

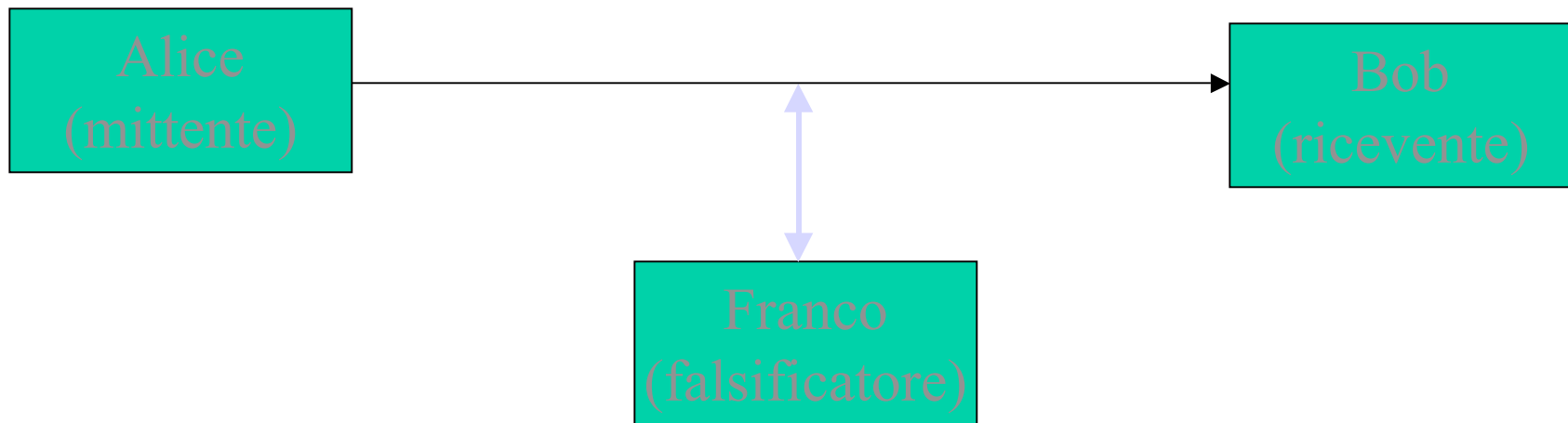
# Crittografia e sicurezza delle reti

Integrità dei dati e loro autenticità  
Message Authentication Codes (MAC)

Firma digitale

# Autenticazione messaggi

Garantire l'integrità dei messaggi anche in presenza di un avversario attivo che manda i suoi messaggi (corretti)



Nota: **Autenticazione** è ortogonale alla **segretezza**  
spesso un sistema deve garantire ambedue

# Algoritmo autenticazione

Chiave di autenticazione-  $k$ , messaggio  $m$

Algoritmo di autenticazione -  $A$  calcola  $A_k(m)$   
impronta (authentication tag) di  $m$

Spazio dei messaggi (stringhe binarie): Alice invia  
a Bob la coppia  $(m, A_k(m))$

L'algoritmo di autenticazione è anche detto **MAC**  
(Message Authentication Code,  $MAC_k(m)$   
denota  $A_k(m)$  )

# Algoritmo di verifica

Bob quando riceve il messaggio  $(m, A_k(m))$   
esegue un Algoritmo di verifica -  $V$  che accetta  
o rifiuta il messaggio:

calcola  $V_k$   $V_k(m, A_k(m)) = \text{accetta/rifiuta}$

La verifica si effettua calcolando l'autentica  
di  $m$  e confrontando con  $MAC_k(m)$

# Proprietà funzioni MAC

Requisito di sicurezza

avversario non può costruire una nuova coppia corretta  $(m, MAC_k(m))$  anche dopo aver visto  $(m_i, MAC_k(m_i))$  ( $i=1,2,\dots,n$ )

Output deve essere corto

Funzioni MAC non sono 1-a-1

# Firma Digitale

Analogo MAC con chiave pubblica

Uno schema di firma digitale include

- Una chiave privata  $d$
- Una chiave pubblica
- Un algoritmo di firma

Applicazioni

Commerciali, necessarie per e-commerce

Legali, valide in Italia

# Firma digitale vs. MAC

Supponi che  $A$  e  $B$  condividano una chiave segreta  $K$ .

Allora  $M, MAC_K(M)$  convince  $A$  che  $M$  è scritto da  $B$ .

Ma in caso di disputa

$A$  non può convincere un giudice che  $M, MAC_K(M)$  è stato inviato da  $B$ , poichè  $A$  potrebbe generarlo lei stessa

# MAC: Modello Avversario

Dati disponibili:

algoritmo MAC è noto

- Known plaintext

- Chosen plaintext

Nota: chosen plaintext MAC è realistico?

Scopo: Date n coppie legali

$(m_1, MAC_k(m_1)), \dots, (m_n, MAC_k(m_n))$

trova una nuova coppia legale  $(m, MAC_k(m))$

# Modello Avversario

L'avversario ha successo anche se il messaggio che trova non ha senso.

