Aspetti crittografici dell'online banking

Prof. Massimiliano Sala

Università degli Studi di Trento, Lab di Matematica Industriale e Crittografia

Trento, 27 Febbraio 2012

1. IL PROBLEMA CRITTOGRAFICO
2. CIFRATURA SIMMETRICA
3. CIFRATURA ASIMMETRICA
4. FUNZIONI HASH
5. NETWORK SECURITY
6. TWO FACTORS AUTHENTICATION
7. RANDOMICITÁ
8. PRODOTTI E VENDORS

IL PROBLEMA CRITTOGRAFICO

Introduzione

Obiettivo crittografico classico:

Consentire a due utenti di comunicare su un canale potenzialmente insicuro, senza permettere ad una terza persona di comprendere il contenuto dei messaggi.

Impedire ad una terza persona di impersonare uno dei due utenti.


Alice


Bob


Eve

Algoritmi crittografici e robustezza matematica

Comprende 4 aspetti problematici:

a) algoritmo di cifratura (simmetrica)

b) algoritmo di scambio chiavi (cifratura asimmetrica)

c) autenticazione ed integrità (funzione hash)

d) generazione delle chiavi (randomicità).

La nostra analisi si concentrerà principalmente su queste quattro caratteristiche.

CIFRATURA SIMMETRICA

Crittografia a chiave privata

Esiste un'unica chiave privata K, sia per l'operazione di cifratura che per l'operazione di decifratura.

Possiamo pensare l'insieme dei messaggi da trasmettere come l'insieme delle stringhe binarie lunghe n.
Formalmente $\mathcal{M} = \mathbb{F}^n$, dove $\mathbb{F} = \{0, 1\}$.

Scelta una chiave K una funzione di crittazione è una mappa $f_k : \mathcal{M} \to \mathcal{M}$ (bigettiva) che trasforma plaintext in ciphertext, in modo che a ciascun plaintext corrisponda un solo ciphertext e viceversa.

Cifrario a blocchi

Solitamente $\mathcal{K} = \mathbb{F}^k$.

Un cifrario a blocchi, φ, è un insieme di funzioni di crittazione

$$\{ \varphi_K \}_{K \in \mathcal{K}}$$

dove ogni chiave K nello spazio delle chiavi $\mathcal{K}$ definisce l'azione di una funzione di crittazione.

ATTENZIONE: sappiamo calcolare

$$x \longrightarrow \varphi_K(x)$$

ma non "sappiamo" φ_K.

Cifrario a blocchi

Un buon cifrario non deve permettere di capire φ_K da

$$\{ (x, \varphi_K(x) \}_{x \in X}$$

per un piccolo $X \subset \mathcal{M}$. Tantomeno deve permettere di capire K.

Robustezza del sistema crittografico

In particolare, l'azione del cifrario deve apparire completamente casuale.

$$\begin{array}{ccc} 1 & \rightarrow & m \\ 2 & \rightarrow & ? \\ 3 & \rightarrow & ?? \\ \ldots & \ldots & \ldots \end{array}$$

Quindi l'unico modo per rompere un cifrario ideale è provare tutte le chiavi, che costa

$$2^k.$$

La sicurezza di un cifrario qualunque si rapporta a quella di un cifrario ideale. Useremo k per confronto (es. per il caso ideale $k{=}k$, in generale $k \geq k$).
Ad esempio se un sistema ha un insieme di $2^{10} = 1024$ chiavi, ma si rompe usando solo 2^6 cifrature, allora si dice che:
$k = 10$
$k = 6$.

Principali cifrari

Su internet, i principali cifrari usati sono:

DES	$k = 56$	$k = 43$
3DES	$k = 168$	$k = 113$
AES256	$k = 256$	$k = 254$
RC4	$k = 256$	$k = 256$ (?)

Principali cifrari

Su internet, i principali cifrari usati sono:

DES	$k = 56$	$k = 43$
3DES	$k = 168$	$k = 113$
AES256	$k = 256$	$k = 254$
RC4	$k = 256$	$k = 256$ (?)

RC4 bisogna usarlo con attenzione, in particolare bisogna scartare i primi byte che genera e bisogna gestire il re-keying process.

CIFRATURA ASIMMETRICA

Crittografia a chiave pubblica

Non esiste solo una chiave privata, perchè siamo costretti ad usare un canale pubblico.

Diffie-Hellman

che consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione pubblico.

Cifratura asimmetrica

grazie alla quale si cifra e decifra usando una combinazione di chiave pubblica e privata (non condivisa).

L'Algoritmo Diffie - Hellman (DH)

L'algoritmo è particolarmente adatto per scambiarsi una chiave segreta attraverso un canale non sicuro.

