

# metodi crittografici

primitive crittografiche

# sommario

- richiami di crittografia e applicazioni
  - hash crittografici
  - crittografia simmetrica
  - crittografia asimmetrica
- attacchi e contromisure
  - birthday
  - rainbow
  - key rollover
  - generatori di numeri casuali

# richiami di crittografia e applicazioni

# funzioni hash crittografiche

dette anche *message digests* o *one-way transformations*

- l'hash di un messaggio (cioè della stringa)  $m$  è denotato  $h(m)$ 
  - $h(m)$  è “apparentemente casuale” e di lunghezza costante
- proprietà:
  - per ogni  $m$  il calcolo di  $h(m)$  è efficiente
    - tempo lineare nella lunghezza di  $m$
  - dato  $H$ , è computazionalmente difficile trovare  $m$  tale che  $H=h(m)$ 
    - pre-image resistance, non invertibilità, one-way
  - data  $h$ , è computazionalmente difficile trovare  $m \neq m'$  tale che  $h(m)=h(m')$ 
    - strong collision resistance
  - data  $h$  e  $m$ , è computazionalmente difficile trovare  $m \neq m'$  tale che  $h(m)=h(m')$ 
    - weak collision resistance o second pre-image resistance

# hash: algoritmi famosi

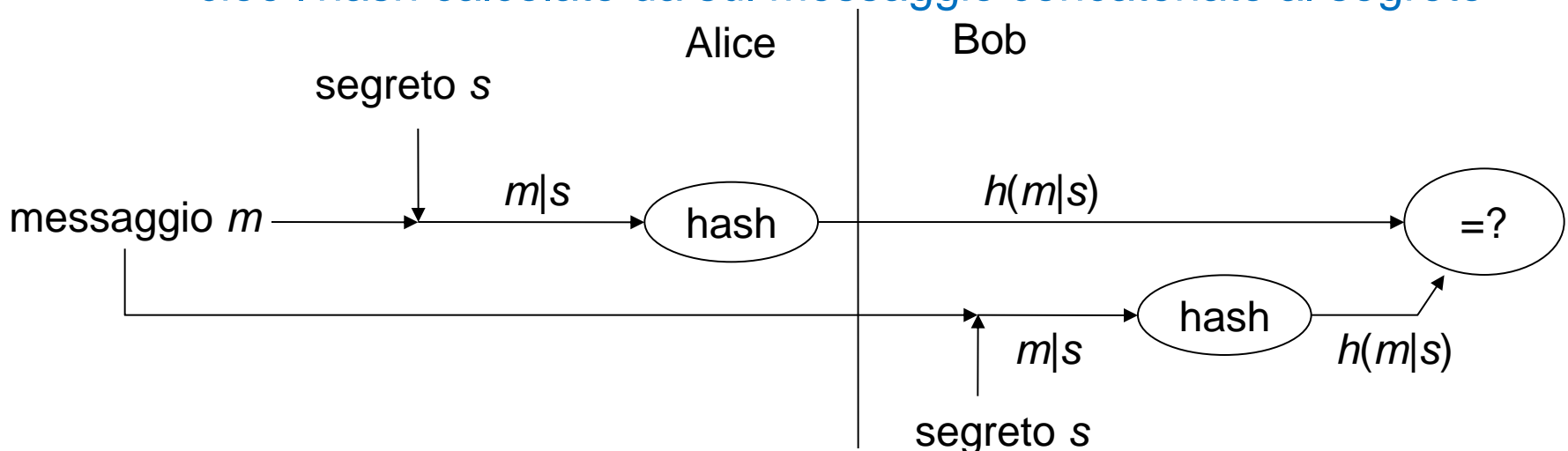
- MD2, MD4, MD5 (output: 128 bits, Rivest)
  - vulnerabili (collisioni trovate)
- ripemd160 (output: 160 bits, sviluppato in un progetto europeo)
  - ok
- SHA-1 (output: 160 bits)
  - vulnerabile (collisioni trovate), ma ok per gran parte delle applicazioni
- **SHA-2** (output: 224-512 bits, 6 varianti)
  - ok
- **SHA-3** (output: 224-512 bits, 4+2 varianti)
  - ok
- vedi anche  
[https://en.wikipedia.org/wiki/SHA-3#Comparison\\_of\\_SHA\\_functions](https://en.wikipedia.org/wiki/SHA-3#Comparison_of_SHA_functions)

# hash crittografici: applicazioni

- password hashing
  - anziché memorizzare la password in chiaro si può memorizzare l'hash
  - la conoscenza del db permette comunque di fare un attacco off-line molto più vantaggioso rispetto a quello on-line
- message digest (riassunto del messaggio)
  - è una stringa di lunghezza fissa (limitata) che identifica il messaggio (cioè messaggi diversi → digest diversi)
  - utile per verificare/memorizzare pochi bytes anziché l'intero messaggio
    - efficienza della firma digitale con chiave asimmetrica
    - verifica di integrità di file negli HIDS
    - verifica di integrità di file scaricati
    - sincronizzazione di file efficiente via rete (es. Dropbox)
    - reti peer-to-peer (ciascun file è identificato dal suo hash)
    - ecc.