La ragione per questo è dovuta al fatto che spesso -  
in un contesto nuovo

- non è facile distinguere cosa ha significato e cosa no
- il ricevente reagisce a messaggi considerati autentici

# Modello Avversario Efficienza

Scopo avversario: Date  $n$  coppie legali

$(m_1, \text{MAC}_k(m_1)), \dots, (m_n, \text{MAC}_k(m_n))$

trovare una nuova coppia legale  $(m, \text{MAC}_k(m))$

efficientemente e con probabilità non trascurabile.

NOTA: Se  $n$  è grande abbastanza allora  $n$  coppie

$(m_i, \text{MAC}_k(m_i))$  determinano la chiave  $k$

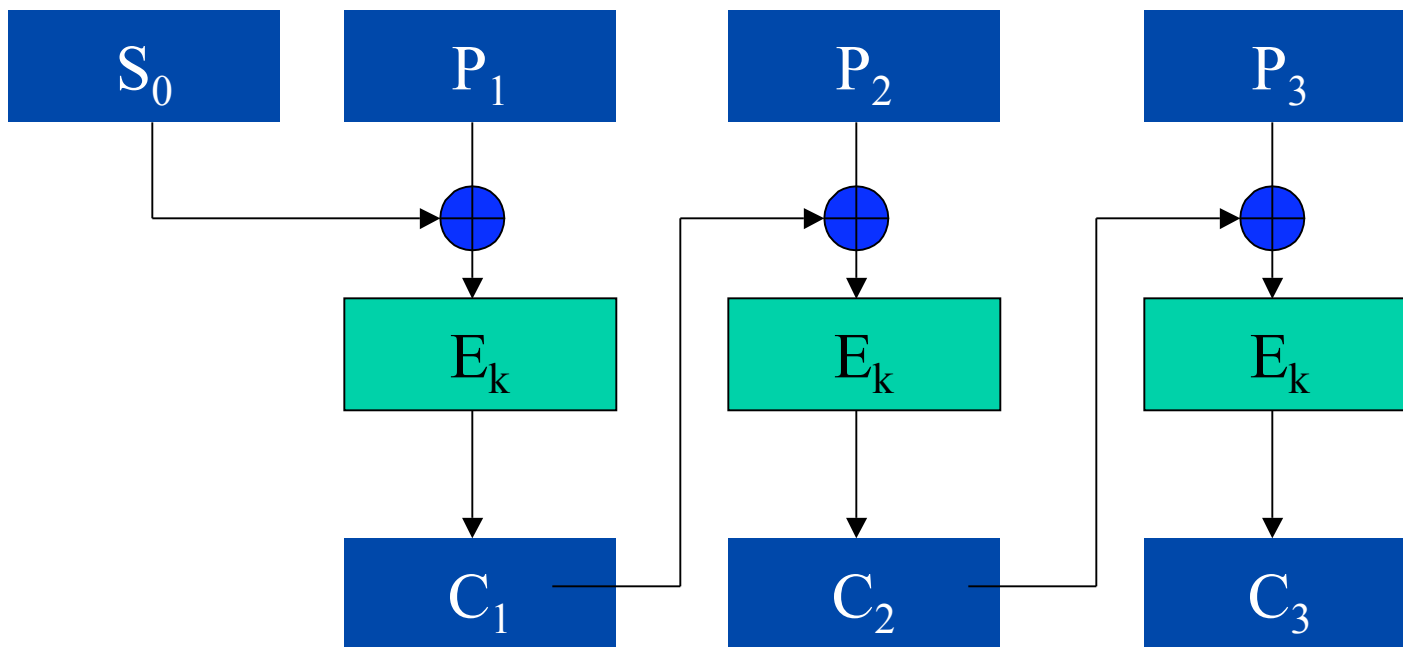
unicamente (con alta prob.).

Quindi si può eseguire una ricerca esaustiva provando tutti i  $k$  e verificare. Eseguire questo deve essere un compito computazionalmente difficile.

# MAC Usati in pratica

- MAC basato su modalità codifica CBC
  - usa codifica crittog. a blocchi
  - lento
- MAC basato su funzioni hash crittografiche
  - veloce
  - nessun problema esportazione