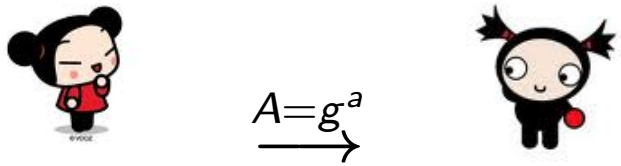
Alice  e Bob  vogliono negoziare una chiave segreta.

Innanzitutto Alice sceglie un primo p e $1 < g < p$.

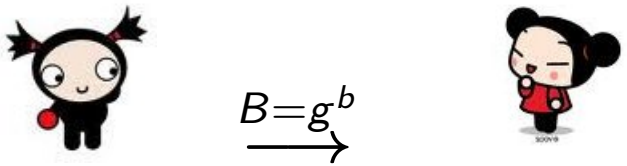
(p, g) è la chiave pubblica k_{p_a}.

L' Algoritmo Diffie - Hellman (DH)

Alice sceglie un numero segreto a random e trasmette g^a a Bob:



Bob sceglie un numero segreto b random e trasmette g^b ad Alice:



A questo punto Alice calcola $B^a = g^{a \cdot b}$;

Bob calcola $A^b = g^{a \cdot b}$.

Quindi hanno ottenuto la stessa chiave $g^{a \cdot b}$.

DLOG (Discrete Logarithm)

Il problema del Logaritmo Discreto su $\mathbb{F}_p$ è il seguente:

$$\text{dati } a, b \in \mathbb{F}_p, \quad \text{trovare } x \in \mathbb{F}_p \text{ t.c. } a^x = b$$

$$[n = \log_2(p)]$$

Si conoscono solo attacchi subesponenziali con l'algoritmo Function Field Sieve, che gira con complessità

$$2^{n^{\frac{1}{3}}(\log_2(n))^{\frac{2}{3}}}$$

$k \sim \sqrt[3]{n}$, ma per $80 \leq k \leq 256$ si ha $n \sim k^{1.7}$.

L' Algoritmo Diffie - Hellman (DH)

Se si riesce a risolvere il problema del logaritmo discreto (DLP), è anche possibile attaccare questo crittosistema.
Infatti ad un attaccante basta calcolare il logaritmo di A o di B per poi ricostruire la chiave segreta $g^{a \cdot b}$.

$$\text{Risolvere DLP} \longrightarrow \text{Attaccare DH}$$

Il viceversa non è noto:

$$\text{Attaccare DH} \not\longrightarrow \text{Risolvere DLP ?}$$

non sono noti al giorno d'oggi algoritmi che risolvono DH senza calcolare il logaritmo discreto.

RSA

L'algoritmo di cifratura asimmetrica RSA si basa sulla difficoltà del seguente problema.
Siano p e q numeri primi, sia $N = pq$, dato N trovare p e q.

$$[n = \log_2(N)]$$

Il miglior algoritmo è una versione del General Number Field Sieve, che gira con complessità

$$2^{n^{\frac{1}{3}}(\log_2(n))^{\frac{2}{3}}}$$

$k \sim \sqrt[3]{n}$, ma per $80 \leq k \leq 256$ si ha $n \sim k^{1.7}$.

Quindi RSA fornisce in pratica la stessa (in)sicurezza di DH.

Descrizione di RSA

Possiamo così passare a descrivere il cifrario. Alice  vuole trasmettere il messaggio m a Bob  .

- Bob deve creare una chiave pubblica $k_{p_b} = (N, e)$ ed una privata $k_{s_b} = (N, d)$ stando attento a non divulgare d
- Bob trasmette (N, e) ad Alice

Descrizione di RSA

Come crea Bob la chiave pubblica e privata?

- Bob sceglie due numeri primi p e q, molto grandi e li moltiplica $N = p \cdot q$
- Bob sceglie un numero e, coprimo con $\varphi(N)$ e più piccolo di $(p-1)\cdot(q-1)$
- Bob calcola d tale che sia l'inverso moltiplicativo di e, cioé $(x^d)^e \equiv x \bmod N$, qualunque x.
- solo Bob  conosce la chiave segreta (N, d) mentre (N, e) è pubblica

Descrizione di RSA

Come avviene la cifratura e decifratura?

- Alice calcola $c = m^e \mod N$
- Alice trasmette c a Bob
- Bob riceve c e lo decripta calcolando:

$$c^d \equiv m \mod N$$

Attacchi ad RSA

Ci sono 2 strategie generali per attaccare l'algoritmo RSA:

1. trovare un algoritmo che dato un messaggio cifrato c e la chiave pubblica (N, e) restituisce il messaggio decifrato m senza però recuperare la chiave privata (N, d).
   Nessuno ne ha mai trovato un'istanza significativa
2. ricavare la chiave privata (N, d) da quella pubblica (N, e)

Attacchi ad RSA

Come si può ricavare la chiave privata da quella pubblica?

