Funcions hash

PID_00235162

Jordi Herrera Joancomartí
Cristina Pérez Solà

Temps mínim de dedicació recomanat: 3 hores

Universitat Oberta de Catalunya

Jordi Herrera Joancomartí

Llicenciat en Matemàtiques per la Universitat Autònoma de Barcelona i doctor per la Universitat Politècnica de Catalunya. Els seus àmbits de recerca són la criptografia, les criptomonedes i la tecnologia *blockchain*. Ha publicat nombrosos textos docents i més de cent articles de recerca en revistes i congressos nacionals i internacionals. Ha dirigit nou tesis doctorals i ha estat investigador principal de diversos projectes de recerca nacionals. Ha participat com a avaluador per a agències de recerca de diversos països europeus i també per a la Comissió Europea. Actualment és professor agregat del departament d'Enginyeria de la Informació i les Comunicacions a la Universitat Autònoma de Barcelona.

Cristina Pérez Solà

Doctora en Informàtica per la Universitat Autònoma de Barcelona i la Universitat Catòlica de Lovaina. Actualment és professora dels Estudis d'Informàtica, Multimèdia i Telecomunicacions de la Universitat Oberta de Catalunya. Els seus àmbits de recerca són les criptomonedes basades en *blockchain* i, en especial, els aspectes relacionats amb la seguretat i la privadesa d'aquestes. També està interessada en els problemes de privacitat que sorgeixen arran de l'ús de les xarxes socials i en l'adaptació de tècniques de mineria de dades a la naturalesa específica d'aquest tipus de xarxes.

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats per la professora: Helena Rifà Pous

Segona edició: febrer 2021

Av. Tibidabo, 39-43, 08035 Barcelona
Autoria: Jordi Herrera Joancomartí, Cristina Pérez Solà
Producció: FUOC

Índex

Introducció

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 mòdul didàctic 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, però les veurem en detall en altres mòduls didàctics.

Objectius

Els objectius que es pretenen assolir amb aquest mòdul didàctic són els següents:

1. Entendre les propietats de les funcions hash.

2. Conèixer diferents tipus de funcions hash.

3. Comprendre el funcionament de la funció SHA256.

4. Identificar diferents aplicacions de les funcions hash.

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 i quina diferència hi ha entre una funció hash i una funció hash criptogràfica.

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. Així mateix, encara que 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 manera similar a un oracle aleatori, i això pot induir a pensar que el seu funcionament no és determinista.

Exemple de funció hash

Un exemple de funció hash de mida 3 dígits decimals seria el 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'ls 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 propietats següents:

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 que cal destacar sobre les funcions hash criptogràfiques és que, tal 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 què se'ls associï una clau, tal com veurem a l'apartat 4.

Exemple 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 què 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 l'enunciat de les propietats d'una funció hash criptogràfica és molt simple, no és gens fàcil construir una funció que les compleixi, 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 mòdul, abusarem del llenguatge i assumirem que totes les funcions hash a les quals fem referència a partir d'aquest punt són funcions hash criptogràfiques.

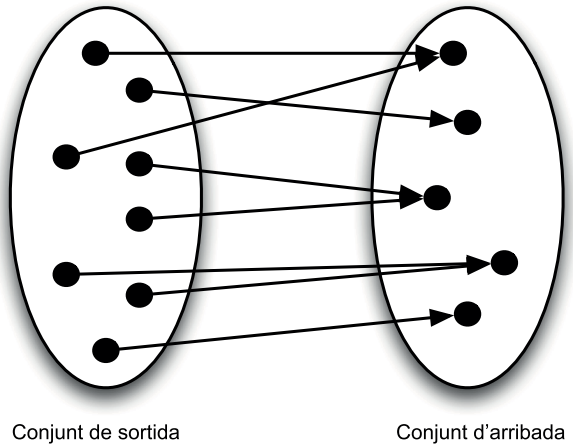
1.2. Propietats

És important aturar-se a mirar amb deteniment les tres propietats de les funcions hash criptogràfiques, ja que una anàlisi en detall d'aquestes 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 com es mostra a la figura 1.

Figura 1. L'exhaustivitat de les funcions hash



Un altre punt que cal 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 que la distribució dels naixements al llarg dels dies de l'any fos uniforme). Aquests valors contradiuen la nostra intuïció, ja 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 (col·lisions 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.

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 aquest 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 el 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 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 hi afegim una tercera persona, la probabilitat que aquesta nova tingui l'aniversari un dia diferent de les dues serà de $\frac{2}{365}$, però com que les dues primeres també han de tenir l'aniversari un dia diferent, ens queda que per tal 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 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.

1.3. Seguretat de les funcions hash

Per parlar de seguretat d'una funció hash primer ens cal 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 col·lisions (febles o fortes).

Com ja hem comentat al subapartat 1.2., 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 fórmula següent:

$$t \approx 2^{\frac{n+1}{2}} \sqrt{\ln\left(\frac{1}{1-\lambda}\right)}$$

Aquesta fórmula ens proporciona el nombre de missatges t dels quals hem de calcular el hash per a trobar una col·lisió amb una probabilitat λ, en què n és la mida en bits de la funció hash.

A la taula 1 veiem les dades per a diferents mides de funció hash.

Taula 1. Nombre de missatges per a 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 a 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 perquè una funció hash tingui un nivell de seguretat d'x bits necessitarem que la seva mida sigui, com a mínim, de $2x$.

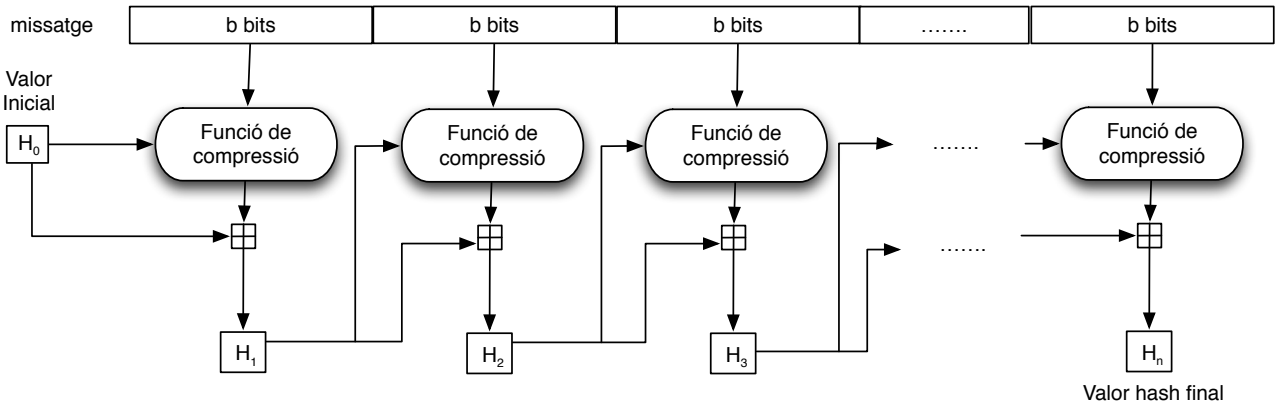
Finalment, és important indicar que malgrat que trobar una única col·lisió en una funció hash és un fet que en posa en entredit la seva seguretat, ja que es tracta d'un fet probabilístic, cal analitzar com s'ha trobat la col·lisió, ja que una funció hash es considera trencada només quan es pot reduir la complexitat de l'atac a valors més petits dels que determina la taula 1.