# Ricorda: CBC Mode Encryption (Cipher Block Chaining)



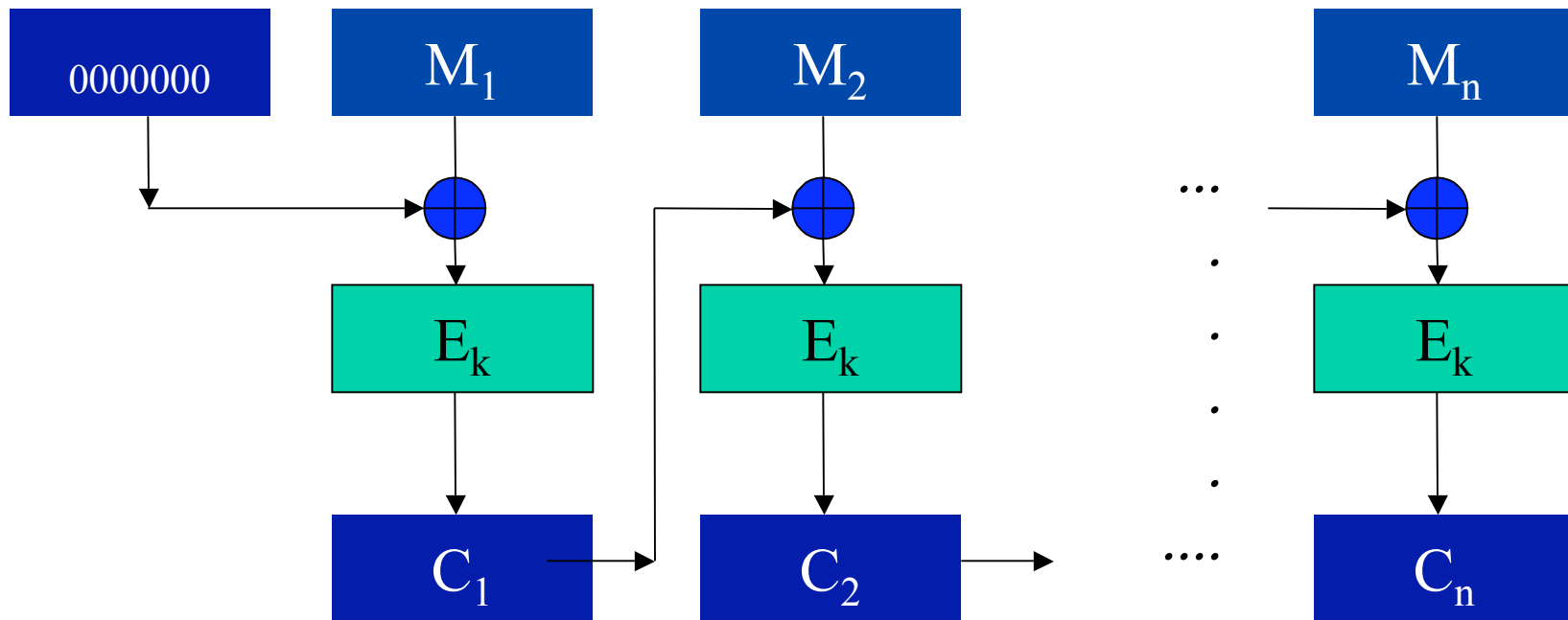
Il blocco codificato precedente è in XOR con il blocco corrente in chiaro prima della codifica

Si utilizza un vettore  $S_0$  di inizializzazione (seed - seme) del processo.  $S_0$  può essere trasmesso in chiaro

# MACs con CBC

Inizia con il seme di tutti zero.

dato un messaggio di  $n$  blocchi  $M_1, M_2, \dots, M_n$ , applica CBC (usando la chiave segreta  $k$ ).



- Scarta i primi  $n-1$  "cipertext" blocchi  $C_1, C_2, \dots$ , usa  $C_n$ .
- Invia  $M_1, M_2, \dots, M_n$  & il tag di autenticazione  $MAC_k(M) = C_n$ .

# Sicurezza di CBC MAC

Proposizione: Se  $E_k$  è una funzione pseudo casuale, allora CBC MAC è robusto

Funzione pseudo casuale funzione che appare casuale rispetto ad analisi dei risultati (usando algoritmi polinomiali)

Se CBC MAC può essere falsificato allora si trasforma l'algoritmo di falsificazione in uno che distingue efficientemente  $E_k$  da una funzione casuale.

# Segretezza & MAC

Dato un messaggio di  $n$  blocchi  $M_1, M_2, \dots, M_n$ , applica CBC (usando la chiave segreta  $k_1$ ) per produrre  $MAC_{k_1}(M)$ .

- Produci  $n$  blocchi di ciphertext  $C_1, C_2, \dots, C_n$  con una diversa chiave,  $k_2$ .
- Invia  $C_1, C_2, \dots, C_n$  & il tag di autenticazione  $MAC_{k_1}(M)$ .

# Funzioni Hash

Grande dominio e piccolo codominio

Esempio  $h: \{0,1,\dots,p^2\} \rightarrow \{0,1,\dots,p-1\}$

$$h(x) = ax + b \text{ mod } p$$

Usati molto per la ricerca (tavole hash)

Collisioni si risolvono in diversi modi-  
chaining, double hashing, etc.

