Funzioni crittografiche di hash

Codici di checksum

- Un codice di checksum è una “firma” di dimensione fissa (ad es. 32 bit) calcolata a partire da un messaggio(*) di lunghezza variabile.
- Ogni messaggio ha una sua firma caratteristica
- Il checksum viene usato per rilevare modifiche accidentali ad un messaggio, ad esempio dovute ad errori di trasmissione su una linea rumorosa (usati ad esempio nei modem, nelle reti ethernet, ecc.)

(*) nel senso più generico del termine, inteso come sequenza di bit

Esempio di codice checksum: CRC

- CRC (Cyclic Redundancy Check) è un codice di checksum ampiamente usato fin dagli anni ‘70 per rilevare errori di trasmissione nelle comunicazioni

- Esistono diversi codici CRC, a seconda della lunghezza della “firma” finale. Uno dei più usati: CRC-32.

- Per ogni algoritmo di CRC-n esiste un “numero magico” *c* lungo n bit da utilizzare nel seguente modo
  - Far scorrere da sinistra a destra c finché il suo bit più significativo (che vale sempre 1) corrisponde ad un 1 nel messaggio
  - Fare lo XOR (estendere c con degli 0)
  - Ripetere l’algoritmo sul valore risultante, finché non si arriva alla fine del messaggio. Gli ultimi n-1 bit del risultato finale sono il checksum

CRC - Esempio

- CRC-4 – numero magico: 1011
- Messaggio: 11010011101100

- Il CRC finale è 101

- Il messaggio 10010011101101 dà invece il risultato 010

un errore di trasmissione che ha modificato un bit può essere rilevato da un controllo CRC

```
11010011101100
1011
01100011101100
 1011
 0111011101100
  1011
  010111101100
   1011
   00001101100
        1011
        0110100
         1011
         011000
          1011
          01110
           1011
           0101
```

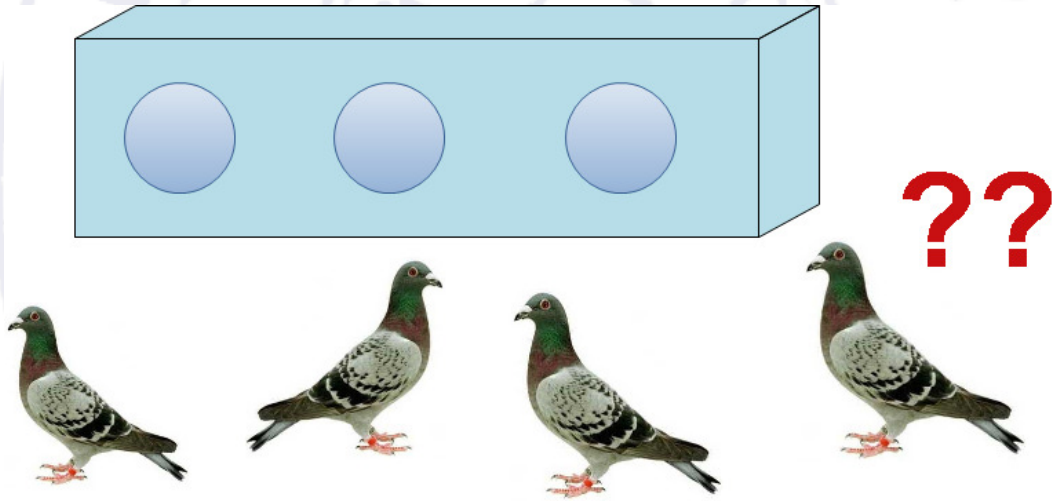
Efficienza dei checksum CRC

- Un checksum CRC a n bit è tipicamente in grado di rilevare qualsiasi “burst” di errori di lunghezza ≤ n, e ogni altro burst con una probabilità di $1-2^{-n}$

- I checksum CRC (e più in generale tutti i codici di checksum) non sono dunque infallibili. Questo deriva banalmente dal cosiddetto principio dei “buchi di colombaia”

Principio dei buchi di colombaia

- Data una colombaia con *n* buchi e *m* > *n* colombe, esiste necessariamente un buco in cui entrerà più di una colomba



- Domanda: esistono a Milano due persone con lo stesso numero di capelli in testa?

