# CriptoTema4.pdf


onafolch


Criptografia i Seguretat

3º Grado en Ingeniería de Datos


Escuela de Ingeniería
Universidad Autónoma de Barcelona

# CRIPTOGRAFIA

## FUNCIONS HASH

### 1. FUNCIONS HASH

Com veurem, una funció hash és una funció que permet obtenir un valor fixat de mida reduïda a partir d'una entrada arbitràriament gran, i és eficientment calculable i determinista, és a dir, donades dues entrades iguals sempre ens proporcionarà la mateixa sortida.

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' ≠ 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 x1 i x2 diferents (x1 ≠ x2) tals que h(x1) = h(x2).

**Exemple:** 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 és 1/365.

Quanta gent ha d'haver perquè la probabilitat que una altra persona tingui l'aniversari al mateix dia sigui de 0.5? És (1- 1/365)^n, n = 253.

Ara bé, quanta gent ha d'haver perquè la probabilitat que dos persones tinguin l'aniversari al mateix dia sigui del 0.5? $\left(\frac{364}{365}\right)^{n(n-1)/2}$, n = 23.

Amb això podem dir que 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.

### 2. CONTRASENYES

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 (normalment es guarden moltes iteracions de la funció hash → sha512-crypt).

Les contrasenyes emmagatzemades únicament amb el seu hash permeten atacs com ara el següent. Suposem un sistema que té les contrasenyes emmagatzemades en un fitxer, i un atacant pogués aconseguir aquest fitxer. Aquest podria agafar un diccionari de contrasenyes habituals i pot anar comparant les funcions hash i comparar el resultat amb cada un dels valors del fitxer. Només que algun dels usuaris del sistema tingui una de les contrasenyes del diccionari, l'atacant pot trobar el hash correcte.

Per evitar-ho, 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

Ona Folch

s'haurà d'afegir en el procés d'autenticació quan s'està validant la correcció de la contrasenya proporcionada per l'usuari.

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.

Per escollir una contrasenya, serà més segura si l'escollim amb valors aleatoris o paraules que no es puguin pronunciar.

## 3. FILTRES DE BLOOM

Són estructures de dades que ens serveixen per saber si un element està a una llista o no de manera eficient. Està format per un vector binari V de n bits de mida inicialitzat a 0, i un conjunt de funcions hash independents que tenen rang [0, n-1].

Per afegir un element al filtre, primer s'apliquen les k funcions hash a l'element, obtenint valors k entre 0 i n-1., i s'assignen a 1 les k posicions del vector.

Per saber si un element està al vector, faré 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 conté un 0, sabem 100% que no està l'element. Si ens donen totes 1, direm que l'element si que està, però no ho podem afirmar amb un 100% de probabilitat. Mai pot retornar falsos negatius, però si falsos positius.

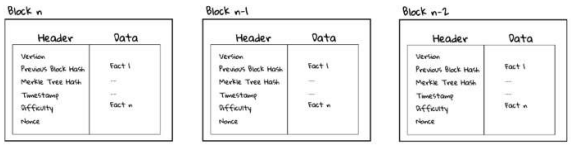
## 4. BLOCKCHAIN

La presa de decisions per consens és un procés de presa de decisions grupal en què els membres del grup desenvolupen i accepten donar suport a una decisió en el millor interès del conjunt.

En sistemes distribuïts, un mecanisme de consens és un mecanisme tolerant a errors que s'utilitza per aconseguir l'acord necessari sobre un valor de dades únic o un estat únic de la xarxa entre processos distribuïts o sistemes multiagent.

Un ledger distribuït és un consens de dades digitals replicades, compartides i sincronitzades distribuïdes geogràficament per múltiples entitats sense administrador central. Les regles d'operació de lectura i escriptura s'han de definir correctament.

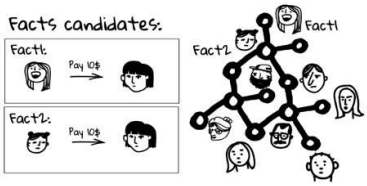
Un blockchain és una llista creixent de registres, anomenats blocs, que s'uneixen mitjançant criptografia. L'estructura de dades de la blockchain és molt diferent de l'estat mundial perquè un cop escrita, no es pot modificar; és immutable.



