

Teoria

04/02/2023

Introduzione

- Docente: Gianluca Dini, Pericle Perazzo
 - Mail: gianluca.dini@unipi.it
 - la prova finale sarà qualche domanda con risposta a scelta multipla e una piccola modifica lato codice sugli esercizi visti durante le esercitazioni con l'ing.Perazzo
-

Dobbiamo considerare che abbiamo un avversario che vuole attaccare il nostro sistema.

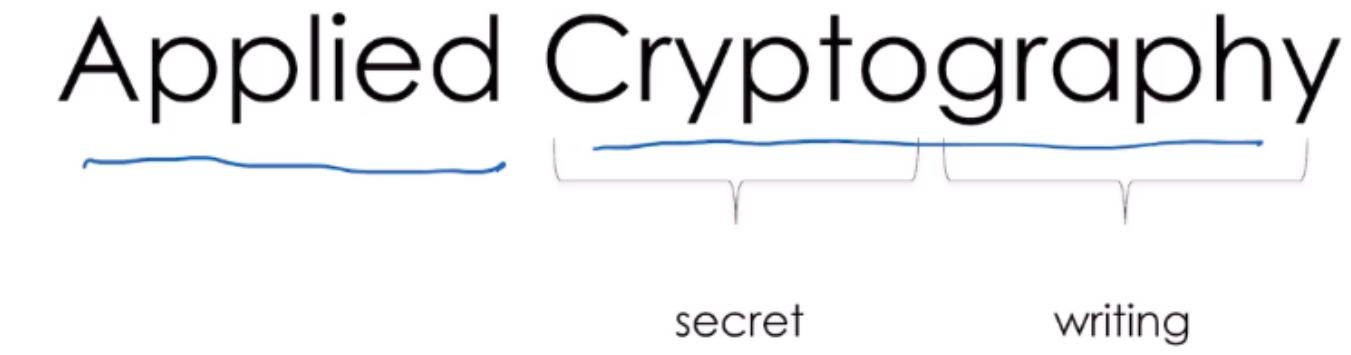
L'avversario cercherà di violare:

1. Confidenzialità
2. Integrità
3. Disponibilità

La crittografia ci permette di proteggere la confidenzialità e l'integrità dei dati.

La disponibilità si protegge in altri modi che non sono metodi crittografici.

Parleremo di **crittografia applicata** in quanto non progetteremo algoritmi crittografici perchè inerente l'ambito matematico. Noi invece ci chiederemo come usare gli algoritmi per proteggere i dati trasmessi in rete. Ci interessa la parte applicativa. Cos'è un certificato, quali vantaggi mi da una firma digitale, etc...



La storia ci insegna che nella maggior parte dei casi gli algoritmi crittografici "home-made" possono essere facilmente rotti. La best practice è quella di usare sempre algoritmi standard, ovvero progettati dai migliori crittografi in circolazione che sono stati attaccati e analizzati e alla fine di questo processo è stato deciso che si trattava di algoritmi sicuri e prestazionali.

La crittografia è importante in quanto nei servizi e apparati che usiamo la troviamo ovunque:

- HTTPS: Web Traffic
- Wireless Traffic: 802.11i WPA2, GSM, Bluetooth
- File cifrati su disco: EFS, TrueCrypt
- Protezione dei media: DVD(CSS) e Blu-ray (AACS)

Cosa significa "Security" ?

Applied Cryptography - 01 Introduction

...

1. What does "security" mean? Sort these notions of security in decreasing order of strength

1 Here is a mathematical proof, accepted by experts, that shows it is secure

↑ ↓

2 Here is a strong argument why breaking it is as hard as solving a problem we believe is hard

3 Many very smart, highly motivated people tried to break it but couldn't

4 There are 834 quadrillions possible keys so it must be secure

1. Se abbiamo una dimostrazione matematica per cui un algoritmo è sicuro quell'algoritmo è sicuro matematicamente. Esiste un algoritmo di questo tipo ma è male utilizzabile nella pratica quindi si usa molto poco. Ne vedremo un esempio dopo.
2. "Rompere quell'algoritmo matematicamente parlando o in termini di tempo è tanto difficile quanto risolvere un problema complesso". Se abbiamo un numero con 100 cifre decimali, fare la fattorizzazione è molto difficile, quindi anche usando i migliori elaboratori ci vogliono migliaia di anni. Se dimostro che rompere quell'algoritmo corrisponde a velocizzare la fattorizzazione, posso stare sicuro. Si tratta del caso della crittografia asimmetrica.
3. Non ha dietro una dimostrazione matematica ma una prova sperimentale in cui i migliori crittografi ben equipaggiati hanno provato a romperlo ma non ci sono riusciti.
4. Scenario in cui c'è un numero enorme di chiavi di crittografia per cui potrei provarle tutte ma sono talmente tante che ci vorrebbe una quantità di tempo impraticabile. Mettendoci anche un microsecondo ci vuole una quantità di tempo inestimabile. Questa non è neanche una definizione di sicurezza.

Ci sono 4 livelli diversi di sicurezza e ogni algoritmo cerca di coprirne almeno 3. Vedremo anche in quale di questi ogni algoritmo si colloca.

Cryptography

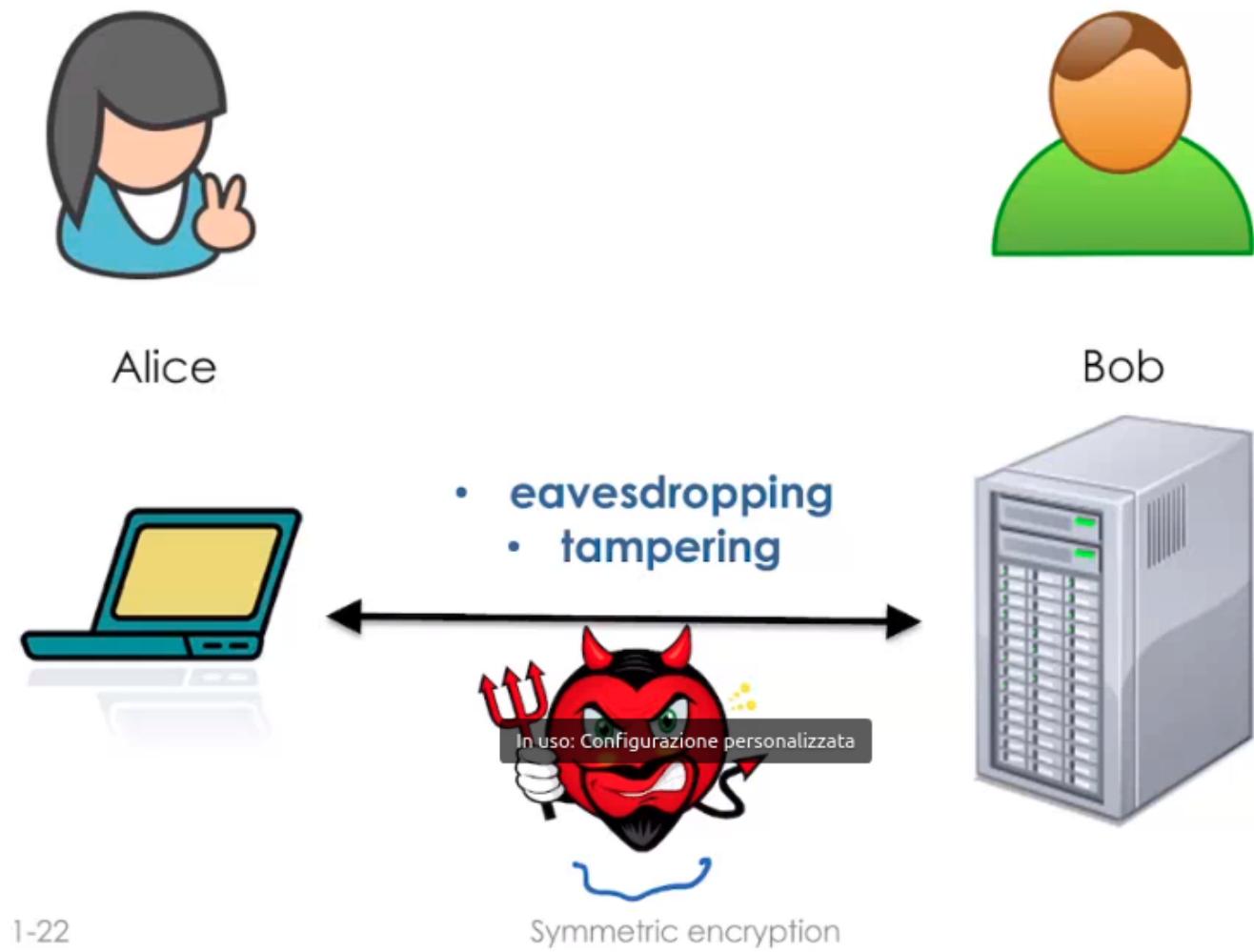
È: uno strumento molto utile e le fondamenta per molti meccanismi di sicurezza

Non è: la soluzione a tutti i problemi, affidabile se implementata e usata in modo sbagliato e qualcosa da inventare e usare per conto nostro

Primitive crittografiche

Cyphers

Cifrari simmetrici



1-22

Lo scenario di riferimento contiene due interpreti principali (**Alice** e **Bob**) che vorranno comunicare tra di loro attraverso una rete di calcolatori. Un esempio potrebbe essere che Alice usa il suo laptop per collegarsi al server di Bob.

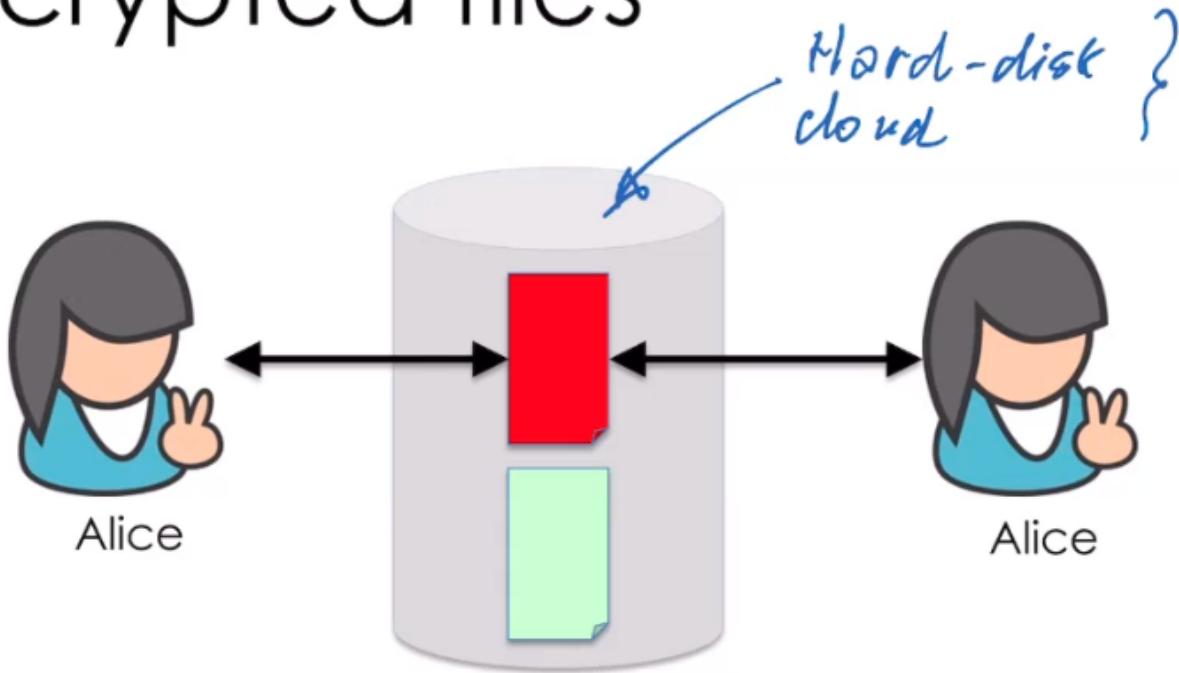
L'attaccante cercherà di violare la confidenzialità delle comunicazioni (eavesdropping) e/o alterare l'integrità della comunicazione (tampering).

- **eavesdropping**: ad esempio Alice invia la password a Bob e l'attaccante può rubare la password
- **tampering**: durante una transazione bancaria da Alice a Bob, l'attaccante modifica il destinatario della somma di denaro inviandoli a se stesso

Per lo meno vorremmo che il tampering sia rilevato.

Lo scenario è il medesimo del seguente:

Encrypted files



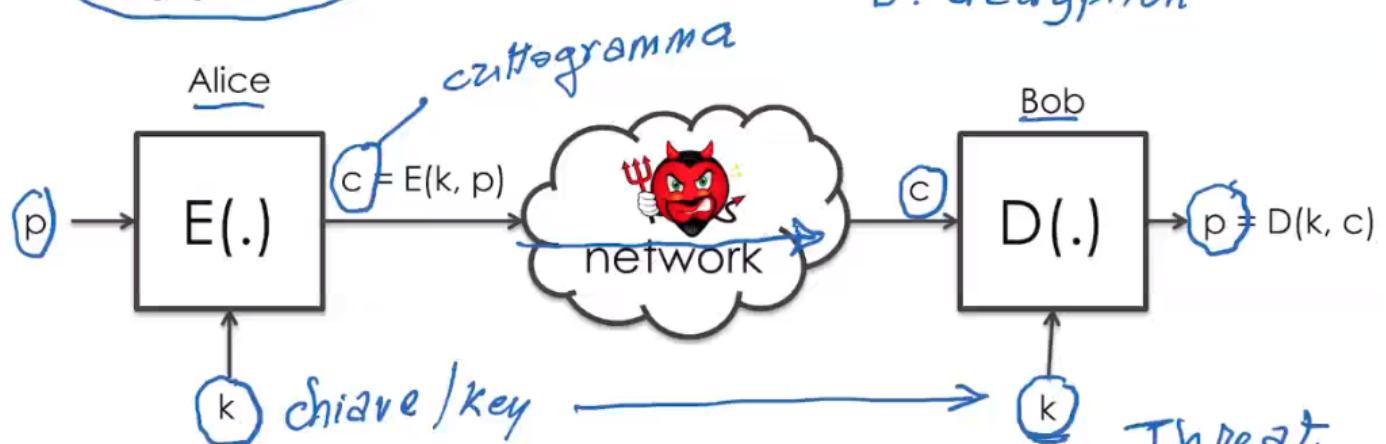
Alice salva un file in un repository e poi lo va a rileggere. E' come se Alice parlasse con se stessa, perchè salvare un file su un disco è come se la Alice di oggi parlasse con la Alice di domani, in quanto il file lo rileggerà nel futuro (domani, o tra un mese). Vogliamo proteggere la modifica del file.

Building block: symmetric



cypher = (E, D)

E : encryption
 D : decryption



- E, D : cipher **k : shared secret key** (128 bits)
- p, c : plaintext, ciphertext

Alice e Bob dispongono di un algoritmo di crittografia che è detto **cifrario** costituito da una coppia di funzioni o algoritmi:

- La funzione **E** è l'algoritmo di cifratura (**encryption**). Prende in ingresso una chiave **K** che è un segreto da immaginare come una sequenza di bit (es: 010100010) che tipicamente ad oggi è dell'ordine di 128 bit. Come si genera una chiave **k** in modo intuitivo? Prendo una "moneta", lanciandola, in base al risultato stabilisco il valore dei bit, ovvero se esce testa ==> 1, se esce croce ==> 0, la lancio 128 volte e segno i 128 valori. Questa funzione usando i parametri di input mi restituisce il crittogramma **c** che è la cifratura del testo in chiaro da cifrare **p**. Questo **c** verrà inviato sulla rete e noi ipotizzeremo che il nostro avversario sia così bravo da intercettare **c**. Questo è il threat model d'esempio.
- La funzione **D** è l'algoritmo di decifratura (**decryption**). **c** arriverà a Bob che prende in ingresso anche la stessa chiave **k** e l'algoritmo **D** restituirà il messaggio originario inviato da Alice.

Sia Alice che Bob avranno la loro coppia identica di funzioni **D** ed **E** (es. librerie python o chip elettronici sulla scheda di rete). La chiave **k** è un segreto condiviso tra Alice e Bob e non deve essere pubblico. La sicurezza di questo sistema che protegge la confidenzialità della comunicazione si basa sulla complessità di **k**.

Se chiunque conosce la chiave **k** ovvero la sequenza esatta di 128 bit ad esempio, prenderebbe **c**, userebbe l'algoritmo **D** che è pubblico, e otterrebbe **p**.

❓ Come fanno Alice e Bob ad avere un segreto condiviso **k**

Potrebbero incontrarsi e scambiarselo al bar, ma funziona meno bene questa soluzione se il client e il server si trovano in locazioni geografiche molto distanti. Come lo stabiliscono è un problema e lo vedremo più avanti. Alice da casa sua potrebbe lanciare la moneta e definire **k** e inviarla a Bob ma mandandola in chiaro l'avversario la intercetterebbe e non sarebbe più sicura la comunicazione. Per mandarla in rete in modo protetto dovrei cifrarla ma per farlo ci vorrebbe un'altra chiave **K₁** e il problema si ripeterebbe. Si può interrompere o con metodi OFFLINE (si incontrano al bar o usando un altro mezzo di comunicazione, la lettera postale con dentro il pin che la banca ci invia a casa è un esempio di mezzo di comunicazione alternativo utilizzato nella realtà) oppure ONLINE ovvero raggiungere un segreto condiviso attraverso la rete stessa definita come insicura, ad esempio il TLS mi permette di raggiungere questo obiettivo ma per farlo è necessaria la crittografia a chiave pubblica.

Per ora il nostro assunto è:

- ✓ Alice e Bob hanno un segreto **k** condiviso
- ✓ L'attaccante conosce gli algoritmi utilizzati **D** ed **E**
- ✓ L'attaccante può intercettare **c**

Properties

Correctness property

For all p and k : $D(k, E(k, p)) = p$

There exist many functions that satisfy the Consistency property

We are interested in those that are also **secure**

Per far sì che tutta la sicurezza sia concentrata in k e non nel resto è necessario che dato c deve essere difficile determinare p senza conoscere la chiave k . Di tutte le funzioni D ed E che posso inventarmi devo trovare quelle per cui deve essere molto difficile trovare p non conoscendo k e avendo solo c . Questa proprietà dipende dal fatto che il mio avversario può intercettare qualsiasi messaggio. Se il mio sistema è sicuro deve esserlo per ogni comunicazione. Dati c e p deve essere inoltre difficile ottenere la chiave k , a meno che non sia usata una volta sola (questa clausola verrà compresa più avanti quando parleremo del cifrario di Vernam):

Security of a cipher (informal)

Security property

- A symmetric cipher is secure iff for each plaintext-ciphertext pair (p, c) ,
1. ⚡ given c , it is “difficult” to determine p without knowing k , and vice versa
 2. ⚡ given c and p , it is “difficult” to determine k , unless it is used just once

L'avversario cercherà di violare almeno una di queste 2 proprietà tramite la **crittoanalisi**, ovvero *lo studio delle tecniche matematiche per rompere l'algoritmo crittografico ovvero violare queste 2 proprietà*. Le ipotesi sono: l'avversario ha accesso ai dati trasmessi in forma cifrata ovvero intercettare qualsiasi cosa e l'altra è l'ipotesi di Kerckhoffs ovvero l'avversario conosce tutti i dettagli degli algoritmi di cifratura in quanto pubblicamente reperibili.



In crittografia, il principio di Kerckhoffs (conosciuto anche come legge di Kerckhoffs), enunciato per la prima volta da Auguste Kerckhoffs alla fine del 1880, afferma che un crittosistema deve essere sicuro anche se il suo funzionamento è di pubblico dominio, con l'eccezione della chiave.

Tenere segreti gli algoritmi **E** e **D** storicamente non ha mai funzionato: la Security through Obscurity (vedi film ENIGMA). In genere **E** e **D** sono scritti nel firmware delle schede elettroniche, se fossero sconosciute a tutti tranne che al produttore e dovessero essere venir rotte da un attaccante, il produttore dovrebbe ritirare tutte le schede elettroniche dal mercato, cambiare **E** e **D** e ridarle ai propri clienti. Cosa diversa se le funzioni hanno passato una fase di crittoanalisi condivisa a livello globale da diverse istituzioni e standardizzata e dunque conosciuta da tutti.