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 el 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 mòdul didàctic.

Tot i la seva complexitat, les funcions hash tenen una estructura general estàndard que s'esquematitza a la figura 2.

Figura 2. Estructura general d'una funció hash



Com es pot veure a la figura, les funcions hash processen els missatges partint-los en blocs (de manera similar als criptosistemes de blocs), tractant cada bloc de manera específica i combinant les sortides que proporciona la funció per a cada bloc amb la resta de sortides dels altres blocs (també d'una manera semblant als modes de xifrat de bloc).

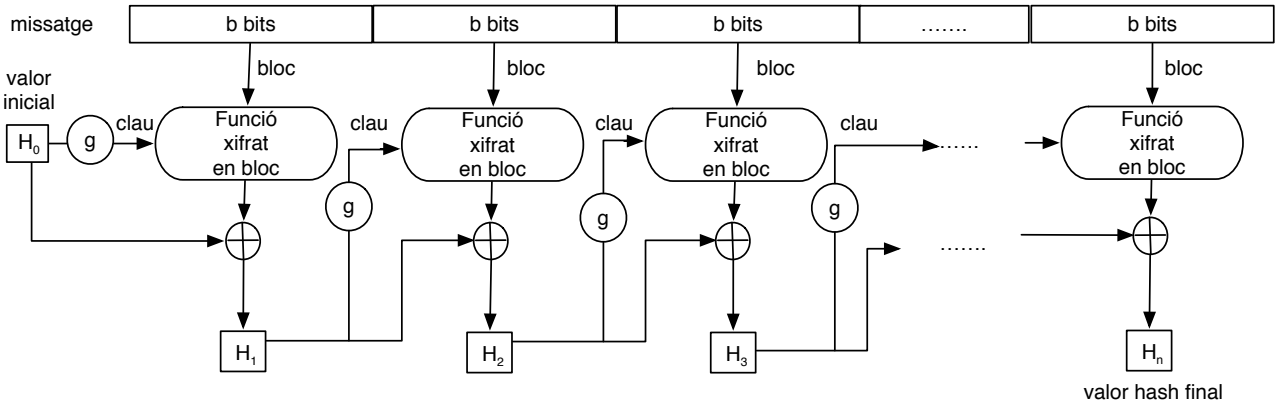
La base de les funcions hash és una funció interna que s'identifica com a funció de compressió. Aquesta funció processa cada bloc del missatge que es vol tractar proporcionant una sortida de mida igual o més petita que el mateix bloc, d'aquí la seva denominació de compressió. La mida de la sortida d'aquesta funció de compressió serà la mida de la mateixa 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.

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, tot i que la més coneguda és el sistema Matyas-Meyer-Oseas.

Figura 3. Funció hash a partir de criptosistema de bloc



A la figura 3 es pot veure l'esquema d'una funció hash a partir d'un criptosistema de bloc. El valor b indica la mida dels blocs en què es partirà el missatge que es vol 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 bloc següent, ens caldrà una funció g que converteixi cadenes de b-bits en 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 acabar, la figura 3 també mostra que la mida de la funció hash és justament b, la mida del criptosistema de bloc que estem fent servir.

D'una manera més analítica, podem partir d'un missatge d'entrada m i dividir-lo en blocs de b bits, de manera que obtenim $m_1, m_2, \cdots, m_n$. Per a cada bloc m_i, per a $i = 1, \cdots, n$, apliquem la funció següent definida de manera recursiva:

$$h_i = E_{g(h_{i-1})}(m_i) \oplus h_{i-1}$$

Padding

En el cas que la mida del missatge no sigui múltiple del bloc, caldrà fer el *padding* del missatge i forçar-ho.

En aquest ca, $E_k(\cdot)$ és la funció de xifrat en bloc amb la clau k i $h_0 = VI$, en què VI és un vector inicial públicament especificat per a la funció hash en qüestió.

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 a $m_0m_1m_2m_3$ 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, la següent:

$$c = E_k(m) = c_0c_1c_2c_3 = (m_0m_1m_2m_3) \oplus k_0k_1k_1k_0$$

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 manera següent: $g(x_0x_1x_2x_3) = (x_0 \oplus x_1)(x_2 \oplus x_3)$.

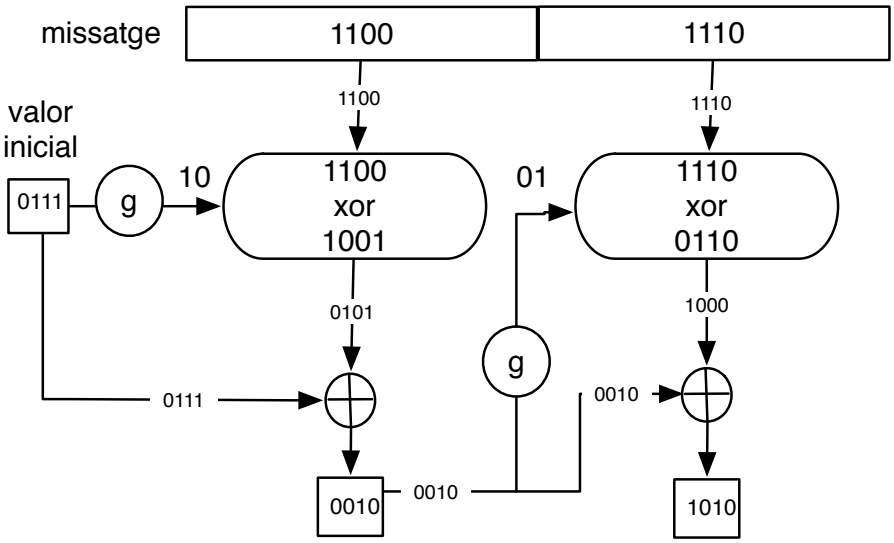
Sobre la base d'aquests paràmetres, veiem com es calcularia el valor hash del missatge $m = 11001110$, es a dir $h(11001110)$.