Checksum e colombi

- Se n è la lunghezza in bit del checksum e m > n quella del messaggio, allora esistono 2^n checksum possibili e 2^m messaggi possibili
- Di conseguenza, per il principio dei buchi di colombaia, anche nel migliore dei casi ad ogni checksum saranno associati $2^m/2^n = 2^{m-n}$ messaggi distinti!
- Ad esempio, se n=32 e m=128, ad ogni checksum sono associati mediamente 2^{96} messaggi (ipotizzando che siano uniformemente distribuiti)
- Questo non significa che i codici di checksum sono inutili. In realtà sono studiati affinché gli errori di trasmissione più comuni generino messaggi con checksum diversi.

Funzioni di hash

- Le funzioni di checksum non sono generalmente studiate per rilevare modifiche *volontarie* da parte di un attaccante

- Per affrontare questo problema, sono state sviluppate le funzioni crittografiche(*) di hash, dette anche *message digest*, o semplicemente *digest*

- Le funzioni di hash, per garantire la robustezza agli attacchi, devono soddisfare i seguenti requisiti…

(*) per distinguerle da altre funzioni di hash usate in informatica…

Requisiti di una funzione di hash

- È facile calcolare l'hash di un messaggio
- È difficile trovare il messaggio che ha generato un dato hash *(non invertibilità delle funzioni di hash)*
- È difficile modificare un messaggio senza modificare il relativo hash *(resistenza debole alle collisioni)*
- È difficile trovare due messaggi che abbiano lo stesso hash *(resistenza forte alle collisioni)*

Usando le solite nozioni di "facile" e "difficile" in termini computazionali…

Requisiti di una funzione di hash

- I primi due punti ricordano i concetti già visti in ambito crittografico, tuttavia in quel contesto era richiesta la non invertibilità solo in assenza della chiave.

- In merito agli ultimi due punti, si noti che esistono necessariamente più messaggi con lo stesso hash (principio dei buchi di colombaia), tuttavia deve essere difficile per un attaccante trovare due messaggi con tale proprietà.

Utilizzi delle funzioni di hash

- Gli hash sono una sorta di “impronta digitale” a lunghezza fissa di un messaggio. Sono quindi utili in diversi settori applicativi, ad esempio…

- **Controllo degli errori:**

  in questo caso la funzione di hash viene utilizzata come un checksum, per identificare eventuali errori di trasmissione dei dati. Le proprietà aggiuntive degli hash non vengono in questo caso sfruttate

Utilizzi delle funzioni di hash / 2

- **Integrità dei dati:**

  la funzione di hash viene utilizzata per garantire che un messaggio non sia stato modificato da un eventuale attaccante.

  Utilizzato ad esempio negli IDS (intrusion detection systems) per controllare l'integrità dei file critici di un sistema operativo (*).

  Esempi software: tripwire http://www.tripwire.com

  fastsum http://www.fastsum.com

  …

(*) gli hash devono essere memorizzati in un luogo sicuro. In alternativa, usare un HMAC (vedi ultime slide).

Utilizzi delle funzioni di hash / 3

- **Verifica manuale dell'integrità dei dati**

  Bob spedisce via mail ad Alice la propria chiave pubblica di un algoritmo di cifratura asimmetrica. Alice telefona a Bob per verificare la paternità della chiave pubblica…

  Anziché leggere tutti i byte della chiave (può essere molto lunga), Bob può comunicare un breve hash

Un software free per calcolare gli hash di file e di testi:
HashCalc http://www.slavasoft.com/hashcalc/index.htm

(Esempio: chiavi PGP)

Utilizzi delle funzioni di hash / 4

- **Memorizzazione di password e autenticazione:**

  nella maggior parte dei casi, quando un sistema informatico deve memorizzare una password, questa non viene salvata in chiaro per motivi di sicurezza, né viene cifrata per evitare che sia possibile risalire alla password originale. Si memorizza in questi casi l'hash della password.