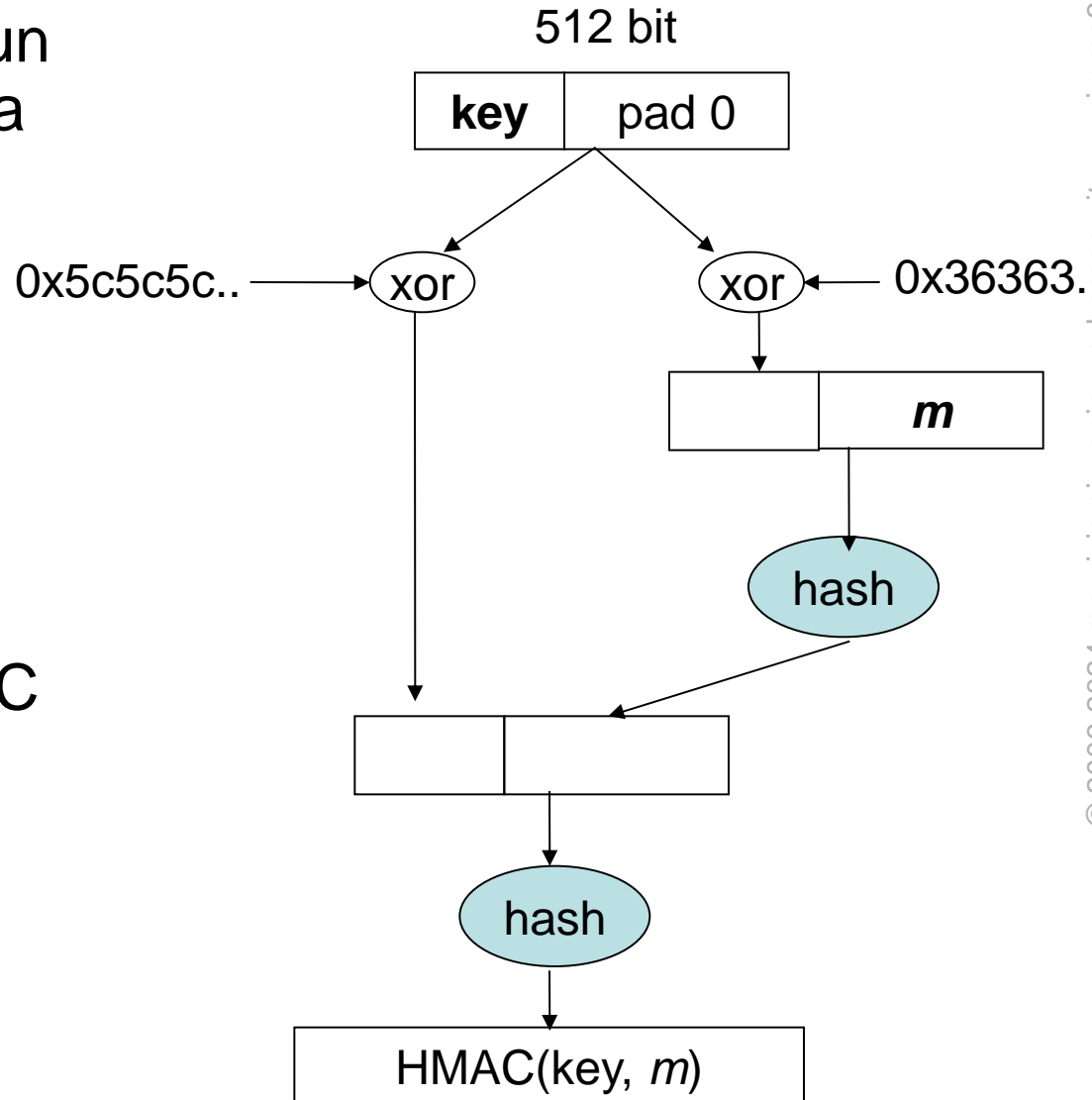
# hash applicazioni: MAC (MIC)

- un *MAC* (*message authentication code* o *MIC message integrity code*) è un codice che associato al messaggio assicura **l'integrità del messaggio e dell'origine**
  - è una firma senza la proprietà di non ripudio
- si può generare un MAC per mezzo di una funzione hash crittografica
- supponiamo che Alice e Bob conoscano un segreto  $s$  condiviso (*shared secret*)
- il MAC di un messaggio  $m$  è  $h(m|s)$ 
  - cioè l'hash calcolato da sul messaggio concatenato al segreto



# HMAC(key,m)

- standard per creare un MAC di  $m$  a partire da una funzione hash qualsiasi e da una chiave
- la lunghezza del risultato è dipende dalla funzione hash scelta
- si dimostra che HMAC è sicuro quanto la funzione hash usata