Apparentemente ci sono 3 alternative:

1. fattorizzare N
2. Calcolare $\varphi(N)$
3. ricavare il valore d direttamente dalla chiave pubblica (N, e)

Si può facilmente dimostrare che queste tre metodi sono equivalenti:

$$(1) \Leftrightarrow (2) \Leftrightarrow (3)$$

Quindi in pratica si fattorizza N.

In pratica:

Usando l'algoritmo GNFS si riescono a fattorizzare numeri N dell'ordine di $2^{768} \sim 10^{231}$.

Spendendo un milione di euro per dotarsi di un cluster potente, si potrebbe rompere una chiave RSA-768 (cioé N è dell'ordine di 2^{768}) ogni 6 mesi (in media).

Per cui chiavi RSA-1024 o superiori (purché non siano chiavi deboli) sono al momento fuori dalla portata di un attacco pratico.

In pratica:

Il miglior attacco (aiuto?) noto a AES256 (che recupera la chiave) ha un costo di 2^{254} crittazioni.
Supponendo che

- un core medio riesca a eseguire 2^{50} crittazioni al secondo,
- esistano 1000 core ogni abitante del pianeta, compresi neonati e le persone che vivono nella giungla amazzonica,
- al mondo ci siano 20 miliardi di persone,

In pratica:

Il miglior attacco (aiuto?) noto a AES256 (che recupera la chiave) ha un costo di 2^{254} crittazioni.
Supponendo che

- un core medio riesca a eseguire 2^{50} crittazioni al secondo,
- esistano 1000 core ogni abitante del pianeta, compresi neonati e le persone che vivono nella giungla amazzonica,
- al mondo ci siano 20 miliardi di persone,

Allora TUTTA la potenza del mondo unita dovrebbe lavorare senza sosta per 10^{31} MILIARDI di anni.

Chiavi deboli

Una chiave debole, K, in un sistema crittografico (sia pubblico che privato) è una chiave che permette la rottura del sistema.

Invece, esistono numerose chiavi deboli sia per DH ed RSA.

Riteniamo quindi fondamentale per la sicurezza valutare attentamente la robustezza delle chiavi usate (e quindi la loro generazione)

Attacchi ad RSA - Chiavi Deboli

É facile rompere il sistema se ad esempio:

- se m ed e sono così piccoli che $m^e < N$; allora risulta facile trovare la radice e-esima di c, poichè $c = m^e$
- se $p - q < N^{\frac{1}{4}}$, ci sono algoritmi veloci per trovare p e q
- se $p - 1$ o $q - 1$ hanno solo fattori piccoli, N si fattorizza velocemente con l'algoritmo di Pollard $p - 1$
- se $d < \frac{1}{3} N^{\frac{1}{4}}$ (attacco di Wiener)

Purtroppo molte implementazioni dell'algoritmo RSA non controllano l'eventualità di avere una chiave debole.

Quanto deboli le deboli?

Le seguenti chiavi RSA 1024:

- $N = 313256375876377212440015318323108678520006173008034636063302469362957128066154999976915271523113164049627718408844530703484989822647023510143641161477819303748834981751956165887681798612125473753385171353058142794346935884074455798705326089160080254660362883833231925389841878822795695169771993196329575554015$
  $e = 29$
- $N = 703606728829425222386754525735867532353561794249517072279524442134219740346812313911122376577798291308343173684600484528948470625632095674346511670057314481944944614776388555094301742500113459518088266875640891456768243692575911682249129488446432250644872863589855553619827615234588027031314797877915$
  $e = 37$

Le ho date come sfida ai team di studenti impegnati nelle CryptoWars e le hanno rotte in

Quanto deboli le deboli?

Le seguenti chiavi RSA 1024:

- $N = 313256375876377212440015318323108678520006173008034636063302469362957128066154999976915271523113164049627718408844530703484989822647023510143641161477819303748834981751956165887681798612125473753385171353058142794346935884074455798705326089160080254660362883833231925389841878822795695169771993196329575554015$
  $e = 29$
- $N = 703606728829425222386754525735867532353561794249517072279524442134219740346812313911122376577798291308343173684600484528948470625632095674346511670057314481944944614776388555094301742500113459518088266875640891456768243692575911682249129488446432250644872863589855553619827615234588027031314797877915$
  $e = 37$

Le ho date come sfida ai team di studenti impegnati nelle CryptoWars e le hanno rotte in meno di un secondo.

Hanno rotto anche una chiave di curve ellittiche da 300 bit in meno di un minuto!