  Ad esempio, sotto Windows, Linux e MacOS le password degli utenti sono salvate sotto forma di hash. Quando un utente scrive la propria password per accedere al sistema, ne viene calcolato l'hash e confrontato con quello memorizzato nel sistema.

  (gli hash di Windows pre-Vista sono notoriamente deboli e facilmente craccabili usando tool come saminside http://www.insidepro.com/eng/saminside.shtml o L0phtcrack http://www.l0phtcrack.com)

Utilizzi delle funzioni di hash / 5

- **Firme digitali**:

  Si è già detto che la crittografia a chiave pubblica è computazionalmente molto onerosa, per cui solitamente viene usata solo per trasmettere la chiave di una crittografia simmetrica

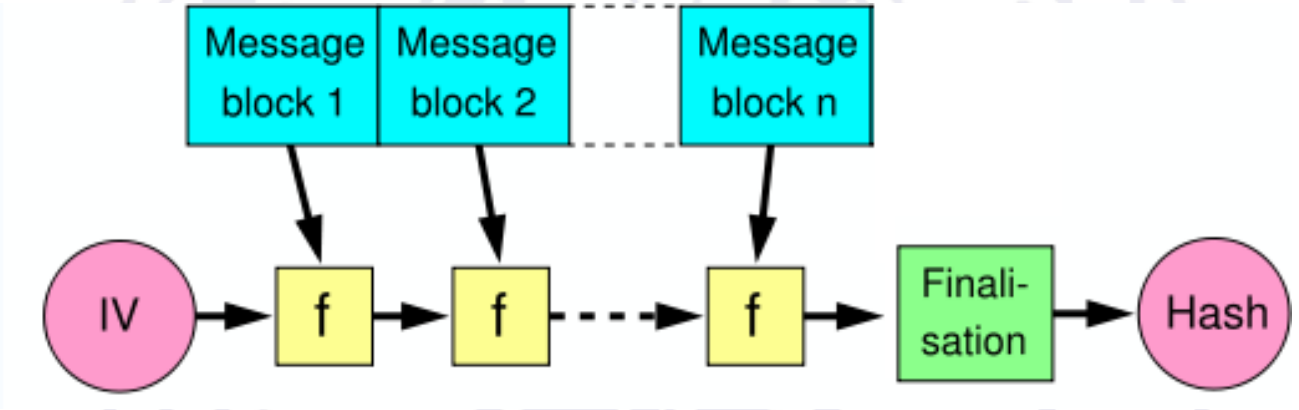
  Per le firme digitali esiste un problema di prestazioni analogo: firmare l'intero documento è computazionalmente pesante

  Soluzione: **firmare solo un hash del documento.**

  (ha anche il vantaggio di lasciare il documento in chiaro, leggibile anche da chi non dispone degli strumenti per verificare la firma digitale)

Struttura di Merkle–Damgård

- È la struttura della maggior parte delle funzioni di hash attualmente utilizzate



- L'input è diviso in blocchi di dimensione fissa (con eventuale zero-padding sull'ultimo)
- IV è un vettore iniziale predefinito
- Una funzione f, detta "compression function", viene ripetutamente applicata ai blocchi di messaggio e all'output della f precedente

La compression function

- La compression function è così chiamata perché il suo uso concatenato ha il compito di “comprimere” l’informazione di un messaggio in un numero minore di bit

- Per la sua struttura, spesso la funzione f è “presa in prestito” dagli algoritmi di crittografia. Più in generale, molte tecniche crittografiche, con le opportune modifiche, possono essere utilizzate per creare algoritmi di hash e viceversa.