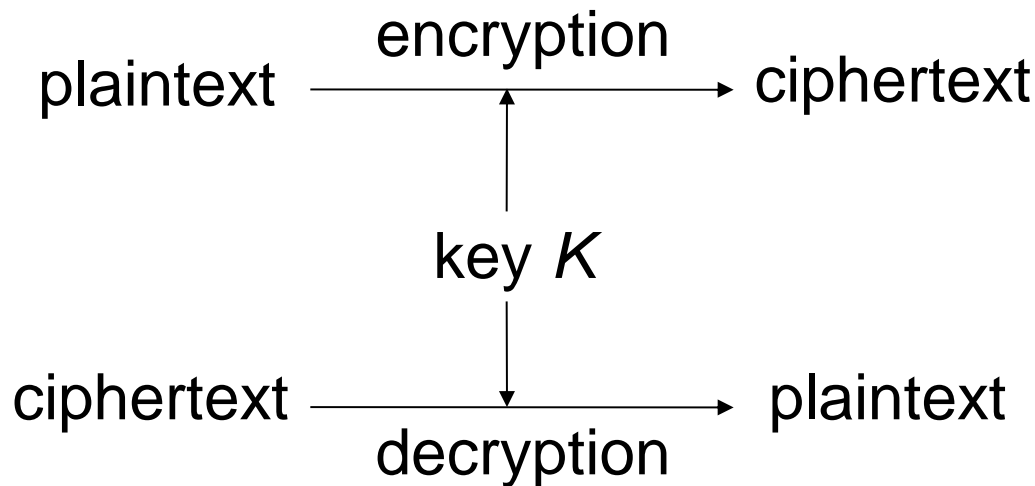


# hash applicazioni: strong authentication

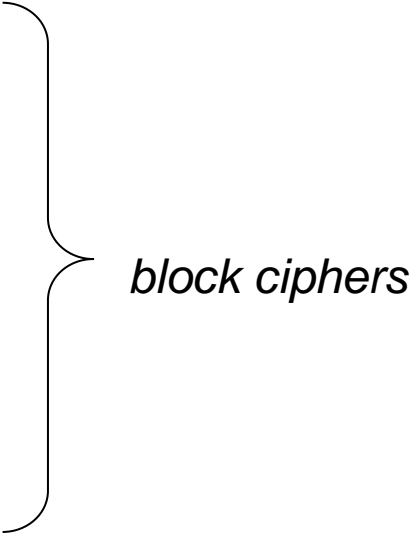
- **strong authentication:** chi si autentica prova che conosce un segreto  $k$  senza rivelarlo
- l'implementazione con hash sfrutta il concetto di MAC
- Bob sceglie a caso un  $m$  (*challenge*), Bob sa che Alice è veramente chi dice di essere se  $\text{MAC}(k,m)$  calcolato da Alice coincide quello calcolato da lui

# cirttografia simmetrica

- impiega una sola chiave  $K$  che è un segreto condiviso da chi usa il canale crittografico
- la notazione  $K\{m\}$  indica che  $m$  (plaintext) è trasformato crittograficamente (in ciphertext) mediante la chiave  $K$ 
  - $K\{m\}$  è lungo circa quanto  $m$
  - $K\{m\}$  è “apparentemente casuale” cioè la probabilità di avere un dato valore per ciascun bit è  $\frac{1}{2}$  ed (idealmente) indipendente dai valori assunti dagli altri bit. Dalla teoria dell’informazione tali stringhe hanno il *massimo valore di informazione* (nessuna ridondanza) per cui sono **non comprimibili**
- la stessa chiave  $K$  è usata per decifrare il messaggio



# c. simmetrica: algoritmi famosi

- DES (1977, 56-bit key)
    - **insicuro**
  - IDEA (1991, 128-bit key)
    - **brevettato, poco efficiente, simile a DES, sospetto**
  - 3DES (2x56-bit key, )
    - **applicazione tripla di DES (EDE), poco efficiente**
  - AES (standardizzato nel 2001, 128, 196, 256-bit key)
    - **NIST**
    - **deriva da rijndael (1999)**
    - **standard FIPS01**
  - RC4 (pubblicato nel 2001, 1-256-bytes key)
    - **Rivest**
    - **appartiene alla categoria degli *stream cipher* (o *one-time-pad*)**
    - **stream di bit random xor'ed con  $m$** 
      - la sequenza è generata dalla chiave che viene usata come *seed* del generatore
    - **efficiente e semplice (10-15 linee di codice)**
  - blowfish, RC5, twofish, CAST-128 (block ciphers)
  - ChaCha20-Poly1305 (stream cipher, with authentication, TLS 1.3, HTTP 3)
- 
- block ciphers*

# c. simmetrica - applicazioni

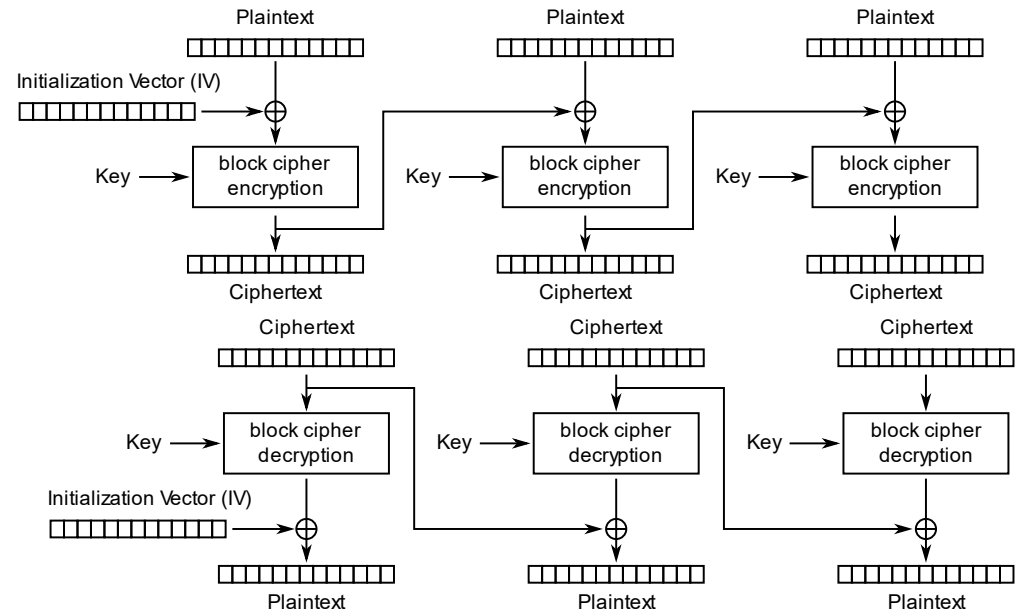
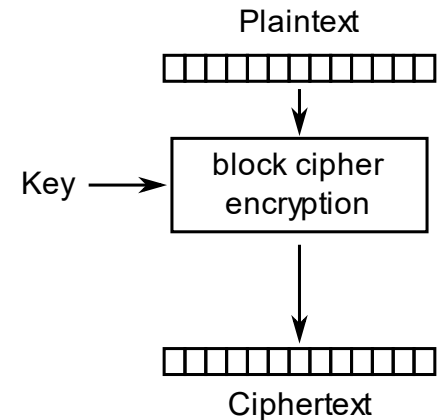
- trasmissione confidenziale su canale non fidato
  - la chiave deve essere trasferita su canale sicuro
  - nasce il problema della distribuzione sicura delle chiavi
- memorizzazione confidenziale su media non fidato
  - filesystem cifrati
- strong authentication
  - Bob sceglie a caso un challenge  $m$ , e chiede ad Alice di cifrarlo con la chiave condivisa