En primer lloc, partirem el missatge en blocs de 4 bits. En aquest cas tenim dos blocs $m_1 = 1100$ i $m_2 = 1110$. En segon lloc, apliquem la funció de xifrat sobre m_1 amb la clau $g(VI)$. En aquest cas, $g(VI) = g(0111) = 10$, 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: $h(11001110) = 1010$.

A la figura 4 es pot veure la versió gràfica dels càlculs.

Figura 4. Funció hash a partir de criptosistema de bloc



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 manera general, s'utilitzen funcions hash de disseny específic, com les es descriuen al subapartat 2.2.

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, en llistem les més rellevants; així mateix, les descriurem breument.

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, ja van ser provades el 1991, i el 1995 ja es podien realitzar atacs de col·lisions en pocs segons, fet que posteriorment va propiciar la retirada de la funció, explicitada a l'RFC 6150. La funció MD5, definida a l'RFC 1321, va ser desenvolupada per a pal·liar les vulnerabilitats de l'MD4. De tota manera, en l'actualitat es considera també insegura i el seu ús està totalment desaconsellat, ja que és fàcil trobar-hi col·lisions i, fins i tot, generar certificats digitals amb claus públiques diferents que tinguin el mateix valor hash MD5.

Certificats digitals

La descripció i l'ús dels certificats digitals s'inclou en el mòdul didàctic "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 que no se'n coneix cap atac, està poc estès, ja que té una mida igual que altres funcions estandarditzades, com ara la 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 l'Organització Internacional per a l'Estandardització (ISO) i la Comissió Electrotècnica Internacional (IEC) sota l'estàndard ISO/IEC 10118-3.

2.2.1. La família de funcions SHA

Els *secure hash algorithms* són un conjunt de funcions hash estandarditzades per l'Institut Nacional d'Estàndards i Tecnologia (NIST) dels Estats Units. Aquestes funcions s'agrupen bàsicament en tres grans grups: SHA-1, SHA-2 i SHA-3.

El grup SHA-1 inclou una única funció hash de mida 160 bits. Aquesta funció va ser dissenyada per l'Agència de Seguretat Nacional (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 l'ús.

El grup SHA-2, també dissenyat per l'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 del 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 de SHA-256 o de SHA-512 a més d'utilitzar uns vectors inicials diferents.

Aquests dos primers grups, SHA-1 i SHA-2, estan detalladament descrits a l'estàndard FIPS 180-4: *secure hash standard* publicat pel NIST al març del 2012.

L'últim grup de funcions hash, la SHA-3, és el més nou, i és un conjunt de funcions hash definides a 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 de la SHA-3 es va realitzar per mitjà d'una selecció pública i oberta en què van participar investigadors de tot el món, d'una manera 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 amb les sigles SHA-3.

3. L'estàndard SHA-256

En aquest apartat estudiarem en detall una de les funcions hash més utilitzades en l'actualitat: la SHA-256. Veurem quines són les seves característiques i descriurem amb detall tot el seu funcionament.

Com ja hem comentat, la SHA-256 és una de les funcions hash definides en l'estàndard FIPS-180-4 publicat pel NIST i que va ser desenvolupat el 2001 per la NSA. Com a característica general, cal dir que la 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 de la SHA-256 segueix l'esquema mostrat a la figura 2 de l'apartat 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, la 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 al subapartat 3.1.

3.1. *Padding* del missatge

Quan s'utilitzen funcions que processen els missatges en blocs, pot succeir que la mida del missatge 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 què 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 manera següent com s'ha de realitzar aquest procés.

Tipus de *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 la SHA-256 és diferent del que utilitza la SHA-512 i també diferent del que utilitza l'AES.

Suposem un missatge M de mida l bits, en què $l \neq 0 \mod 512$. En aquest cas procedirem a fer el següent:

1) afegir el bit 1 al final del missatge;

2) afegir a continuació k bits a zero, en què $k = 448 - (l+1) \mod 512$, 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 de càlcul de *padding* a la SHA-256

Prenem el missatge `abc`, que expressat en codi ASCII de 8 bits és:

$$\underbrace{01100001}_{a}\quad\underbrace{01100010}_{b}\quad\underbrace{01100011}_{c}$$

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:

$$\underbrace{01100001}_{a}\quad\underbrace{01100010}_{b}\quad\underbrace{01100011}_{c}\quad 1$$

Ara, calculem el valor k, de manera que $k = 448 - (24 + 1) \bmod 512 = 423$. És a dir, caldrà afegir 423 zeros al missatge:

$$\underbrace{01100001}_{a}\quad\underbrace{01100010}_{b}\quad\underbrace{01100011}_{c}\quad 1\ \underbrace{00\cdots0}_{423\text{ zeros}}$$

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:

$$\underbrace{01100001}_{a}\quad\underbrace{01100010}_{b}\quad\underbrace{01100011}_{c}\quad 1\ \underbrace{00\cdots0}_{423\text{ zeros}}\quad\underbrace{00\cdots011000}_{64\text{ bits}}$$

Mida màxima dels missatges

Fixeu-vos que aquest mecanisme de *padding* implica que la mida màxima dels missatges que pot tractar la 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*.

3.2. Funció de compressió de la SHA-256

Tal com ja hem comentat, la SHA-256 treballa amb blocs de 512 bits, els quals processa per mitjà 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 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 a 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.

3.2.1. Definició de les funcions internes de la SHA-256

En aquest subapartat definirem un seguit de funcions que s'utilitzaran tant en la part d'expansió del bloc com en el procés de compressió de la 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 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:

$$SHR^3(abcdefgh) = 000abcde$$

3.2.2. 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 a 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 a $M = M_0||M_1||\cdots||M_{15}$ en què 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 a fer-ho utilitzarem l'expressió següent:

$$W_t = \begin{cases} M_t & 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) \boxplus W_{t-7} \boxplus \sigma_0(W_{t-15}) \boxplus W_{t-16} & 16 \leq t \leq 63 \end{cases}$$

Les funcions σ_0 i σ_1 s'hi defineixen de la manera següent:

- $\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$
- $\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

L'operació $\boxplus$ és una suma mòdul 2^{32}.

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 paraula de 32 bits W_{16}.