L'algoritmo di hash MD5

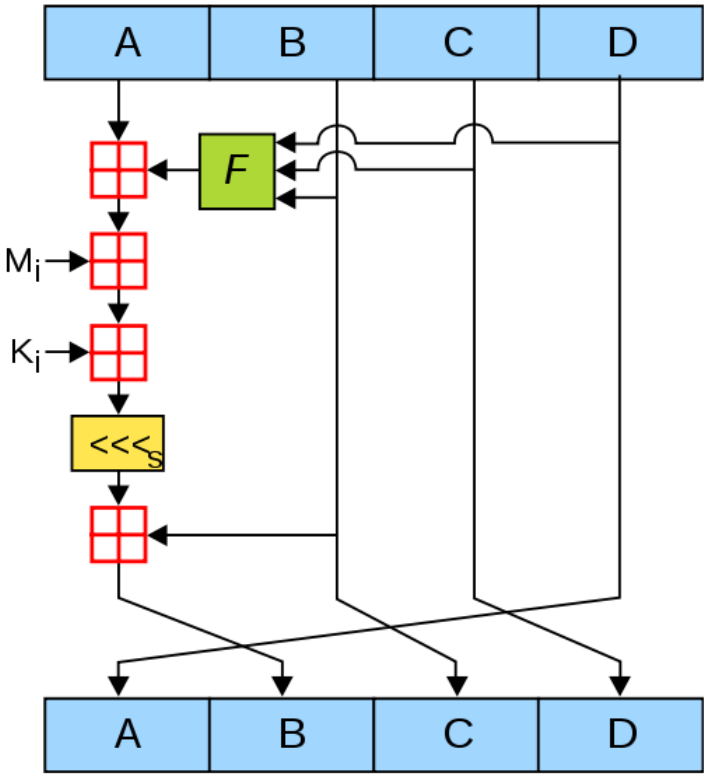
- MD5 (message-digest algorithm 5) è un algoritmo di hash sviluppato da Ronald Rivest (la "R" di RSA)
- Genera hash da 128 bit (32 caratteri esadecimali)
- È stato ampiamente usato finora, ma negli ultimi anni sono state dimostrate alcune sue debolezze crittografiche, per cui non è più considerato sicuro per applicazioni critiche

MD5

- Il messaggio viene diviso in blocchi da 512 bit
- Ogni blocco viene elaborato in 4 “round” successivi
- In ogni round la funzione di compressione viene applicata 16 volte
- La funzione F dipende dal round:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

- M_i è un sottoblocco di 32 bit
- K_i è una costante che cambia a seconda del round
- All’inizio, A B C e D sono inizializzati con un IV da 128 bit