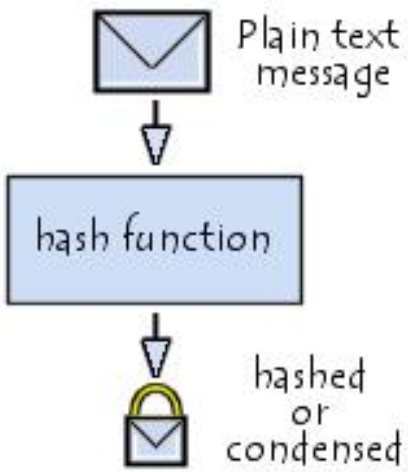
Ma tutto questo solo un gioco?

Presentiamo i risultati di un recentissimo lavoro scientifico appena uscito (14/02/2012), che porta la firma di Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter.

- 270.000 siti che condividono la chiave pubblica con almeno un altro sito;
- 71.052 chiavi sono usate da tanti siti diversi (stessa chiave anche per migliaia di siti);
- 12.720 chiavi RSA1024 sono vulnerabili e si rompono in meno di un millisecondo.

FUNZIONI HASH

Funzioni HASH



Una **funzione hash** è una funzione che converte un vettore arbitrariamente lungo in un vettore di lunghezza fissata:

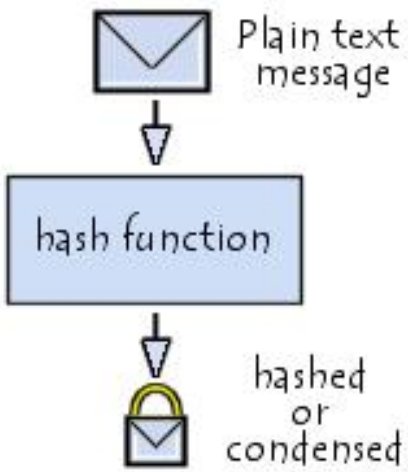
$$H : \mathbb{F}^{\infty} \rightarrow \mathbb{F}^{h}$$

dove $\mathbb{F} = \{ 0, 1 \}$.

Una collisione per H sono due file con lo stesso HASH.

Una buona funzione di HASH non dovrebbe permettere di trovare collisioni facilmente.

SHA-1



- SHA-1: Security Hash Algorithm [FIPS 180-1]
- dato un messaggio in input di lunghezza massima 2^{64} bits, SHA-1 restituisce come output una stringa di 160 bit

input: ''m'illumino di immenso''

output:

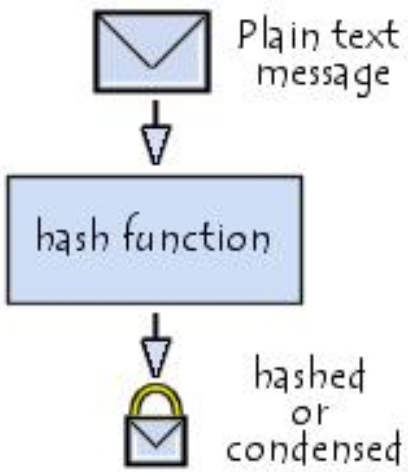
04DEC8C39C14B4E5AB284EE204C81D58F1A59936

input: ''Roma''

output:

DE5429D6F4FA2C86427A50757791DE88A0B75C85

SHA-1



- SHA-1: Security Hash Algorithm [FIPS 180-1]
- dato un messaggio in input di lunghezza massima 2^{64} bits, SHA-1 restituisce come output una stringa di 160 bit

input: ''mi illumino di immenso''

output:

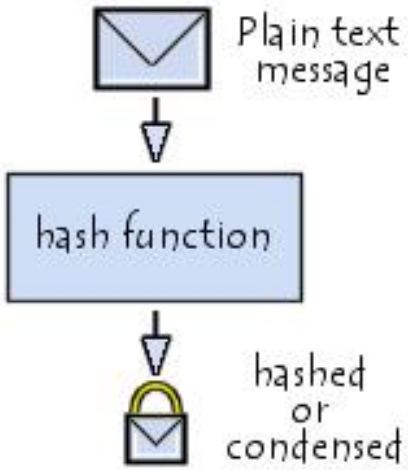
666BCFA1CC6D6580F316AF077B85B9DE34055A57

input: ''roma''

output:

A6B6EA31C49A8E944EFE9ECBC072A26903A1461A

SHA-2



- SHA-256: Security Hash Algorithm [FIPS 180-2]
- dato un messaggio in input di lunghezza massima 2^{64} bits, SHA-256 restituisce come output una stringa di 256 bit

SHA-256 è ritenuto nettamente più sicuro di SHA-1

Funzioni HASH

Complessità di trovare una collisione, ovvero due file con lo stesso HASH.