$W_{16} = \sigma_1(W_{14}) \boxplus W_9 \boxplus \sigma_0(W_1) \boxplus W_0 =$

$= \sigma_1(0x00000000) \boxplus 0x00000000 \boxplus \sigma_0(0x00000000) \boxplus 0x61626380 =$

$= (ROTR^{17}(0x00000000) \oplus ROTR^{19}(0x00000000) \oplus SHR^{10}(0x00000000)) \boxplus$

$\boxplus$ 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

Per tant, W_{16} = 0x61626380.

D'aquesta mateixa manera podem calcular l'element W_{17}:

$W_{17} = \sigma_1(W_{15})\boxplus W_{10}\boxplus\sigma_0(W_2)\boxplus W_1 = \sigma_1(0x00000018)\boxplus 0x00000000\boxplus\sigma_0(0x00000000)\boxplus$ 0x00000000 = $(ROTR^{17}(0x080000018) \oplus ROTR^{19}(0x00000018) \oplus SHR^{10}(0x00000018)) \boxplus$ 0x00000000 $\boxplus$ $(ROTR^{7}(0x00000000) \oplus ROTR^{18}(0x00000000) \oplus SHR^{3}(0x00000000))$ $\boxplus$ 0x00000000 = 0x000f0000 $\boxplus$ 0x00000000 $\boxplus$ 0x00000000 $\boxplus$ 0x00000000 = 0x000f0000

Així doncs, W_{17} = 0x000f0000.

De la mateixa manera es calculen la resta de paraules fins a completar els 2048 bits.

3.2.3. Inicialització del *buffer*

Tal com veurem en aquest subapartat, 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 a l'estàndard:

$$H_0^{(0)} = \texttt{0x6a09e667}$$
$$H_1^{(0)} = \texttt{0xbb67ae85}$$
$$H_2^{(0)} = \texttt{0x3c6ef372}$$
$$H_3^{(0)} = \texttt{0xa54ff53a}$$
$$H_4^{(0)} = \texttt{0x510e527f}$$
$$H_5^{(0)} = \texttt{0x9b05688c}$$
$$H_6^{(0)} = \texttt{0x1f83d9ab}$$
$$H_7^{(0)} = \texttt{0x5be0cd19}$$

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. Aquestes 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 ]`

3.2.4. 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. A la figura 8 es pot veure el diagrama de la funció de compressió de la SHA-256 aplicada a un únic bloc.

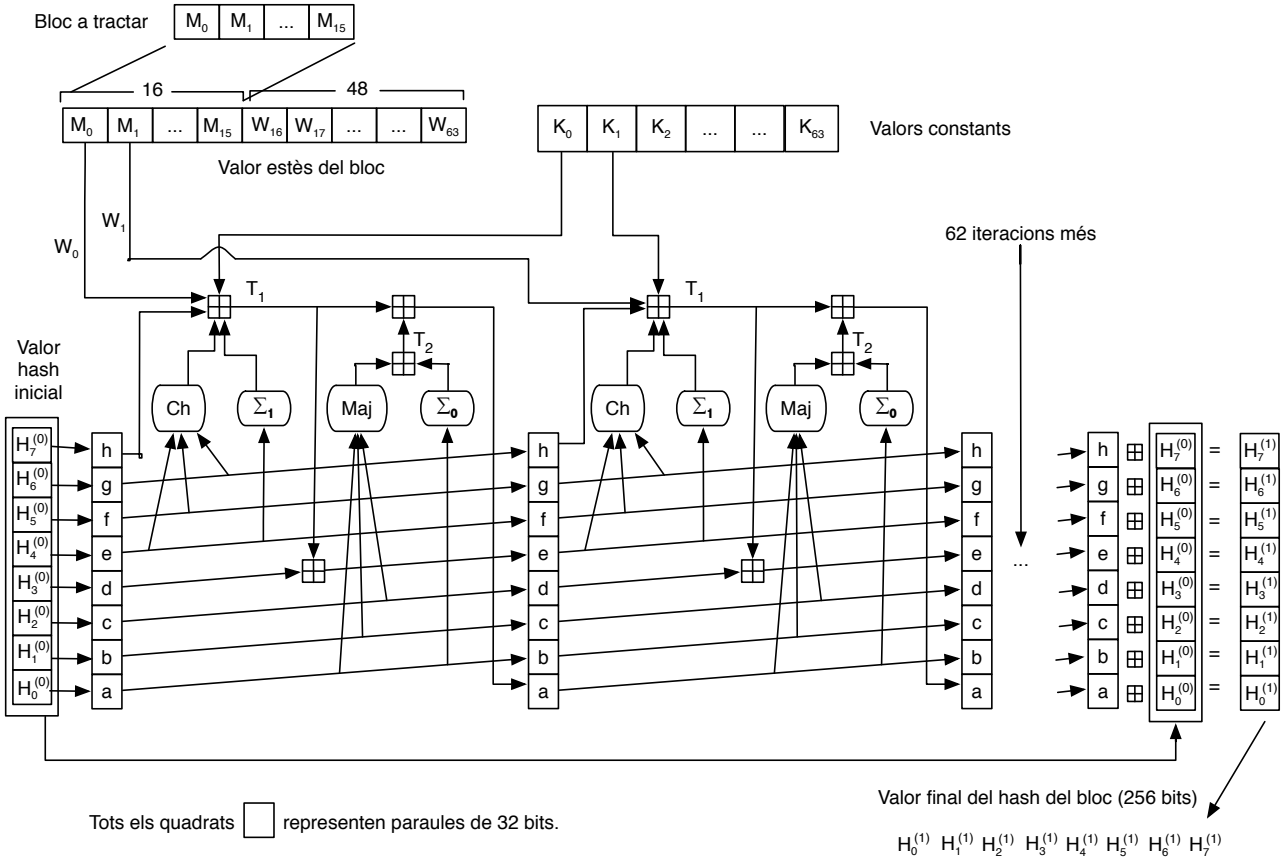
A la mateixa figura 8 es pot veure que en cada una de les 64 iteracions es fa servir tant una de les paraules de 32 bits, W_i, com una de les constants k_i del vector K, també de 32 bits, definides a l'estàndard. També veiem que els valors del bloc que es vol comprimir es combinen amb els valors inicials del hash, definits també a l'estàndard com a $H^{(0)}$. La figura 8 també mostra que aquestes combinacions estan formades per quatre funcions: Ch, Maj, $\sum_0$ i $\sum_1$, que es descriuen a continuació:

Recordeu

Una XOR es pot pensar com una suma mòdul 2 (component per component) $000101 \oplus 000111 = 000010$, i un AND com un producte mòdul 2 (component per component) $000101 \wedge 000111 = 000101$. Recordeu també l'operant de negació $\neg 0100 = 1011$. A més, en la nostra notació, l'operació $\boxplus$ és una suma mòdul 2^{32}.