💡 Esempi di cifrari simmetrici 💡

Algoritmo a sostituzione monoalfabetica



Cryptoanalysis An historical example

Mono-alphabetic substitution

Cleartext alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Key	J	U	L	I	S	C	A	E	R	T	V	W	X	Y	Z	B	D	F	G	H	K	M	N	O	P	Q

P = "TWO HOUSEHOLDS, BOTH ALIKE IN DIGNITY,
IN FAIR VERONA, WHERE WE LAY OUR SCENE"

("Romeo and Juliet", Shakespeare)

P' = "TWOHO USEHO LDSBO THALI KEIND IGNIT
YINFA IRVER ONAWH EREWE LAYOU RSCEN E"

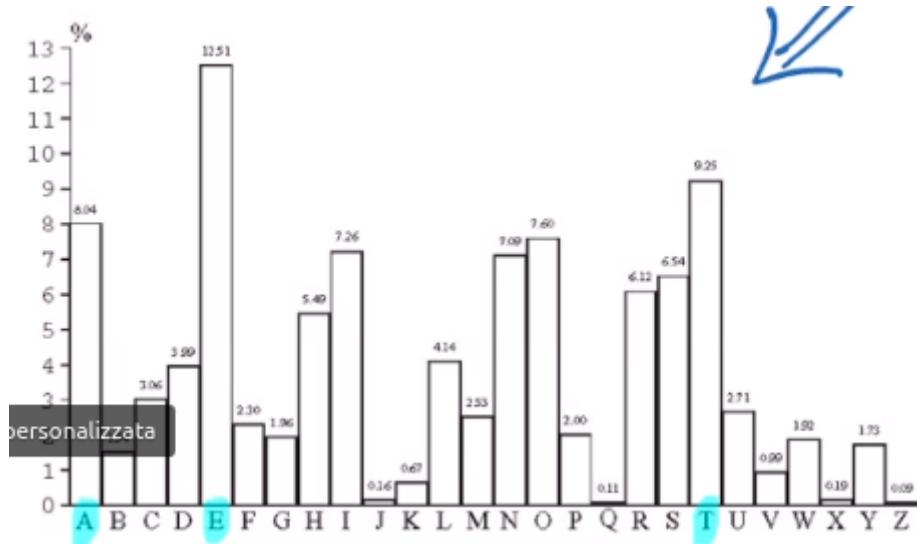
C = "HNZEZ KGSEZ WIGUZ HEJWR VSRYI RAYRH
In uso: Configurazione personalizzata
PRYCJ RFMSF ZYJNE SFSNS WJPZK FGLSY S"

Nell'esempio, **P** contiene le lettere raggruppate a gruppi di 5 solo per evitare di mappare le lettere 1:1 e rendere l'attacco più facile del previsto.

L'avversario in rete intercetta **C** e vorrebbe ricavare **P**. L'avversario potrebbe iniziare a ragionare sui caratteri ma quello che sicuramente può fare l'avversario è eseguire un' **attacco esaustivo alle chiavi** o anche detto **attacco a forza bruta o brute-force**, fin quando non ottiene un messaggio avente un senso compiuto. Se lui fa un attacco a forza bruta deve provare tutte le possibili permutazioni, con **n** elementi le permutazioni sono **n!**.

In questo caso $26!$ è circa 4×10^{26} e se anche ci mettesse un microsecondo per ogni permutazione ci vorrebbe troppo tempo. Questo significa che è **DIFFICILE**. Visto così questo algoritmo sembrerebbe molto sicuro, perchè ci vuole molto tempo per un attacco a forza bruta.

Le lettere però non sono tutte equiprobabili nei testi scritti. Nella lingua inglese la lettera che appare più frequentemente è la lettera 'E' 'T' ed 'A'. Se la lettera più frequente nella lingua inglese è la 'E', ma con l'algoritmo di crittografia dell'esempio io la trasformo in 'S', troverò molte 'S' ed è molto probabile che corrisponda alla lettera 'E'.



↙

Si può ragionare anche per coppie di lettere (es. TH in inglese è molto frequente).

Sfruttando le statistiche del linguaggio è possibile crittoanalizzare il testo cifrato e trovare la chiave.

11/02/2023

L'altra volta abbiamo fatto alcune considerazioni generali e abbiamo visto un primo cifrario ovvero quello a sostituzione monoalfabetica. Abbiamo visto in un caso semplice cosa vuol dire attaccare un cifrario.

Perfect cyphers

Un cifrario perfetto esiste, è molto semplice da realizzare, è stato realizzato in pratica ma capiremo perchè non lo usiamo tutti i giorni.

❓ Un cifrario perfetto cosa vuol dire?

Shannon nel **1949** diede una definizione matematica, in quegli anni di guerra esisteva Turing e al tempo stesso in America c'era Shannon con Von Neuman:

- "Un cifrario è perfetto se non rivela alcuna informazione sul testo in chiaro"

Se la coppia di algoritmi **E** e **D** vuol dire che l'avversario intercetterà tutto il testo cifrato possibile ma non riuscirà ad ottenere alcuna informazione riguardo il testo in chiaro.

- Inoltre se riusciamo a costruire un cifrario perfetto, non ci interessa nemmeno la potenza computazionale dell'avversario, ovvero si può ipotizzare che si trovi in una condizione di capacità di calcolo infinita.

- 💡 **Teorema di Shannon**

Se riuscissimo a costruire un cifrario perfetto , si può dimostrare che il numero delle chiavi deve essere maggiore o uguale del numero dei messaggi.

Perfect cipher

- **Shannon's idea of security (1949)**

- Cipher-text should not reveal any information about plain-text

- **Perfect secrecy (unconditional security)**

- An adversary is assumed to have infinite computing resources

- Observation of the CT provides no information whatsoever to the adversary

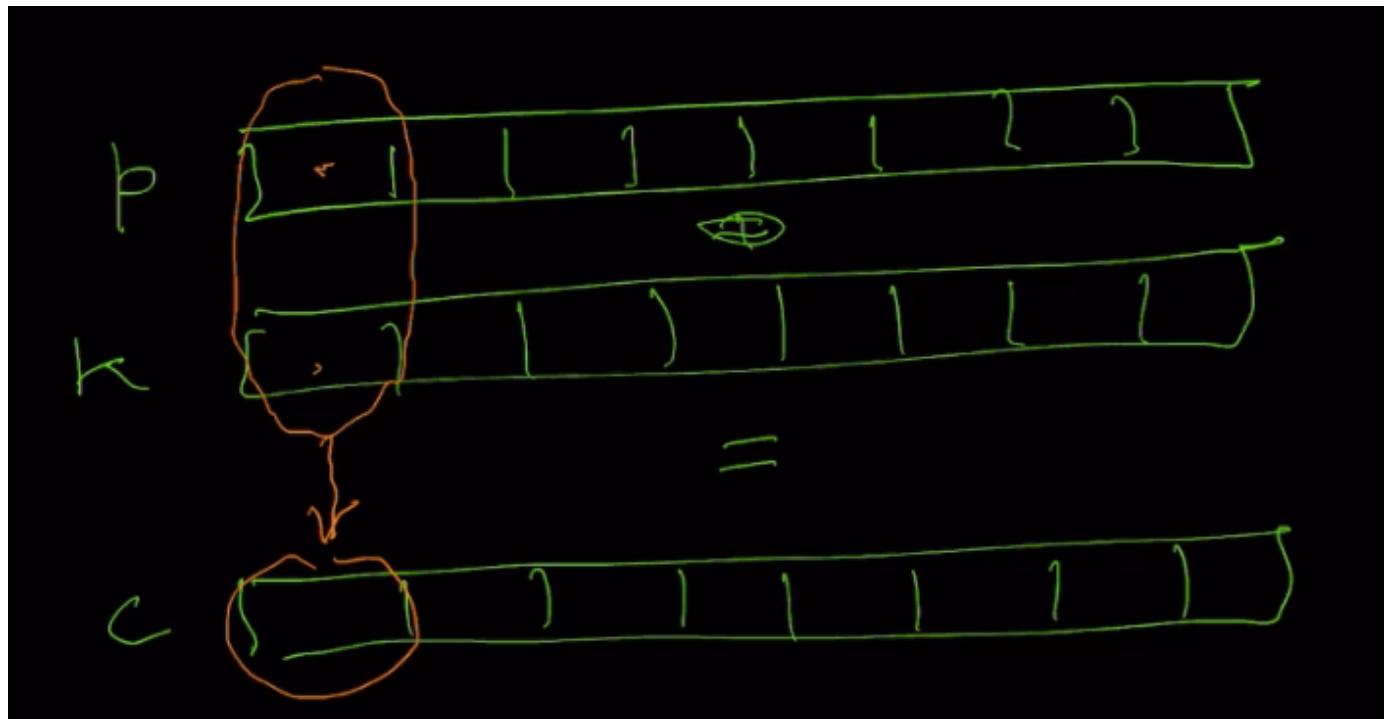
- **Necessary conditions**

- the key is randomly chosen
 - #keys cannot not be smaller than #msg (**Shannon's theorem**)

One Time Pad

Il cifrario perfetto esiste e si chiama **One Time Pad** (Vernam cipher, 1917).

Supponiamo che il testo in chiaro sia sostanzialmente una sequenza di bit. Ogni casella è un bit e può contenere o il bit 0 o il bit 1. Chiaramente se l'avversario la intecetta si troverà con la sequenza di bit, e magari se conoscesse a priori che Alice e Bob si stanno scambiando un PDF potrà interpretare la sequenza come PDF, ovvero l'avversario non esegue mai l'attacco alla cieca ma parte già con una conoscenza pregressa. Il cifrario di Vernam prevede di prendere una chiave **K** che ha tanti bit quanti sono i bit del messaggio. ⚡ **La chiave viene costruita in modo randomico** ⚡ . A questo punto viene eseguita l'operazione di **OR ESCLUSIVO (XOR)** bit a bit. Ovvero per ogni coppia di bit partendo dalla prima coppia, usando l'operazione XOR, produco il cipher text.



Le operazioni bit a bit sono estremamente efficienti e veloci. Si tratta di un'operazione molto semplice, veloce e poco costosa computazionalmente parlando, quindi da un punto di vista implementativo è ideale.

⚠ XOR

Si tratta di un'operazione logica binaria. Per definizione se i due bit sono diversi il risultato è 1 altrimenti se sono uguali ritorna 0.

p	k	c
0	0	0
0	1	1
1	0	1
1	1	0



One Time Pad

(Vernam cipher, 1917)

- Let p be a t -bit cleartext, i.e., $m \in \{0,1\}^t$
- Let k be a t -bit key stream, $k \in \{0, 1\}^t$, where each bit is truly random chosen
- Encryption $E(k, p)$: $c = p \oplus k$ (bitwise)
- Decryption $D(k, c)$: $p = c \oplus k$ (bitwise)

\oplus OR ESCLUSIVO

Si può dimostrare che conoscendo il testo cifrato c , è possibile decriptare riapplicando lo XOR con la chiave k in possesso.

Possiamo dimostrare che se la chiave è perfettamente random, allora il cifrario di Vernam è perfetto.

OTP has perfect secrecy

- **Theorem.** OTP has perfect secrecy iff
 1. For all p, p' : $\text{sizeof}(p) = \text{sizeof}(p')$
 2. For all p : $\Pr(p \text{ is transmitted}) \neq 0$
 3. Key k is truly random

Esempio del perchè il cifrario è perfetto:

OTP has perfect secrecy: intuition



- $c[i] = m[i] + k[i] \text{ mod } 26$
- $m = \text{"SUPPORT JAMES BOND"}$

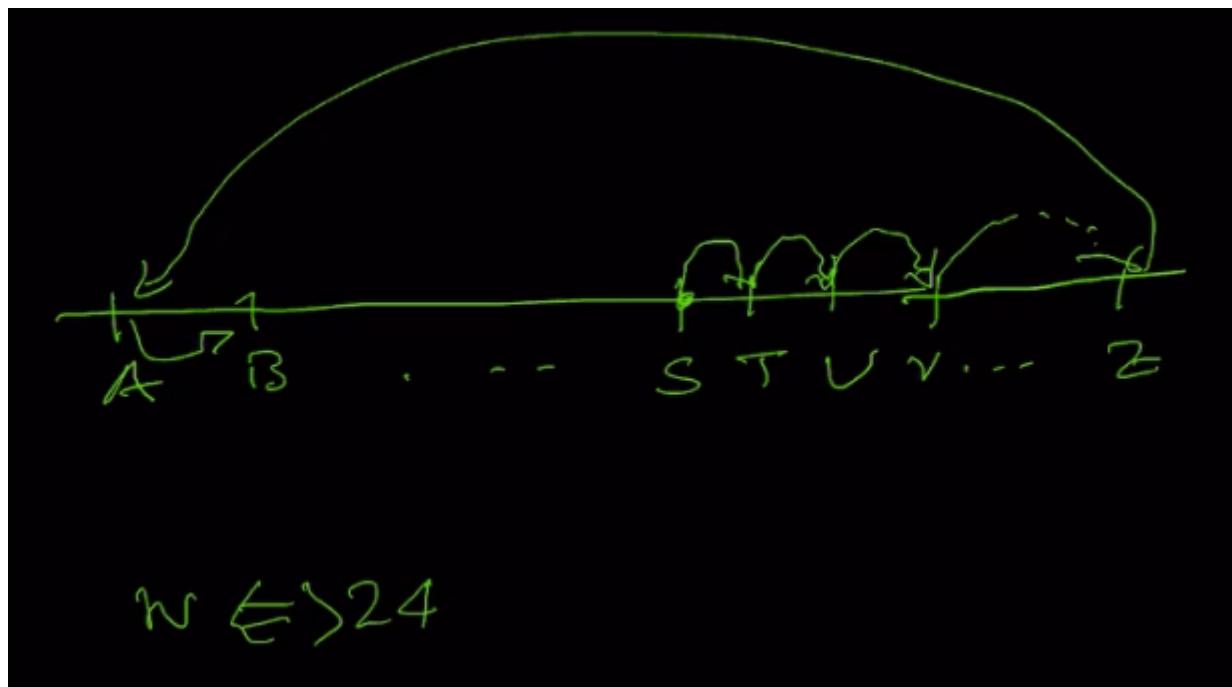
$m =$	S	U	P	P	O	R	T	J	A	M	E	S	B	O	N	D
$k =$	W	C	L	N	B	T	D	E	F	J	A	Z	G	U	I	R
$c =$	O	W	A	C	P	K	W	N	F	V	E	R	H	I	V	U



$c =$	O	W	A	C	P	K	W	N	F	V	E	R	H	I	V	U
$k' =$	M	W	L	J	V	T	S	E	F	J	A	Z	G	U	I	R
$m =$	C	A	P	T	U	R	E	J	A	M	E	S	B	O	N	D

Scenario: james bond deve andare in missione in nazione straniera, quindi il suo capo manda un messaggio m all'agente di supporto in cui dice "supporta james bond". La chiave che viene generata è k ma non sto usando i bit ma i caratteri. Qui c'è una complicazione...

⚠ Lo XOR è una somma a singolo bit in base 2, qui stiamo facendo una somma in singola cifra in base 26. Il concetto è che se prendiamo le prime lettere di m e k ovvero 'S' e 'W', usando l'alfabeto inglese che ha 26 caratteri, alla A associo il valore 0, B il valore 1 ... alla Z il valore 25. S + W mi porta sulla lettera O in quanto essendo modulare in base 26 è ciclico su 26 caratteri possibili, è come se l'alfabeto fosse circolare.



$n \in \mathbb{N}$

Si può dimostrare che questo corrisponde a fare lo XOR.



c è il messaggio che viene trasmesso in rete. Questo viene intercettato dall'avversario il quale vuol cercare di capirci qualcosa, ma essendo perfetto non riesce. Può però provare un attacco a forza bruta ovvero provare tutte le possibili chiavi. Avendo un messaggio di 16 caratteri con un alfabeto di 26 caratteri avremmo 26^{16} possibili chiavi.

Pur provandole tutte lui otterrà da ogni possibile chiave un plaintext, ovvero tutti i possibili messaggi su 16 caratteri. Tra questi messaggi ci sarà anche quello che sta cercando ma ce ne sarà anche qualcuno che ha senso compiuto ma non è quello originale mandato, ad esempio:
"CAPTURE JAMES BOND".

Non si usa nella pratica per i seguenti motivi:

CONS

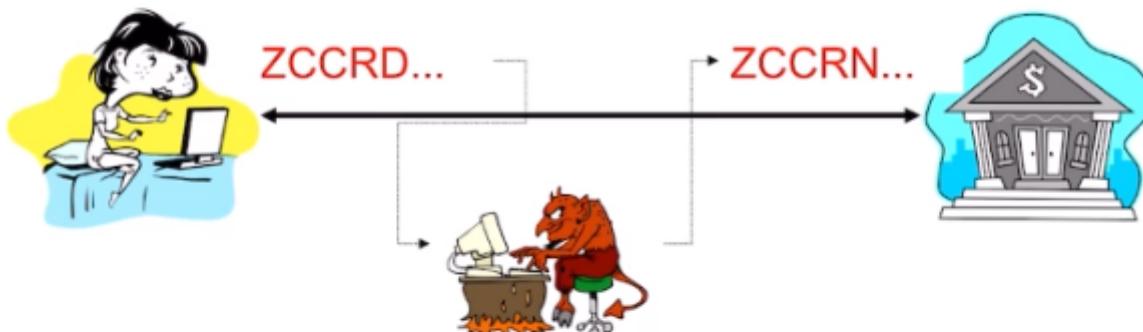
- **Long keys: unpractical!**
 - Key len = msg len
- **Keys must be used once: avoid two time pad!**
 - $C_1 = M_1 \text{ xor } K, C_2 = M_2 \text{ xor } K \Rightarrow$
 - $C_1 \text{ xor } C_2 = M_1 \text{ xor } M_2 \Rightarrow M_1, M_2$ (due to redundancy of English and ASCII)
- **A Known-PlainText attack breaks OTP**
 - Given $(m, c) \Rightarrow k = m \text{ xor } c$
- **OTP does not provide integrity, even worse OTP is malleable**
 - Modifications to cipher-text are undetected and have predictable impact on plain-text

- **la chiave deve essere lunga quanto il messaggio**, per messaggi lunghi diventa impossibile (1gigabit di film da cifrare diventa molto scomodo da un punto di vista pratico, in quanto sarà necessaria una chiave da 1gigabit)
 - **la chiave può essere usata una volta sola**, perchè si può dimostrare che l'avversario dai due testi cifrati usando la stessa chiave potrebbe ottenere informazioni riguardo la chiave stessa. Per le proprietà dello xor infatti:
- $$C1 \oplus C2 = (M1 \oplus k) \oplus (M2 \oplus k) = M1 \oplus k \oplus M2 \oplus k = M1 \oplus M2 \oplus k \oplus k = M1 \oplus M2$$
- se dispongo del testo cifrato e del testo in chiaro, **ricavare la chiave è molto semplice**
 - **il cifrario è malleabile**: se usassi la chiave una volta non ci sono problemi. L'altro obiettivo è il tampering, ovvero l'avversario potrebbe modificare il messaggio, il problema del OTP è che mentre è sicuro rispetto all'eavesdropping, non ho alcuna garanzia sul tampering, ed ancora peggio è **MALLEABILE**: ovvero *delle modifiche al cipher text da parte dell'avversario non sono rilevabili e inoltre l'avversario riesce ad avere un controllo delle modifiche in modo tale che il ricevente decifri quello che l'avversario vuole.*



OTP is malleable

$m =$	D	A	R	E	C	E	N	T	O	E	U	R	O	A	B	O	B
$k =$	W	C	L	N	B	T	D	E	F	J	A	Z	G	U	I	R	X
$c =$	Z	C	C	R	D	X	Q	X	T	N	U	Q	U	U	J	F	Y



$c' =$	Z	C	C	R	N	B	O	P	J	N	U	Q	U	U	J	F	Y
$k =$	W	C	L	N	B	T	D	E	F	J	A	Z	G	U	I	R	X
$m =$	D	A	R	E	M	I	L	L	E	E	U	R	O	A	B	O	B

! I cifrari non sono forti rispetto all'integrità perchè sono progettati per fare altro (confidenzialità). Per questo motivo parleremo di funzioni di hash e firme digitali.