# Utilizzo di funzioni hash per il calcolo del MAC

Obiettivo: calcola Mac usando

- funzione hash  $h$
- messaggio  $m$
- chiave segreta  $k$

Vogliamo che il MAC dipenda dalla chiave e  
dal messaggio

Esempi  $MAC_k(m) = h(k,m)$  o  $h(m,k)$  o  $h(k,m,k)$

oppure primi bit di  $h(k,m)$  o  $h(m,k)$

# Resistenza alle Collisioni

Una funzione hash  $h: D \rightarrow R$  è detta *weakly collision resistant* per  $x \in D$  se è difficile trovare  $x' \neq x$  tale che  $h(x') = h(x)$

Una funzione hash  $h: D \rightarrow R$  è detta *strongly collision resistant* se è difficile trovare  $x', x$  tale che  $x' \neq x$  ma  $h(x') = h(x)$

Teor. i) Strongly collision resistant non implica one-way

ii) One way non implica strongly collision resistant

# Resistenza alle Collisioni

i) *Strongly collision resistant non implica one-way*

$g$  collision res.  $h: \{0,1\}^* \rightarrow \{0,1\}^n$ . Sia  $h(x)$

$h(x) = 0 || x$  se  $|x| = n$ ,  $h(x) = 1 || g(x)$  altrimenti

- $h$  coll. resistant (perché lo è  $g$ )
- $h$  non è one way (si possono invertire metà input (quelli che iniziano con 0))

ii) *One way non implica strongly collision resistant*

$g$  one way: sia  $h(z || 0) = h(z || 1) = g(z) = y$

- facile trovare collisione su  $h$
- $h$  è one way (perché lo è  $g$ )

# Paradosso del compleanno

Se 23 persone sono scelte a caso la prob che due di esse abbiano compleanno lo stesso giorno è maggiore di 0.5

In generale, sia  $h:D \rightarrow R$  una qualunque funzione. Se si sceglie  $1.17|R|^{1/2}$  elementi di  $D$  a caso, la probabilità che due di essi diano lo stesso valore della funzione è maggiore di 0.5.

# Funzioni Hash Crittografiche

Funzioni hash Crittografiche sono funzioni hash che sono strongly collision resistant.

- Non c'è chiave segreta.
- Devono essere veloci da calcolare ma deve essere difficile trovare collisioni (equivalente a problema computazionalmente difficile - es P diverso da NP).
- Di solito definite usando una funzione di compressione con dominio di  $n$  bits (es. 512) e codominio di  $m$  bits (es. 160),  $m < n$ .

# Funzioni Hash usate

Famiglia MD ("message digest")

MD-2, MD-4, MD-5 (128 bits)

SHA-1 (secure hash standard, 160 bits

[www.itl.nist.gov/fipspubs/fip180-1.htm](http://www.itl.nist.gov/fipspubs/fip180-1.htm))

SHA 256, 384 e 512 (standard proposti,  
impronte più lunghe)

Sono state trovate collisioni per MD4; non  
sono note collisioni per MD5 o SHA, SHA-1

# Secure Hash Algorithm (SHA-1)

SHA progettato da NIST & NSA in 1993,  
rivisto nel 1995 - SHA-1

- US standard per uso con firma digitale
  - standard FIPS 180-1 1995, Internet RFC3174
  - nb. l'algoritmo è SHA, lo standard è SHS
  - hash 160-bit (paradosso compleanno  $2^{80}$ )
  - modifica di un vecchio algoritmo MD4
- **Febbraio 2005: SHA-1 rotto!!!**

# Revisione SHA

Febbraio 05: SHA-1 rotto con  $2^{69}$  operaz.

Agosto 2005:  $2^{63}$  operazioni

- un attacco migliore di  $2^{64}$  è importante teoricamente: sono state fatti calcoli (distirbuiti in internet) che richiedono  $2^{64}$  operaz.
- lavoro crittoanalisti abbassa ancora
- comunque, la scrittura di un programma che esegue in pratica attacco, non è ovvia al momento

# SHA-1

1. Padding messaggio per avere lunghezza  $448 \bmod 512$
2. Appendi stringa di 64-bit (lunghezza messaggio)
3. Inizializza buffer di 5 parole (160-bit) (A,B,C,D,E)  
(67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. Processa messaggio in blocchi di 16-parole (512-bit)
  - espandi un blocco di 16 parole in un blocco di 80 parole (usa exor di insiemi diversi di blocchi e shift per generare altri blocchi)
  - aggiorna il buffer (A,B,C,D,E): applica 4 rounds ognuno di 20 operazioni su blocco di 20 parole
  - somma (mod  $2^{32}$ ) input e buffer e ottieni nuovo buffer
  - output: buffer finale di 5 parole