# integrity vs. confidentiality

- stream ciphers do not guarantee integrity
  - those that perform xor with a random bit string R
- e.g.,
  - an attacker does not know the key (and R)
  - but knows the plaintext sent: “no”
  - using xor properties it can transform the encrypted version of “no” in the encrypted version of “yes”
  - $E\{\text{no}\} = R \oplus \text{no}$
  - $E\{\text{no}\} \oplus \text{no} \oplus \text{yes} = R \oplus \text{yes} = E\{\text{yes}\}$
- in general you cannot ask integrity to encryption algorithms
  - unless declared explicitly

# block cipher mode of operation

- block ciphers just encrypt a “block” of bits
- to encrypt a stream of bits
  - encrypt each block independently (ECB)
    - several drawbacks
  - ...or use *chaining*
- Cipher Block Chaining (CBC)
  - note the introduction of the *initialization vector*
  - note that decryption can be parallelized



# other modes

- certain modes make a stream cipher out of a block cipher
  - e.g., CFB, OFB, CTR, etc.
  - essentially, they use a block cipher to build a generator of a random bit stream to be xor'ed with the plaintext
- certain modes perform *authenticated encryption* (AE, i.e., provably secure w.r.t. integrity)
  - single or double pass
  - may ask/provide Additional Data (AEAD)
  - e.g., Galois Counter Mode (GCM)
- certain modes self-generates the IV
  - e.g., Syntetic Initialization Vector
- for encrypted filesystem special purpose modes may be used

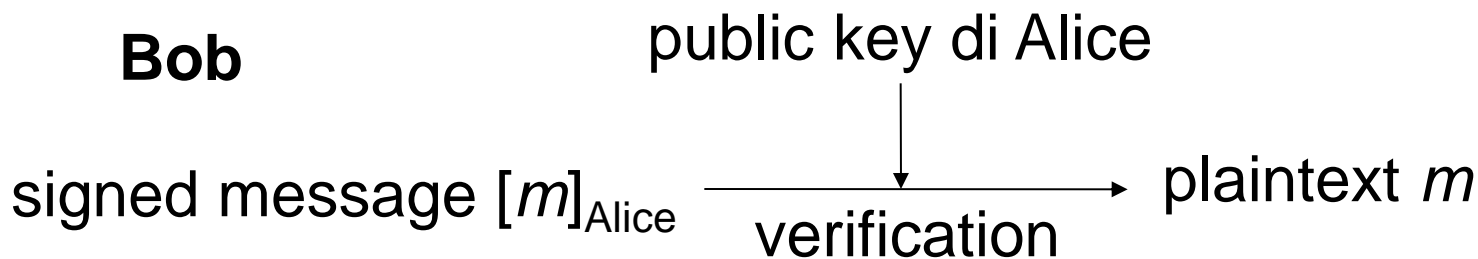
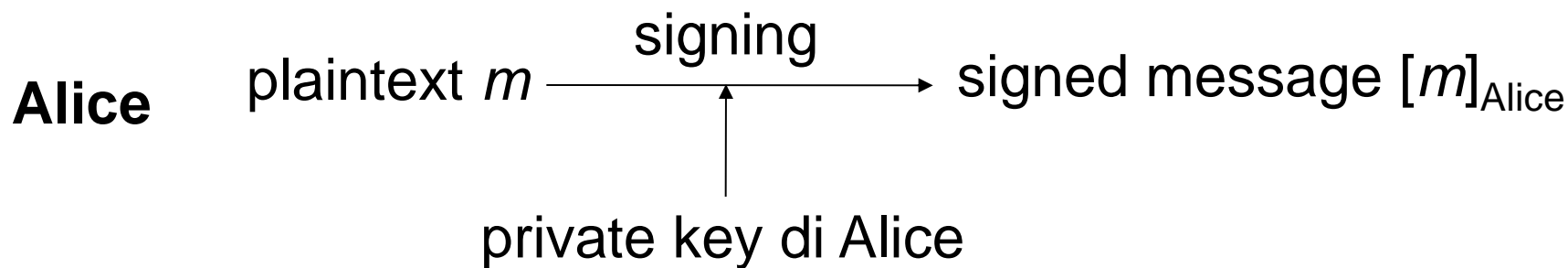
# crittografia asimmetrica (o a chiave pubblica)

- impiega due chiavi una *privata* non divulgata e una *pubblica* nota a tutti
  - tipicamente la coppia di chiavi è associata ad un solo soggetto
  - nessun problema di distribuzione delle chiavi
- il testo cifrato con una delle due chiavi può essere decifrato solo con l'altra chiave
- il risultato della cifratura è
  - lungo circa quanto l'input
  - “apparentemente casuale”
- inefficiente rispetto alle tecniche a chiave simmetrica



# crittografia asimmetrica: *firma*

- la notazione  $[m]_{\text{Alice}}$  indica che  $m$  è cifrato da Alice con la sua chiave privata (*firma*)
  - la crittografia asimmetrica è inefficiente
  - $m$  non può essere molto lungo