Come fa l'avversario a sapere che deve mettere esattamente i caratteri al posto di quelli? Chiaramente l'esempio è didattico, ma l'avversario potenzialmente può conoscere tutto eccetto le chiavi, quindi potrebbe conoscere la struttura del messaggio di massima, ovvero la prima parte è il comando, la seconda parte è la quantità di denaro, la terza parte è la valuta e la quarta parte è il

destinatario:

cmd
 $m = D A R E G E N T O E U R O A B O B$

La dimostrazione matematica di malleabilità:

r è la perturbazione introdotta dall'avversario, si può dimostrare che lato ricevente si riceve il testo in chiaro XOR la perturbazione, che non viene rilevata ed ha un impatto prevedibile. Lo XOR esclusivo tra r e il testo cifrato mi si propaga sul testo in chiaro. Inoltre r è totalmente indipendente dalla chiave k .

- Alice sends Bob: $c = p \oplus k$
- The adversary
 - intercepts c and transmits Bob $c' = c \oplus r$, with r called perturbation
- Bob
 - receives c' and
 - computes $p' = c' \oplus k = (c \oplus r) \oplus k = ((p \oplus k) \oplus r) \oplus k = p$
 - The perturbation goes undetected
 - The perturbation has a predictable impact on the plaintext
 - The perturbation does not depend on the key

Exercise – the flipping problem

- The problem
 - Alice and Bob use one-time pad (OTP) and have agreed on a perfectly random secret key K.
 - Alice will send Bob the answer to the question “Are you coming for dinner?” as either $pt_Y = 'Y'$ (01011001) or $pt_N = 'N'$ (01001110).
 - The adversary intercepts transmission between Alice and Bob, $ct = 00011101$, and flips Alice’s answer.
 - What should the adversary send Bob to flip Alice’s answer?

Example – the flipping problem

- Solution
 - Let $ct = pt \oplus K$ where pt may assume two values pt_Y or pt_N .
 - In order to flip Alice’s message, the adversary computes
 $ct' = c \oplus (pt_Y \oplus pt_N)$.
 - Substituting the exercise data we obtain the following. $(pt_Y \oplus pt_N) = 01011001 \oplus 01001110 = 00010111$.
 - Therefore, $ct' = ct \oplus (pt_Y \oplus pt_N) = 1001110 \oplus 00010111 = 0001010$.

Trovo le differenze in bit tra pt_Y e pt_N :

$pt_Y: 0\ 1\ 0\ \underline{1}\ 1\ \underline{0}\ \underline{0}\ 1$

$pt_N: 010\underline{0}1\underline{1}\underline{1}0$

e noto che sono opposti tra di loro (quelli sottolineati). Per invertire un bit basta fare:
bit_che_si_vuole_invertire \oplus 1 dunque la perturbazione r sarà costituita dagli 1 in corrispondenza dei bit che voglio far flippare e degli 0 in corrispondenza di quelli che voglio lasciare intatti:
 $r: 0\underline{0}0\underline{1}0\underline{1}11$

Se la risposta di Alice è $pt_Y: 010\underline{1}1\underline{0}\underline{0}1$, introducendo la perturbazione, Bob riceverà pt_N e viceversa:

$pt_Y: 01011001 \oplus r: 00010111 \rightarrow 01001110: pt_N$

$pt_N: 01001110 \oplus r: 00010111 \rightarrow 01011001: pt_Y$

Form:

✓ Esatto 1/1 Punti

2. Security of OTP

depends on the skill of the adversary

Is independent of the adversary resources ✓

depends on the amount of resources of the adversary

✓ Esatto 1/1 Punti

3. Let the ciphertext be the sequence of bit 10000110. Let k be the key 11010101. Determine the plaintext.

01010011

✗ Non corretto 0/1 Punti

4. A remote banking system uses the message <src | dst | TRANSFER | amount> to order the transfer of an amount of money from a src to a dst. Each message field is one byte long. The Bank uses OTP for encrypting the message. Alice wants to transfer 100 euros to Bob. Bob wants to make a fraud and double the amount. What's the value of the perturbation that Bob has to use? (Express the value in hexadecimal)

Inserisci la risposta

Risposte corrette: 000000AC

✓ Esatto 1/1 Punti

5. Consider an n-bit OTP. If you remove the key 0000...000

the resulting scheme is not perfect anymore ✓

the resulting scheme remains perfect iff n is prime.

the resulting scheme remains perfect

✓ Esatto 1/1 Punti

6. Let c = "HELLO" a ciphertext obtained by OTP.

"HELLO" can't be the plaintext

"HELLO" may be the plaintext ✓

"FAREWELL" may be the plaintext

Risposta 4.

Bisogna prendere la rappresentazione binaria di 100 e 200, si fa lo XOR e quella sarà la perturbazione.

Risultato: 000000AC. Tutti gli altri bit devono essere lasciati inalterati quindi si lasciano a 0.

Risposta 5.

Rimuovo la chiave a tutti 0 in quanto la chiave con tutti 0 mi restituisce sempre un ciphertext uguale al plaintext allora per questo motivo potrei pensare di rimuoverlo.

Avendo il numero di chiavi che sono meno rispetto al numero di bit violerei il teorema di Shannon.

Quindi non va rimossa come chiave anche se poi l'avversario potrebbe ottenere un testo in chiaro uguale al testo cifrato, ma non è un problema in quanto l'avversario non sa quale chiave abbiamo usato.

Risposta 6.

Se la chiave è costituita da tutti 0, il plaintext e il ciphertext coincidono. FARWELL è più lungo di HELLO quindi non potrebbe ottenersi in quanto viene fatto lo XOR **bitwise**

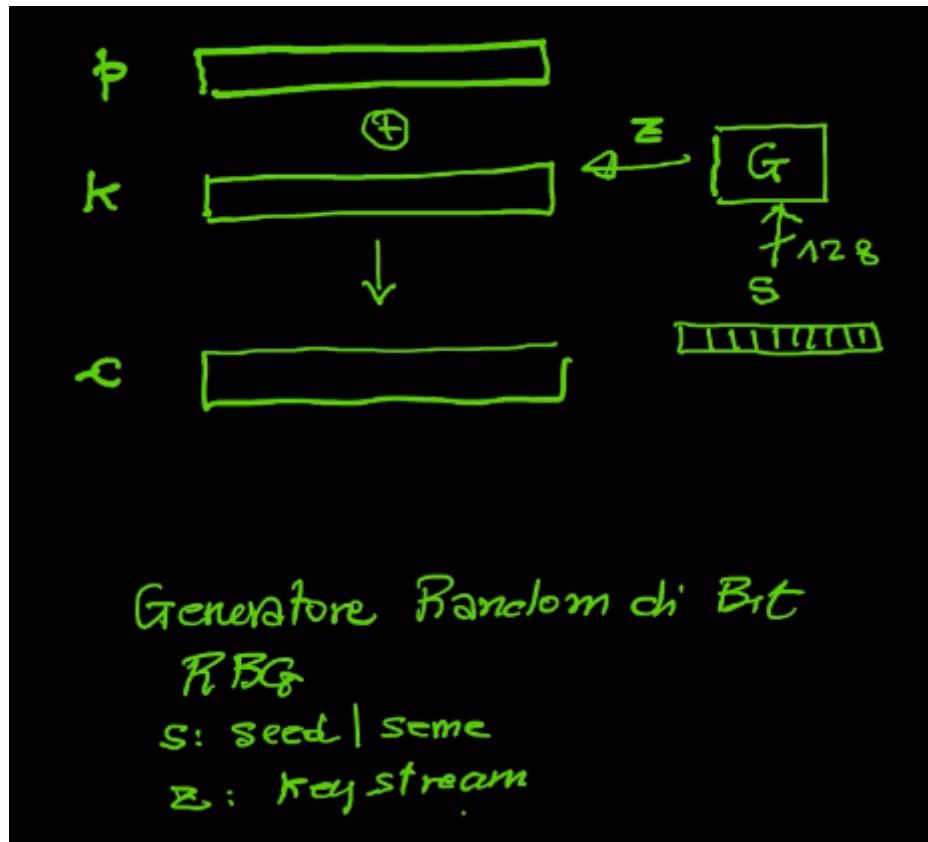
Stream ciphers

Quando viene mandata alla radio base la nostra voce viene digitalizzata e cifrata usando uno **stream ciphers**, ovvero i cifrari a caratteri.

Il problema è che abbiamo il messaggio **p** in chiaro, poi devo costruire una chiave **k** in modo perfettamente random, **la uso una volta sola**, e faccio lo XOR. Se cambio il testo in chiaro dovrò generare una nuova chiave.

Allora invece di generare una chiave in modo perfettamente random, potremmo inventare una operazione **G** (una funzione) che chiamo **GENERATORE RANDOM DI BIT** (**RandomBitGenerator**) che prende in ingresso una quantità detta **seme** **s** e produce in uscita una quantità **z** detto **key stream**. Ovvero supponiamo che riesca a inventarmi questa funzione che prende **s** come sequenza di bit in ingresso che però ha una caratteristica ovvero deve essere perfettamente random ma ha sempre una dimensione fissa, ad esempio 128 bit. Il key stream generato **z** è una sequenza di bit più lunga di **s** in modo tale da usarlo ogni volta che devo cifrare con OTP, prendendo però una porzione di **z** di volta in volta, ovvero G amplifica **s** in una sequenza molto più grande da cui prendere una porzione di bit di

volta in volta per fare OTP.



Nello schema originario (OTP) avevamo solamente il p, c e la chiave k perfettamente random, adesso solo s deve essere perfettamente random.

s diventa la nuova chiave che deve rimanere segreta ma che posso mantenerla per tanto tempo, in quanto il keystream è molto lungo e di volta in volta estraggo solo una piccola sequenza.

La chiave prima era perfettamente random e anche ora lo è (siamo passati da k ad s). Inoltre la chiave è molto lunga quindi sembrerebbe essere tutto OK.

MA dobbiamo ricordarci cosa dice Shannon, ovvero che la chiave deve avere una dimensione pari al messaggio e deve essere perfettamente random, la chiave è s ma non è più della stessa lunghezza del messaggio. Avendo fatto questa scelta il numero delle chiavi sarà 2^s ovvero 2^{128} semi in questo esempio, quindi il numero delle chiavi è molto minore rispetto al numero di messaggi, in quanto potrei avere messaggi di ordine più grande (un film ad esempio!) quindi **questo cifrario non è più perfetto !!!**.

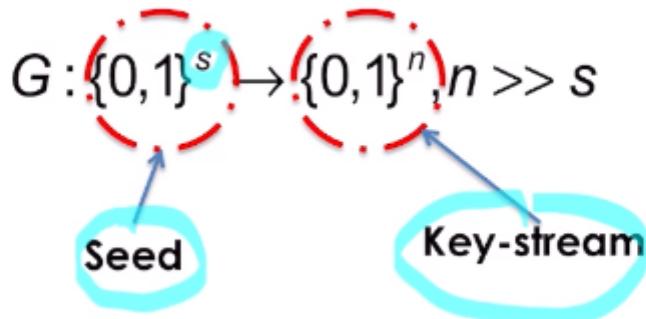
Stiamo quindi cambiando la definizione di sicurezza, trasformando **difficult** in **impossible**.

Un cifrario perfetto infatti è fattibile in teoria ma non in pratica, quindi bisogna necessariamente ritornare a qualcosa di non ideale (ovvero solo **difficult**). Questa riduzione di sicurezza la spendo con un dispositivo più facilmente usabile nella applicazione pratica.

Da un punto di vista matematico, **difficult** vuol dire in teoria della complessità che non esiste alcun algoritmo efficiente che è in grado di violare il cifrario in tempo polinomiale, ovvero ci vorrà una grande quantità di tempo per violare il cifrario.

La grande modifica introdotta nello schema è il componente G , progettandolo bene, sebbene non sia più perfetto il cifrario sarà almeno sicuro da un punto di vista computazionale.

- **Idea:** replace random key by pseudo-random key
- **Pseudo-Random Generator G** is an **efficient** and **deterministic** function that takes a **seed** (short) and produces a **key-stream** (long)



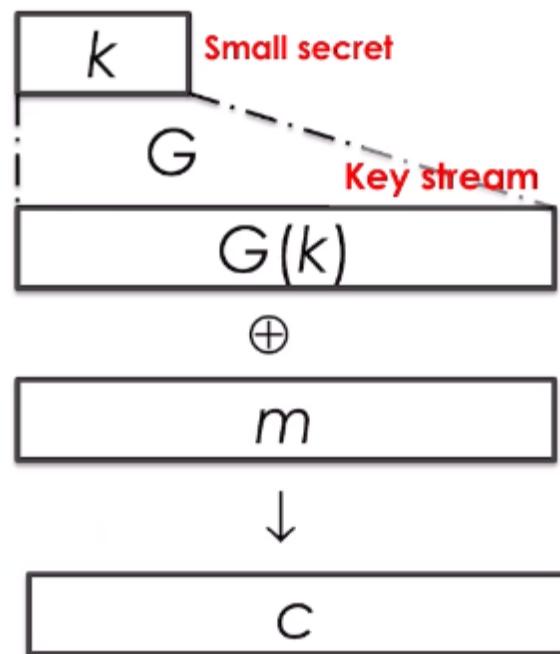
Making OTP practical (2/3)

Larger secret key stream computed from a short seed k

$$c = E(k, m) = G(k) \oplus p$$

$$m = D(k, c) = G(k) \oplus c$$

This cipher is called **stream cipher**



Making OTP practical (3/3)

- Is a **stream cipher perfectly secret?**
 - No, #keys < #msg (Shannon's theorem violated)
- So, we need a new definition of security!
 - Security will depend on the specific PRG
 - PRG must appear random, unpredictable, and indistinguishable from a TRG to a limited adversary
 - There exist no efficient algorithm to distinguish PRNG output from a TRG output (**computational security**)
 - Computationally Secure Pseudo Random Number Generatore (CSPRNG)

Progettare questi G è molto difficile.

La chiave k doveva essere generata in modo *truly random*. L'esempio di truly random era il lancio della moneta. In un'applicazione informatica si usano ovviamente altri fenomeni, ad esempio prendendo un oscillatore elettronico che produce un'onda a una certa frequenza. L'oscillatore però non produrrà a quella frequenza precisa in quanto sono presenti delle fluttazioni (variabili aleatorie).

Quando introduco G non è più truly random ma diventa un algoritmo, una funzione, quindi G non produce numeri random infatti viene chiamato **Pseudo Random Generator**. Se osservo il lancio della moneta, e l'ho lanciata 10 volte, sapere i 10 risultati di prima non influenza l'11° lancio. Usando un algoritmo, i bit prodotti prima invece potrebbero influenzare o potrebbero servirmi a predire quale sarà il prossimo bit. Questo è un problema in quanto se qualcuno mi scopre un pezzo del keystream potrebbe essere in grado di prevedere anche le chiavi future.

Prendo una sequenza di bit in modo perfettamente random la seq1 poi prendo una sequenza seq2 generata da G con un certo seme. Se G è fatto bene, le due sequenze devono essere indistinguibili. L'altra proprietà altrettanto fondamentale è che si gli do un prefisso di 20 bit, non sia in grado un attaccante di prevedere il 21° bit con una probabilità maggiore di 0.5. Se per un attaccante intuire queste due condizioni (distinguere le sequenze e prevedere il 21° bit) è un task difficile, allora G è fatto bene.

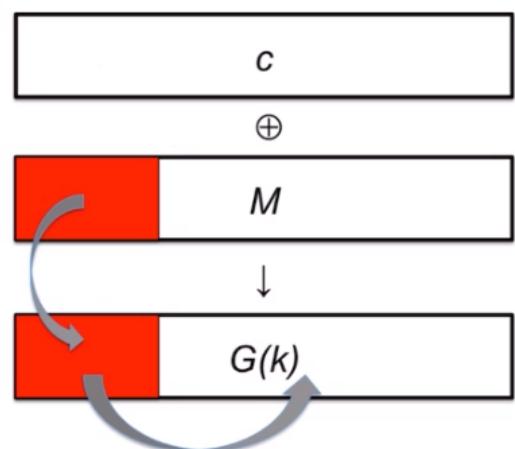
? Perchè deve essere impredicibile?

Supponiamo di avere a disposizione del testo cifrato c e supponiamo che l'attaccante conosca del testo in chiaro un prefisso (quello in rosso all'inizio). Questa situazione nella realtà è probabile in quanto molto spesso i messaggi inviati sulla rete usano degli header sempre simili per trasportare dei metadati utili alla comunicazione ma che non hanno alcun payload. L'attaccante potrebbe fare lo XOR tra il prefisso del messaggio e il prefisso del testo cifrato, ottenendo un "prefisso" della chiave. Se il generatore fosse prevedibile, da questo prefisso iniziale della chiave potrei iniziare ad indovinare i bit che vengono dopo con alta probabilità, dunque potrei andare a ricavare l'intera chiave e quindi anche il pezzo di testo in chiaro che mi manca.

Ecco perchè G deve essere **unpredictable**!

What if PRG is predictable

- If PRG is predictable then OTP is not secure!
 - If the adversary is able to determine a **prefix** of M then
 - (s)he gets able to determine a **prefix** of $G(k)$
 - If $G(k)$ is predictable, the (s)he gets able to determine all $G(k)$ and thus
 - Decrypt C



Statistical tests

- **Good news: it is possible to measure the quality of a random bit generator**
 - **Statistical tests** probabilistically determine whether sample output sequences possess certain attributes that a truly random sequence would be likely to exhibit
 - Ex.: in a sequence X, $| \#1(X) - \#0(X) |$ is “small”
 - Ex.: in a sequence X, $| \#00(X) - \text{len}(X)/4 |$ is “small”
 - A generator may be rejected or accepted (= not rejected)
- **Bad news: tests prove necessary conditions only**

Questi generatori esistono, quelli pseudorandom sono usati in tanti campi ad esempio in tutti quei settori in cui si fa simulazione ad eventi (medicina, economia...). Quei generatori vanno bene per la simulazione ma non per la sicurezza in quanto sono **PREVEDIBILI**. Questo perchè sebbene possano essere indistinguibili da una sequenza random, dato un prefisso riesco a predire con alta probabilità i bit che verranno generati successivamente.

? Se ho un generatore esistono dei test che mi permettono di capire se il generatore è buono? **SI ma** danno solo una condizione necessaria **MA NON SUFFICIENTE**. Pur passando i test, un generatore potrebbe non essere sicuro.

18/02/2023

La sicurezza computazionale è una sicurezza imperfetta in quanto non è definita alla Shannon, in linea di massima è possibile ma magari vuole troppo tempo o una quantità elevata di risorse e denaro. E' definita come *il livello percepito di computazione richiesta per difendersi da un attacco, basato sul miglior attacco conosciuto e sul tipo di avversario ipotizzato*.

Computational security

- A cipher is **computationally (practically)** secure if
 - the **perceived** level of computation required to defeat it, using the **best attack known**, exceeds, by a comfortable margin, the computation resources of the **hypothesized** adversary
 - Now, the adversary is assumed to have a limited computation power

Le parole evidenziate sono importanti in quanto non siamo nel caso di OTP in cui la sicurezza è perfetta e dunque non mi interessa chi ho di fronte, nessuno di loro è in grado di ottenere informazioni dal testo cifrato. Nelle condizioni in cui operiamo adesso invece siamo in un mondo di sicurezza computazionale.

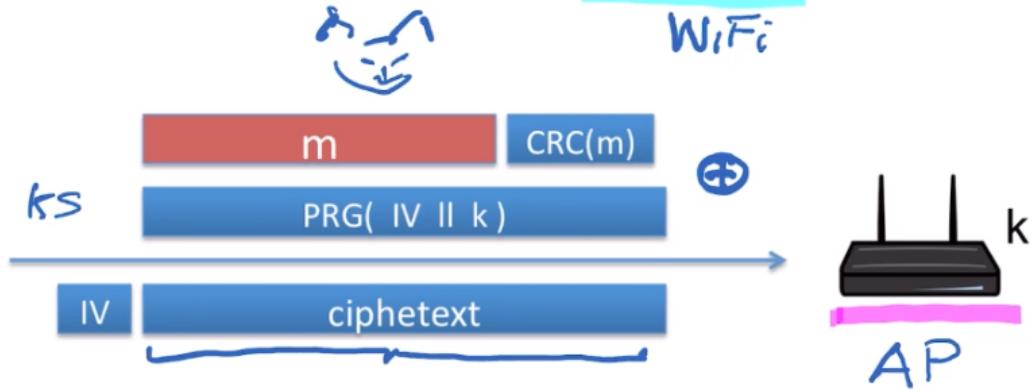
❓ Devo difendermi da che tipo di avversario? Perchè cambia l'expertise e il budget del mio avversario.