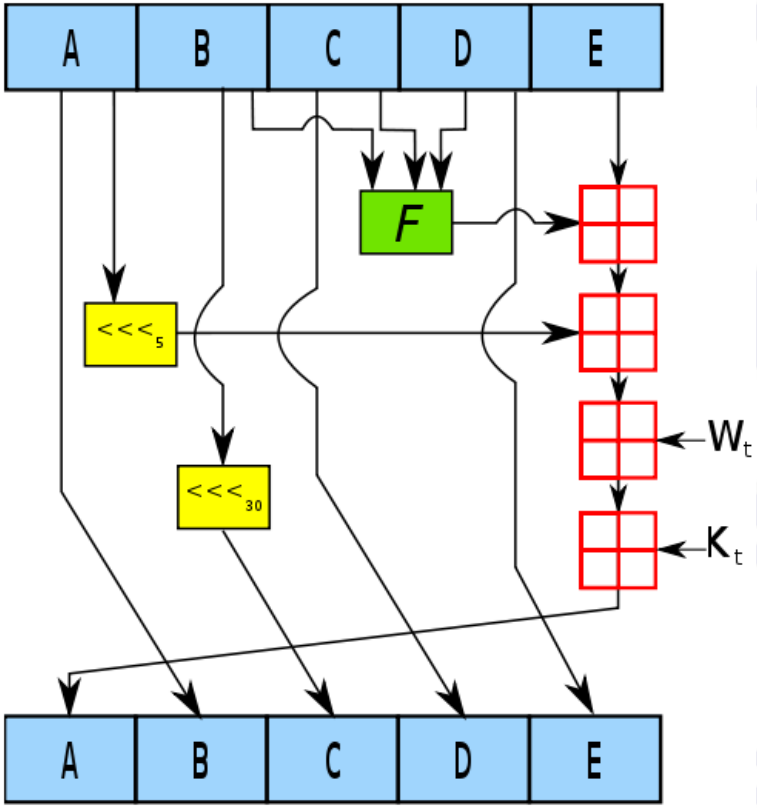


Compression function di MD5

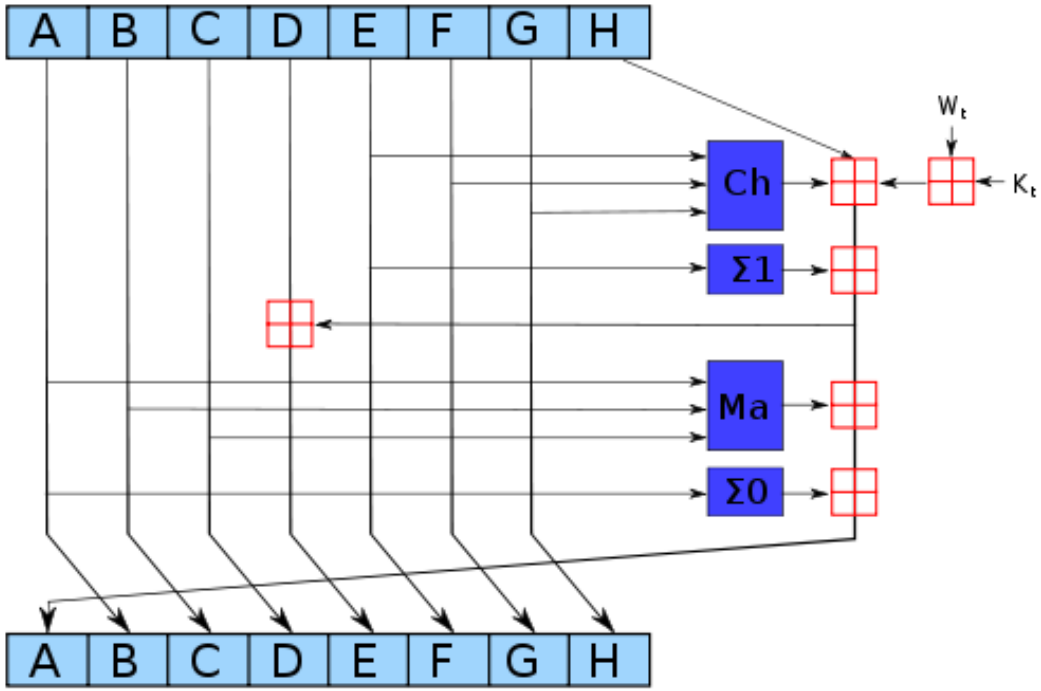
Gli algoritmi di hash SHA

- SHA-1 è l'algoritmo di hash standard adottato dalla NSA (National Security Agency)
- Oggi si usano delle varianti più sicure che vanno sotto il nome di SHA-2, con dimensioni dell'hash maggiori (da 256 a 512 bit, contro i 160 di SHA-1)
- È attualmente in atto il contest per scegliere il nuovo standard SHA-3

Funzioni di compressione di SHA-1 e SHA-2



SHA-1



SHA-2

Attacchi agli algoritmi di hash

- **Attacchi alla preimmagine** : dato un hash h, trovare il messaggio m tale che H(m) = h (H è la funzione di hash)

  (violazione del principio di non invertibilità)

- **Attacchi alla seconda preimmagine:** dato un messaggio m1, trovare un messaggio m2 tale che H(m1) = H(m2)

  (violazione del principio di resistenza debole alle collisioni)

- **Rilevamento di collisioni:** trovare due messaggi m1 ed m2 tali che H(m1) = H(m2)

  (violazione del principio di resistenza forte alle collisioni)