SHA1	$k = 80$	$k = 58$
SHA256	$k = 128$	$k = 128$
SHA3	$k =?$	$k =?$

Hash raccomandate da TLS 1.2 (SSL3.3):

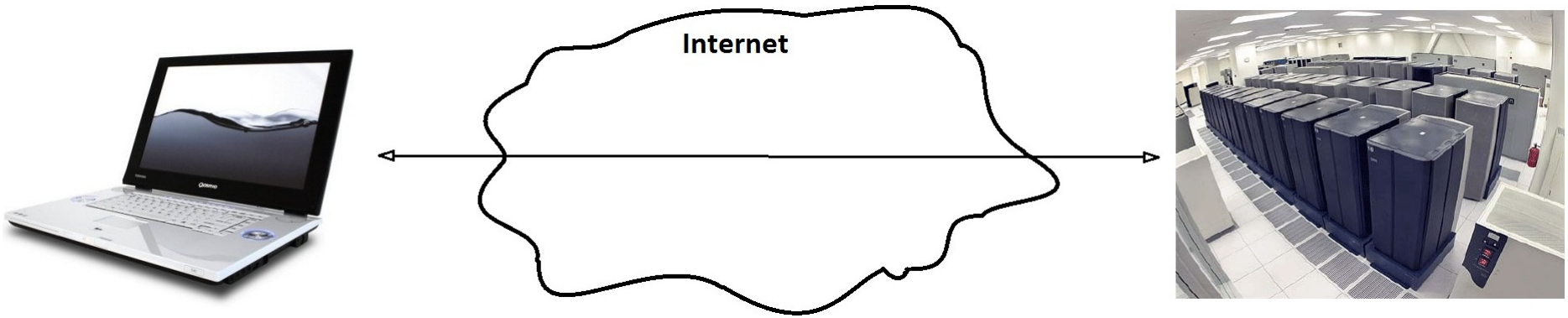
- MD-5
- SHA-1
- SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512)

MD-5 e SHA-1 sarebbero da evitare.

NETWORK SECURITY

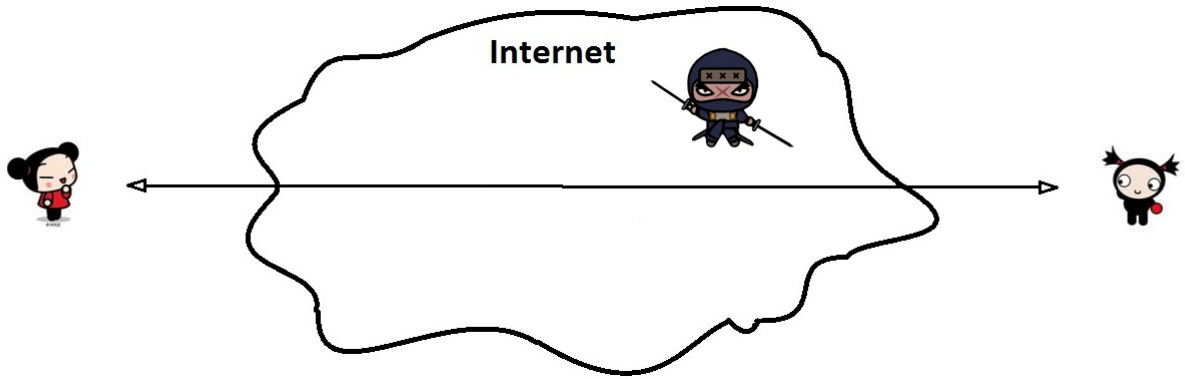
SSL/TLS

É il protocollo usato a livello di comunicazione tra browser e server per la trasmissione sicura dei dati.



SSL/TLS

É il protocollo usato a livello di comunicazione tra browser e server per la trasmissione sicura dei dati.



Fasi SSL/TLS

1. A e B si mettono d'accordo sulla crittografia da usare (Cipher Suite)
2. A si crea per conto suo dei bit segreti
3. B fa lo stesso
4. A e B scambiano una chiave segreta K (segreto condiviso) tramite RSA o DH
5. A e B usano K con AES, 3DES o DES per crittare la conversazione

Complichiamo un pò le cose

1. A e B si mettono d'accordo sulla crittografia da usare (Cipher Suite) e si scambiano del random non cifrato
   - il server dà un certificato al client, ovvero dà una chiave pubblica (tipicamente RSA) e una certification authority (CA) che garantisce per lui
   - bisogna scambiare un premaster secret

Complichiamo un pò le cose

Per scambiare il premaster secret ci sono due alternative:

Si usa la chiave del certificato

Usare la chiave del certificato é piú veloce ma abbassa la qualitá della randomicitá e quindi del segreto condiviso.