❓ Qual'è il miglior attacco conosciuto ad oggi? Ad oggi non esiste un algoritmo ad esempio che permette di distinguere un generatore random da uno non random, ma se domani questa condizione dovesse cambiare, vuol dire che questo generatore che veniva percepito come sicuro deve essere sostituito.

Esempio: 802.11b WEP

Real world examples: 802.11b

WEP



- A new IV for each new message
 - Key is fixed (104-bits)
 - IV avoids 2TP
- WEP used a good PRG (RC4), but...

$PRG(s) \rightarrow \text{keystream}$

Inizialmente l'esigenza era quella di proteggere tutte le comunicazioni tra il client e l'access point, in quanto la comunicazione avviene in aria e quindi il mio avversario potrebbe ascoltarla. **WEP** è un **cifrario a caratteri**, che cercava di proteggere il messaggio m . $CRC(m)$ è il codice di ridondanza per il recupero degli errori. Inoltre viene generato un seme dal quale si genera il **Keystream**. Il seme s è fatto da due quantità:

- **la chiave**: la password del wifi dopo alcune manipolazioni in qualche modo diventava la chiave usata. La dimensione è di 104 bit.
- **IV: il vettore di inizializzazione** è un numero che potrebbe essere creato tramite un contatore. L'idea è che per ogni nuovo messaggio m ci vuole un nuovo **IV**. La dimensione è di 24 bit.

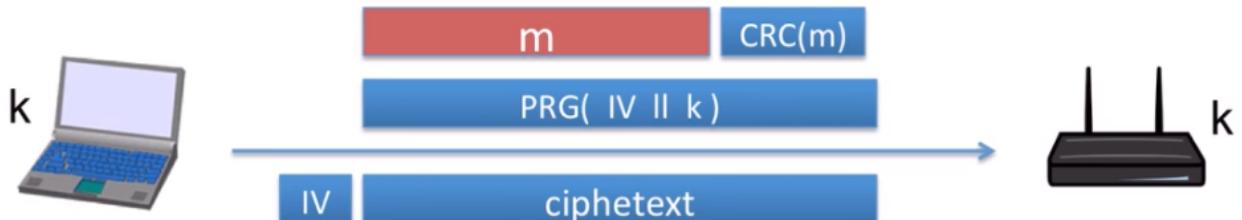
Sul lato ricevitore, si riceve il cyphertext, dunque per trovare il testo m dovrà fare lo XOR tra c e $PRG(IV \parallel K)$.

L'IV viene trovato all'inizio del messaggio ricevuto, spedito in chiaro.

Avendo 24 bit, posso andare a rappresentare tutti gli interi che vanno da 0 a $2^{24} - 1$. In tutto ho **circa 16 milioni** di vettori di inizializzazione. Dato che IV viene creato progressivamente e che il keystream dipende dal IV, significa che avrà al massimo 16 milioni di keystream. Avendo un IV per ogni messaggio, una volta trasmessi 16 milioni di messaggi, inizierà a generare nuovamente gli stessi keystream perché gli IV ricominciano dal primo. Se riutilizzo il Keystream, l'avversario vede che viene riutilizzato (vede riapparire lo stesso IV più volte), dunque inizia a fare quella serie di attacchi mirati ai keystream ripetuti.

⚠️ Problemi ⚠️ :

- Il problema è che 24 bit è piccolo, dunque la ripetizione si verifica abbastanza presto con una quantità di traffico abbastanza piccolo (pochi milioni di messaggi)
- Il numero 24 è scritto nella standard dunque come produttore di schede di rete devo necessariamente rilasciare prodotti che siano standardizzati, dunque come produttore sono costretto a usare un IV da 24 bit.
- Pensando al client, esso all'accensione riparte a generare l'IV dal valore 0, dunque per l'attaccante basterà mettersi in ascolto all'accensione, in quanto tutti i messaggi all'accensione del client partiranno sempre con lo stesso IV.

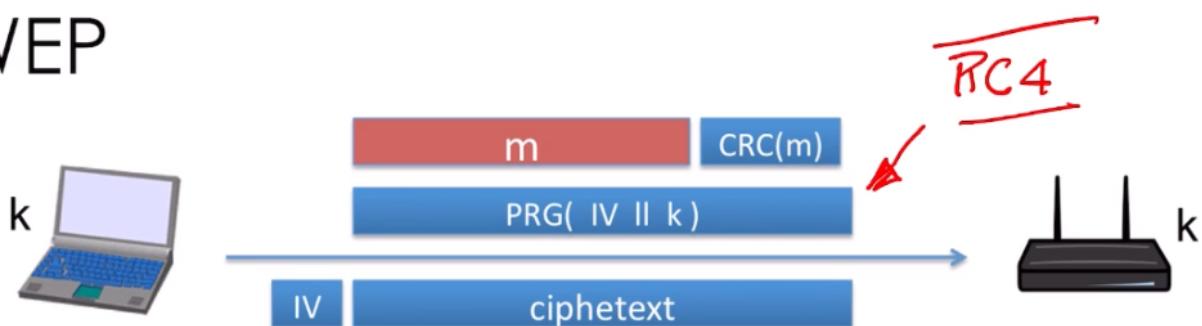


- Length of IV: 24 bits
 - The length is “carved” in the standard
 - Repeated IV after $2^{24} \approx 16M$ frames
 - On some 802.11 cards IV resets to 0 after power cycle

Questo è un esempio di protocollo **INSICURO** costruito con componenti **SICURI**.

Lo pseudo random generator usato è l'**RC4**, che è debole rispetto all'**attacco alle related keys**!

Real world examples: 802.11b WEP



RC4 is weak w.r.t. related keys

Key for frame #1: 1 | | k

Key for frame #2: 2 | | k

Key for frame #3: 3 | | k

...

Related keys

- FMS 2001 attack can recover K in 10^6 frames (now 40 Kframes)
- Avoid related keys!**

Dato che la chiave è fissa e il IV è progressivo, il seme che veniva generato aveva questa forma:

1 || K

2 || K

3 || K

...

queste chiavi sono correlate con K, dunque in questa situazione il generatore non è così impredicibile.

Attacco delle chiavi correlate

Da Wikipedia, l'enciclopedia libera.

In crittografia un **attacco delle chiavi correlate** è una forma di **crittanalisi** in cui l'attaccante può osservare l'operato di un cifrario con svariate **chiavi** diverse i cui valori sono inizialmente ignoti ma legate da alcune relazioni matematiche che egli conosce. Ad esempio, l'attaccante potrebbe sapere che gli ultimi 80 bit delle chiavi sono sempre identici anche se egli non conosce inizialmente il valore di questi bit.

A prima vista appare come un modello irrealistico: sembra infatti improbabile che un attaccante possa persuadere un crittografo umano a cifrare dei testi in chiaro utilizzando numerose chiavi segrete legate tra di loro in una qualche maniera. Però la crittografia moderna è implementata utilizzando protocolli informatici molto complessi, spesso non controllati dai crittografi: ecco perché, in molti casi, un attacco correlato alla chiave è facilmente attuabile.

WEP [modifica | modifica wikitesto]

Un esempio molto noto di un protocollo crittografico caduto sotto un attacco delle chiavi correlate è il **Wired Equivalent Privacy**, meglio noto come **WEP**, utilizzato nelle reti senza fili **WiFi**. In una rete protetta dal WEP ogni scheda WiFi ed ogni **access point** condividono la stessa chiave WEP e la cifratura utilizza l'algoritmo RC4, un **cifrario a flusso**: in questo tipo di cifrari è essenziale che la stessa chiave non sia mai usata più di una volta. Per evitare che ciò accada, il WEP include in ogni pacchetto del messaggio un **vettore di inizializzazione** (VI) di 24 bit: la chiave RC4 per quel pacchetto è il VI concatenato alla chiave WEP. Le chiavi WEP devono essere cambiate manualmente e questo in genere accade di rado: un attaccante può quindi assumere che tutte le chiavi utilizzate per cifrare i pacchetti siano correlate da un VI noto. Questo fatto ha esposto il WEP ad una serie di attacchi che si sono rivelati devastanti: il più semplice di essi si basa sul fatto che un VI di 24 bit permette meno di 17 milioni di combinazioni differenti. A causa del **paradosso del compleanno** ci si può attendere che ogni 4096 pacchetti due di essi condividano lo stesso VI e, quindi, la stessa chiave RC4, permettendo ai pacchetti di essere attaccati.

Attacchi più potenti approfittano di certe **chiavi deboli** dell'RC4 che permettono alla stessa chiave WEP di essere scoperta. Nel 2005 agenti dell'**FBI** hanno dimostrato questa possibilità utilizzando strumenti software ampiamente diffusi [1].

Prevenzione degli attacchi delle chiavi correlate [modifica | modifica wikitesto]

Per prevenire questi attacchi, è stato introdotto un sostituto del WEP denominato **Wi-Fi Protected Access** (WPA), che utilizza 3 livelli di chiave: una **chiave primaria**, una **chiave operativa** ed una **chiave RC4**. La chiave primaria è condivisa con ogni **client** ed **access point** ed è utilizzata in un protocollo denominato **TKIP** il quale cambia le chiavi operative molto frequentemente: questo serve a contrastare i metodi di attacco noti. Le chiavi operative sono poi combinate con un VI più lungo, 48 bit, per formare la chiave RC4 per ogni pacchetto. Questo processo è simile a quello del WEP quel tanto che basta per poter essere utilizzato nelle schede Wi-Fi di prima generazione, alcune delle quali implementano porzioni del WEP in hardware. Comunque non tutti gli **access point** di prima generazione possono eseguire il WPA.

Un altro approccio, più conservativo, è quello di impiegare un cifrario progettato per resistere agli attacchi correlati alla chiave, come ad esempio quelli che incorporano un robusto algoritmo di **schedulazione della chiave**: una nuova versione del protocollo denominata **WPA2** utilizza, anche per questo motivo, il **cifrario a blocchi AES** al posto dell'RC4. Molte delle schede di rete più vecchie non possono però usare il WPA2.

Un approccio più aggressivo è quello di utilizzare un **generatore di numeri casuali** per generare veri numeri casuali oppure, come fanno molti sistemi, usare nuove chiavi per ogni sessione.

In questo caso, non potendo sostituire WEP, hanno rilasciato delle fix software realizzate sul firmware della scheda tramite i driver della scheda, per ottenere la versione più sicura (WPA).

Form

1. A stream cipher

- cannot be a perfect cipher ✓
- can be a perfect cipher
- will be a perfect cipher in the future

Blurred text input field

✓ Correct 1/1 Points

3. Statistical tests

- provide a sufficient condition
- provide a necessary condition ✓
- provide a sufficient and necessary condition

✓ Correct 1/1 Points

4. A Random Number Generator for security applications

- can be predictable
- must be predictable
- must be unpredictable ✓

✓ Correct 1/1 Points

5. A True Random Number Generator

- is based on a random physical process ✓

✓ Correct 1/1 Points

5. A True Random Number Generator

- is based on a random physical process ✓
- is based on a well-designed algorithm
- is periodic

✓ Correct 1/1 Points

6. The output of a Pseudo-Random Number Generator (PRNG)

- is random
- could be random
- cannot be random ✓

Block ciphers

Il cifrario a caratteri che abbiamo visto come evoluzione di OTP è come se cifrasse un bit alla volta. Nel caso dei cifrari a blocchi invece viene preso un blocco di bit (ad esempio 64 bit) di testo in chiaro che viene cifrato ogni volta. Non viene più fatta la cifratura un bit alla volta.

Block cipher

- Block ciphers break up the plaintext in blocks of fixed length n bits and encrypt one block at time



- $E: \{0,1\}^n \rightarrow \{0,1\}^n$ $D: \{0,1\}^n \rightarrow \{0,1\}^n$
- Mathematically, E is a permutation
 - one-to-one, invertible function

$\{0,1\}^n$ significa che vengono considerate sequenze di n bit. Dunque se in ingresso ho n li avrò anche in uscita, usando una chiave k . Da un punto di vista matematico, E definisce una permutazione. La cosa interessante è che di questi algoritmi di encryption E ne posso avere tanti quant'è la lunghezza delle chiavi k .

Ovvero avrò l'algoritmo E_0 associato a k_0 , E_1 associato a k_1 etc...

Ogni algoritmo genera permutazioni diverse ed Alice e Bob dovranno mettersi d'accordo su quale algoritmo usare. Dunque per evitare che l'attaccante inferisca l' E tramite brute-force chi comunica dovrà usare un numero elevato di k . Ad oggi vengono usate chiavi da almeno 128 bit.

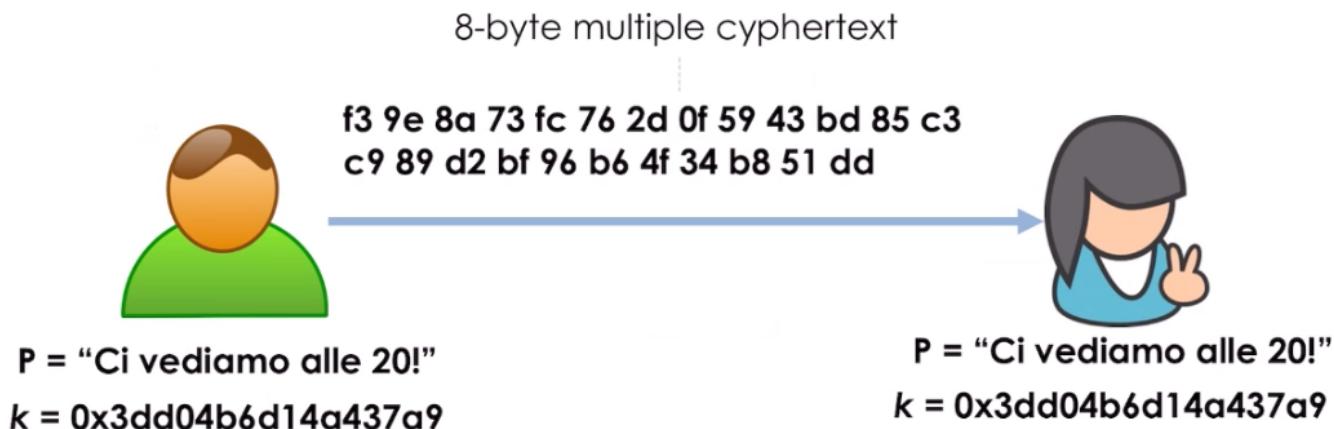
Esempio di block cyphers: DES

La chiave è rappresentata in formato esadecimale, con 56 bit (chiave molto corta!) infatti DES non è più sicuro al giorno d'oggi, ma lo è stato per molti anni (fino agli 90 circa) grazie al fatto che fosse "difficult" ricavare la chiave tramite brute-force.

An example: DES (CBC)

$n = 64$ bit

$k = 56$ bit



Il numero delle chiavi deve essere molto grande per rendere un cifrario a blocchi "sicuro alla Shannon" dunque è impossibile nella pratica!



Secure block cipher

- Perfectly secure block cipher
 - implements a truly random permutation
 - is not physically feasible: too many keys!
- Practical block cipher
 - the encryption function corresponding to a randomly chosen key **appears** as a randomly chosen permutation to a **limited adversary**

Esistono alcune caratteristiche di un cifrario che interessano:

- **n**: la dimensione del blocco, quanti bit alla volta andiamo a cifrare
- **k**: la dimensione della chiave

Canonical example

Block ciphers

- DES n = 64 bits, k = 56 bits
- 3DES n = 64 bits, k = 168 bits
- AES n = 128 bits k = 128, 192, 256 bits

Performance (AMD Opteron, 2.2 GHz)

- RC4 126 MB/s
- Salsa20/12 643 MB/s
- Sosemanuk 727 MB/s
- 3DES 13 MB/s
- AES-128 109 MB/s

Lo standard ad oggi in vigore è AES-128, è abbastanza probabile che in futuro si migrerà sui 256 bit di chiave.

Di ogni algoritmo, per esempio DES nell'immagine in basso, ci interessa una tabella che mi dica quale è il best-known attack, ovvero il migliore attacco conosciuto al momento e quali sono e quante sono le

risorse necessarie per fare quell'attacco.

Crypto-analysis of DES

attack method	data complexity		storage complexity	processing complexity
	known	chosen		
exhaustive precomputation	—	1	2^{56}	1^*
exhaustive search	1	—	negligible	2^{55}
linear cryptanalysis	2^{43} (85%)	—	for texts	2^{43}
	2^{38} (10%)	—	for texts	2^{50}
differential cryptanalysis	—	2^{47}	for texts	2^{47}
	2^{55}	—	for texts	2^{55}

* Table lookup

%: probability of success

LC is the best known analytical attack but it is considered “unpractical”

Ad esempio la ricerca esaustiva della chiave prevede di prendere, per ogni possibile chiave, il testo in chiaro p e cifrarlo con k . Se ottengo lo stesso c intercettato sulla rete, ho trovato la chiave. Questa metodologia in caso di cifrario perfetto non servirebbe, ma essendo DES non perfetto può funzionare. Questo attacco mi costa solamente per il numero di chiavi, in quanto dovrò provare 2^{56} chiavi.

Statisticamente posso essere sfortunato (**dunque provarle tutte e 2^{56}**) oppure fortunato (**trovarla al primo colpo**), dunque essendo le probabilità al 50% dovrò provare $2^{56}/2$ volte = 2^{55} . Questo numero AD OGGI è troppo basso, ovvero compiere 2^{55} cicli si riescono a completare in poco tempo e con risorse alla portata di quasi tutti ed è il motivo per cui DES non viene più considerato uno standard di sicurezza per le applicazioni moderne.

L'attacco di crittoanalisi differenziale è un attacco che per implementarlo in questo caso ho bisogno di 2^{47} coppie p e c . Inoltre la memoria di storage di cui ho bisogno in questo caso non è stimata in quanto non è particolarmente interessante, ma dovrò fare 2^{47} cicli for. Ecco perchè questo tipo di attacco non è stato mai messo in pratica nella realtà contro DES in quanto il numero di coppie necessarie da procurarsi è altissimo, in quanto poi ognuna di queste chiavi dovrà essere memorizzata.

Storia del DES

DES viene standardizzato nel **1978**, e prima di essere standardizzato il NIST lanciò una competizione per chiedere alla comunità accademica e non solo di poter proporre il proprio algoritmo di cifratura. Questa infatti fu anche la prima volta nella storia che un algoritmo di crittografia veniva concepito in ambito civile e non ambito militare. Questo perchè nei primi anni 70 cominciava ad affermarsi Internet e

ci si rese conto che bisognava andare a crittografare le transazioni economico finanziarie. La competizione venne vinta da **IBM** con il proprio algoritmo **Lucipher**. **DES** ha un blocco a 64 bit e una chiave a 56 bit. **Lucipher** nella sua versione originaria aveva un blocco a 64 bit e una chiave a 64 bit. **La chiave nel caso del DES è più piccola.** Le modifiche richieste dalla **National Security Agency** sono state di ridurre la chiave e modificare il funzionamento dell'algoritmo. Le motivazioni dietro queste richieste rimangono ad oggi ancora sconosciute. Nel **1980** due ricercatori inventano l'algoritmo di crittoanalisi differenziale. Quando NSA richiede quelle modifiche, tutti pensano che siano state richieste apposta affinchè l'NSA potesse romperlo in qualsiasi momento. L'NSA probabilmente conosceva già la crittoanalisi differenziale nel 1980, dunque Lucifer nel 1978 sarebbe potuto essere vulnerabile ad un attacco di tipo crittonalisi differenziale ma al tempo stesso richiedendo una riduzione della lunghezza della chiave avrebbe permesso, a patto di avere grandi risorse (che solo l'NSA o enti governativi avevano) di poter rompere in qualsiasi momento il DES. Hanno dunque rinforzato l'algoritmo per il pubblico ma lo hanno reso vulnerabile solo per entità molto potenti computazionalmente come il governo americano.

Ci si rese conto che con 56 bit di chiave, DES non era più adeguato, allora venne lanciato intorno agli anni 90 un nuovo concorso per standardizzare un nuovo algoritmo di cifratura. Nel frattempo, come soluzione temporanea, fu deciso di usare la cifratura multipla...

Cifratura multipla

Venne lanciata la DES challenge, in cui si rese pubblica una parte del testo in chiaro e la challenge consisteva nel trovare il restante testo in chiaro.

DES challenge

p = "The unknown messages is: c1 c2 c3 XXX ..."

