 11, tindrà com a grau $m-1 = 3-1 = 2$ i com a coeficients podem triar aleatòriament, per exemple, els nombres $x_1 = 8$ i $x_2 = 7$. D'aquesta manera el polinomi ens queda determinat per $a(x) = 7x^2 + 8x + 11 \bmod 13$
Per generar els fragments prenem 5 valors qualssevol menors que p i calculem les seves imatges pel polinomi $a(x)$. Prenent com a valors $\{1,2,3,4,5\}$ tindrem:

$$\begin{aligned}
a(1) &= 7+8+11 = 0 \pmod{13} \\
a(2) &= 28+16+11 = 3 \pmod{13} \\
a(3) &= 63+24+11 = 7 \pmod{13} \\
a(4) &= 112+32+11 = 12 \pmod{13} \\
a(5) &= 175+40+11 = 5 \pmod{13}
\end{aligned}$$

Per tant els fragments dels participants són: $(1,0),(2,3),(3,7),(4,12),(5,5)$.

Exercici 10.3:

Donat que tenim un sistema de compartició llindar amb $m = 3$ podem triar, d'entre els diferents fragments, $(58,137),(11,48),(50,99),(80,50),(104,33),(39,114)$, qualsevol conjunt de 3 punts per recuperar el secret. Per exemple, si triem $(50,99),(80,50),(39,114)$ podem plantejar el següent sistema d'equacions:

$$\begin{aligned}
S + a_1 \cdot 50 + a_2 \cdot 50^2 &= 99 \pmod{149} \\
S + a_1 \cdot 80 + a_2 \cdot 80^2 &= 50 \pmod{149} \\
S + a_1 \cdot 39 + a_2 \cdot 39^2 &= 114 \pmod{149}
\end{aligned}$$

Com que només ens interessa resoldre el sistema per la variable S, que és el secret, podem aplicar el mètode de Cramer i obtenim:

$$\frac{\begin{vmatrix} 99 & 50 & 116 \\ 50 & 80 & 142 \\ 114 & 39 & 31 \end{vmatrix}}{\begin{vmatrix} 1 & 50 & 116 \\ 1 & 80 & 142 \\ 1 & 39 & 31 \end{vmatrix}} = \frac{36}{120} = 36 \cdot 113 = 45 \pmod{149}$$

Exercici 10.4:

El gestor ha utilitzat el polinomi $a(x) = S + a_1x + a_2x^2$, on S és la clau del sistema.

Quan els tres usuaris es reuneixen poden escriure el següent sistema:

$$\begin{aligned} f_1 + 2 &= S + a_1 + a_2 \\ f_2 + x &= S + 2a_1 + 4a_2 \\ f_3 + 2 &= S + 3a_1 + 9a_2 \end{aligned}$$

on f_1, f_2, f_3 són els fragments respectius d'A, B i C i x és la trampa que ha fet l'usuari B.

La solució per la incògnita S en aquest sistema és la mateixa que pel sistema en el que cap participant fa trampa, ja que l'enunciat indica que han recuperat el mateix secret, per tant:

$$\begin{aligned} f_1 &= S + a_1 + a_2 \\ f_2 &= S + 2a_1 + 4a_2 \\ f_3 &= S + 3a_1 + 9a_2 \end{aligned}$$

Així doncs podem plantejar la següent igualtat:

$$\frac{\begin{vmatrix} f_1+2 & 1 & 1 \\ f_2+x & 2 & 4 \\ f_3+2 & 3 & 9 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{vmatrix}} = \frac{\begin{vmatrix} j & 1 & 1 \\ m & 2 & 4 \\ s & 3 & 9 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{vmatrix}}$$

i si realitzem les operacions dels determinants ens queda que $6x = 3$ i finalment $x = 3 \cdot 6^{-1} = 7$, sempre treballant a $\mathbb{Z}_{13}$. Per tant, la trampa que ha fet l'usuari B ha estat sumar 7 al seu fragment.

Exercici 10.5:

Per comprovar que efectivament, amb els valors $m_1 = 29$ i $r_1 = 90$ es pot obrir el compromís de Pedersen $C(m_1) = 24$, cal calcular el valor del compromís com $27^{29} \cdot 94^{90} \pmod{113} = 27^{29} \cdot 94^{90} \pmod{113} = 66 \cdot 62 \pmod{113} = 4092 \pmod{113} = 24$ que efectivament coincideix amb $C(m_1)$.