(il segreto viene solo dal client)

Si fa Diffie-Helmann

(il segreto viene sia dal client che dal server)

Complichiamo un pò le cose

Per ovviare al problema dell'uso della chiave del certificato si crea una master key.

$$\text{Premaster key} \longrightarrow \text{master key } K$$

PRF : Pseudo Random Function

$K = \text{PRF}(\text{premaster secret, random server, random client}) \rightarrow 384 \text{ bit}$

Se si usa una chiave RSA-1024 la sicurezza pura scende a 80 bit.

TWO FACTORS AUTHENTICATION

One factor authentication

Una static password è un segreto condiviso e passato per canali sicuri (esempio: raccomandata), tipicamente un PIN o una coppia login/password (oppure entrambi).

Chiamiamo questo tipo di autenticazione *one factor authentication*.

L'eventuale funzionalità del cambio di password a scelta dell'utente non aggiunge niente alla sicurezza teorica perché la nuova password viene a sua volta trasmessa sotto la protezione della vecchia.

Questo è chiaramente lo scenario più debole di autenticazione.

Two factors authentication

Si parla di *two factor authentication* quando si adottano due diversi metodi di autenticazione tra i seguenti:

- "Qualcosa che conosci" (password o PIN).
- "Qualcosa che hai" (token, bancomat).
- "Qualcosa che sei" (tracce biometriche).

Unione dei segreti

Non è chiaro come nel server siano conservati i diversi segreti.

- in chiaro? $\rightarrow$ chi víola il server è in grado di recuperare i dati degli utenti
- con una hash fra i vari segreti? $\rightarrow$ anche se il server venisse violato, l'attaccante non è in grado (teoricamente) di recuperare il valore dei segreti conoscendo solo il valore di hash

Tipi di autenticazione (token)

Tipi di autenticazione (token)

tipi di token	seed random	pseudo-random gen
static password	✓	
event-based token + pswd	✓	✓
time-based token + pswd	✓	✓
sms + pswd	✓	✓
interaz. tel.	✓	✓
token con PIN + pswd	✓	✓
card + PIN	✓	✓
token multi OTP device	✓	✓

RANDOMICITÁ

Creazione randomicità

Esistono principalmente tre modi per ottenere una chiave random:

- utilizzare sorgenti random naturali (es. campionamento del campo elettrico e della voce, rilevamento fenomeni quantistici)
- utilizzare algoritmi pseudo-random (PRF)
- utilizzare una commistione di sorgenti random e pseudo-random

Quanti random?

	CONDIVISI	NON CONDIVISI
TANTI BIT RANDOM	impossibile (o costosissimo)	• segreti in DH • segreti in RSA • segreti collegati al piccolo segreto condiviso
POCHI BIT RANDOM	• K chiave simmetrica (AES, DES, . . .) • seed di PRF	

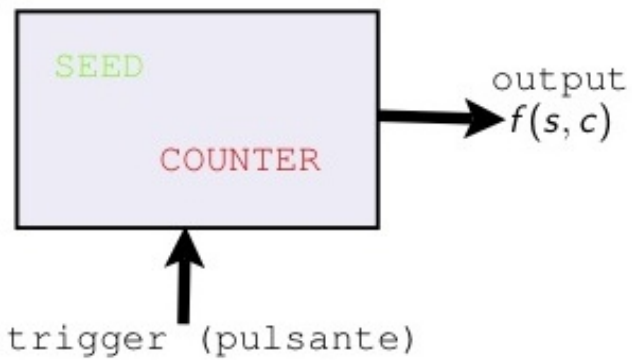
PRF: seed$\rightarrow$ 101000111 . . . . . .

Funzionamento teorico dei dispositivi

SEED segreto scambiato su un canale sicuro?

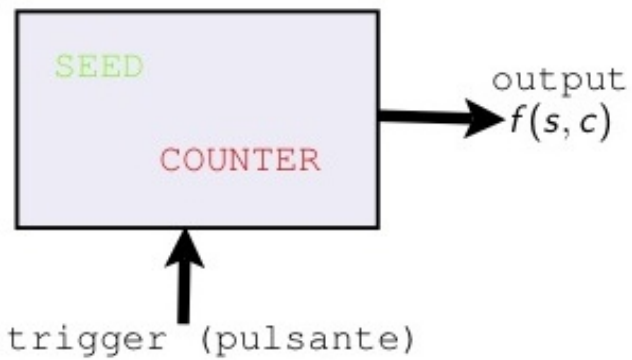
Funzionamento teorico dei dispositivi

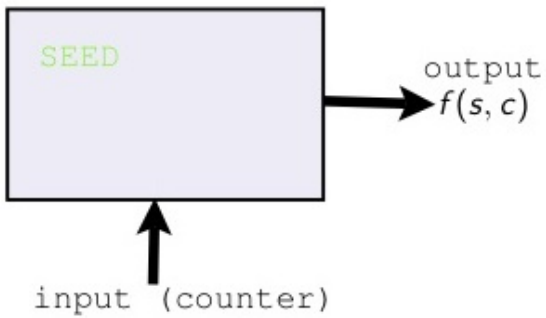
SEED segreto scambiato su un canale sicuro?