Goal. Find $k \in \{0,1\}^{56}$ s.t. $c_i = \text{DES}(k, p_i)$, $i = 1, 2, 3, \dots$

- **1997:** Internet search – 3 months
- **1998:** EFF machine (deep crack) – 3 days (250K\$)
- **1999:** Combined search – 22 hours
- **2006:** COPACABANA (120 FPGAs) – 7 days (10K\$)

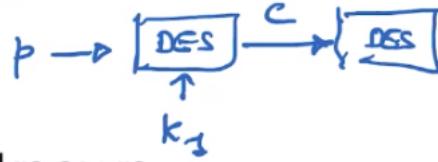
⇒ **56-bit ciphers should not be used**

Ci furono vari tentativi, partendo da tentativi puramente software nel 1997 in cui si prese una parte delle chiavi e si distribuì su un certo numero di macchine su internet disponibili a partecipare a questo progetto. Ci vollero circa 3 mesi per trovare tutto il testo in chiaro. Fin quando nel 2006 ci fu

l'esperimento di COPACABANA in cui in 7 giorni provando tutte le chiavi era possibile ottenere il plain text completo. Dunque si capì che DES era sicuro ma la chiave usata dall'algoritmo era troppo piccola!

In attesa di un nuovo standard l'intuizione fu quella di cifrare più volte il medesimo testo in chiaro:

Multiple encryption



- DES is a secure cipher
 - No efficient cryptanalysis is known
- DES key has become too short
- Can we improve the security of DES by encrypting two or more times?
 - Double-encryption brings no advantage
 - Meet-in-the-middle attack
 - Triple Encryption
 - 3DES-EDE (ANSI X9.7 / ISO 8732)
 - Backward compatibility
 - Exhaustive key search: 2^{118}

Si vide che anche il double encryption era soggetto ad un attacco di tipo Meet In The Middle, allora si provò ad inserire un terzo stadio di cifratura ottenendo la cifratura tripla ottenendo un sistema molto più sicuro. In realtà anche nel **3DES** è possibile riprodurre il Meet In The Middle che però richiede un numero pari a 2^{118} cicli. Dunque 3DES viene considerato sicuro e negli utilizzi pratici è usatissimo. Lo svantaggio è che dovendo cifrare 3 volte, **3DES è molto lento**. Inoltre DES essendo stato progettato negli anni 70 fu progettato per essere eseguito su hardware. Dunque la versione hardware di DES è più efficiente della versione software di 3DES. Ad esempio AES, che è stato poi definito come standard minimo per la sicurezza odierna, è stato progettato per essere prestazionale sia in hardware che in software.

Form

Incorrect 0/1 Points

2. A block cipher with $n = 64$ and $k = 64$

- is certainly perfect
- is certainly not perfect ✓
- may be perfect

Correct 1/1 Points

3. A perfect block cipher

- does not exist.
- exists but it is not physically feasible ✓
- exists and it is physically feasible

Correct 1/1 Points

4. 5DES is not implemented because

- it is insecure
- its increased security level does not justify the loss of performance ✓
- it is as secure as 3DES

Correct 1/1 Points

5. Consider a triple-monoalphabetic substitution cipher in which any given character is encrypted three times by means of three different permutations of the alphabet.

- The cipher is three times less secure than the original monoalphabetic substitution cipher
- The cipher is three times more secure than the original monoalphabetic substitution cipher
- The cipher is as secure as the original monoalphabetic substitution cipher ✓

Correct 1/1 Points

6. In a block cipher, a large key

- is a sufficient condition for security
- is a necessary condition for security ✓
- is a necessary and sufficient condition for security

[Go back to thank you page](#)

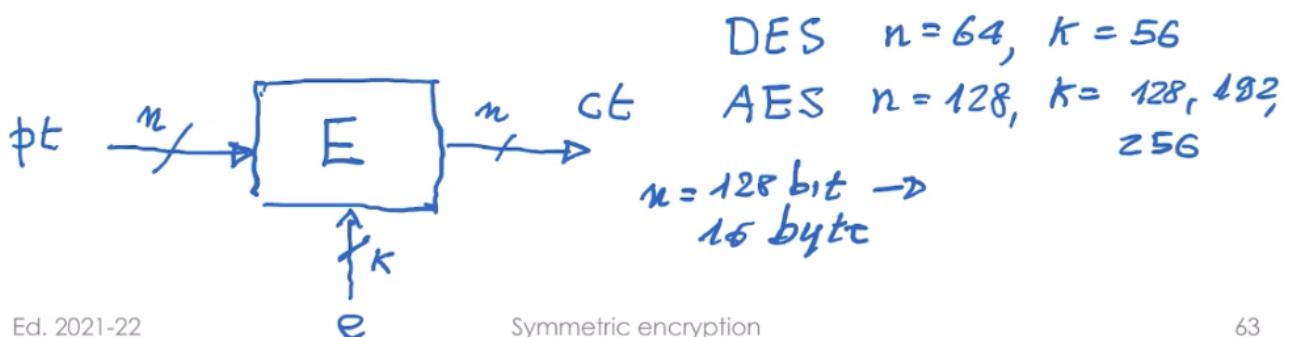
25/02/2023

Un cifrario a blocchi abbiamo detto che cifra blocchi di bit in ingresso. Dunque in ingresso ho \boxed{n} bit, in

uscita ho n bit e utilizzo k bit di chiave. Dunque un cifrario a blocchi ha 2 ingressi e una sola uscita:

Encryption Modes

- A block cipher encrypts PT in fixed-size n -bit blocks
- When the PT len exceeds n bits, there are several modes to the block cipher
 - Electronic Codebook (ECB)
 - Cipher-block Chaining (CBC)



Ed. 2021-22

Symmetric encryption

63

Se supponiamo di usare l'algoritmo AES che lavora con $n = 128$ bit \rightarrow 16 byte, avremmo un problema in quanto se il mio cifrario lavorasse solo su 16 byte come mostrato in figura (che corrispondono a 16 caratteri ASCII) potremmo trasmettere poca informazione alla volta usando una sola chiave.

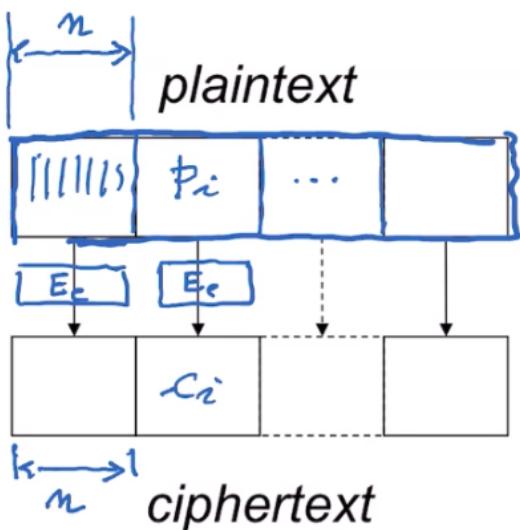
? Cosa succede quando la quantità di dati da cifrare è molto più grande della dimensione della chiave?

Per risolvere questo problema si usano le **modalità di cifratura**. Ne esistono tante che servono a risolvere anche problemi diversi da quello presentato. Noi ci limiteremo ai due più conosciuti: **ECB** e **CBC**.

✓ L'ipotesi che facciamo è che il **plaintext è un multiplo del blocco**. Questa ipotesi la rilasseremo più avanti quanto introdurremo il padding.

ECB Electronic Codebook

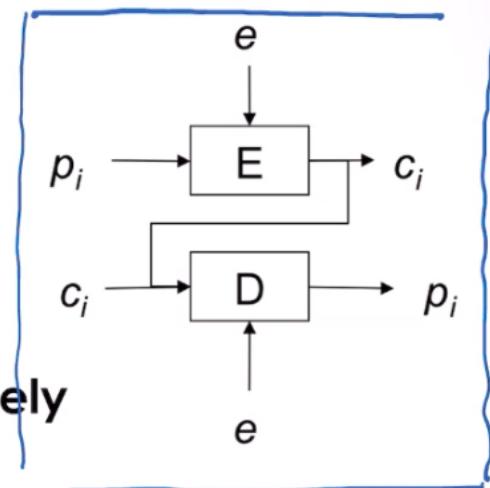
Electronic codebook



IPOTESI
plaintext è
un multiplo del
blocco

$$\forall 1 \leq i \leq t, c_i \leftarrow E(e, p_i)$$

$$\forall 1 \leq i \leq t, p_i \leftarrow D(e, c_i)$$



PT blocks are encrypted separately

Ed. 2021-22

Symmetric encryption

65

La modalità di cifratura ECB prende il plaintext, lo affetta, ogni fetta ha la dimensione di un blocco, al cui interno ad esempio per AES per ogni blocco ci sono 128 bit (16 byte), ed ogni blocco viene cifrato con chiave e . Quello che ottengo è il testo cifrato suddiviso in n blocchi.

ECB - properties

• PROS

- No block synchronization is required
- No error propagation
 - One or more bits in a single CT block affects decipherment of that block only
- Can be parallelized

• CONS

- Identical PT results in identical CT
 - ECB doesn't hide data pattern
 - ECB allows traffic analysis
- Blocks are encrypted separately
 - ECB allows block re-ordering and substitution

Pro:

✓ La cifratura può essere parallelizzata: se ho ogni blocco può essere cifrata per conto proprio

posso avere diverse CPU che parallelizzano il calcolo di cifratura

 **Non c'è propagazione dell'errore tra i diversi blocchi**

Contro:

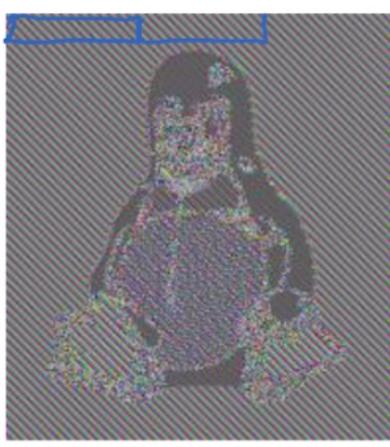
 **Testi in chiaro uguali producono testi cifrati uguali.** Infatti i blocchi vengono cifrati separatamente con la chiave e , ma se i blocchi sono uguali otterrò sempre lo stesso output. Da un punto di vista della sicurezza è un problema in quanto ECB:

- **non nasconde i data pattern:** i blocchi uguali di testo in chiaro me li ritrovo cifrati in modo uguale anche nel testo cifrato. Mi sto allontanando dal cifrario perfetto perchè stiamo rilasciando all'attaccante delle informazioni sulla struttura del testo in chiaro.

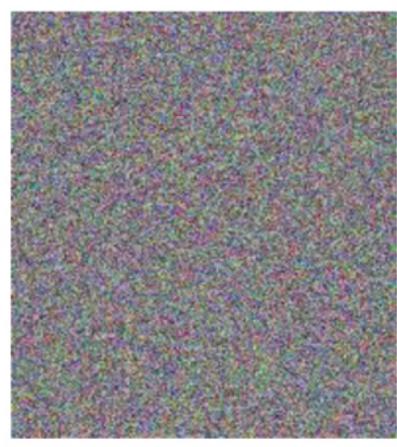
Ad esempio se prendiamo questa immagine come sequenza di byte, dove ogni byte mi rappresenta un pixel dell'immagine, prendendo ad esempio i pixel di colore bianco in alto a sinistra indicati, questi vengono cifrati in modo uguale nell'immagine al centro:



Plaintext



ECB encrypted



Non-ECB encrypted

BITMAP

128 1G

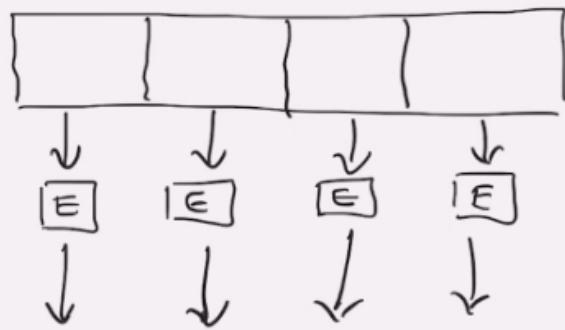
I colori sono alterati a causa del processo di cifratura ma la struttura dell'immagine anche nel testo cifrato viene un pò mantenuta, non è esattamente quella di partenza ma una qualche similitudine la ottengo. Ricorda infatti un pò una cifratura monoalfabetica, in quanto ho solo "spostato" il colore (dal bianco al grigio rigato) ma il layout del contenuto viene mantenuto.

Usando invece CBC (la terza immagine sulla destra) i data pattern vengono nascosti come vedremo dopo.

- **permette l'analisi del traffico:** permette ovvero di capire quali tipi di messaggi vengono scambiati andando a sfruttare ad esempio i metadati sempre uguali e quindi cifrati sempre nello stesso modo utilizzati nella maggior parte dei protocolli.

 **Permette il riordino o sostituzione dei blocchi:** supponiamo infatti di aver cifrato il testo in chiaro p_t con la modalità ECB ottenendo il testo cifrato c_t . Lato ricevente, applicherà la decifratura usando la medesima chiave e ottenendo il testo in chiaro di partenza. Dato che ciascuno di questi blocchi è cifrato separatamente, l'attaccante potrebbe prendere due blocchi casualmente e scambiarli di posto:

PT

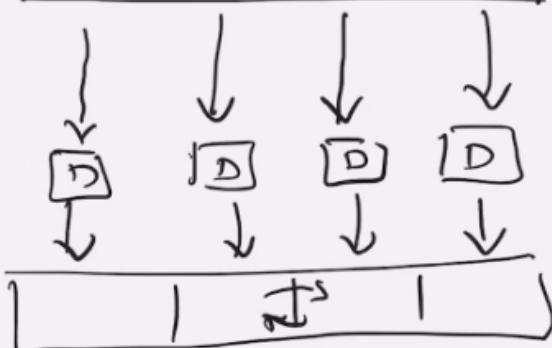


CT



→ spedito in rete

PT

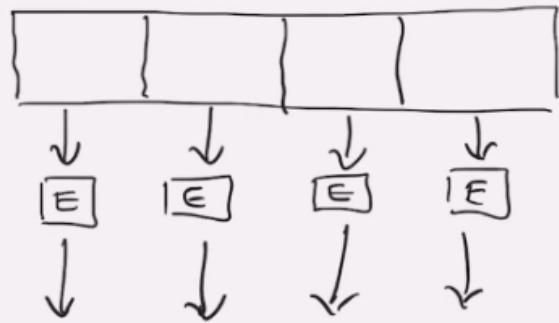


12 34
30 12

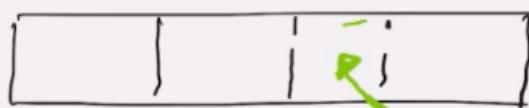
In fase di decifratura è difficile accorgersene di questo scambio (a meno di utilizzare altre tecniche di verifica dell'integrità che vedremo più avanti) a maggior ragione se entrambe le informazioni (quella originaria e quella costruita a seguito dello scambio) sono plausibili.

Inoltre se prendo un blocco proveniente da una **cifratura precedente** cifrato con medesima chiave (in **verde rigato**) e lo sostituisco al posto di altri blocchi, quando andrò a decifrare, otterò un valore diverso da quello voluto (**attacco di malleabilità**):

PT



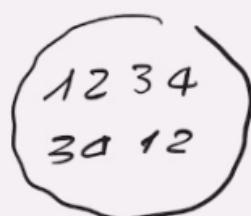
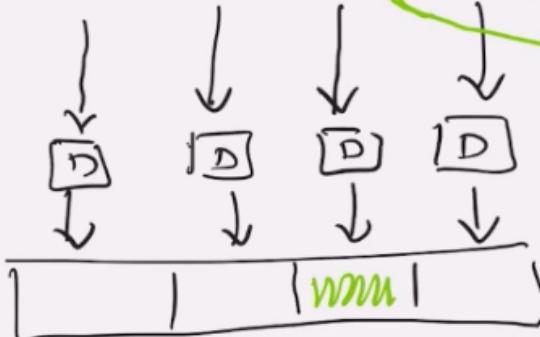
CT



→ spedito in rete

(chiavi K)

PT



12 78

12 34
34 12

L'ECB viene usato nella pratica ma presenta questi contro, quindi dobbiamo tenerne conto.

Block replay attack

Supponiamo di avere un utente U che dispone di 2 conti bancari, uno presso la banca 1 e uno presso la banca 2 . Vuole trasferire una quantità di soldi da banca 1 a banca 2 . Se devo progettare il protocollo, la banca 1 addebita D euro ad U , poi invia un messaggio alla banca 2 , dicendogli di accreditare D euro sul conto di U , la quale banca 2 aggiunge questi D euro al conto di U .

Il formato dei messaggi sarà:

ECB – block replay attack

- Bank transaction that transfers a client U's amount of money D from bank B1 to bank B2
 - Bank B1 debits D to U
 - Bank B1 sends the "credit D to U" message to bank B2
 - Upon receiving the message, Bank B2 credits D to U
- Credit message format
 - Src bank: M (12 byte)
 - Rcv bank: R (12 byte)
 - Client: C (48 byte)
 - Bank account: N (16 byte)
 - Amount of money: D (8 byte)
- Cipher: n = 64 bit; ECB mode



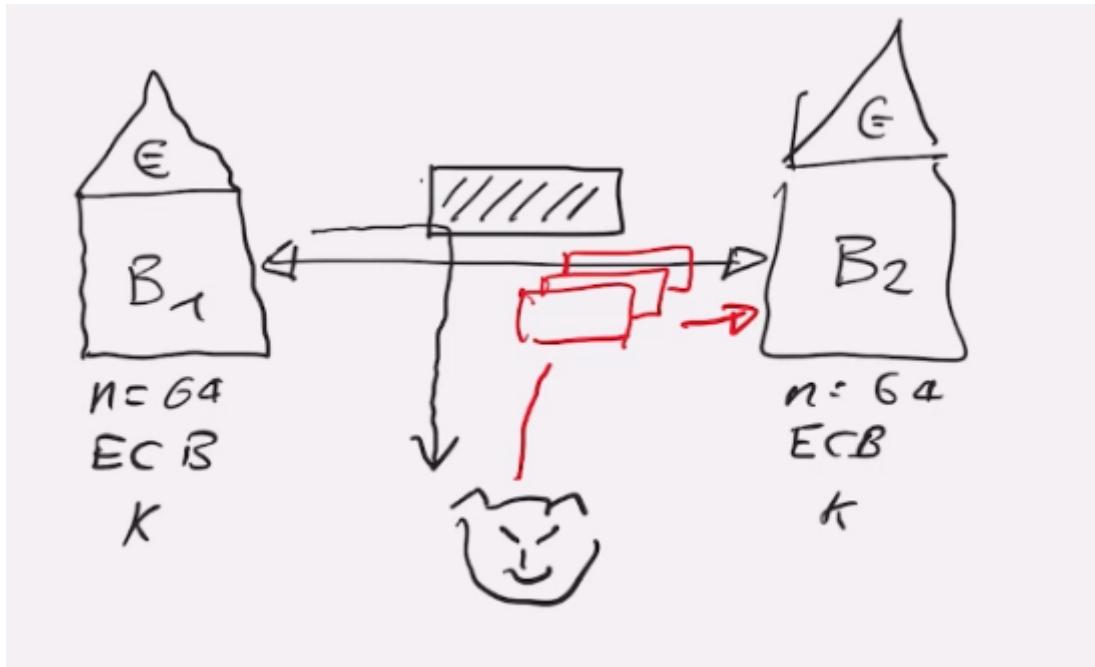
Supponiamo di usare un algoritmo di cifratura come DES con chiave a 64 bit (8 byte).