# crittografia asimmetrica: *firma*

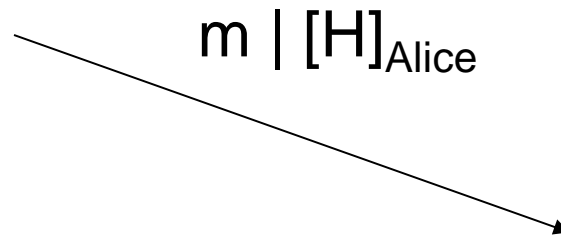
- per efficienza si firma un hash del messaggio  $m$  ottenendo  $[h(m)]_{\text{Alice}}$

## Alice

plaintext  $m$

$H=h(m)$

$[H]_{\text{Alice}}$



## Bob

$m \rightarrow h(m)=H'$

$[H]_{\text{Alice}} \rightarrow H$  (decifra)

verifica se  $H=H'$

- la sicurezza si basa sulla proprietà **strong collision resistance** di  $h(m)$
- alle volte abbreviamo la notazione  $m \mid [h(m)]_{\text{Alice}}$  con  $[m]_{\text{Alice}}$

# c. asimmetrica applicazioni: firma digitale (per integrità)

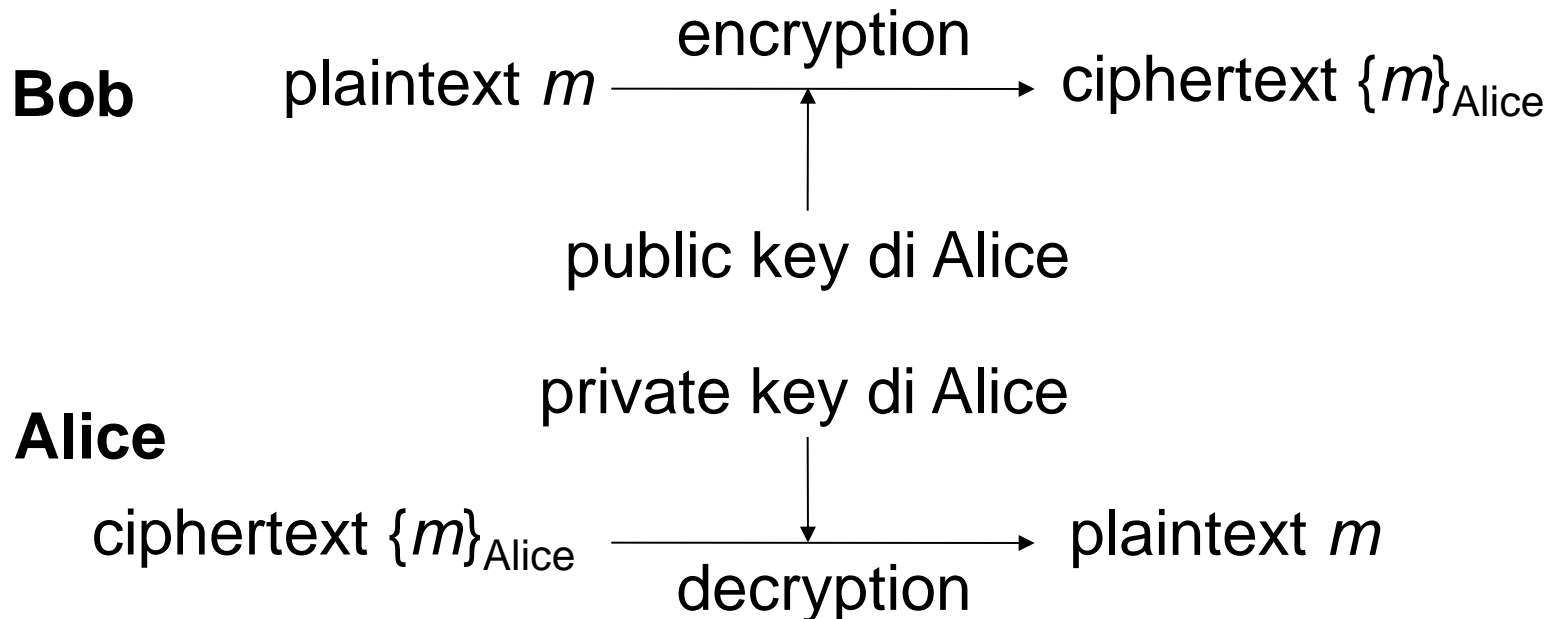
- garantisce...
  - autenticità (integrità della sorgente)
  - integrità (del messaggio)
  - **non ripudio**
- la tecnica del MAC con **segreto condiviso non garantisce il non ripudio**
  - Bob può creare  $MAC(m)$  esattamente come Alice, Bob sa che Alice è l'autore ma non può mostrarlo come prova a nessuno
  - nella firma digitale Bob non può creare la firma perché non conosce la chiave privata di Alice

## c. asimmetrica applicazioni: autenticazione

- Bob chiede ad Alice di firmare un challenge
- la chiave pubblica di Alice deve essere associata ad Alice in maniera inequivocabile
  - certificati

# crittografia asimmetrica: *cifratura*

- la notazione  $\{m\}_{Alice}$  indica che  $m$  è *cifrato* da Bob con la chiave pubblica di Alice (*cifratura*)
- la crittografia asimmetrica è inefficiente
  - *$m$  non può essere molto lungo*



# c. asimmetrica applicazioni

- trasmissione confidenziale su canale non fidato
  - **Inefficiente**
    - non usato direttamente per messaggi lunghi
  - usata molto per **distribuire chiavi simmetriche**
- problema della associazione tra chiave pubblica e soggetto
- memorizzazione confidenziale su media non fidato
  - cifratura di chiavi simmetriche usate per cifrare files
  - per ciascun utente  $A$  che ha diritto di accedere al file cifrato con la chiave  $K$ , viene memorizzato  $\{K\}_A$  assieme al file