# SHA-1 - Compressione round $t$

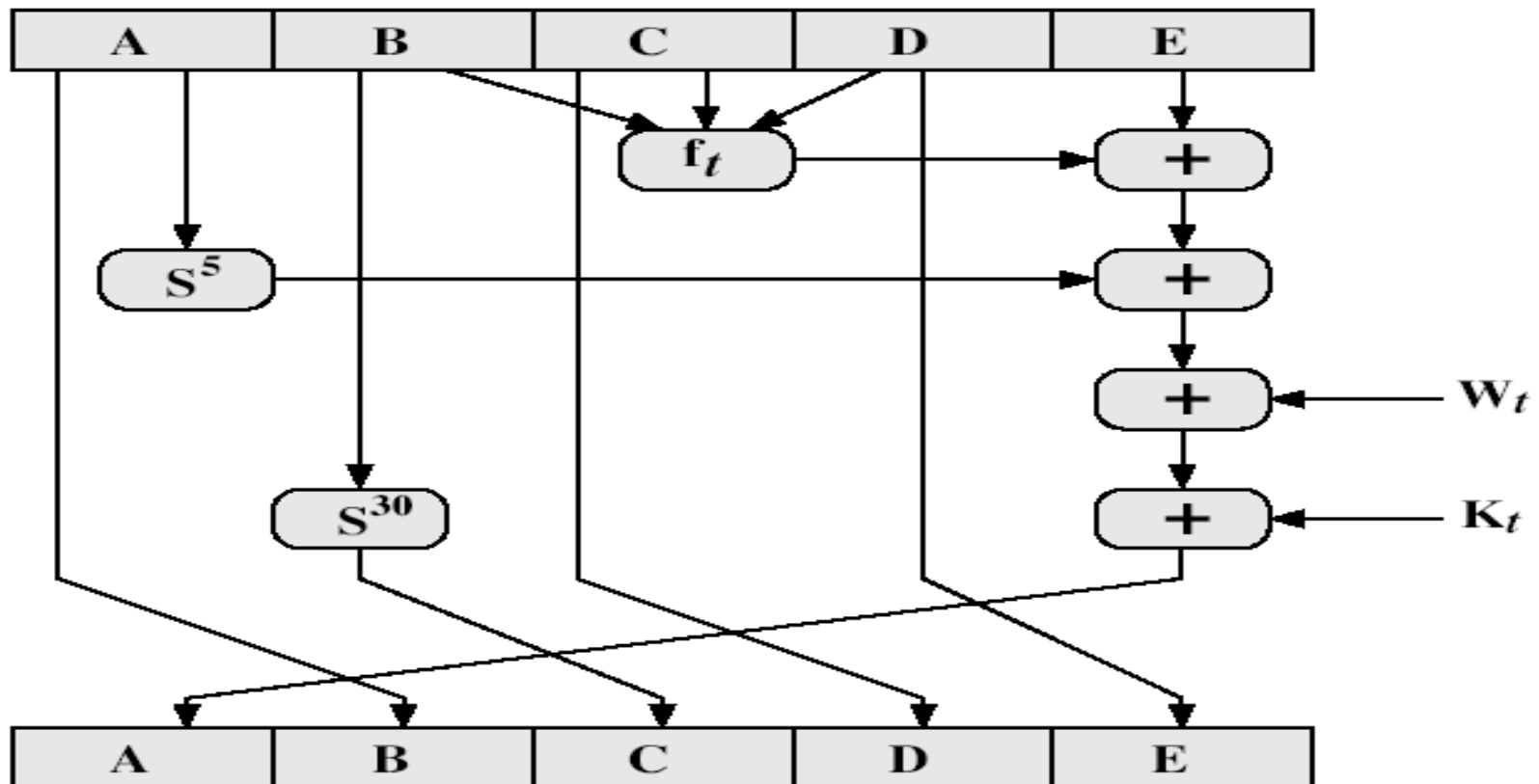
80 iterazioni: ciascuna modifica buffer di 5 parole  $(A, B, C, D, E)$ :

$(A, B, C, D, E) \leftarrow$

$(E + f(t, B, C, D) + (A \ll 5) + W_t + K_t), A, (B \ll 30), C, D)$

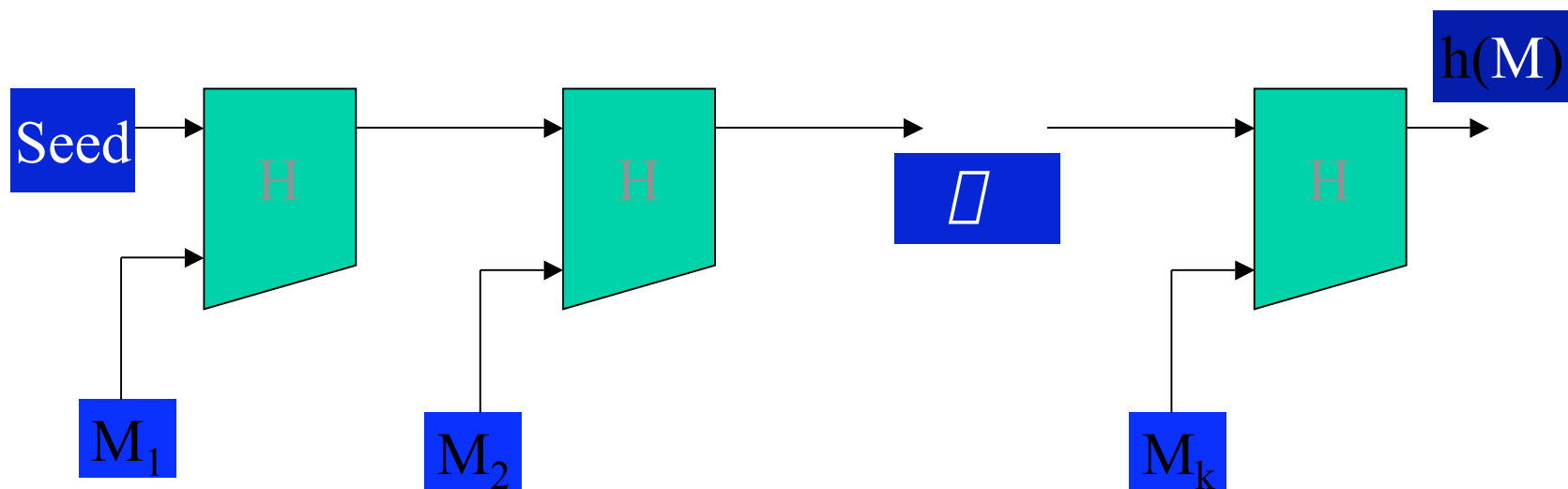
- $t$  è il no. del round,  $\ll$  è oper. shift a sin
- $f(t, B, C, D)$  è funzione nonlineare del round
- $W_t$  è blocco ottenuto dall'espansione in 80 par.
- $K_t$  è costante ottenuta dal seno