ECB – block replay attack

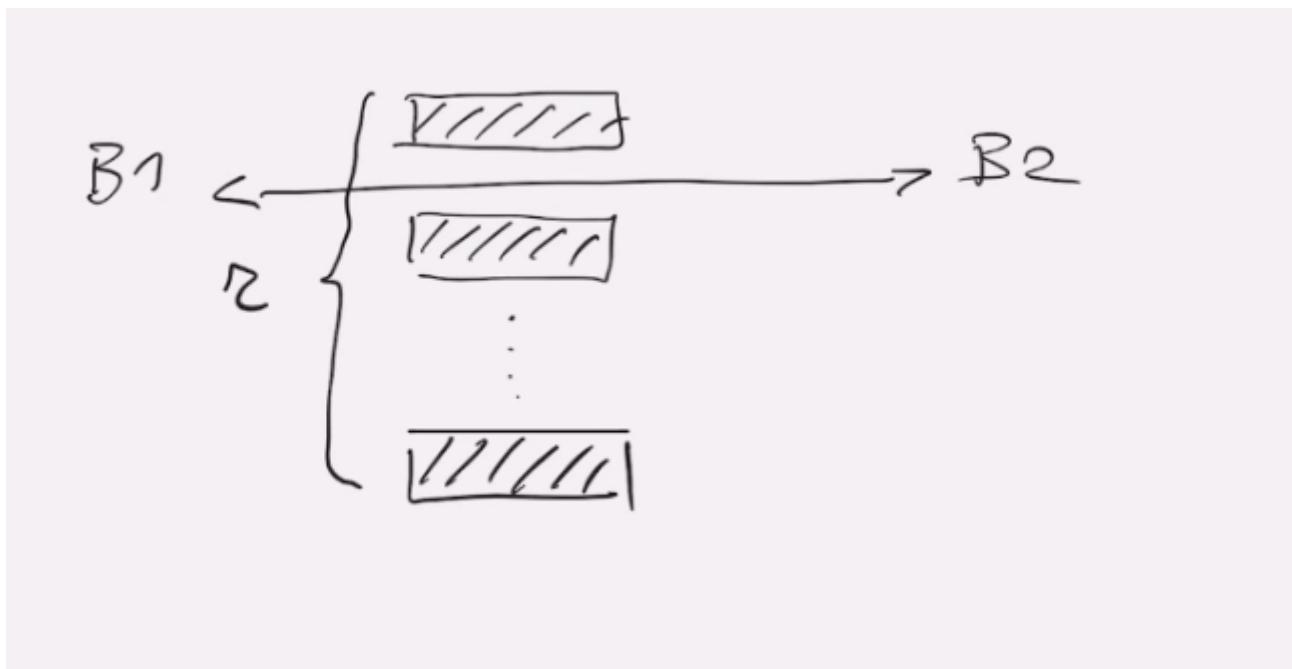
- Mr. Lou Cipher is a client of the banks and wants to make a fraud
- Attack aim
 - To replay Bank B1's message "credit 100\$ to Lou Cipher" many times
- Attack strategy
 - Lou Cipher activates multiple transfers of 100\$ so that multiple messages "credit 100\$ to Lou Cipher" are sent from B1 to B2
 - The adversary identifies at least one of these messages
 - The adversary replies the message several times

Supponiamo che l'utente U voglia fare una truffa. Ogni volta che la banca 2 riceve il messaggio

replicato (in rosso) con una certa somma di denaro, verrà accreditata quella somma di denaro:

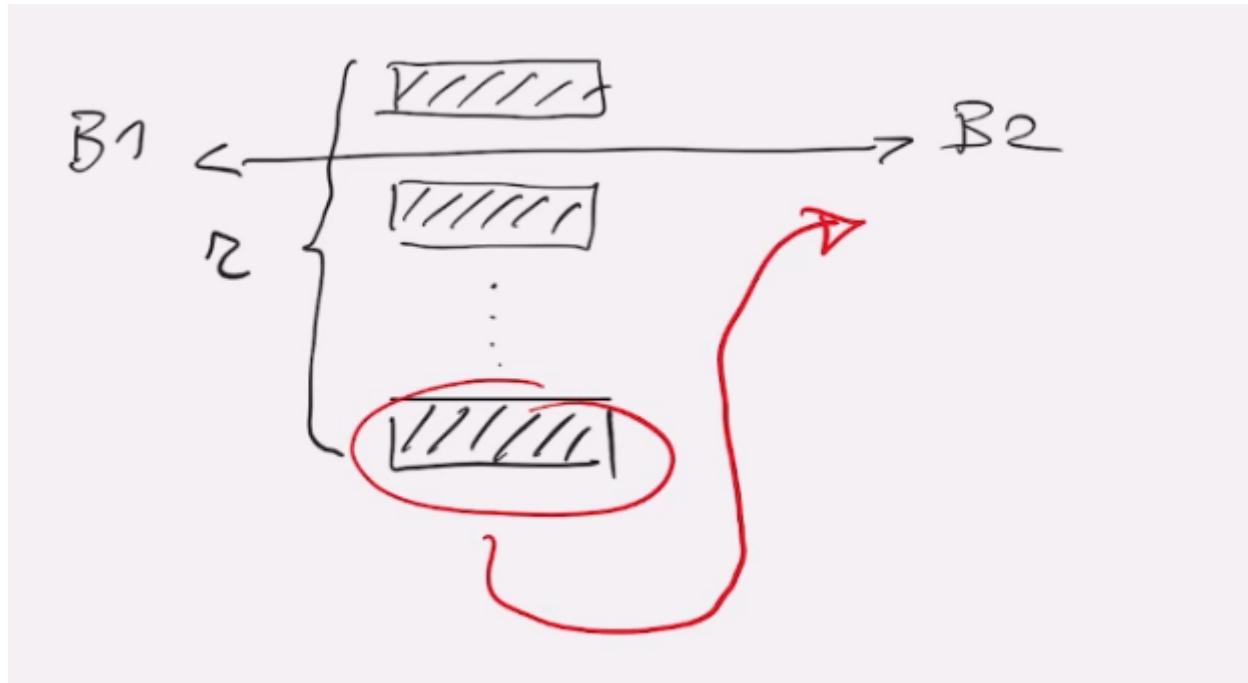


L'attaccante farà un primo trasferimento in cui trasferirà 100 euro dalla banca 1 alla banca 2 diretto sempre al suo stesso conto. Non riuscirà a distinguerlo in quanto cifrato e "affogato" tra gli altri messaggi che le banche si inviano per i loro clienti. Allora per distinguere i suoi messaggi invierà un altro messaggio uguale in cui cambierà solamente la somma trasferita (es: 200 euro). Se vuole aumentare le probabilità di riuscire a identificare i suoi messaggi, invierà molti messaggi, tutti uguali tra di loro al netto della cifra trasferita:



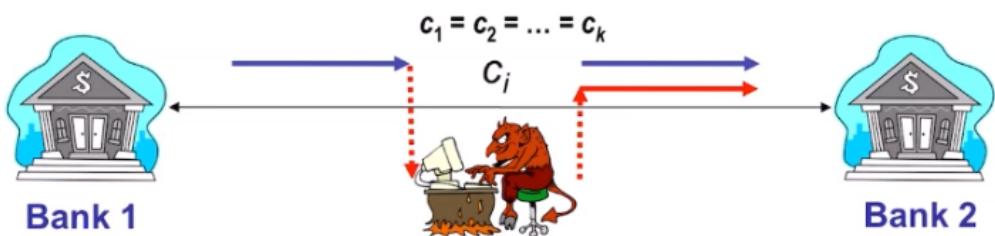
a quel punto, una volta identificati i suoi messaggi nel calderone dei messaggi della banca, può

procedere con l'attacco della replica dei blocchi (freccia in rosso):



ECB – block replay attack

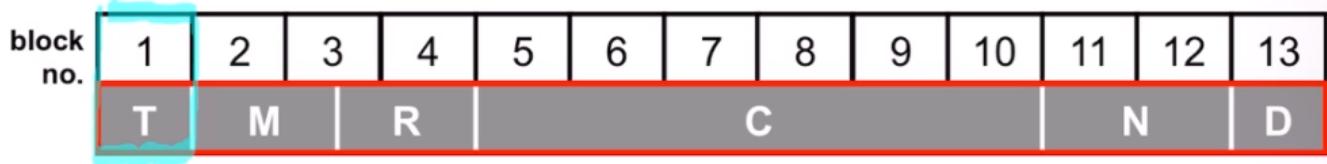
- Mr. Lou Cipher performs k equal transfers
 - credit 100\$ to Lou Cipher $\rightarrow \mathbf{c}_1$
 - credit 100\$ to Lou Cipher $\rightarrow \mathbf{c}_2$
 - ...
 - credit 100\$ to Lou Cipher $\rightarrow \mathbf{c}_k$
- Then, he searches “his own” CT in the network
 - k equal CTs!
- Finally he replies one of these cryptograms



In questo protocollo sembra quindi non esserci alcun "ELEMENTO DI FRESCHEZZA", invece usando un **timestamp** ad esempio, il ricevente può capire se si tratta di un messaggio recente o un messaggio vecchio, quindi capire se siamo in caso di **Replay attack**.

Alteriamo allora il formato dei messaggi aggiungendo un blocco di 8 byte per indicare le marche temporali dei messaggi:

- An 8-byte timestamp field T (block #1) is added to the message to prevent replay attacks



- However, Mr Lou Cipher can
 - Identify “his own” CT by inspecting blocks #2-#13
 - Intercept any “fresh” CT
 - Substitute block #1 of “his own” CT with block #1 of the intercepted “fresh” block
 - Replay the resulting CT

a questo punto i messaggi non potranno essere più tutti uguali !!! ma per lo meno saranno uguali dal blocco 2 al blocco 13 in quanto il blocco 1 corrispondente al timestamp cambia ogni volta. Inoltre è da notare che il blocco 1 corrispondente al timestamp sta su 8 byte dunque il blocco 1 è cifrato separatamente.

Se adesso intercetto il blocco, abbiamo detto che il primo pezzo ha la marca temporale, e quindi come avversario cerco di fare il replay attack, non potrò metterci una marca temporale nuova, in quanto dovrei conoscere la chiave per cifrarne una, quindi la banca 2 si accorgerà del replay attack scartando il messaggio. Posso però individuare in rete un messaggio di altre transazioni cifrate con la medesima chiave, posso prenderla e sostituirla all'interno della mia transazione e a quel punto l'attaccante può bypassare la soluzione del timestamp. **Ovvero ho fatto una sostituzione di blocco.**

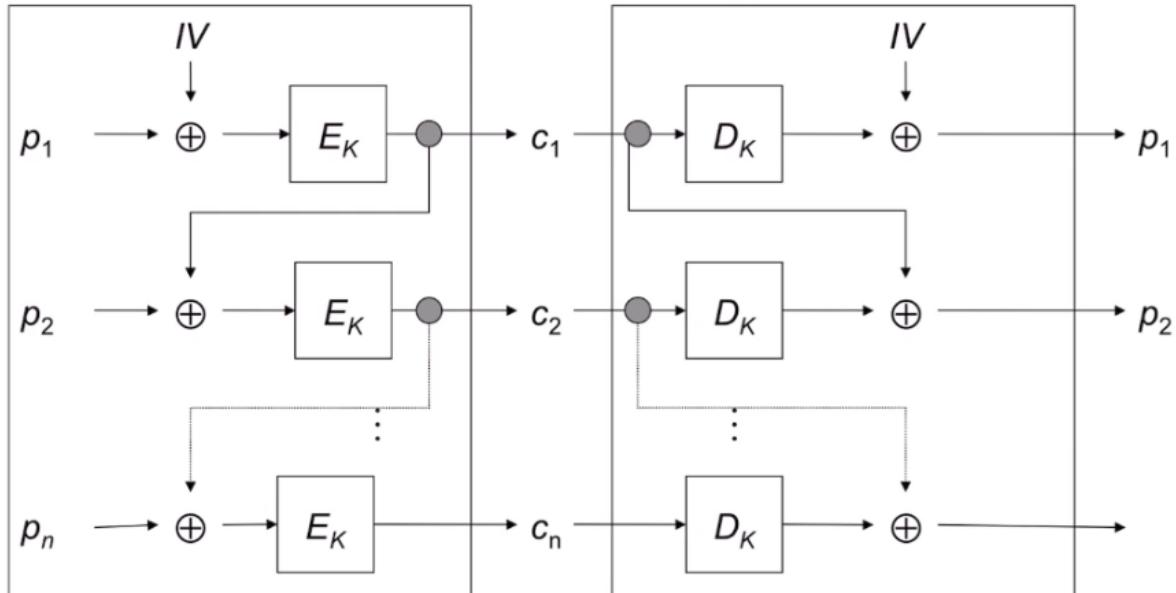
Dunque questo tipo di cifrario va usato in modo critico, usandolo solo quando la mia specifica applicazione può non soffrire di queste problematiche o comunque quando non sono importanti per il mio caso d'uso.

CBC Cipher block chaining

Cipher block chaining (CBC)

$$c_0 \leftarrow IV. \forall 1 \leq i \leq t, c_i \leftarrow E_k(p_i \oplus c_{i-1})$$

$$c_0 \leftarrow IV. \forall 1 \leq i \leq t, p_i \leftarrow c_{i-1} \oplus D_k(c_i)$$



La modalità di cifratura è più complessa ed esiste una forma di retroazione in questo caso di cifratura. Prendiamo ad esempio il blocco p_2 , prima di cifrarlo faccio lo XOR con il blocco c_1 di testo cifrato precedente.

Quando vado a cifrare p_1 che è il primo blocco, non esistendo un blocco precedente, dovrò usare un **IV o vettore di inizializzazione**. Con questa modalità di cifratura, in fase di decifratura, lato ricevente, quando arriverà c_2 , lo decifrerò e ne farò lo XOR con c_1 per ottenere p_2 .

In questo caso supponiamo che ci siano 2 blocchi uguali, ad esempio p_1 e p_2 . p_1 viene messo in XOR con l'IV mentre p_2 viene messo in XOR con c_1 . Questo vuol dire che quello che entra in ingresso al cifrario sono quantità diverse, ovvero anche se p_1 e p_2 sono uguali, dato che viene fatto lo XOR, gli input saranno diversi anche se i plaintext sono uguali, proprio grazie all'operazione di "concatenamento". In modo più macroscopico, se cifro due volte lo stesso dato, vorrà dire che i p_1, p_2, \dots, p_n dei 2 dati saranno uguali, ma sarà sufficiente per ogni cifratura (in questo caso 2) cambiare l'IV per avere due testi cifrati diversi anche a parità del plaintext, dunque con l'analisi del traffico un attaccante non riesce più a capire la struttura della comunicazione.

Inoltre abbiamo visto che con ECB si potevano riordinare i blocchi o sostituirli.

Se ci concentriamo su questa formula:

$$c_0 \leftarrow IV. \forall 1 \leq i \leq t, c_i \leftarrow E_k(p_i \oplus c_{i-1})$$

$$c_0 \leftarrow IV. \forall 1 \leq i \leq t, p_i \leftarrow c_{i-1} \oplus D_k(c_i)$$

essi ci dice in fase di decifratura come calcolare il blocco in chiaro a partire dai blocchi cifrati. Se supponiamo che il blocco c_i venga scambiato oppure sostituito con un blocco precedente dall'attaccante, quello che succede è che vado ad alterare la sequenza, quindi avrò un valore diverso da c_i (detto c'_i), quando vado a decifrare andrò a fare lo XOR con c'_i che non è il vero blocco precedente, dunque la decifratura non fallisce ma il risultato netto che ottengo sarà una serie di caratteri completamente random e privi di senso (oppure nel caso peggiore potrebbe venir fuori per sfortuna delle vittime un valore sensato per l'applicazione in essere, ma è molto improbabile!).

CBC - properties

- Chaining dependencies
 - c_i depends on p_i and c_{i-1} (IV)
 - Enc is sequential, Dec can be in parallel
- Encryption is randomized by using IV
 - CBC is non-deterministic: identical ciphertext results from the same PT under the same key and IV
 - IV is a nonce
 - IV can be sent in the clear but its integrity must be guaranteed
- CT-block reordering affects decryption
- CBC suffers from error propagation
 - Bit errors in c_i affect decryption of c_i and c_{i+1} (error propagation)
 - CBC is self-synchronizing (error recovery)
 - CBC does not tolerate “lost” bits (framing errors)
- ! Dunque il riordino o sostituzione di un blocco provoca una rottura in fase di decifratura. !
- La cifratura deve essere fatta in modo sequenziale, mentre la decifratura può essere fatta in modo parallelo in quanto per poter decifrare un qualunque blocco ho bisogno del suo blocco corrispondente e di quello di prima.
- Il vettore di inizializzazione deve essere nuovo per ogni nuovo messaggio, dunque non bisogna mai usare lo stesso IV due volte, si dice che **CBC non è deterministico** perché a parità di dati, basta cambiare l'IV per ottenere testi cifrati differenti. Dunque l'IV randomizza la cifratura.
L'IV, come tutti gli elementi in generale che servono per conferire freschezza, viene detto **NONCE** può essere realizzato tramite un contatore oppure può essere generato in modo completamente

random. L'IV può essere inviato in chiaro, ma la sua integrità deve essere verificata, in quanto se l'avversario riuscisse a modificarlo, in fase di decifratura si otterrebbe un valore diverso del messaggio voluto.

Esistono altre modalità di cifratura, che non verranno trattate:

Other encryption modes

- Other encryption modes
 - Cipher Feedback mode (CFB)
 - Output Feedback mode (OFB)
 - Counter mode (CTR)
 - Galois Counter mode (GCM)
 - and many others (e.g., CCM, CTS, ...)
- Use of encryption modes
 - In CFB, OFB, CTR a block cipher is used as stream cipher / pseudo-random generator
 - In GCM and CCM a block cipher guarantees confidentiality and authentication and integrity (authenticated encryption)
 - CTS avoids ciphertext expansion
- Block ciphers are very versatile components

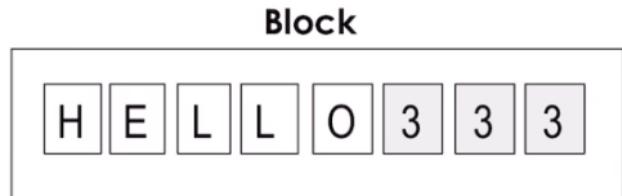
Padding

L'ipotesi iniziale che avevamo fatto che il testo in chiaro fosse un multiplo del blocco la andiamo a rilassare, infatti:

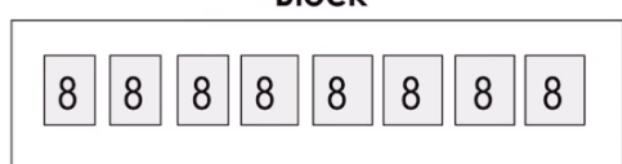
Padding – the PKCS#5 standard

- Padding is necessary when PT len is not a block multiple

If PT len is NOT a block multiple
Each padding byte \leftarrow #(bytes to complete a block)



If PT is a block multiple
Padding = block
Each padding byte \leftarrow 8



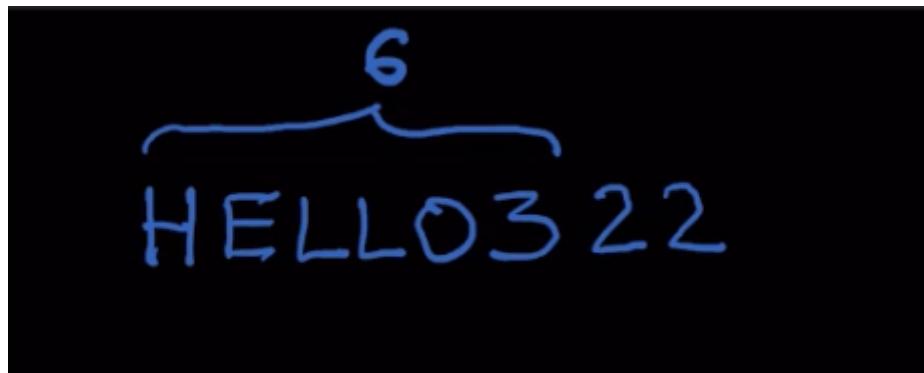
il problema che vogliamo affrontare è una situazione più generica in cui ci troviamo a fare l'encryption ovvero

il testo in chiaro che vogliamo cifrare viene affettato in blocchi e l'ultimo blocco non è pieno ma gli manca qualche byte.

Sia che si utilizzi CBC che ECB il cifrario in ingresso vuole sempre un blocco della stessa lunghezza. Quindi viene effettuata una operazione di **padding** utilizzando dei byte fittizi in modo tale che sul lato della decifratura, vengano scartati i byte aggiuntivi. Però chi riceve non sa quali sono i byte fittizi e quale è la vera lunghezza del messaggio, quindi in qualche modo dobbiamo comunicare questa informazione aggiuntiva. Supponiamo di avere il messaggio **HELLO** completato con dei byte tutti a 0 come valore fittizio.