- $\mathrm{Ch}(e,f,g) = (e \wedge f) \oplus (\neg e \wedge g)$
- $\mathrm{Maj}(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$
- $\sum_0(a) = ROTR^2(a) \oplus ROTR^{13}(a) \oplus ROTR^{22}(a)$
- $\sum_1(e) = ROTR^6(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e)$

Figura 8. Esquema de compressió d'un bloc de la funció SHA-256



La funció Ch actua sobre tres paraules de 32 bits amb operacions lògiques bàsiques.

Exemple de càlcul de la funció Ch

Càlcul de la funció Ch(e,f,g) per als valors:

e = 0x510e527f

f = 0x9b05688c

g = 0x1f83d9ab

e	01010001000011100101001001111111
f	10011011000001010110100010001100
$e \wedge f$	00010001000001000100000000001100

$\neg e$	10101110111100011010110110000000
g	00011111100000111101100110101011
$\neg e \wedge g$	00001110100000011000100110000000

$e \wedge f$	00010001000001000100000000001100
$\neg e \wedge g$	00001110100000011000100110000000
$(e \wedge f) \oplus (\neg e \wedge g)$	00011111100001011100100110001100

Per tant, el resultat en hexadecimal serà Ch(e,f,g) = 0x1f85c98c.

La funció Maj també actua sobre 3 paraules de 32 bits i, igual que la funció Ch, també opera utilitzant operacions lògiques bàsiques.

Exemple de càlcul de la funció Maj

Càlcul de la funció Maj(a,b,c) per als valors:

a = 0x6a09e667

b = 0xbb67ae85

c = 0x3c6ef372

a	01101010000010011110011001100111
b	10111011011001111010111010000101
$a \wedge b$	00101010000000011010011000000101

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 Maj(a,b,c) = 0x3a6fe667.

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 de càlcul de la funció $\sum_0$

Càlcul de la funció $\sum_0(a)$ per al valor: a = 0x6a09e667.

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 $\sum_0(a)$ = 0xce20b47e.

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 de càlcul de la funció $\sum_1$

Càlcul de la funció $\sum_1(e)$ per al valor: e = 0x510e527f.

e	01010001000011100101001001111111
$ROTR^6(e)$	11111101010001000011100101001001

e	01010001000011100101001001111111
$ROTR^{11}(e)$	01001111111010100010000111001010

e	01010001000011100101001001111111
$ROTR^{25}(e)$	10000111001010010011111110101000

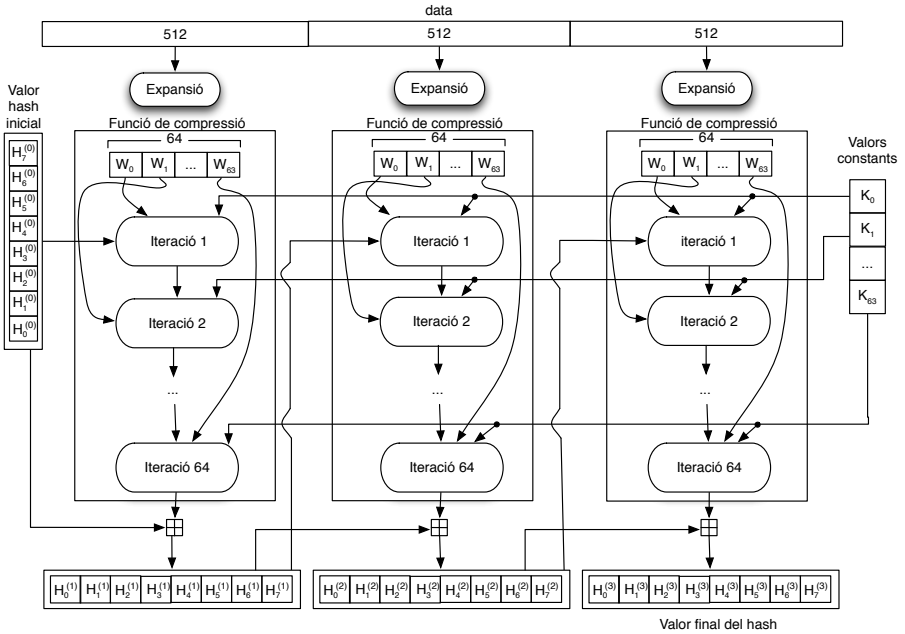
$ROTR^6(e)$	11111101010001000011100101001001
$ROTR^{11}(e)$	01001111111010100010000111001010
$ROTR^{25}(e)$	10000111001010010011111110101000
$ROTR^6(e) \oplus ROTR^{11}(e) \oplus ROTR^{25}(e)$	00110101100001110010011100101011

Per tant, el resultat en hexadecimal serà $\sum_1(e)$ = 0x3587272b.

3.3. SHA-256 sobre múltiples blocs

Als subapartats 3.1. i 3.2. hem parlat detalladament 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 què 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 manera recursiva sobre cada bloc, encadenant la sortida de cada bloc amb l'entrada del següent. A la figura 6 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.

Figura 6. SHA-256 mostrant el processat de 3 blocs



Com es pot apreciar a la figura 6, 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 a l'estàndard. Fixeu-vos que en cada bloc s'utilitzen els valors W_i corresponents obtinguts del mateix bloc, però, en canvi, els valors K_i utilitzats en el processat de cada bloc són sempre els mateixos. Per acabar, cal notar que el resultat del hash de cada bloc s'utilitza com a valor inicial per al càlcul del hash del bloc següent.

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, veurem les múltiples aplicacions en què s'utilitzen les funcions hash.

4.1. Codis d'autentificació de missatges

Un **codi d'autentificació de missatge** o MAC (en anglès, *message authentication code*) és una cadena curta d'informació relacionada amb el missatge per mitjà d'una clau simètrica; això permet que se'n pugui fer l'autentificació.

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, sense anar més lluny, l'autentificació, així com la integritat del missatge. Tot i això, els codis d'autentificació no ofereixen la propietat de no-repudi, propietat que sí que ofereixen les signatures digitals. Ara bé, els codis d'autentificació són molt més ràpids i eficients de calcular i és per aquest motiu que s'utilitzen en entorns en què la propietat de no-repudi no és essencial.

Signatures digitals

La definició i el funcionament de les signatures digitals la trobareu al mòdul didàctic "Criptografia de clau pública".