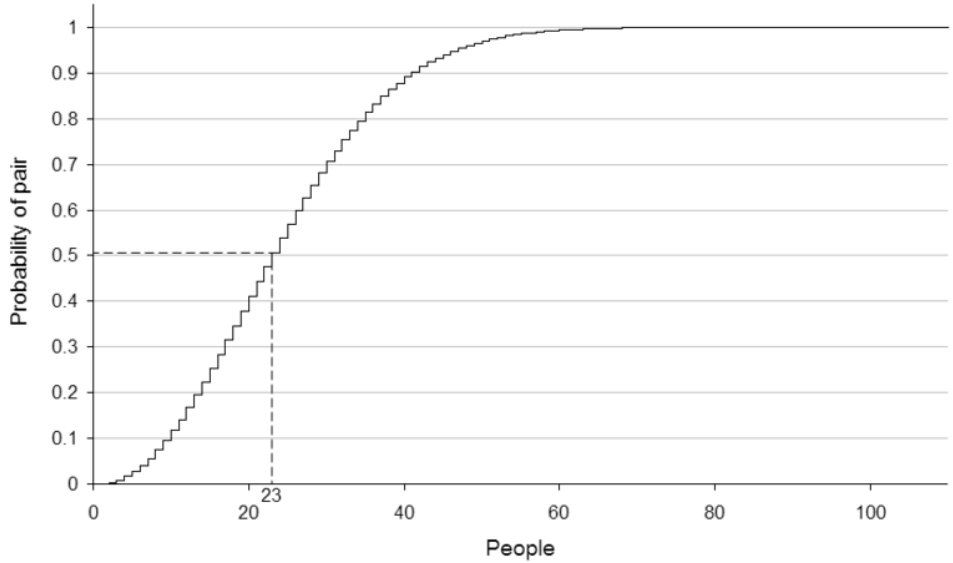
Attacchi agli algoritmi di hash / 2

- L'analisi della sicurezza degli algoritmi di hash si è concentrata soprattutto sulle tecniche di rilevamento delle collisioni.
- Se una funzione di hash ha n bit di output, esistono 2^n hash possibili. Per il principio dei buchi di colombaia, cercare una collisione richiede di valutare al massimo 2^n+1 messaggi
- Si potrebbe quindi pensare che, in media, si possa trovare una collisione dopo aver effettuato metà tentativi: $2^n/2 = 2^{n-1}$. SBAGLIATO!
- In realtà trovare una collisione è molto più semplice, mediante il cosiddetto *attacco del compleanno*

Il paradosso del compleanno

- Qual è la probabilità che in un gruppo di 23 persone ve ne siano almeno due che compiono gli anni nello stesso giorno?

- Contrariamente a quanto ci suggerisce l'intuito, la probabilità è leggermente superiore al 50%



Il paradosso del compleanno / 2

- Dimostrazione:
- Sia A l'evento "almeno due persone compiono gli anni nello stesso giorno" e $\bar{A}$ il suo negato ("nessuno compie gli anni nello stesso giorno")

- $P(A) = 1 - P(\bar{A})$
- $P(\bar{A}) = \underbrace{\frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{343}{365}}_{\text{23 volte}}$

$$= \frac{365!}{342!365^{23}} = 0.49270276$$

- $P(A) = 1 - P(\bar{A}) = 0.507297 \ (50.7297\%)$

Attacco del compleanno

- Quanti messaggi casuali bisogna analizzare affinché ci sia una probabilità di collisione del 50%? Il paradosso del compleanno ci dice che il numero di messaggi da considerare è più basso di quanto ci si aspetterebbe

- Emerge che in media la prima collisione è attesa dopo aver valutato solamente circa $2^{n/2}$ (ovvero $\sqrt{2^n}$) messaggi