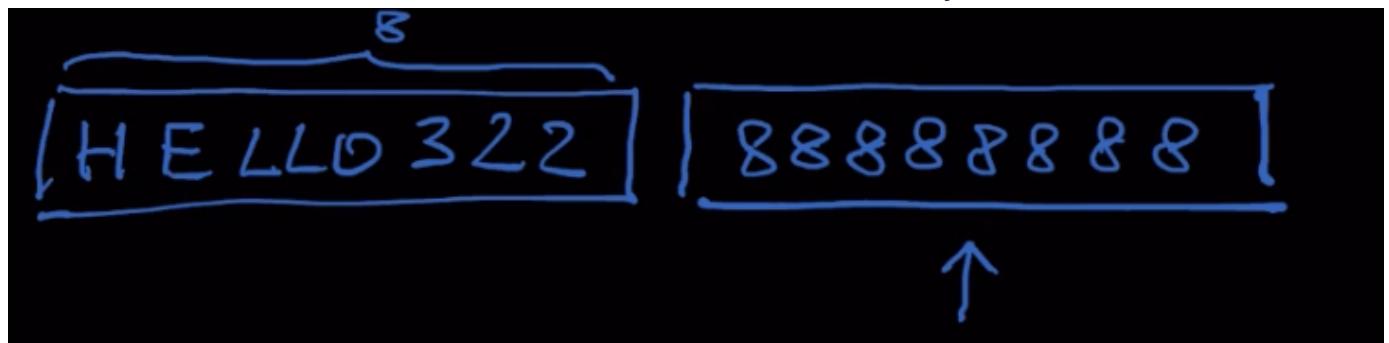
⚠ Il problema di questa soluzione è che se volessi trasmettere il messaggio HELLO0? Potrei avere dei casi in cui lo 0 non è padding ma è parte del mio payload! Inoltre in molti linguaggi di programmazione lo 0 è il carattere di fine stringa, quindi quello 0 dovremmo distinguerlo dal carattere di fine stringa.

Il trucco che si usa è il seguente: supponiamo di voler scrivere HELLO e mi mancano 3 byte da completare. In questi 3 byte scriverò il numero 3, in quanto sono 3 i byte che devo riempire. Fossero stati solo 2 byte da dover riempire avrei scritto il numero 2 in ciascuno dei 2 byte. Il problema sembrerebbe ripresentarsi nel caso in cui io voglia mandare **HELL03**, il riempimento che mi serve in questo caso è pari a 2 byte, dunque scriverò il carattere 2 come riempimento:



Con questo metodo riesco quindi sempre a identificare il padding, eccetto in un caso:

supponiamo che l'ultimo blocco contenga l'informazione **HELL0322**, quindi **non ci sarebbe bisogno di padding**, quando però viene ricevuto, gli ultimi 2 byte verrebbero scartati perché sono posti a 2. Posso allora introdurre un **intero blocco fittizio** in cui scrivere un blocco da 8 byte con il carattere 8:



a questo punto l'ultimo blocco diventa quello identificato con la freccia e quindi verrà scartato interamente.

Con questo metodo di padding si riesce a tenere conto del fatto che il messaggio non deve avere necessariamente una dimensione che non è un multiplo integrale del blocco.

Dunque il padding espone due attenzioni da prendere in fase di progettazione e utilizzo delle librerie:

1. Se un messaggio in chiaro ha una certa dimensione, il buffer da allocare per contenere il testo cifrato dovrà essere più grande in quanto il padding aumenta la dimensione
 2. Se devo trasmettere dei dati, i dati trasmessi sono più grandi di quelli in chiaro, questo normalmente non è un problema se non nel caso di dispositivi embedded o real-time application in cui c'è un dispendio maggiore di energia o storage. Per questo motivo esistono delle modalità di cifratura che mi permettono di evitare il rigonfiamento dovuto al padding.
-

Form

✓ Correct 1/1 Points

2. The ECB encryption mode

- makes it possible traffic analysis, block reordering and block substitution ✓
- does not make it possible traffic analysis, block reordering and block substitution
- makes it possible traffic analysis only

✓ Correct 1/1 Points

3. In the CBC encryption mode

- two identical plaintexts produce the same ciphertext if encrypted by means of the same key
- two identical plaintexts produce the same ciphertext if encrypted by means of the same key and the same IV ✓
- two identical plaintexts produce the same ciphertext if encrypted by means of the same IV

✓ Correct 1/1 Points

4. In the CBC encryption mode

- encryption only can be parallelized
- decryption only can be parallelized ✓
- both encryption and decryption can be parallelized

✗ Incorrect 0/1 Points

5. Consider a cipher with $n = 64$. If the last block of a plaintext contains "REGARDS", determine the value of the PKCS#7 padding (express the value in hexadecimal)

0x1

Correct answers:

01

✗ Incorrect 0/1 Points

6. Consider a cipher with $n = 64$. Consider a plaintext of 122 byte. With reference to the PKCS#7 padding, determine the number of blocks of the padded plaintext

14

Correct answers:

16

✗ Incorrect 0/1 Points

7. Consider a cipher with $n = 64$. Consider a plaintext of 128 byte. With reference to the PKCS#7 padding, determine the number of blocks of the padded plaintext

0

Correct answers:

16

! domanda 7: non dovrebbe essere necessario il padding in questo caso? quindi dovrebbero essere 16 padded plaintext

03/03/2023

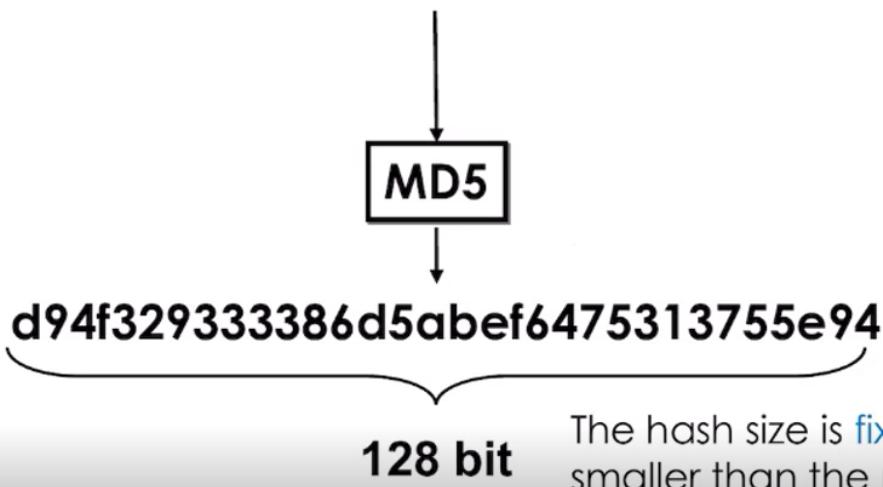
Hash Functions

Sono un'altra primitiva crittografica che svolgono una funzione diversa rispetto ai cifrari i quali si occupano di confidenzialità. Le funzioni Hash e MAC si occupano di un altro requisito che è l'**INTEGRITÀ** al fine di contrastare il tampering.

An example

**Nel mezzo del cammin di nostra vita
mi ritrovai per una selva oscura
che' la diritta via era smarrita.**

**Ahi quanto a dir qual era e` cosa dura
esta selva selvaggia e aspra e forte
che nel pensier rinova la paura!**



Intuitivamente si tratta di una funzione che prende in **ingresso** un dato che può avere una **dimensione FINITA** ma non **LIMITATA** mentre produce un **output** su un numero di bit che è **PREDEFINITO**. Ad esempio la funzione MD5 produce un output su 128 bit.

Nell'esempio viene mostrato come dando in ingresso un testo codificato in ASCII, viene dato come output una sequenza di 128 bit che per semplicità è stata rappresentata in esadecimale. Dunque la dimensione dell'hash in genere è **più piccola** della dimensione dell'ingresso. In genere l'output viene chiamato anche Hash, Digest o Fingerprint.

Informal properties

- Applicable to messages of any size
- “Easy” to compute
- Output of fixed length
- “Difficult” to invert
- “Unique”: the hash of a message can be used to "uniquely" represent the message
 - The output should be highly sensitive to all inputs
 - if we make minor modifications to the input, the output should look like very different (like a block cipher)

- Le funzioni hash si applicano a qualsiasi dimensione in ingresso
- Sono facili nel senso che sono molto efficienti da calcolare
- L'output ha una lunghezza fissata
- E' difficile da invertire. Ovvero conoscendo l'hash è **MOLTO** difficile computazionalmente parlando ricavare il testo che ha prodotto quella sequenza di bit. Matematicamente non è corretto affermare una cosa del genere in quanto per esempio usando una funzione hash che produce 2^{128} bit di uscita, il numero di bit in uscita sarà sempre inferiore rispetto al numero in ingresso che può essere variabile e molto grande (2^{64} come anche 2^{256}) dunque potrei avere 2 input che finiscono sullo stesso output e queste vengono dette **collisioni**.
- Ci piacerebbe che ogni messaggio avesse la propria hash, se così fosse praticamente la hash diventerebbe una specie di identificatore univoco per un determinato insieme di informazioni. Avere una rappresentazione a lunghezza fissa e molto facile da manipolare rispetto a una lunga sequenza di byte è vantaggioso in quanto invece di calcolare la firma digitale di un quantitativo enorme di informazioni posso calcolare la firma digitale sull'hash, guadagnandoci in termini di prestazioni. Però dato che è possibile (ma molto improbabile) avere 2 input che hanno la stessa hash, questa unicità potrebbe venire meno!

! Possiamo avere collisioni proprio per definizione di funzione di hash, ma è importante che le collisioni siano molto difficili da trovare. Dunque l'output è molto sensibile all'input: ovvero se cambio anche di un solo carattere l'input, l'output viene stravolto e dovrebbe apparire molto diverso.

Nell'esempio è bastato aggiungere una r per stravolgere l'hash:

- The fingerprint must be highly sensitive to all input bits
 - Input «I am not a crook»
 - Hash (MD5): 6d17fcd4ae0e82fa4409f4ea6f4106a6 
 - Input «I am not a cook»
 - Hash (MD5): 9ebe3d42d5c01fc59fe3daacbf42f515

Collisions

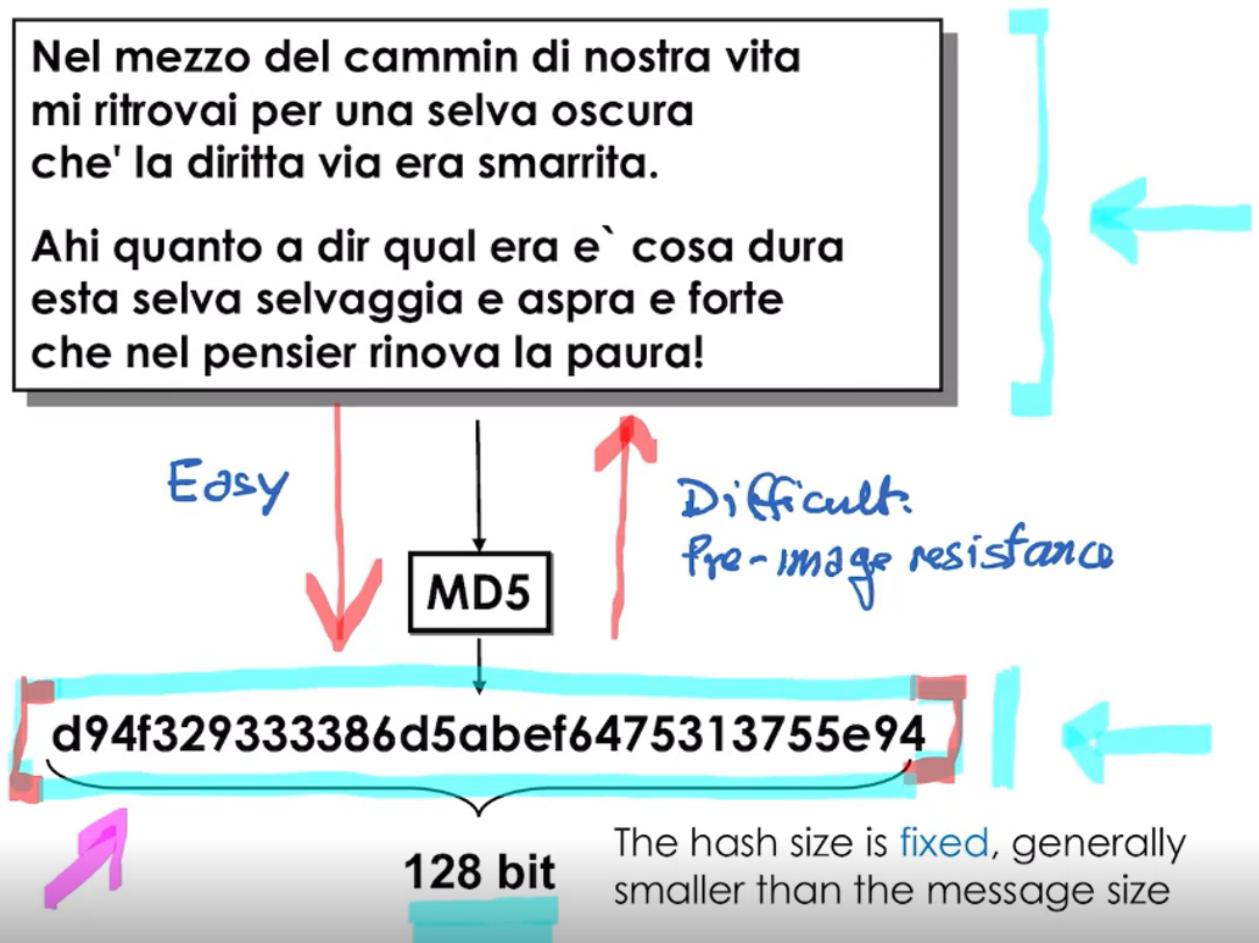
- **Hash function $H: \{0,1\}^* \rightarrow \{0,1\}^m$**
- **Properties**
 - **Compression** – H maps an input x of arbitrary finite length into an output $H(x)$ of fixed length m
 - **Ease of computation** – given x , $H(x)$ must be “easy” to compute
 - **Many-to-one**, so it implies **collisions**
 - **DEF.** A **collision** for H is a pair x_0, x_1 s.t. $H(x_0) = H(x_1)$ and $x_0 \neq x_1$

Proprietà di sicurezza

Security properties

- Preimage resistance (one-way)
 - For essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output
 - i.e., to find x such that $y = h(x)$ given y for which x is not known
- 2nd-preimage resistance (weak collision resistance)
 - It is computationally infeasible to find any second input which has the same output as any specified input
 - i.e., given x , to find $x' \neq x$ such that $h(x) = h(x')$
- Collision resistance (strong collision resistance)
 - It is computationally infeasible to find any two distinct inputs which hash to the same output,
 - i.e., find x, x' such that $h(x) = h(x')$

- Preimage resistance (one-way): ovvero data una sequenza di bit è difficile ricavare il messaggio la cui funzione hash è proprio quella sequenza di bit



- 2nd-preimage: ovvero se abbiamo x_1 che dato in ingresso alla funzione hash mi produce y_1 , deve essere computazionalmente difficile trovare un valore x_2 che abbia in uscita lo stesso y_1 .

- **Collision resistance:** deve essere molto difficile computazionalmente trovare 2 valori che collidano. Questa proprietà sembra molto simile alla seconda proprietà ma sono profondamente diverse in quanto nella proprietà di 2nd-preimage ho un x dato e devo trovare un altro input in modo tale che i 2 abbiano la stessa hash. In questo caso non abbiamo alcun input all'inizio e sono "libero" di scegliermi la coppia come mi pare. Dunque questo problema è **PIU' FACILE** del secondo.

Relationship between properties

- Collision resistance implies 2-nd preimage resistance
- Collision resistance does not imply preimage resistance
 - In practice, CRHF almost always has the additional property of preimage resistance

Dunque se una funzione hash soddisfa la proprietà di collision resistance allora anche la proprietà di 2nd-preimage

Attacking Hash Functions

- An attack is successful if it produces a collision (forgery)
- Two types of forgery
 - Selective forgery
 - The adversary has a complete, or partial, control over the colliding input x
 - Existential forgery
 - The adversary has no control over the colliding input x

Dato che queste proprietà di sicurezza mi dicono che è difficile trovare delle collisioni dunque sfrutterò queste proprietà per firmare una hash anzichè il file originale, un attaccante proverà a produrre una collisione ovvero fare un forgery, una contraffazione.

Birthday paradox: intuition

- **Problem 1.** (*2nd preimage res.*)

- In a room of 23 people, the probability that at least a person is born on 25 December is $23/365 = 0.063$

- **Problem 2.** (*collision resistance*)

- In a room of 23 people, the probability that at least 2 people have the same birthdate is 0.507

$$0,507 \Rightarrow 0,063$$

circa un ordine di grandezza

Intuitivamente si può pensare che la 2nd-preimage resistance sia equivalente al problema 1 in figura, ovvero in una stanza di 23 persone calcolare la probabilità che almeno una persona sia nata il 25 dicembre. La probabilità che viene fuori è 0.063.

Facendo riferimento invece al problema 2, calcolando la probabilità che in una stanza di 23 persone, almeno 2 persone abbiano lo stesso compleanno, questo è un problema equivalente alla collision resistance. In questo caso la probabilità è del 0.507 ovvero quasi il 50%.

Questo tipo di problema passa sotto il nome di **Paradosso del Compleanno**, perché si tratta di un problema poco intuitivo.

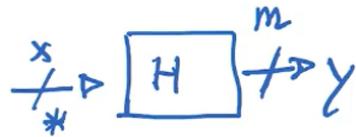
Quello che l'avversario vuole provare a fare è una forgery, quindi a trovare delle collisioni, può procedere in 2 modi:

I black box attacks sono 2 attacchi che non sfruttano la conoscenza dell'algoritmo e questo succede

quando l'algoritmo è considerato sicuro ovvero non ha vulnerabilità da sfruttare:

Black box attacks

- Black box attacks



- Consider H as a black box
- Only consider the output bit length m ;
- Approximate H as a random variable

- Specific BB attacks

- Guessing attack: find a 2nd pre-image ($O(2^m)$)
- Birthday attack: find a collision ($O(2^{m/2})$)
- These attacks constitute a security upper bound

$$\text{MD5 } m = 128 \quad 2^{128} \quad 1\mu\text{s} \quad 2 \times 10^{-6} \text{ s}$$

Quanti anni?

1. **Guessing attack**: provare a violare la 2nd-preimage.

Guessing attack

- Objective

- Given x_0 find a 2nd preimage
 $x_1 \neq x_0$ s.t. $H(x_0) = H(x_1)$

- Complexity

- Time complexity
 - Every step requires
 - 1 random number generation: efficient!
 - 1 hash function computation: efficient!
 - Number of steps 2^m
- Data/storage complexity
 - Constant and negligible

```
GuessingAttack(x₀)
```

```
repeat
```

```
    x ← random() // guessing
```

```
until h(x₀) = h(x)
```

```
return x
```

Supponiamo di conoscere un valore x_0 e vogliamo trovare x_1 in modo tale che si generi una collisione. Consiste nel provare iterativamente a generare x_1 in modo randomico calcolandone

l'hash e comparandolo con quello di x_0 . I tentativi sono nell'ordine di 2^m con m pari al numero di bit in uscita dall'hash function.

2. **Birthday attack**: ovvero provare a trovare delle collisioni. In questo caso si può dimostrare che i tentativi sono nell'ordine di $2^{m/2}$ ovvero meno rispetto al numero di tentativi del guessing attack.

Birthday attack →

- Intuition
 - Start with
 - x_1 = «Transfer \$10 into Oscar's account»
 - x_2 = «Transfer \$10.000 into Oscar's account»
 - Alter x_1 and x_2 at nonvisible locations so that semantics is unchanged
 - Spaces, tabs, return,...
 - Continue until $H(x_1) == H(x_2)$

Si può alterare x_1 e x_2 in modo non visibile (ad esempio in coda al messaggio aggiungendo TAB e BACK-TAB in modo tale che la semantica non cambi) però di fatto cambiando il testo e quindi dopo un certo numero di tentativi (mediamente $2^{m/2}$) riuscire a trovare 2 stringhe che hanno la stessa hash.

Questi attacchi costituiscono un limite di sicurezza superiore, ovvero meglio di così non è possibile fare in quanto quello che potrebbe succedere è che un "matematico brillante" riesca a trovare dei metodi più efficienti per riuscire a trovare delle collisioni.