Com veurem a continuació, els MAC es poden implementar d'una manera 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 sobre el fet que les funcions hash no incorporen cap clau ni cap element secret. La manera de fer servir la clau, però, simplement és per a variar d'alguna manera la forma del missatge que es vol autenticar. Per exemple, d'una funció hash $h(\cdot)$ en podem derivar dos MAC:

$$HMAC1_k(m) = h(k \parallel m)$$
$$HMAC2_k(m) = h(m \parallel k)$$

Notació matemàtica

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 de càlcul d'un HMAC

A continuació, veurem un exemple de com utilitzar la funció hash definida a l'exemple del subapartat 2.1. per a calcular un *secret prefix* HMAC.

El missatge sobre el qual calcularem l'HMAC serà el següent: m = 11101010 i utilitzarem la clau k = 1100.

Per tant, $HMAC_k(m) = h(k \parallel m) = HMAC_{1100}(11101010) = h(1100 \parallel 11101010) = h(110011101010)$.

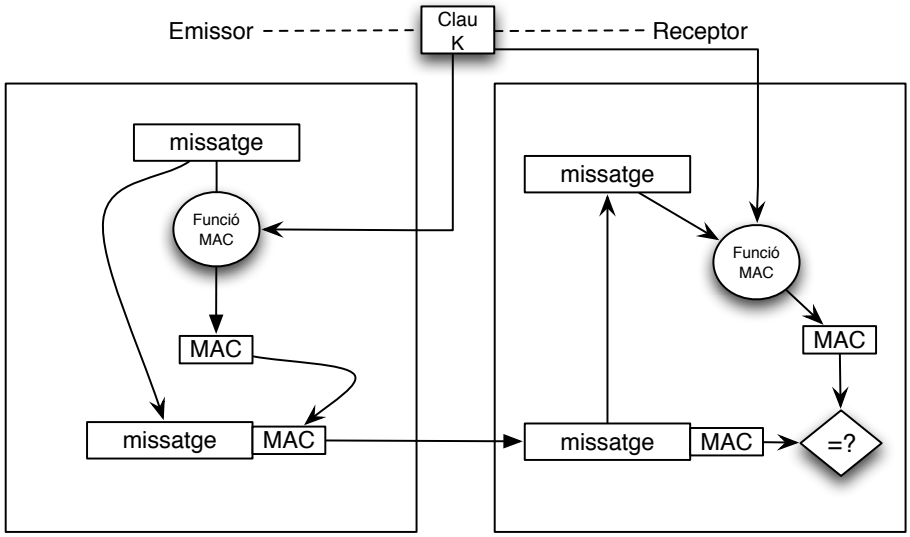
En aquest cas tenim tres blocs: m_1 = 1100, m_2 = 1110 i m_3 = 1010. Si ens hi fixem, els dos primers blocs són els mateixos que els de l'exemple de la funció hash, per tant, tenim que h_2 = 1010.

Així doncs, 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, $HMAC_k(m) = HMAC_{1100}(11101010) = 1111$.

La figura 7 esquematitza des del punt de vista pràctic com els HMAC s'utilitzen per a autentificar missatges.

Figura 7. Utilització d'un HMAC per autentificar missatges



Com es pot veure en la figura 7, emissor i receptor comparteixen una clau secreta. Cada vegada que l'emissor vol enviar un missatge al receptor, 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 a 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.

Autentificació contra confidencialitat

Fixeu-vos que els codis HMAC s'utilitzen per a 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 en podrà conèixer el contingut.

4.2. Resum de missatges

Una altra de les aplicacions en què s'utilitzen les funcions hash és per a obtenir una representació compacta d'un missatge més gran. Gràcies que el valor hash d'un missatge pot permetre identificar-lo d'una manera 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 només el valor hash a la mateixa base de dades i una localització externa. D'aquesta manera, es pot referenciar el contingut i fer-ne cerques fins i tot partint del mateix 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.

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 a 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'autentificació. Si la contrasenya proporcionada per l'usuari coincideix amb la que el sistema emmagatzema, l'autentificació 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 a una funció hash. D'aquesta manera, en el procés d'autentificació hi ha un pas més que cal fer, tal com es mostra a la figura 8.

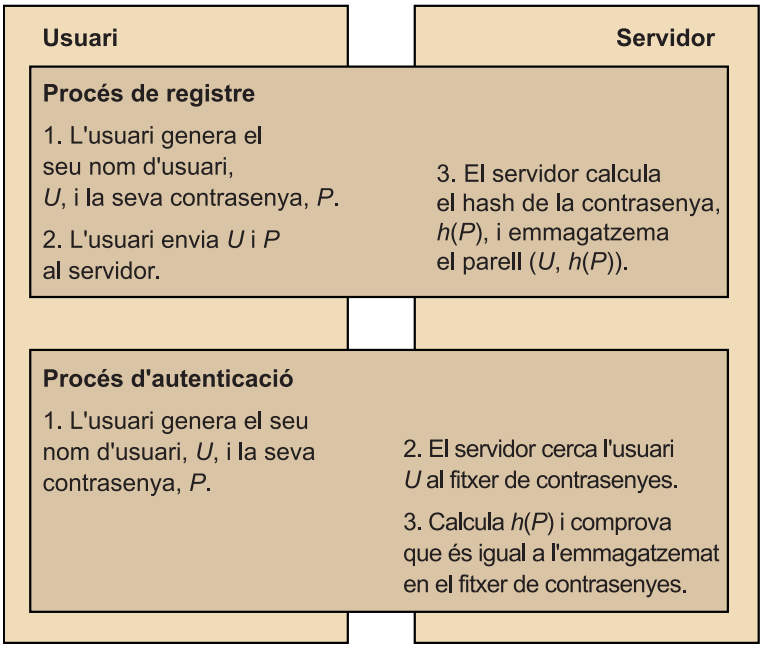
Contrasenyes no xifrades

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.

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 hash i el compara amb el valor que té emmagatzemat. En el cas que els dos valors coincideixin, l'usuari queda autentificat correctament.

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.

Figura 8. Esquema d'autentificació amb contrasenya



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.

Exemple d'atac a una contrasenya

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 l'atac següent.

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 si només un dels usuaris del sistema ja té 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 a evitar aquest 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 a 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'autentificació 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*, ha de calcular el hash de cada un dels valors del diccionari de contrasenyes per a cada un dels usuaris del sistema al qual està atacant.

Rainbow tables

Les *rainbow tables* són unes taules construïdes per a 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*. Les *rainbow tables* no són útils amb contrasenyes desades amb *salt*.

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 a aconseguir que les persones puguin generar i recordar claus

d'una manera senzilla, 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 per a la mateixa contrasenya d'entrada, donat que aquesta és determinista, 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, és basen en una funció hash.