# SHA-1 - Compression round $t$



# Calcolo per stringhe lunghe

*Problema: dato blocco come fare hash di messaggi lunghi?*



$H: D \rightarrow R$  (dimensione fissata,  $\{0,1\}^n$  e  $\{0,1\}^m$  )

# Calcolo per stringhe lunghe

Il seme iniziale (seed) è di solito costante

Si usa ,padding (includendo la lunghezza del messaggio originale) per ottenere messaggi di lunghezza multipla di  $n$ .

**Teorema:** se la funzione hash base  $H$  è resistente alle collisioni, anche la sua estensione lo è.

# Calcolo per stringhe lunghe

La lunghezza del messaggio di Input è arbitraria, in pratica si assume che sia limitata da  $2^{64}$  bit, che è grande abbastanza per tutti gli scopi pratici.

La lunghezza del blocco di input è di solito 512 bits.

La lunghezza del blocco di output è di almeno 160 bits per prevenire attacchi basati sul paradosso del compleanno.

# Hash e MAC

Data funzione hash per messaggi lunghi come calcolare MAC

- $MAC_k(m)=h(k,m)$  : problema grave
- $MAC_k(m)=h(m,k)$  : problema piccolo (se trovi due messaggi che collidono, allora trovi due messaggi con stesso MAC)
- $h(k,m,k)$ : OK (simile a HMAC)
- uso primi bit (ad esempio la prima metà) di  $h(k,m)$  o  $h(m,k)$ : OK (attaccante non può verificare correttezza)

# HMAC

Proposto nel 1996 da [Bellare Canetti  
Krawczyk] Internet engineering task force  
RFC 2104; FIPS Standard

Scopi:

- calcolare MAC basandosi su una funzione hash (considerata come una scatola nera),  
senza degrado significativo
- input messaggio  $m$ , chiave  $k$  funzione hash  $h$
- usare le chiavi in modo semplice
- risultato notevole di analisi formale della sicurezza di un meccanismo di autenticazione

# HMAC

Riceve in input un messaggio  $m$ , una chiave  $k$  e una funzione hash  $h$

Calcola MAC come:

$$\text{HMAC}_k(m, h) = h(k \oplus \text{opad}, h(k \oplus \text{ipad}, m))$$

la chiave  $k$  è riempita con zeri a sin. per ottenere  $\text{lungh. } k = \text{lungh. di un blocco}$

$\text{opad: } 01011010, \text{ ipad: } 00110110, \text{ ripetuti } |k|/8 \text{ volte}$

**Teorema [BCK]:** HMAC può essere falsificato se e solo se può esserlo la funzione hash (si trovano collisioni).

# HMAC - Attacchi

Attaccante vuole trovare due messaggi  $m$  e  $m'$   
t.c.  $\text{HMAC}_k(m,h) = \text{HMAC}_k(m',h)$ ; conosce  $IV, h$  ma  
non conosce  $k$

Si applica il paradosso del compleanno (blocco lungo  $n$   
implica collisione con  $2^{n/2+1}$  tentativi)? No:

- attaccante non conosce  $k$  e quindi non può generare messaggi autentici;
- deve quindi osservare  $2^{n/2+1}$  messaggi generati con la stessa chiave (es.  $n=128$  almeno  $2^{64+1}$  bit)

Nota paradosso del compleanno non si applica anche a  $h(k,m,k)$

# HMAC in Pratica

SSL / TLS

WTLS

IPSec: AH, ESP

# RSA e Data integrity: OAEP

Problema: inviare dati con RSA non modificabili

codif.  $c = \text{RSA}[M \parallel 0^{k_1} \text{ exor } G(r)] \parallel$

$[H(\text{RSA}[M \parallel 0^{k_1} \text{ exor } G(r)]) \text{ exor } r]$

$k_0, k_1$  noti;  $G, H$  funzioni hash, note;  $r$  stringa casuale di  $k_0$  bit (scelta dal mittente);

decod.  $cd = s \parallel t$  (con  $t$  di  $k_0$  bit);  $u = t \text{ exor } H(s)$ ;  $v = s \text{ exor } G(u)$ ; accetta se  $v = m \parallel 0^{k_1}$ ; altrimenti rifiuta