Si el compromís de m_2 és $C(m_2) = 91$ i el compromís de m_1 val $C(m_1) = 24$ podem calcular $C(m_1 + m_2)$ com $C(m_1) \cdot C(m_2)$, és a dir $24 \cdot 91 \pmod{113} = 37$. Tot i poder-lo calcular, aquest compromís de la suma no es pot obrir perquè per fer-ho necessitaríem el valor r_2 que permet obrir $C(m_2)$.

Exercici 10.6:

La solució de l'exercici es mostra en la Taula 10.8.

Exercici 10.7:

Com que A vol fer creure a B que coneix el logaritme discret amb un probabilitat de 0,75 això vol dir que caldrà executar de forma satisfactòria 3 vegades del protocol. Com que A coneix el generador pseudoaleatori sap que en la primera execució del protocol, al Pas 2, V triarà $b = 0$, en la segona execució del protocol triarà $b = 1$ i en la tercera execució triarà $b = 0$. Per tant, per tal d'enredar a V:

- En el primer protocol, en el pas 1 triarem qualsevol valor aleatori, per exemple $r = 45$ que serà el mateix valor que retornarem en el pas 3, $h = 45$. La validació del pas 4 feta per V serà correcta.

Pas	Alice		Bob
1.	Genera 5 claus públiques: $(3,8,10,11,14)$ Prepara els 5 missatges per signar: $m_1 = (3\|\|5) = (35), m_2 = (85)$ $m_3 = (105), m_4 = (115), m_5 = (145)$ Genera els 5 valors per tapar-los: $(5,8,15,23,4)$ $t_1 = 5^{19} = 718 \pmod{899}$ $t_2 = 8^{19} = 872 \pmod{899}$ $t_3 = 15^{19} = 773 \pmod{899}$ $t_4 = 23^{19} = 895 \pmod{899}$ $t_5 = 4^{19} = 473 \pmod{899}$ Tapa els 5 missatges: $m_1 \xrightarrow{t_1} m'_1 = 35 \cdot 718 = 857 \pmod{899}$ $m_2 \xrightarrow{t_2} m'_2 = 85 \cdot 872 = 402 \pmod{899}$ $m_3 \xrightarrow{t_3} m'_3 = 105 \cdot 773 = 255 \pmod{899}$ $m_4 \xrightarrow{t_4} m'_4 = 115 \cdot 895 = 439 \pmod{899}$ $m_5 \xrightarrow{t_5} m'_5 = 145 \cdot 473 = 261 \pmod{899}$	$\xrightarrow{(m'_1,m'_2,m'_3,m'_4,m'_5)}$	
2.		$\xleftarrow{i=2}$	Tria $i = 2 \in_R \mathbb{Z}_5$
3.	Envia els valors t_j menys el t_2 seleccionat.	$\xrightarrow{(t_1,t_3,t_4,t_5)}$	
4.		$\xleftarrow{s'_2=371}$	Destapa els valors i comprova que el servei sigui $S = 5$ (últim dígit) $m'_1 \xrightarrow{t_1} m_1 = 35$ $m'_3 \xrightarrow{t_3} m_3 = 105$ $m'_4 \xrightarrow{t_4} m_4 = 115$ $m'_5 \xrightarrow{t_5} m_5 = 145$ Signa el valor no destapat: $s'_2 = 402^{619} = 371 \pmod{899}$
5.	Destapa el valor per obtenir la signatura de m_2 $s'_2 \xrightarrow{t_2} s_2 = \frac{371}{8} = 833$ que veiem que coincideix $85^{619} = 833 \pmod{899}$		

Taula 10.8: Solució de l'Exercici 10.6

- En la segona execució del protocol, en el pas 1 triarem, per exemple, $r = 5$. Però enviarem a V el valor $c = \frac{g^r}{y} \bmod p = \frac{7^5}{94} \bmod 97 = 56$. Aleshores en el pas 3 enviarem $h = r = 5$ i la validació que farà V en el pas 4 també serà correcta ja que $c \cdot y^b = 56 \cdot 94^1 = 26 \bmod 97$ i $g^h = 7^5 = 26 \bmod 97$.
- En la tercera execució, aplicarà la mateixa estratègia que en la primera.

Exercici 10.8:

La solució d'aquest exercici es mostra en la Taula 10.9.