4.4.1. La funció PBKDF2

La funció **PBKDF2** (de l'anglès, ***password-based key derivation function***) és una funció definida a l'RFC2898 que proporciona un mecanisme segur per a 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. A l'expressió següent 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 al subapartat 4.1., i en aquest punt 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 a 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 funcionament. L'expressió següent proporciona el sistema per a calcular la clau utilitzant la funció PBKDF2:

$$K = \mathtt{T}_1 \parallel \mathtt{T}_2 \parallel \cdots \parallel \mathtt{T}_{\lceil \mathtt{dkLen/hLen} \rceil}$$

El símbol $\parallel$ indica la concatenació i els valors $\mathtt{T}_i$ es descriuen a continuació.

Cada valor $\mathtt{T}_i$ el definim com a $\mathtt{T}_i$ = F(Contrasenya, Salt, c, i), en què la funció F queda explicitada en l'expressió següent:

$$\mathtt{F(Password, Salt, c, } i\mathtt{)} = \mathtt{U}_1 \oplus \mathtt{U}_2 \oplus \cdots \oplus \mathtt{U}_\mathtt{C}$$

En aquesta expressió, els valors $\mathtt{U}_i$ són els següents:

$$\begin{aligned} \mathtt{U}_1 &= \mathtt{PRF(Contrasenya,\ Salt} \parallel \mathtt{INT\_32\_BE(i))} \\ \mathtt{U}_2 &= \mathtt{PRF(Contrasenya,\ U}_1\mathtt{)} \\ &\vdots \\ \mathtt{U}_\mathtt{C} &= \mathtt{PRF(Contrasenya,\ U}_{\mathtt{C}-1}\mathtt{)} \end{aligned}$$

I 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 que el valor dkLen/hLen no sigui un enter i, per tant, la part entera superior de la divisió, és a dir, $\lceil$dkLen/hLen$\rceil$, proporcionarà un nombre total de bits de la clau superior a l'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.

4.5. 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 manera 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 manera 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 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 manera 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 a produir-la. És per aquest motiu que les funcions unidireccionals utilitzades en criptografia, com ara les funcions hash, són un bona 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 anticorreu 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 calcularà la imatge per a una funció hash determinada, que haurà consensuat amb B. Prèviament, A i B també hauran 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 en realitzar el càlcul del hash 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 a enviar el missatge a B.

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 sigui 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.

Fixeu-vos que la necessitat que té l'emissor del missatge per a 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.

Aquest mateix mecanisme de prova de treball és el que fan servir moltes criptomonedes, com ara els bitcoins, per incloure un nou bloc a la cadena i assegurar que un usuari no pot gastar de nou uns diners que ja havia gastat prèviament.

Resum

Com hem pogut veure en aquest mòdul didàctic, 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 coneguda i assumint que la definició de la funció hash queda totalment determinada de manera pública. A més, aquesta predicció no es pot realitzar encara que es conegui 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 a 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 analitzar-les. Per aquest motiu, la simple definició d'una funció hash, com ara la SHA-256, ja implica una complexitat força elevada.

Gràcies a aquestes propietats, les funcions hash es poden utilitzar per a realitzar autentificació de missatges amb criptografia de clau simètrica, per a obtenir resums quasi unívocs de missatges, 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.

Exercicis d'autoavaluació

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?

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 a l'expressió següent:

$$E_k(m) = m \oplus k_{ext}$$

En aquesta expressió k_{ext} s'obté a partir de la clau k amb l' 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 3 del subapartat 2.1. 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 $h(01010101)$.

3. Calculeu la cadena de bits que processarà la funció SHA256 una vegada s'ha realitzat el *padding* al missatge d'entrada $m =$ `SALA`, on els caràcters s'han codificat en ASCII amb 8 bits.

4. Realitzeu els càlculs de les funcions internes de la SHA256 tenint en compte els valors de les cadenes següents:

$$m_1 = 00000000000000000111111111111111$$
$$m_2 = 11110000000000000111111111110000$$
$$m_3 = 11111111000000000111111100000000$$

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. 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 a assegurar la seva integritat, és a dir, per a 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 manera correcta, ja que l'atacant desconeix la clau. Fan servir una funció HMAC basada en la funció hash $h(\cdot)$ definida a l'exemple del subapartat 2.1., 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 $m' = 01111111$?

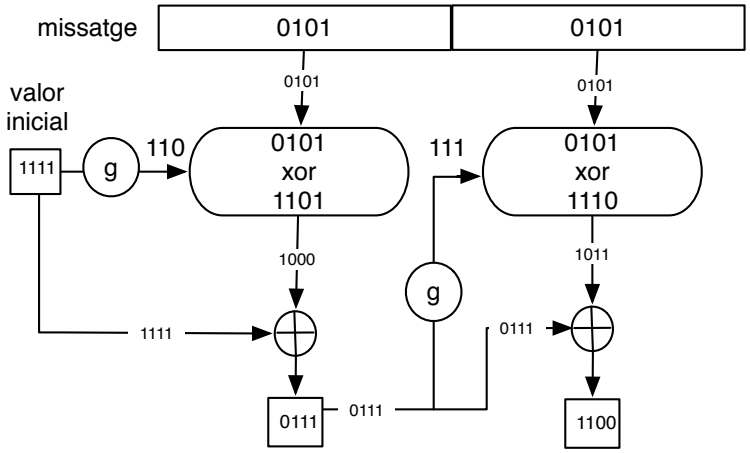
6. Tenim un sistema que utilitza una prova de treball per mitjà d'una funció hash. Aquesta funció hash té una mida de 64 bits i la potència de càlcul de la xarxa que l'utilitza està fixada en 100.000 hashs per segon. Fixeu un valor objectiu de la funció hash per tal que, amb la potència de càlcul que s'indica, es trobi una imatge del hash menor que el valor objectiu, de mitjana, cada 10 minuts.

Solucionari

1. Utilitzant les expressions del subapartat 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. Així mateix, la probabilitat que almenys una d'elles hagi nascut el dia 1 de gener és de $1-\left(1-\frac{1}{365}^{49}\right)$ = 0,12.

2. El resultat de la funció hash és el valor 1100 i el procés de càlcul es mostra a la figura 9.

Figura 9. Procés de càlcul de la funció hash



3. En primer lloc, cal passar el valor SALA a una cadena de bits, amb la qual cosa obtenim:

$$01010011010000010100110001000001$$

A continuació afegim un 1 a la cadena, obtenint 01010011010000010100110001000001**1**.