# c. asimmetrica: algoritmi famosi

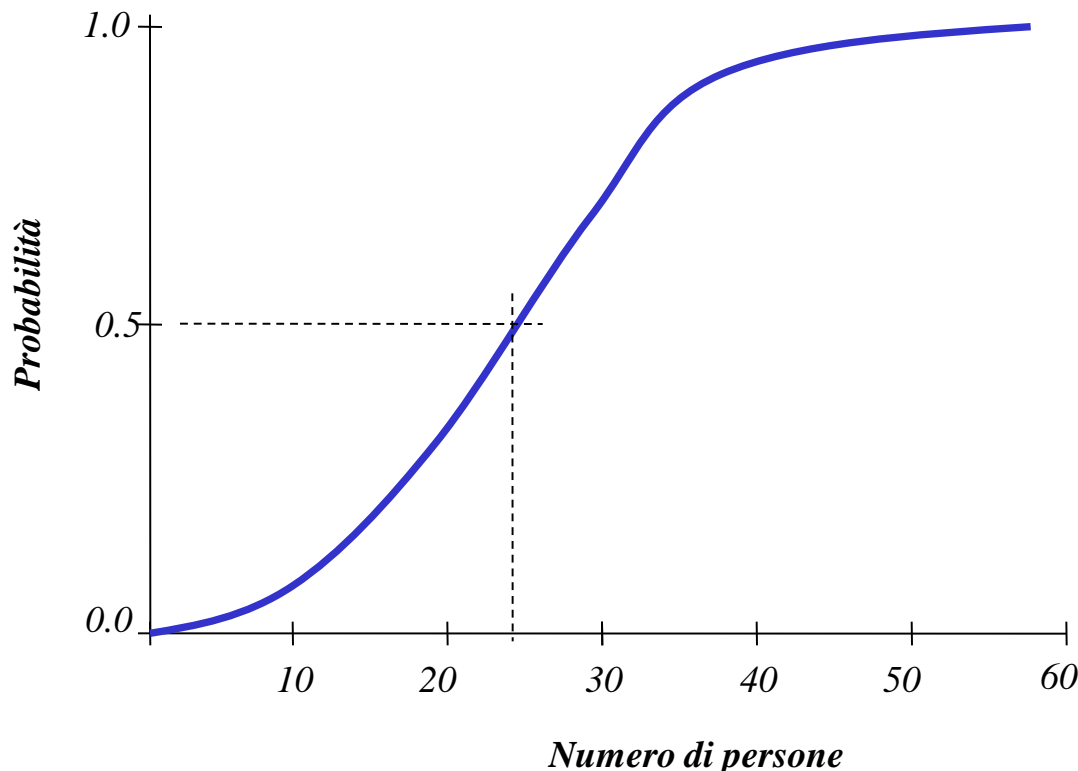
- Diffie Hellman
  - solo scambio di shared secret
- RSA
  - criptazione, firma, scambio di shared secret
- ElGamal, DSA (digital signature algorithm)
  - firma, derivato da diffie-hellman
- DSS (digital signature standard NIST, basato su ElGamal)
  - Firma
- Ed25519, variante di DSA, più efficiente

# attacchi e contromisure



# hash: birthday attacks

- serve a trovare una collisione (attacco alla firma digitale)
- paradosso del compleanno
- la probabilità che in un gruppo di  $N$  persone ne esistano almeno due che sono nate lo stesso giorno aumenta velocemente con  $N$



# hash: birthday attacks

- il compleanno è distribuito uniformemente come il valore di hash
  - persone → messaggi
  - data di compleanno → valori di hash
- si può dimostrare che...
  - in uno spazio degli hash di cardinalità  $N$
  - cercando tra  $1.2 \sqrt{N}$  messaggi si ha alta probabilità di trovare una coppia  $m_1 \neq m_2$  tale che  $h(m_1) = h(m_2)$

# attacchi birthday e firma

Vogliamo trovare due messaggi che dicano cose opposte ma abbiano lo stesso hash

- consideriamo una famiglia di testi che sia abbastanza vasta
- almeno  $1.2 \sqrt{N}$  elementi

```
{Egr. |Spett.} direttore,  
la ringrazio {del suo interessamento|della sua proposta}.  
Relativamente {a questa|ad essa} {ho|abbiamo}  
{deciso|preso la decisione} di non {acquistare|comprare}  
le {azioni|quote azionarie} {della|relative alla}  
securebank.com.
```

```
{Distinti saluti|cordiali saluti|cordialità}
```

```
hash: ??????????????????????
```

Se il numero di varianti è abbastanza alto ho buone probabilità di trovare due messaggi con significato opposto e stesso hash.

# hash: brute force

- serve a invertire l'hash
- su messaggi brevi  $p$  (es. passwords)
- si crea un db che contiene “tutte” le coppie  $(p, h(p))$
- si indicizza per  $h(p)$
- richiede uno spazio enorme

# rainbow tables

- come attacco brute force ma...
- compromesso tra tempo e spazio
  - l'idea è di perdere un po' di tempo pur di guadagnare molto spazio nel db
  - rappresentazione implicita di un gran numero di coppie (password, hash)
- funzione di riduzione
  - $r$ : hashes  $\rightarrow$  passwords
  - di fatto un'altra funzione di hash arbitraria dallo spazio degli hash a quello delle password
- rainbow chain
  - $p_1 \rightarrow h_1=h(p) \rightarrow p_2=r(h(p)) \rightarrow h_2=h(r(h(p))) \rightarrow \dots \rightarrow h_n$
  - la chain associa ad un insieme di passwords  $p_1 \dots p_n$  un solo hash  $h_n$