Funzionamento teorico dei dispositivi

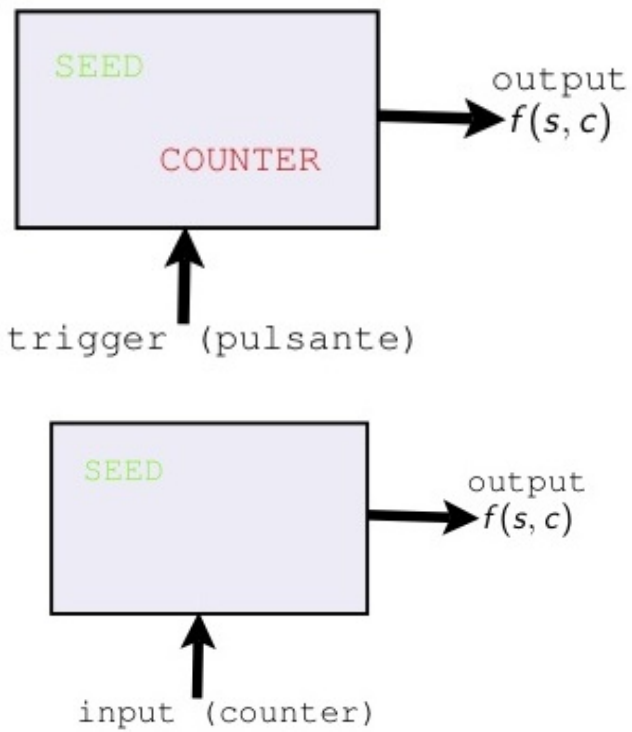
SEED segreto scambiato su un canale sicuro?





Funzionamento teorico dei dispositivi

SEED segreto scambiato su un canale sicuro?



Fissato un seed s, esiste un periodo Π tale che, per ogni counter c:

$$f(s, c + \Pi) = f(s, c)$$

Aspetti della randomicità

Esistono essenzialmente due aspetti fondamentali che riguardano la randomicità in quest'ambito:

1. creazione di sequenze pseudo-random da parte dei dispositivi
2. generazione (pseudo-random) dei seed.

Per tali ragioni si ricorre all'uso di PRF (Pseudo Random Function).

Purtroppo non è possibile dimostrare matematicamente la (pseudo-)randomicità di un algoritmo.

Sequenze pseudo-random

Per una buona PRF deve valere:

1. Per ogni SEED s e COUNTER c (usato), anche se l'attaccante conosce $f(s,1),\ldots,f(s,c)$ non è in grado di trovare $f(s,c+1)$.
   - 1-a in particolare, Π deve essere grande per ogni s
   - 1-b anche conoscendo $f(s,1),\ldots,f(s,c)$ non si può trovare s
2. idealmente anche se l'attaccante conosce il seed s, non deve essere in grado di recuperare il counter c da $f(s,c)$.

Generazione seed (pseudo-random)

Non solo è importante che le sequenze siano generate pseudo-random, ma a maggior ragione che la distribuzione dei seed sia il più possibile random
Ciò che non vogliamo è che

$$\text{conoscere } s_1,\ s_2, \ldots, s_{i-1} \implies \text{conoscere } s_i.$$

Per questo la distribuzione dei seed dovrebbe essere il più possibile vicino all' entropia pura. Ciò si può ottenere mediante generatori hardware di randomicità che sfruttano fenomeni quantistici quali:

- variazione del campo elettro-magnetico
- effetto fotoelettrico

PCI Express

PCI

Generazione pseudorandomicità

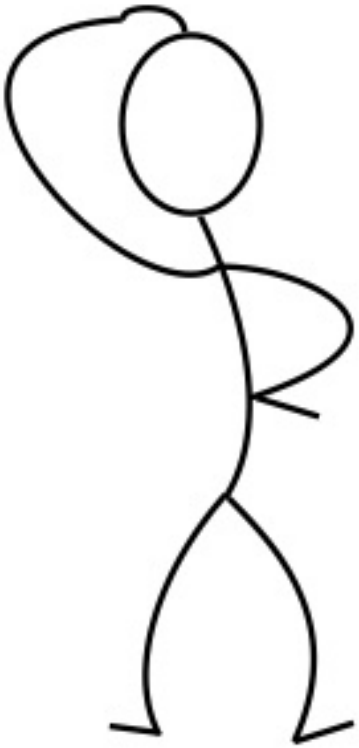
Ogni stringa generata pseudo-random viene chiamata OTP (One Time Password).