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:

$$01010011010000010100110001000001\underbrace{1}\ \underbrace{00\cdots0}_{415\text{ zeros}}\ \underbrace{00\cdots00100000}_{64\text{ bits}}$$

4. Els resultats de les funcions són els següents:

$$ROTR^7(m_1) = 11111111000000000000000111111111$$
$$SHR^{10}(m_1) = 00000000000000000000000000111111$$
$$\sigma_0(m_1) = 11000001111111111101111000000000$$
$$\sigma_1(m_1) = 01100000000000000110000000111111$$
$$\sum_0(m_1) = 00111100000001111100001111111000$$
$$\sum_1(m_1) = 00000011100111111111110001100000$$
$$\mathrm{Ch}(m_1,m_2,m_3) = 11111111000000001111111111110000$$
$$\mathrm{Maj}(m_1,m_2,m_3) = 11110000000000001111111111110000$$

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 del qual cal calcular el hash. Si realitzem els càlculs, trobarem que l'HMAC per al missatge 01111111 serà 0001.

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 hashs per segon, haurem provat $100.000\cdot60\cdot10=60.000.000$ valors. Per tant, el valor objectiu per a la nostra prova de treball serà $\frac{2^{64}}{60.000.000}\approx 307445734561$. Expressat en una cadena de 64 bits tenim:

$$0000000000000000000000001000111100101010011000110011100101000001$$

Glossari

codi d'autentificació de missatge *m* Cadena curta d'informació relacionada amb el missatge per una clau simètrica de manera que en permet l'autentificació.
sigla **MAC**
en message authentication code

funció de derivació de claus *f* Funció que implementa un mecanisme segur per a obtenir una clau a partir d'una contrasenya.

família de funcions SHA *f* Conjunt de funcions hash identificades com a *secure hash algorithms* que estan estandarditzades pel NIST. Tenen mides entre d'160 i 512 bits.

funció hash *f* Funció que pren com a entrada un missatge (o cadena) d'una mida arbitràriament gran i en retorna una cadena de mida fixa

funció hash criptogràfica *f* Funció hash amb les propietats de resistència a preimatge, resistència a segona preimatge i resistència a col·lisions.

funció resistent a col·lisions febles *f* Una funció f que compleix que donat un valor x tal que $y = h(x)$ no és possible trobar un valor x amb $x' \neq x$ de manera que $y = h(x)$.

funció resistent a col·lisions fortes *f* Una funció f tal que no és possible trobar dos valors x_1 i x_2 diferents ($x_1 \neq x_2$) de manera que $h(x_1) = h(x_2)$.

funció unidireccional *f* Funció que és fàcil de calcular però molt difícil d'invertir.

HMAC *m* Codi d'autentificació de missatge implementat utilitzant una funció hash.

KECCAK *f* Família de funcions hash proposada pels criptògrafs Guido Bertoni, Joan Daemen, Michael Peeters i Gilles van Assche l'any 2008 i estandarditzada pel NIST el 2015. Proposa funcions de mida 224, 256, 384, 512.

MAC *m* Vegeu **codi d'autentificació de missatge**.

message authentication code *m* Vegeu **codi d'autentificació de missatge**.

mida d'una funció hash *f* Mida de la cadena que retorna la funció.

padding *m* Mecanisme pel qual un sistema que processa la informació en blocs defineix com completarà l'últim bloc en cas que el missatge que s'ha de tractar no sigui múltiple exacte de la mida del bloc.

PBKDF2 *f* Funció de derivació de claus definida a l'RFC2898.

prova de treball *f* Mecanisme que permet a l'usuari d'un sistema demostrar a la resta d'usuaris de manera fidedigna que ha realitzat una certa quantitat de feina, normalment, una certa quantitat de càlculs.

rainbow table *f* Taula precalculada de valors hash que permet que un atacant d'un sistema de contrasenyes desades utilitzant hashs (sense salt) pugui establir un equilibri entre la informació que ha de tenir precalculada i, per tant, emmagatzemada, i el volum de càlculs que ha de realitzar durant l'atac.

RIPEMD *f* Família de funcions hash creades pels criptògrafs belgues Hans Dobbertin, Antoon Bosselaers i Bart Preneel l'any 1996 i que tenen mides de 128, 160, 256 i 320 bits.

SHA-256 *f* Funció hash de mida 256 bits àmpliament utilitzada que forma part de la família de funcions hash SHA estandarditzada pel NIST.

SHA-3 *f* Nom que rep la funció KECCAK en la seva versió estandarditzada pel NIST.

WHIRLPOOL *f* Funció hash creada pels criptògrafs Vincent Rijmen i Paulo S. L. M. Barreto l'any 2000 amb una mida de 512 bits i estandarditzada a la ISO/IEC 10118-3.

Bibliografia

Dobbertin, H.; Bosselaers, A.; Preneel, B. (1996). "RIPEMD-160: A Strengthened Version of RIPEMD". *Proceedings of FSE, LNCS* (núm. 1039, pàg. 71-82). Nova York: Springer-Verlag.

Institut Nacional d'Estàndards i Tecnologia (2015). "Secure Hash Standard (SHS)". *Federal information processing standards* (núm. 180-4). Washington: NIST.
DOI: 10.6028/NIST.FIPS.180-4

Institut Nacional d'Estàndards i Tecnologia (2015). "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions". *Federal information processing standards* (núm. 202). Washington: NIST. DOI: 10.6028/NIST.FIPS.202

Lenstra, A.; Wang, X.; Weger, B. (2005). "Colliding X.509 Certificates". *Cryptology ePrint Archive Report* (núm. 2005/067).

Paar, C.; Pelzl, J. (2010). *Understanding Cryptography* (vol. 11). Heidelberg: Springer-Verlag. DOI: 10.1007/978-3-642-04101-3

Rijmen, V.; Barreto, P. (2000). *The WHIRLPOOL Hash Function*. Lovaina: Universitat Catòlica de Lovaina.

Rivest, R. (1992). "The MD4 Message-Digest Algorithm". *Request for Comments* (núm. 1320). Fremont: Internet Engineering Task Force.

Rivest, R. (1992). "The MD5 Message-Digest Algorithm". *Request for Comments* (núm. 1321). Fremont: Internet Engineering Task Force.

Rivest, R. (1992). "PKCS #5: Password-Based Cryptography Specification Version 2.0". *Request for Comments* (núm. 2898). Fremont: Internet Engineering Task Force.

Rivest, R. (2011). "MD4 to Historic Status". *Request for Comments* (núm. 6150). Fremont: Internet Engineering Task Force.