# rainbow tables

- rainbow table
  - la tabella memorizza  $(p_1, h_n)$
- query nella rainbow table
  - se l'hash  $g_1$  si trova tra gli  $h_n$  la password è  $p_n$  (calcolabile da  $p_1$ ), altrimenti...
  - si cerca tra gli  $h_n$   $g_2 = h(r(g_1))$ 
    - se c'è la password è al penultimo posto della catena
  - e poi  $g_3 = h(r(g_2))$  ecc...
  - se c'è la password è al terzultimo posto della catena
  - la catena è comunque nota a partire dalla password
- rispetto a brute force: divido lo spazio per  $n$ , moltiplico il tempo per  $n$
- db generati random
  - probabilità alta di trovare una password nel db (es  $>0.9$ )
- <http://www.antsight.com/zsl/rainbowcrack/rcracktutorial.htm>

# salting

- invece di fare hash della password si fa hash di un derivato randomizzato
- NO:  $h(p)$ , SI:  $h(p,s)$ 
  - es.  $s$  stringa random,  $s$  è detto “sale”
  - $p$  ed  $s$  concatenati
- si memorizza la coppia  $\langle s, h(p,s) \rangle$
- $p,s$  è in uno spazio molto più ampio di  $p$
- quindi  $h(p,s)$  è molto più difficile da invertire di  $h(p)$ 
  - indipendentemente dall’attacco: rainbow, brute force, ecc.

# c. simmetrica: attacchi

in ordine di complessità

- ciphertext only
  - è l'attacco più ovvio, tipicamente inevitabile, gli algoritmi devono assolutamente resistere a questo tipo di attacco
  - l'attaccante deve essere in grado di riconoscere quando ha successo
    - deve conoscere la struttura del plaintext (lingua inglese, http, ecc.)
- known plaintext
  - su alcune coppie <ciphertext, plaintext>
- chosen plaintext
  - come known plaintext ma il plaintext può essere scelto dall'attaccante
  - i protocolli che usano tecniche crittografiche dovrebbero cercare di evitare che questo attacco sia possibile.



# c. simmetrica: lunghezza della chiave e del messaggio

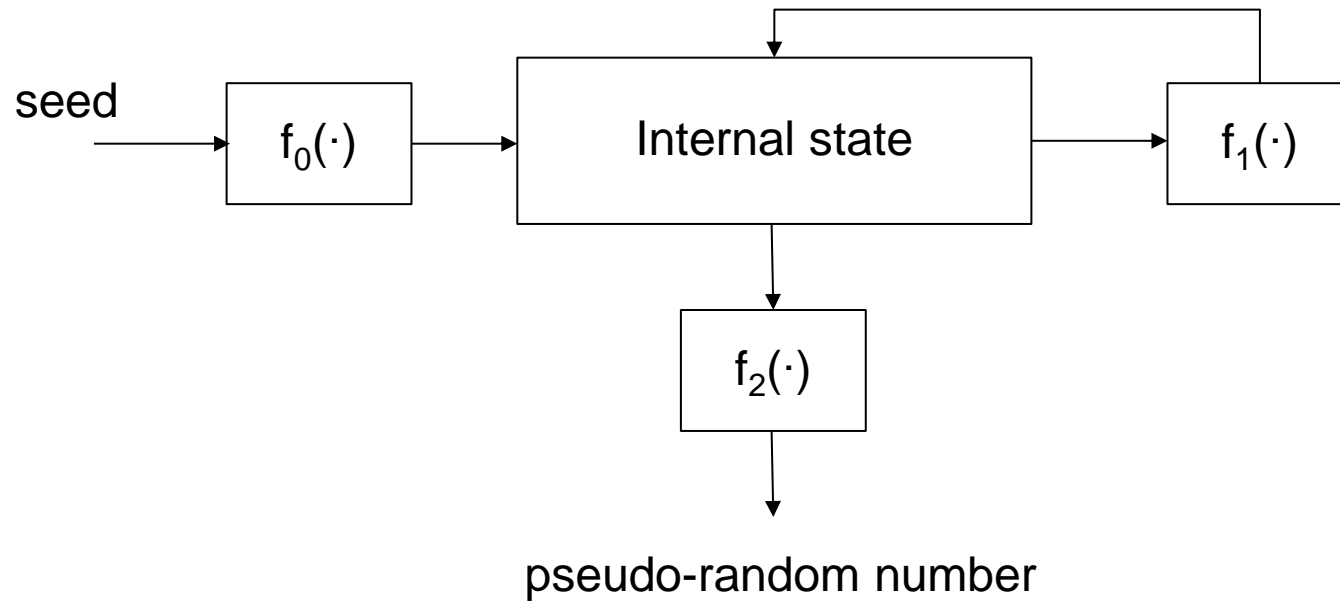
- gli attacchi sono semplici quanto più  $m$  è lungo rispetto a  $K$
- la chiave migliore è quella lunga quanto  $m$ 
  - la tecnica viene detta one-time-pad
  - una chiave infinita può essere generata pseudo-casualmente
    - il problema è creare numeri pseudo-casuali “buoni”
    - es. algoritmo RC4 (stream cipher)
- la chiave si deteriora con l’uso e col tempo
  - tanti messaggi cifrati facilitano gli attacchi
  - più passa il tempo più aumenta la probabilità che
    - la chiave sia stata pubblicata
    - la chiave sia stata scoperta mediante crittoanalisi
- ogni tanto dobbiamo cambiare la chiave
  - generazione casuale

# generatori di numeri pseudo-casuali

## pseudo random number generators (PRNG)

- i generatori di numeri casuali sono fondamentali per l'applicazione sicura dei metodi crittografici
  - ad esempio sono usati per generare chiavi crittografiche
- i generatori pseudo-casuali sono degli automi a stati finiti deterministici
  - ad ogni passo si pubblica **parte** dello stato
  - il numero degli stati è finito e quindi il sistema è periodico
    - il periodo deve essere abbastanza lungo! (facile)
  - l'evoluzione è determinata dallo stato iniziale (*seed*)