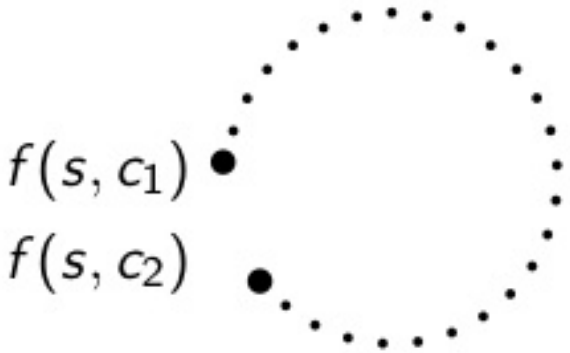
In modo algoritmico la pseudo-randomicità si può ottenere, a partire da un seed, principalmente in due modi:

- mediante funzione hash
- mediante cifrario a blocchi

Il punto di vista dell'osservatore esterno

$$
\begin{array}{rcl}
f(s, c_1) & \rightarrow & 101011011110 \\
f(s, c_2) & \rightarrow & 101010001110 \\
\vdots & & \vdots \\
f(s, c_i) & \rightarrow & 101110011001 \\
f(s, c_{i+1}) & \rightarrow & ?? \\
 & \searrow & \\
 & & s??
\end{array}
$$





Scambio seed

Il seed deve essere "scambiato su un canale sicuro".

In pratica, sul token c'è un numero di serie. Il seed del token è stampato all'interno e non è modificabile.
Su un qualche server del produttore di token c'è la corrispondenza biunivoca tra seed e numeri di serie.

Il server della banca conosce questa corrispondenza relativamente ai token che ha comprato.
Siccome la banca sa quale token ha il cliente, nei suoi server ci deve essere quindi la corrispondenza tra cliente (login/password/PIN e ogni altro suo dato) e seed del token.

Funzioni HASH

Esistono vari modi per ottenere da una funzione hash una sequenza pseudo-random, ad esempio:

$$\mathrm{HMAC}_K(c) = H(K_1, H(K_2, c))$$

con chiavi K_1, K_2 ottenute dalla chiave K, che rappresenta il seed.

HOTP

OATH per generare l'OTP utilizza l'algoritmo HOTP, basato su HMAC.

In particolare, dato un seed $s = K$ ed un counter c, HOTP ottiene un output di 160 bit

$$\begin{aligned} \text{HOTP}(K, c) &= \text{HMAC-SHA-1}(K, c) \\ &= \text{SHA-1}\left(K_1, \text{SHA-1}\left(K_2, c\right)\right) \end{aligned}$$

con chiavi K_1, K_2 ricavate da K

HOTP e SHA-1

HOTP prevede l'utilizzo di SHA-1, una funzione hash per cui sono noti attacchi che ricavano collisioni. Questo fatto non preoccupa i promotori di OATH che nel 2005 dicevano:

"Is SHA-1 broken? For most practical purposes, we would say probablynot, since the resources needed to mount the attack are huge. Here is one way to get a sense of it: we can estimate it is about the same as the time we would need to factor a 760-bit RSA modulus, and this is currently considered out of reach."

Ma ormai, come si sa, le chiavi RSA-760 si riescono a fattorizzare.

PRODOTTI E VENDORS

RSA

- RSA SecureID
- token time based
- fino al 2002 creava pseudo-random con un algoritmo proprietario, che però è stato rotto
- ora usa un sistema basato su cifrari a blocchi (DES, 3DES, AES)
- non è specificata la generazione dei seed (HSM? Entropia pura?)

GEMALTO

- Ezio suite tokens
- implementa OATH (con HOTP)
- fino a febbraio 2011 implementavano HOTP con SHA-1
- no informazioni sulla generazione dei seed (Entropia pura?)

VASCO

- Digipass GO
- i dispositivi supportano sia algoritmi time ed event based con DES, 3DES e AES
- sia (su richiesta) il modello OATH event based (HOTP) o event based (TOTP)

VASCO

- Digipass 250
- i dispositivi supportano sia algoritmi time ed event based con DES, 3DES e AES
- sia (su richiesta) il modello OATH event based (HOTP) o event based (TOTP)
- inserimento PIN mediante tastiera con creazione dinamica di OTP

I reader con carta

Questi dispositivi differiscono dai precedenti poiché la creazione dell'OTP non dipende direttamente dal dispositivo, ma dalla smart card.

Il loro funzionamento si può riassumere in questo modo:

- si inserisce la card nel dispositivo
- si inserisce il PIN per "sbloccare" la carta
- la sequenza pseudo-random viene prodotta dal seed e dagli algoritmi di cifratura contenuti nella card

I reader con carta

GEMALTO

Ezio suite card readers

VASCO

Digipass reader

Grazie per l'attenzione