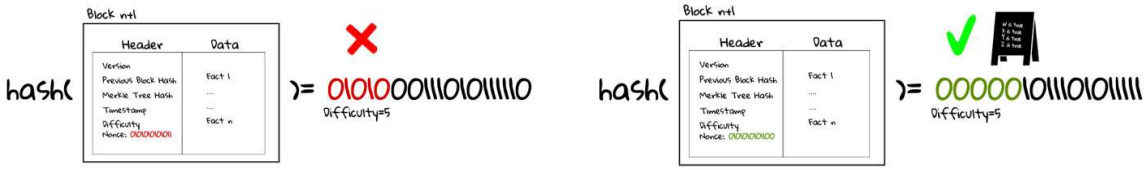

- **Previous Block Hash**: hash del bloc anterior que forma la cadena.
- **Timestamp**: hora en què es tanca el bloc.
- **Dificultat**: manté constant el temps mitjà entre blocs tancats a mesura que canvia la potència hash de la xarxa.
- **Nonce**: el valor l'ajusten els miners de manera que el hash del bloc serà inferior o igual a l'objectiu actual de la xarxa.
- **Merkle Tree Root**: un arbre hash binari que ajuda a codificar les dades de la blockchain de manera més eficient i segura.

Ona Folch

La mineria de la blockchain s'utilitza per assegurar i verificar els fets de la cadena de blocs. La mineria implica miners de blockchain que afegeixen fets de les dades al Ledger públic global de blockchain de transaccions passades.




La prova de treball és lo bàsic pels algorismes de consens de blockchain on els validadors (anomenats miners) trien les dades que volen afegir fins que produeixen una solució específica. Els miners van provant valors de nonce fins que tinguin (en aquest cas), tants zeros com la dificultat.



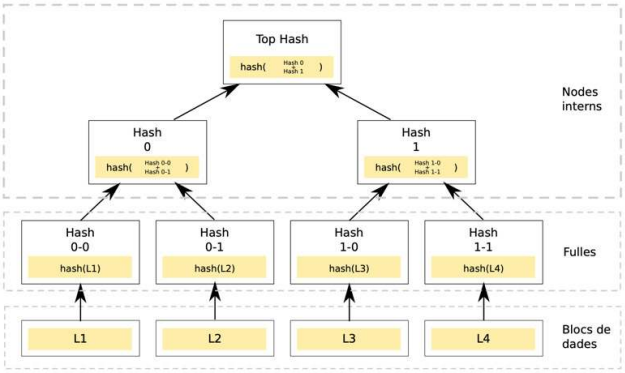

Les cadenes de blocs solen ser gestionades per una xarxa P2P per utilitzar-les com a ledger distribuït públicament, on els nodes s'adhereixen col·lectivament a un protocol per comunicar i validar nous blocs. Tot i que els registres de blockchain no són inalterables, ja que són possibles les bifurcacions, les blockchains es poden considerar segures per disseny i exemplifican un sistema informàtic distribuït amb una alta tolerància a errors bizantina.

## 5. MERKLE TREES

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.

Si divideixo el missatge en 4, i faig els hash, ho estic fent de manera optimitzada. A envia el merkle tree i el fitxer, i quan B ho rep, veu que no li dona igual el hash de tot el fitxer. Doncs el que farà ara és el hash de la primera divisió, i descartarà el que li doni bé, i això fins trobar on està l'error. La busqueda és logarítmica, enlloc de n.

També serveixen per fer proves de pertinença. Si volem mirar si L4 està a l'arbre i tenim el hash de L3, podem fer el hash de L3+L4. Si tenim el hash de L1+L2, podem calcular el de L1+L2+L3+L4. Si el hash dona el mateix, és una prova que la transacció L4 estava inclosa al bloc. Aquests arbres ajuden a codificar les dades de blockchain de manera més eficient i segura. Quan feu mineria, només heu de fer hash la capçalera del bloc, en lloc de tot el bloc.




Ona Folch

## 6. CODIS D'AUTENTICACIÓ DE MISSATGES

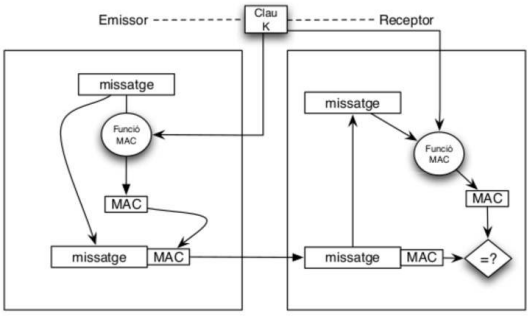
Un codi d'autentificació de missatge (MAC) és una cadena curta d'informació relacionada amb el propi missatge a través d'una clau de manera que permet la seva autenticació.

Permeten autentificar missatges, i comparteixen algunes propietats amb les signatures digitals com ara la pròpia autentificació així com la integritat del missatge.

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(·) 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 || representa la concatenació de cadenes. La primera expressió es coneix com a secret prefix HMAC i la segona com a secret suffix HMAC.




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

Ona Folch