Taula 10.9: Exercici 7

Pas	Alice ($x=4$)		Bob ($y=2$)
1.	Tria $t_a = 1349 \in_R \mathbb{Z}$		Tria $t_b = 1547 \in_R \mathbb{Z}$
2.	Calcula $k_a = E_B(t_a) = 1465$ $K_a = k_a - x = 1461$		Calcula $k_b = E_A(t_b) = 2212$ $K_b = k_b - y = 2210$
3.		$\xrightarrow{K_a=1461}$	
4.		$\xleftarrow{K_b=2210}$	
5.	Calcula: $f_1 = D_A(K_b+1) = 4217$ $f_2 = D_A(K_b+2 = 1547)$ $f_3 = D_A(K_b+3) = 3556$ $f_4 = D_A(K_b+4) = 3569$ $f_5 = D_A(K_b+5) = 884$		Calcula: $f'_1 = D_B(K_a+1) = 1177$ $f'_2 = D_B(K_a+2) = 573$ $f'_3 = D_B(K_a+3) = 4426$ $f'_4 = D_B(K_a+4) = 69$ $f'_5 = D_B(K_a+5) = 674$
6.	Tria $p_a = 239 < t_b$ Calcula: $g_1 = f_1 \pmod{p_a} = 154$ $g_2 = f_2 \pmod{p_a} = 113$ $g_3 = f_3 \pmod{p_a} = 210$ $g_4 = f_4 \pmod{p_a} = 223$ $g_5 = f_5 \pmod{p_a} = 167$ Crea la seqüència: $G = \{154, 113, 210, 223, 168, 239\}$		Tria $p_b = 739 < t_a$ Calcula: $g'_1 = f'_1 \pmod{p_b} = 438$ $g'_2 = f'_2 \pmod{p_b} = 573$ $g'_3 = f'_3 \pmod{p_b} = 731$ $g'_4 = f'_4 \pmod{p_b} = 69$ $g'_5 = f'_5 \pmod{p_b} = 674$ Crea la seqüència: $G' = \{438, 573, 732, 70, 675, 739\}$
7.		$\xrightarrow{G}$	
8.		$\xleftarrow{G'}$	
9.	Com que $G'_4 = 70 \neq 1349 = t_a \pmod{739}$, aleshores $y < x$ i per tant A té més diners que B.		Com que: $G_2 = 113 = 1547 = t_b \pmod{239}$, aleshores $x \geq y$ i per tant A té tants o més diners que B.

10.12 Bibliografia

Brassard, G.,Crepeau, C., Chaum, D. *Minimum Disclosure Proofs of knowledge.* Journal of Computer and System Sciences, v. 37, n 2. (1988)

Chaum, D.*Blind signatures for untraceable payments.* Advances in Cryptology Proceedings of Crypto. 82 (3): 199-203. (1983)

Chaum, D., Evertse, J.-H., Van de Graaf, J. *"An improved protocol for demonstrating possession of discrete logarithms and some generalizations"*, Proceedings of Eurocrypt'87. Springer-Verlag. (1988).

Even, S.; Goldreich,O.; Lempel, A. *A Randomized Protocol for Signing Contracts.* Communications of the ACM, Volume 28, Issue 6, pg. 637-647 (1985).

Jakobsson, M.; Yung, M. *Proving without knowing: On oblivious, agnostic and blindfolded provers.* Advances in Cryptology - CRYPTO '96, volume 1109 of Lecture Notes in Computer Science. Berlin. pp. 186-200. (1996)

Shamir, A. *How to Share a Secret.* Communications of the ACM, v. 24, n.11, pp. 612-613 (1979)

Les imatges que il·lustren les portades de cada capítol han estat obtingudes de Unsplash amb una llicència lliure.

La imatge de la portada és de Joana Abreu.

La imatge de la taula de continguts és d'Anna Zakharova.

La imatge del capítol 1 és de Ilona Frey.

La imatge del capítol 2 és de Moritz Kindler.

La imatge del capítol 3 és de olieman.eth.

La imatge del capítol 4 és de La-Rel Easter.

La imatge del capítol 5 és de Thomas Foster.

La imatge del capítol 6 és de Paula Hayes.

La imatge del capítol 7 és de Daniel Diesenreither.

La imatge del capítol 8 és de Taneli Lahtinen.

La imatge del capítol 9 és de Quentin Rey.

La imatge del capítol 10 és de Jeremy Bezanger.