- Molto meno dei 2^{n-1} messaggi che avevamo ipotizzato inizialmente

Dimostrazione

- Ipotesi: 2^n hash possibili, k messaggi analizzati. A = evento «tra i k messaggi ce ne sono almeno due con lo stesso hash»
- $P(A) = 1 - P(\bar{A})$
- $P(\bar{A}) = \frac{2^n-1}{2^n} \cdot \frac{2^n-2}{2^n} \cdot \ldots \cdot \frac{2^n-k}{2^n} = \left(1-\frac{1}{2^n}\right)\left(1-\frac{2}{2^n}\right)\ldots\left(1-\frac{k}{2^n}\right)$
- Approssimazione al primo ordine di Taylor per x piccoli:

  $$e^x \approx 1 + x$$

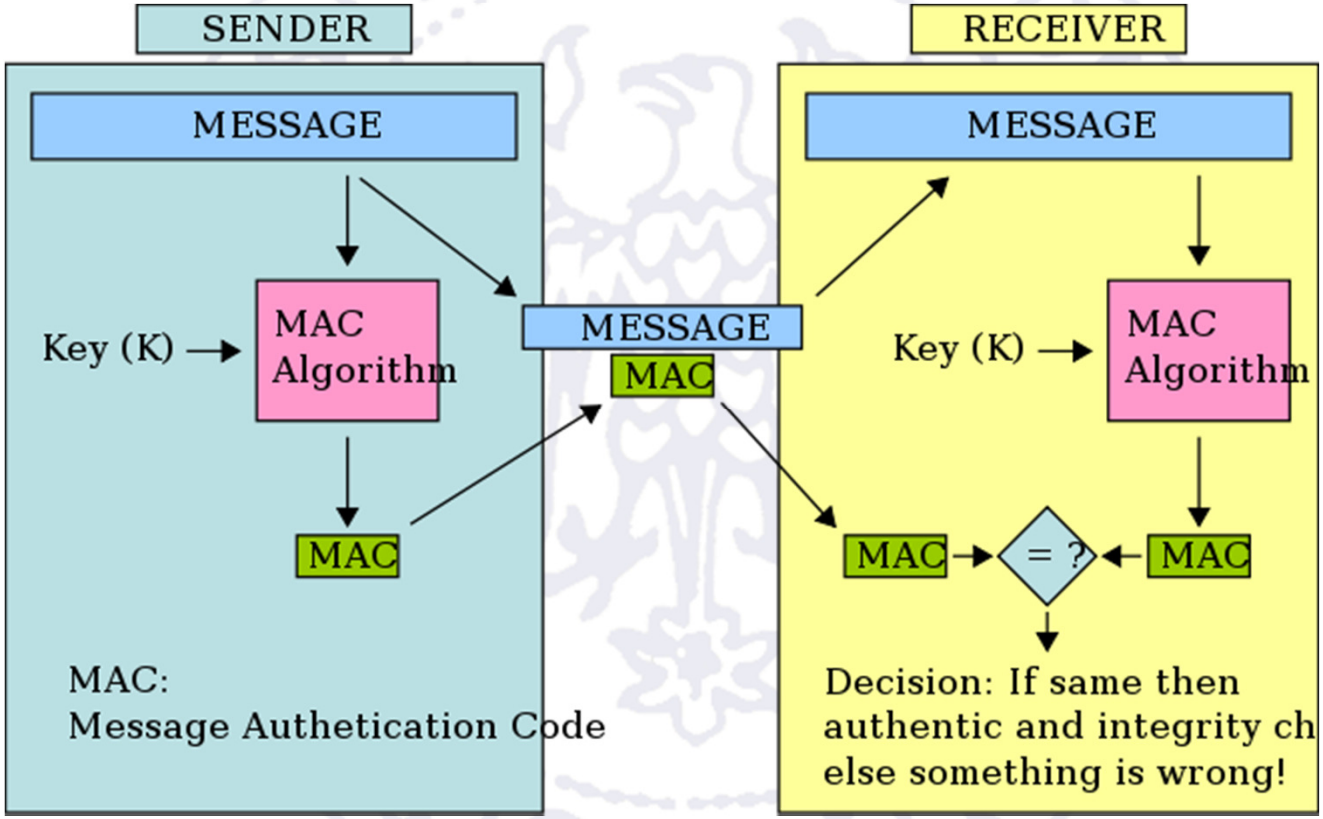
- $P(\bar{A}) \approx e^{-1/2^n} e^{-2/2^n} e^{-3/2^n} \ldots e^{-k/2^n} = e^{-\frac{k(k+1)}{2^{n+1}}} \approx e^{-k^2/2^{n+1}}$
- Volendo P(A) = 0.5 allora…
- $0.5 \approx 1 - e^{-k^2/2^{n+1}}$
- $k^2 \approx -2\ln(0.5)\, 2^n$
- $k \approx \sqrt{-2\ln(0.5)\, 2^n} = \sqrt{-2\ln(0.5)}\; 2^{\frac{n}{2}} \approx 1.17 \cdot 2^{\frac{n}{2}}$