$r$  casuale implica che OAEP è robusto anche nel caso di attacco chosen ciphertext (assumendo  $H$  e  $G$  buone funzioni hash)

Nota: RSA non fornisce identificazione del mittente

# Esercizi

mostrare che il seguente metodo non è un buon metodo per calcolare l'impronta di un testo indipendentemente dalla funzione usata:

$$E_k(M_1 \oplus M_2 \oplus \dots \oplus M_n) \quad \text{oppure}$$

$$E_k(M_1) \oplus M_2 \oplus \dots \oplus M_n$$

# Proprietà firma digitale

- Firma facile da verificare
- Difficile da falsificare
- Hanno valore legale (convincere una terza persona - esempio un giudice).

# Diffie and Hellman (76)

## "New Directions in Cryptography"

Sia  $D_A$  la chiave segreta di Alice e  $E_A$  la sua chiave pubblica

Per firmare  $M$ , Alice calcola  $y = D_A(M)$  e invia  $(M, y)$  a Bob.

Per verificare la firma di Alice, Bob calcola  $x = E_A(y)$  e verifica che  $x = M$ .

Intuizione: Solo Alice può calcolare  $y = D_A(M)$ , quindi la falsificazione è computazionalmente intrattabile

## Problemi con il Paradigma di DH "Puro"

Facile falsificare messaggi casuali anche senza conoscere  $D_A$ :

**Bob** sceglie  $R$  e calcola  $S = E_A(R)$ .

La coppia  $(S, R)$  è una firma valida di **Alice** del "messaggio"  $S$ .

Lo schema è soggetto a **existential forgery**.

"Cosa implica" ?

## Problemi con il Paradigma di DH "Puro"

Considera RSA. La proprietà moltiplicativa implica che

$$D_A(M_1M_2) = D_A(M_1) D_A(M_2).$$

Se  $M_2 = \text{"Devo a BOB \$20"}$  e  $M_1 = \text{"100"}$   
allora per qualche codifica

$$M_1M_2 = \text{"Devo a BOB \$2000" ...}$$

# Soluzione Standard : Hash

Sia  $D_A$  la chiave segreta di Alice e  $E_A$  la sua chiave pubblica

Per firmare  $M$ , Alice calcola  $y = H(M)$  e  $z = D_A(y)$  e invia  $(M, z)$  a Bob.

Per verificare la firma di Alice, Bob calcola  $y = E_A(z)$  e verifica che  $x = H(M)$

La funzione  $H$  deve essere **collision resistant**, per cui è difficile trovare  $M'$  con  $H(M) = H(M')$ .

# Meccanismo Generale

**Generazione** di chiavi pubbliche e private (casuali).

**Firma** (o deterministica o con algoritmi probabilistici)

**Verifiche** (accetta/rifiuta) - deterministico.

# Schemi usati nella pratica

- RSA
- El-Gamal Signature Scheme (85)
- DSS (digital signature standard, utilizzato da NIST in 94 basato sullo schema El-Gamal).

# Firma con RSA

FIRMA: codifica con la chiave segreta hash del messaggio

Solo chi conosce la chiave segreta può firmare

Tutti possono utilizzare la chiave pubblica e verificare la firma

Il metodo si applica a qualunque sistema crittografico a chiave pubblica

# Firma RSA: Public-Key Crypto. Standard (PKCS)

FIRMA: codifica con la chiave segreta hash del messaggio

PKCS-1: standard per formato messaggi (byte)

0||1||almeno 8 byte FF in base 16|| 0|| alg. hash,||  
codifica con chiave segreta hash(M)

(M rappresenta i dati da firmare)

- primo byte 0 implica che il messaggio è minore n
- secondo byte (1) indica firma
- byte di 11111111 implicano messaggio grande;
- la specifica del tipo di funzione hash usata aumenta sicurezza

# El-Gamal Signature Scheme

DL: problema del logaritmo discreto (esponenziale è considerata funzione one way)

**Generazione** Scegli primo  $p$  di 1024 bits tale che DL in  $Z_p^*$  è intrattabile.

Sia  $g$  un generatore di  $Z_p^*$ .

Scegli  $x$  in  $[2, p-2]$  a caso.

Calcola  $y = g^x \bmod p$ .

- Chiave pubblica :  $p, g, y$ .
- Chiave Privata :  $x$ .

# El-Gamal Signature Scheme

## Firma $M$

Hash: Sia  $m=H(M)$ .

1. Scegli  $k$  in  $[1, p-2]$  relativamente primo a  $p-1$   
scelto a caso

2. Calcola  $r=g^k \bmod p$ .

3. Calcola  $s=(m-rx)k^{-1} \bmod (p-1)$

(nota:  $s=(m-rx)k^{-1} \bmod (p-1)$  e le scelte fatte implicano  $sk+rx=m$ )

4. Firma  $(M)= r || s$

Nota che il calcolo di  $r$  non dipende dal messaggio,  
quindi può essere fatto prima

# El-Gamal Signature Scheme

Chi verifica la firma conosce  $M, r, s, p, g, y$

Nota:  $k$  è ignoto al verificatore

Verifica: Calcola  $m=H(M)$

Accetta se  $0 < r < p$  e  $y^r r^s = g^m \pmod{p}$ .

altrimenti rifiuta.