# an simple PRNG schema



# PRNG and security

problems are essentially two

- bad PRNG
  - e.g. non cryptographic PRNG
- bad use of a good PRNG
  - e.g. bad seed

# PRNG and security

- typical attack objective:  
**prediction of the next random number**
  - a predictable random number usually makes any security application vulnerable

in fact,

- sometimes random numbers are published
  - e.g. authentication challenges
- sometimes they should be kept secret
  - e.g. encryption keys
- info about seed or published random numbers may allow the attacker to obtain information on the internal state of the PRNG and hence help to predict the next random number

# seed

- il seed deve essere il più possibile casuale
- errori tipici:
  - seed pubblicato (perchè usato in altro contesto)
  - seed da fonte pubblica (real time clock)
  - spazio del seed troppo piccolo (es. 16 bit, oppure uptime granularità del secondo)
  - sorgente casuale non abbastanza casuale (es. uptime)
    - il comando nmap spesso permette di conoscere l'uptime di un calcolatore remoto

# numeri casuali veri

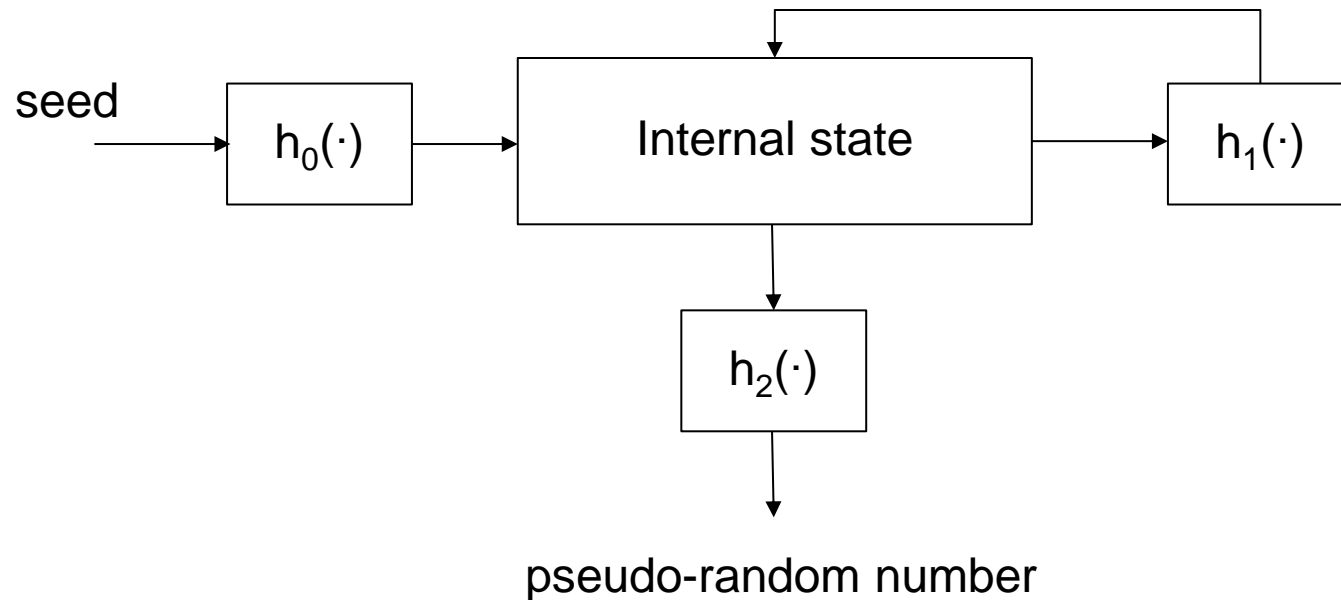
- necessari per ottenere buoni seed per PRNGs
  - keystrokes timing
  - mouse movements
  - process scheduling events
  - ecc.
- librerie crittografiche danno generatori di numeri casuali di questo tipo
  - **ATTENZIONE:** la generazione è molto lenta  
**Non possono essere usati al posto dei PRNG!!!!**
  - latenza non prevedibile
    - ad esempio dipende se l'utente interagisce
  - **bloccante finché non si vede abbastanza "entropia"**
- generatori basati su rumore termico
  - necessita di hardware dedicato (TPM)
  - non costoso ma non è detto che sia presente nei pc

# PRNG e necessità crittografiche

- un buon seed non è sufficiente
- per applicazioni crittografiche è essenziale la non predicibilità del prossimo numero casuale a partire dai precedenti
  - per altre applicazioni basta avere ad esempio distribuzione uniforme dei valori prodotti
  - **le librerie standard non soddisfano il requisito di non predicibilità**
    - hanno altri obiettivi, es. efficienza + distribuzione uniforme



# an simple cryptographic PRNG schema



- $h_0(\cdot)$ ,  $h_1(\cdot)$ ,  $h_2(\cdot)$  are cryptographic hash functions
- it is non-predictable since  $h_2(\cdot)$  is one-way.

# (P)RNG recap

	<b>suited for cryptography</b>	<b>usability</b>	<b>blocking</b>	<b>speed depends on...</b>
Scientific Pseudo-RNG	<b>NO</b>	easy	no	the CPU speed
Cryptographic Pseudo-RNG	yes	easy	no	the CPU speed
True RNG (software)	yes (seeds only)	it requires a source of entropy	yes (long)	the entropy source
True hardware RNG (e.g., TPM)	yes (mostly seeds)	it requires a specific hardware	yes (short)	the speed of the hardware, usually not comparable with CPU speed