Questo è il numero di messaggi da considerare affinché la probabilità di avere una collisione sia del 50%.
Il numero medio di messaggi da analizzare prima di trovare una collisione è leggermente diverso, 1.25 * 2^(n/2)

HMAC

- Un HMAC (Hash-based Message Authentication Code) non è altro che una funzione di hash che richiede anche una chiave segreta.
- Viene utilizzato per garantire l'integrità e l'autenticazione di un messaggio
- È molto simile alla firma digitale, ma in questo caso si usa una chiave simmetrica segreta anziché un protocollo a chiave pubblica/privata
- Autenticazione: il messaggio è stato generato da qualcuno che conosce la chiave. Non è una forma di autenticazione abbastanza forte da garantire la non ripudiabilità (sia mittente che destinatario conoscono la chiave → di fronte ad una terza parte entrambi possono ripudiare il messaggio e accusare l'altro di esserne l'autore)

HMAC - Schema di utilizzo



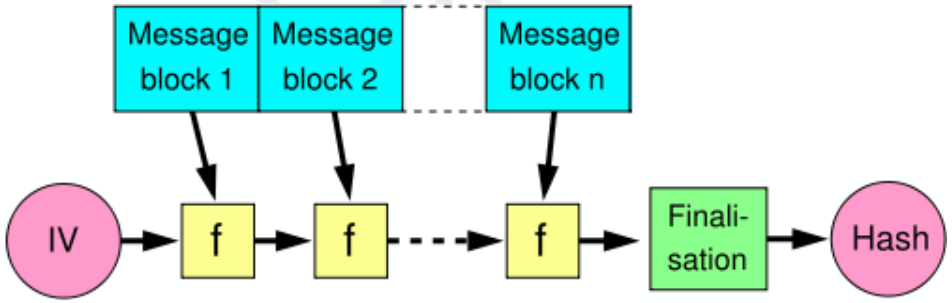
Fonte: http://en.wikipedia.org/wiki/Message_authentication_code

Costruire un HMAC da una funzione di hash

- Primo tentativo:

  $S(k,m) = H(k \,||\, m)$

- Pessima idea…



- Se conosco $H(k \,||\, m)$ per un certo messaggio m, posso calcolare facilmente $H(k \,||\, m \,||\, w)$ (basta aggiungere altri stadi nella struttura di Merkle–Damgård). Riesco quindi a calcolare il MAC per un messaggio nuovo senza avere la chiave.
- Versione sicura:

  $S(k,m) = H(k \,||\, H(k \,||\, m))$

HMAC + cifratura simmetrica

- Similmente a quanto visto per la cifratura a chiave pubblica, gli HMAC possono essere usati in congiunzione con un algoritmo di cifratura simmetrico per garantire contemporaneamente sia la riservatezza che l'integrità dei dati.

- Schema sempre sicuro:
  - Prima cifrare, poi applicare HMAC

- Si ottiene in questo modo la cosiddetta «cifratura autenticata», che garantisce sia la riservatezza (grazie alla cifratura) che l'integrità del messaggio (grazie al HMAC). Garantisce la sicurezza contro attaccanti attivi.