Infatti

- $s = (m - rx)k^{-1} \pmod{p-1}$  (definizione di  $s$ ) implica  $sk + rx = m$ .
- $r = g^k$  implica  $r^s = g^{ks}$  ; inoltre  $y = g^x$  implica  $y^r = g^{rx}$ ,
- quindi  $y^r r^s = g^m$ .

# Digital Signature Standard (DSS)

NIST, FIPS PUB 186

DSS usa SHA come funzione hash e  
DSA come metodo di firma

DSA simile a El Gamal

Più veloce per chi firma piuttosto che  
per chi verifica (firma con smart  
cards)

# Digital Signature Algorithm (DSA)

Trova  $p$  - primo (512 bit),  $q$  - primo (160 bit),  
tale che  $p-1 = 0 \pmod q$ ,

$g = 1^{(1/q)} \pmod p$  ( $g$  generat. gruppo ordine  $q$  in  $Z_p$ )

Chiave Privata :  $a, 1 \leq a \leq q-1$  (a random, casuale).

Chiave Pubblica:  $(p, q, g, y = g^a \pmod p)$

Firma del messaggio  $M$  (con SHA come funz. hash)

Scegli a caso  $k, 1 \leq k \leq p-1, k$  segreto!!

Parte I:  $r = ((g^k \pmod p) \pmod q)$

Parte II:  $s = ((\text{SHA}(M) + a r) / k) \pmod q$

*Nota: Parte I non dipende da  $M$  (preprocessing); Parte II veloce da calcolare*

# Digital Signature - Verifica

$$e1 = \text{SHA}(M) / (\text{PARTE II}) \bmod q$$

$$e2 = (\text{PART I}) / (\text{PARTE II}) \bmod q$$

Accetta se  $(\alpha^{e1} y^{e2} \bmod p) \bmod q = \text{PARTE I}$

Prova di correttezza: dalla definizione segue che :

1.  $\text{SHA}(M) = (-a(\text{PARTE I}) + k(\text{PARTE II})) \bmod q$  quindi

$$\text{SHA}(M) / (\text{PARTE II}) + a(\text{PARTE I}) / (\text{PARTE II}) = k \bmod q$$

2.  $y = \alpha^a \bmod p$  implica  $\alpha^{e1} y^{e2} \bmod p =$

$$\alpha^{\text{SHA}(M) / (\text{PART II}) + a(\text{PART I}) / (\text{PART II})} \bmod p = \alpha^{k+cq} \bmod p$$

$$= \alpha^k \bmod p \text{ (poiché } \alpha^q = 1).$$

3. Eseguendo  $\bmod q$  si ottiene

$$(\alpha^{e1} y^{e2} \bmod p) \bmod q = (\alpha^k \bmod p) \bmod q = \text{PARTE I}$$

# DSS: sicurezza

Non si rivela la chiave segreta ( $a$ ) e non si può falsificare senza la chiave segreta

Usa un numero segreto per messaggio ( $k$ )!

- Non esistono duplicati con la stessa firma
- Se  $k$  è noto allora si può calcolare  $a$
- Due messaggi con lo stesso  $k$  possono rivelare  $k$ , e quindi  $a$

Esiste un canale subliminale: si può inserire nella firma un messaggio segreto. Attacco: un implementatore disonesto può inserire informazioni sulla chiave segreta usata

# DSS: efficienza

Trovare i primi  $p$  e  $q$  tale che  $p-1 = 0 \pmod q$  è lungo

Il metodo permette di utilizzare gli stessi  $p$  e  $q$  per più utenti:  $p$  e  $q$  pubblici

Problemi di sicurezza:

- mi devo fidare di chi crea per me la tripla  $(p,q,g)$
- attaccante rompe una tripla  $p,q,g$  ne rompo tante

DSS più lento nella verifica della firma di RSA

DSS e RSA stessa velocità in firma (DSS più veloce se si permette preprocessing)

Richiede uso numeri casuali che non sono sempre facili da generare

# Firma con DSS verso RSA

DSS: (+) più veloce nella firma di RSA (con preproc meglio per smart card)

(+ / - ?) firma usa numero casuale (+)

MA ci sono problemi nella implementazione:

- numeri casuali veri richiedono hw speciale (no smartcard);
- pseudo rando generator richiede memoria (no smart card)
- numero casuale che dipende dal messaggio non permette preprocessing e rallenta la firma

(+) standard senza royalties

RSA: (+) studiato e attaccato più a lungo e quindi considerato più sicuro

(+) più veloce nella verifica della firma

# Esercizi

DSA cosa succede se la PARTE II della firma è 0?  
verificare che con DSA se uso lo stesso valore di k  
due volte allora si può falsificare la firma