Sample hash functions

Hash Function	m	Preimage	Collision	Speed (Mb/sec)
MD5	128	2^{128}	2^{64}	
RIPEMD-128	128	2^{128}	2^{64}	
RIPEMD-160	160	2^{160}	2^{80}	153
SHA-1	160	2^{160}	2^{80}	153
SHA-256	256	2^{256}	2^{128}	111
SHA-512	512	2^{512}	2^{256}	99

Guessing
Attack

Birthday
attack

2^{128}

X

X

Broken in 2017



Announcing the first SHA-1 collision

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

- Collision

- Given two images produce two PDFs that hash to the same tag
- After 20 years

Announcing the first SHA1 collision

February 23, 2017

Posted by Marc Stevens (CWI Amsterdam), Elie Bursztein (Google), Pierre Karpman (CWI Amsterdam), Ange Albertini (Google), Yarik Markov (Google), Alex Petit-Bianco (Google), Clement Baisse (Google)

Cryptographic hash functions like SHA-1 are a cryptographer's swiss army knife. You'll

- Large scale computation

- Nine quintillion (9×10^{18} SHA1 computations) in total
- 6,500 years of CPU computation to complete the attack first phase
- 110 years of GPU computation to complete the second phase
- Still 10^5 times faster than brute-force attack

Black Box attack / Birthday attack

- Move soon to SHA-256, SHA-3

Scopi delle funzioni hash

Use of CRHF

- The purpose of a CRHF, in conjunction with other mechanisms (authentic channel, encryption, digital signature), is to provide message integrity and authentication



Le funzioni hash si usano principalmente per l'integrità del messaggio e autenticazione. Quando trasmettiamo il messaggio in rete infatti l'avversario può alterare il messaggio. Inoltre quando riceviamo un messaggio vorremmo essere sicuri dell'origine/mittente di quel messaggio.

Integrity vs authentication

- Message integrity**

- The property whereby data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source

- Message origin authentication**

- A type of authentication whereby a party is corroborated as the (original) source of specified data created at some time in the past

- Data origin authentication includes data integrity**

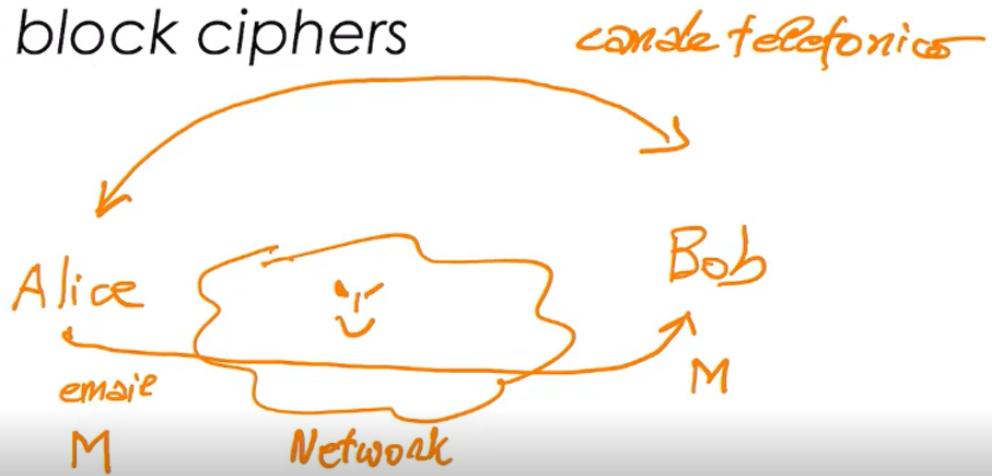
Le funzioni hash devono essere usate insieme ad altri meccanismi come indicato in figura infatti fino ad ora non abbiamo mai parlato di chiavi.

Integrity with CRHF

CRHF and an authentic channel

- physically authentic channel
- digital signature

CRHF and block ciphers



Pensiamo ad un applicazione in cui Alice e Bob si scambiano un messaggio M . In rete c'è l'avversario che potrebbe intercettare il messaggio e modificarlo. Bob vuole essere sicuro che il messaggio ricevuto sia quello inviato da Alice e non quello modificato dall'attaccante. Bob allora potrebbe telefonare ad Alice non usando più la rete ma usando un altro canale che posso considerare sicuro, e leggerle l'intero messaggio M ricevendo la conferma via telefono da Alice che il messaggio è stato ricevuto correttamente. **Data la poca fattibilità pratica di questa soluzione, potrebbero invece calcolare Alice l'hash del messaggio inviato e Bob l'hash del messaggio ricevuto e vedere se coincidono.** Però anche in questo caso sarebbe necessario usare ogni volta un secondo canale di comunicazione **autentico** (nel senso di sicuro, affidabile, privo di interferenze) dunque per risolvere questo problema vedremo che gli hash vengono usati insieme ai cifrari a blocchi o alle firme digitali.

Hash + Block Cipher

Alice (A) vuole mandare un messaggio (x) a Bob (B) assicurandosi che la comunicazione sia confidenziale e integra. La chiave (e) viene usata per le operazioni simmetriche di encryption/decryption. Alice può prendere x , ne calcola l'hash $H(x)$ e lo inserisce in append al testo che vuole mandare x . Infine cifra $x \parallel H(x)$ usando la chiave di cifratura e . Questo processo è descritto nel primo punto della figura ed è quello **consigliato**. Gli altri due schemi sono invece

sconsigliati per diversi motivi:

cipher

$y = E(k_A, x \mid H(x))$

– Confidentiality and integrity

– As secure as E

• $x, E(e, H(x))$

– Sender has seen $H(x)$

• $E(e, x), H(x)$

– $H(x)$ can be used to check a guessed x

Diagram illustrating the security of different cipher schemes:

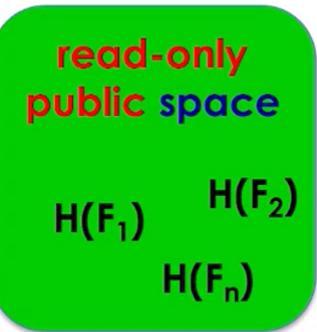
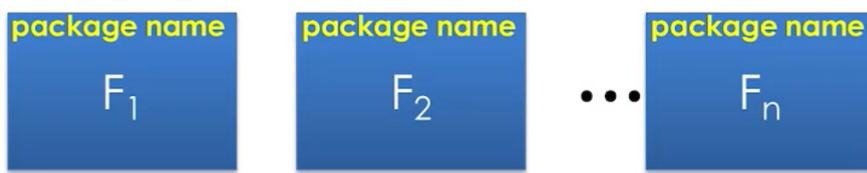
- $A \xrightarrow{x} y \xrightarrow{k_B} B$
- For $y = E(k_A, x \mid H(x))$:
 - Left arrow labeled "yes" with "conf." handwritten.
 - Right arrow labeled "no".
- For $x, E(e, H(x))$:
 - Left arrow labeled "no".
 - Right arrow labeled "no".
- For $E(e, x), H(x)$:
 - Left arrow labeled "no".
 - Right arrow labeled "scary".

Esempi pratici

Un utilizzo frequente è quello di usare la funzione hash per verificare l'integrità di un file scaricato da internet.

Example: protecting files

- **Software packages**



- Upon downloading a package, the user can verify that contents are valid
 - H collision resistant \Rightarrow attacker cannot modify package without detection
 - no key needed (public verifiability), but requires read-only space

Oppure per la memorizzazione delle password:

Storage of password

- Passwords are stored in hashed form

– <username, hash>



- Example

– **alice**

4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b ||

– **jason**

695ddcccd984217fe8d79858dc485b67d66489145afa78e8b27c1451b27cc7a2b

– **mario**

cd5cb49b8b62fb8dca38ff2503798eae71bfb87b0ce3210cf0acac43a3f2883c

– **teresa**

73fb51a0c9be7d988355706b18374e775b18707a8a03f7a61198eefc64b409e8

– **bob**

4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b ||

– **mike**

4b529ac375b4217be17fef1a4a6f1624185cc99909e92278c0759e12ab3d61fa

I computer non memorizzano le password in chiaro ma una hash di quella password. Il sistema infatti effettua la validazione dello username e password andando a fare un confronto tra l'hash della password ricevuta dall'utente che vuole fare login e l'hash della password memorizzata. In caso positivo il login va a buon fine altrimenti no. Quindi nel caso di attacco informatico l'attaccante si ritroverebbe con un file contenenti gli hash delle password degli utenti del sistema (come in figura) e per la prima proprietà degli hash sarebbe difficile risalire alla password rompendo l'hash. Ma questo meccanismo ha dei limiti in quanto se Alice e Bob hanno definito la medesima password come nell'esempio in figura, **le stringhe hash sono uguali!!!.**

Inoltre effettuando un attacco alle password, per ogni parola del dizionario inglese o italiano o una wordlist connessa al contesto, calcolerà una hash andando a vedere se quella hash è presente nel file delle password.

Storage of password

password manager LastPass

- Criticalities

- If different users choose the same password, they have the same hash
 - Example: Alice and Bob
- Dictionary attack (brute force attack)
 - E.g.: <https://www.onlinehashcrack.com/>
- Rainbow table attack
 - Pre-computed database of hashes for fast access
 - Trade storage for computation
 - E.g. <https://crackstation.net/>
 - E.g.: Mike / “friendship”

Ci si potrebbe costruire una **rainbow table**, ovvero pre-calcolo gli hash di tutte le parole in un dizionario, poi nel momento in cui vorrò attaccare l'hash di una password mi basterà andare a vedere se l'hash della password vittima è presente nella mia rainbow table. Un esempio di sito online che usufruisce di una rainbow table per effettuare il cracking degli hash è <https://crackstation.net/>

CrackStation

CrackStation ▾ Password Hashing Security ▾ Defuse Security ▾

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
4420d1918bbcfcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b  
695ddcccd984217fe8d79858dc485b67d66489145afa78eb27c1451b27cc7a2b  
cd5cb49bb862fb8edca30ff2503798ae71bf8b7b0ce3210cf0cac43e3f2883c  
73fb51a0c9be7d98835570618374e775b18707a8a03f7a61198efcf4ab409e8  
4420d1918bbcfcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b  
4b529ac375b4217be17fe1a4a6f1624185cc99999e92278c0759e12ab3d61fa
```

I'm not a robot reCAPTCHA Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sh1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
4420d1918bbcfcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b	Unknown	Not found.
695ddcccd984217fe8d79858dc485b67d66489145afa78eb27c1451b27cc7a2b	Unknown	Not found.
cd5cb49bb862fb8edca30ff2503798ae71bf8b7b0ce3210cf0cac43e3f2883c	Unknown	Not found.
73fb51a0c9be7d98835570618374e775b18707a8a03f7a61198efcf4ab409e8	Unknown	Not found.
4420d1918bbcfcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b	Unknown	Not found.
4b529ac375b4217be17fe1a4a6f1624185cc99999e92278c0759e12ab3d61fa	sha256	friendship

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

[Download CrackStation's Wordlist](#)

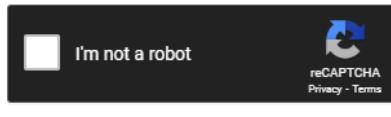
Inoltre è interessante notare come questi strumenti siano in grado di riconoscere anche delle euristiche

come sostituzione di caratteri con numeri o caratteri speciali:

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

8EFE310F9AB3EFEAE8D410A8E0166EB2



[Crack Hashes](#)

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sh1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
8EFE310F9AB3EFEAE8D410A8E0166EB2	md5	P4ssw0rd

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

[Download CrackStation's Wordlist](#)

Per contrastare l'attacco che fa uso di rainbow table è necessario usare il **Salting**.

Salting

Questa tecnica rende il rainbow-table attack impossibile:

Salting password →

- Salting a password
 - Upon creation of a new password `pwd`
 - Define salt = `random()`
 - Compute hash = $H(\underline{\text{salt}} \mid \underline{\text{pwd}})$
 - Store username, salt, hash
- Advantages
 - Salting makes a Rainbow Table Attack infeasible
 - If stored elsewhere than hash, salt also makes a Dictionary attack infeasible

in quanto se si fosse usato il salting in fase di hashing della password di Mike ad esempio, si sarebbe dovuta pre-creare una rainbow table per ciascuna combinazione possibile di salt. Con un salt di 32 bit ad esempio avremmo 2^{32} rainbow table dunque la quantità di memoria necessaria all'attaccante per

completare l'attacco sarebbe troppo grande:

Salting password



- Example
 - Alice
 - Password: admin
 - salt: 317029;
 - hash: f9ea5ab02d83138e4f0f1f87ffd2c62a |
 - Bob
 - Password: admin
 - salt: 450982
 - hash: 8c13e26985d3972bff4063861194c98c |

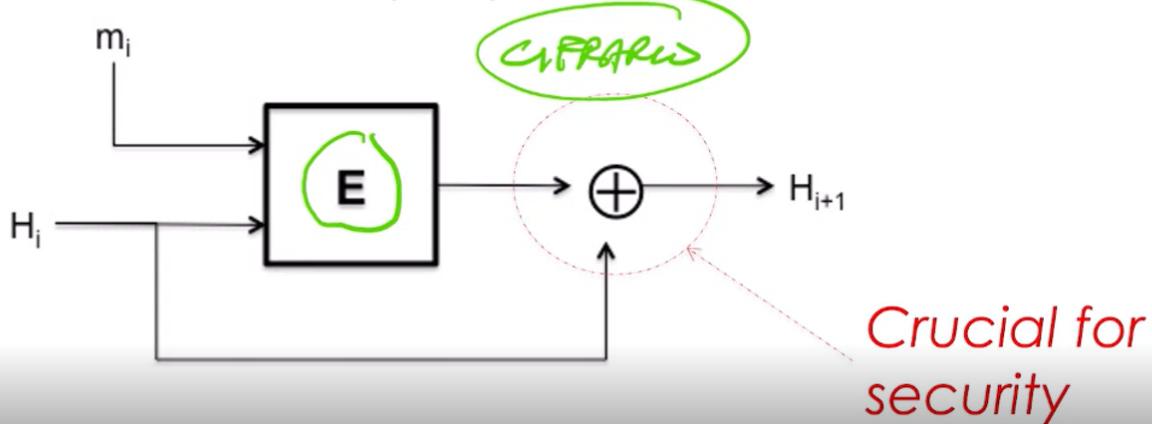
Compress function

Molte funzioni hash (non tutte) usano un cifrario, dunque il componente usato per la confidenzialità può essere anche usato per realizzare funzioni hash. Ad esempio per componenti hardware embeded con poca memoria avrà bisogno di caricare solamente un'unica componente software per effettuare

entrambe le operazioni di encryption e hashing:

Compression function

- E: block cipher
- Davies-Meyer compression function
 - Finding a collision $h(H, m) = h(H', m')$ requires $2^{m/2}$ evaluations of (E, D) \Rightarrow best possible!



Funzioni hash:

2. Determine the hash of the following sentence "La follia sta nel fare sempre la stessa cosa aspettandosi risultati diversi". Use SHA-256. □₄₈

e000db04de32c3f8d13fbb4a4545f07ce8459a26e869f97f7ad5a71e0c0326d6

✗ **Incorrect** 0/1 Points

3. In hash functions, collisions are □₄₈

- impossible
- inevitable ✓
- possible, depending on the algorithm

✓ **Correct** 1/1 Points

4. Violating 2nd-preimage resistance is □₄₈

- easier than collision resistance
- more difficult than collision resistance ✓
- as difficult as collision resistance

✓ **Correct** 1/1 Points

5. Black box attacks

- consider the hash algorithm
- don't consider the hash algorithm
- consider the size of the input

✓ **Correct** 1/1 Points

6. Let n be the output size of hash function H . Complexity of a birthday attack is in the order of

- 2^n
- $2^{(n/2)}$
- n^2

✓ **Correct** 1/1 Points

7. Let n be the output size of hash function H . Complexity of a guessing attack is in the order of

- 2^n
- $2^{(n/2)}$
- n^2

Message Authentication Code (MAC)

MAC



MAC Generation $S: \mathcal{P} \times \mathcal{K} \rightarrow \{0, 1\}^m$

MAC Verification $V: \mathcal{P} \times \mathcal{K} \times \{0, 1\}^m \rightarrow \text{Boolean}$

$$t \leftarrow S(k, p)$$

$$\{\text{true}, \text{false}\} \leftarrow V(p, k, t)$$

MAC generation

$$t = S(k, p)$$

MAC verification

$$t' = S(k, p)$$

return $(t == t')$; ■

Abbiamo Alice e Bob che hanno una chiave condivisa k e due algoritmi S e V , dove:

- S prende in ingresso un messaggio p , la chiave k e produce un hash t . Dunque in questo caso la hash viene calcolata usando **sia la chiave che il messaggio**. Il messaggio e il fingerprint vengono inviati, passano attraverso l'avversario, Bob riceverà la coppia (testo,fingerprint) → (p, t) e per verificare l'integrità del messaggio e l'autenticità del mittente, Bob calcolerà autonomamente sul suo computer una nuova fingerprint (t') a partire dal testo ricevuto e dalla chiave di cui è in possesso e verificherà che t' sia uguale al t ricevuto. Se sono uguali allora la comunicazione è integra e p non è stato modificato dall'attaccante. In questo esempio p è stato trasmesso in chiaro per mettere in evidenza che questo è un problema di integrità e autenticazione non di confidenzialità. Dunque in questo scenario l'avversario vede tutte le coppie (messaggio, relativo hash). Se supponiamo che Alice e Bob si scambino tanti messaggi, l'avversario potrà collezionare tutte queste coppie. Dunque l'algoritmo S deve soddisfare la proprietà indicata con la freccia blu:

Definition of secure MACs

- Ease of computation
 - Given a function S , a key k and an input x , $S(k, x)$ is easy to compute
- Compression
 - S maps an input x of arbitrary finite bit-length into an output of fixed length m
- Computation-resistance
 - For each key k , given zero or more (x_i, t_i) pairs, where $t_i = S(k, x_i)$ (*chosen message attack*), it is **computationally infeasible** to compute (x, t) , s.t. $t = S(k, x)$, for any new input $x \neq x_i$ (including possible $t = t_i$ for some i)
(existential forgery)

ovvero anche se l'avversario può collezionare tante coppie (messaggio, relativo hash) per lui deve essere computazionalmente difficile calcolarsi la hash sul messaggio di sua scelta senza conoscere la chiave. Ovvero se l'avversario crea un messaggio p' e lo vuole mandare a Bob o Alice per imbrogliarlo, non conoscendo la chiave per lui deve essere molto difficile trovare la hash corrispondente.

Ovviamente venendo usata una chiave per tale processo, l'avversario può sempre provare ad usare un attacco alle chiavi fin tanto che non trova la chiave k tale per cui $S(k, p) = t$.

Tirando le somme dunque:

- ✓ efficiente dal punto di vista computazionale
- ✓ effettua una compressione in quanto funziona come una funzione hash
- ✓ per un avversario che non conosce la chiave devono comunque valere le stesse proprietà di prima ovvero pre-image resistance, 2nd pre-image resistance e collision resistance

Esempi di utilizzo del MAC

Combining MAC and ENC

- Let
 - PT message: m ; transmitted message: m' ;
encryption key: e ; MAC key: a
- Schemes
 - SSL
 - $t = S(a, m); c = E(e, m \parallel t), m' = c$
 - IpSec
 - $c = E(e, m); t = S(a, c); m' = c \parallel t$
 - SSH
 - $c = E(e, m); t = S(a, m); m' = c \parallel t$

Supponiamo di voler trasmettere un messaggio m e di avere una chiave di cifratura e e una chiave per calcolare il MAC a .

- Il protocollo **SSL** ad esempio prende il messaggio m , ne calcola la hash usando la chiave di autenticazione a ottenendo t , poi concatena la hash t al messaggio m e cifra il tutto con la chiave e .
- **IPSec** invece effettua prima la cifratura del messaggio $c = E(e, m)$. Poi calcola la **hash del testo cifrato**: $t = S(a, c)$ e infine trasmette il testo cifrato concatenato con la hash $m' = c \parallel t$.
- **SSH** effettua prima la cifratura del messaggio $c = E(e, m)$. Poi calcola la **hash del messaggio in chiaro**: $t = S(a, m)$ e infine trasmette il testo cifrato concatenato con la hash $m' = c \parallel t$.

MAC:

2. MAC guarantees

- confidentiality
- non-repudiation
- authentication and integrity ✓

✗ Incorrect 0/1 Points

3. In a secure MAC

- it is very difficult for the adversary to produce some new text-MAC pairs
- it is very difficult for the adversary to produce a new text-MAC pair ✓
- it is very difficult for the adversary to produce a new text-MAC pair for a text of his choice only

✓ Correct 1/1 Points

4. Which approach to authenticated encryption is always recommended?

- MAC then Encrypt
- Encrypt and MAC
- Encrypt then MAC ✓

✓ Correct 1/1 Points

5. HMAC is a MAC built from

- a cipher
- a hash function ✓
- an ad-hoc algorithm

✓ Correct 1/1 Points

6. CBC-MAC is a MAC built from

- an ad-hoc algorithm
- an hash function
- a cipher ✓

11/03/2023

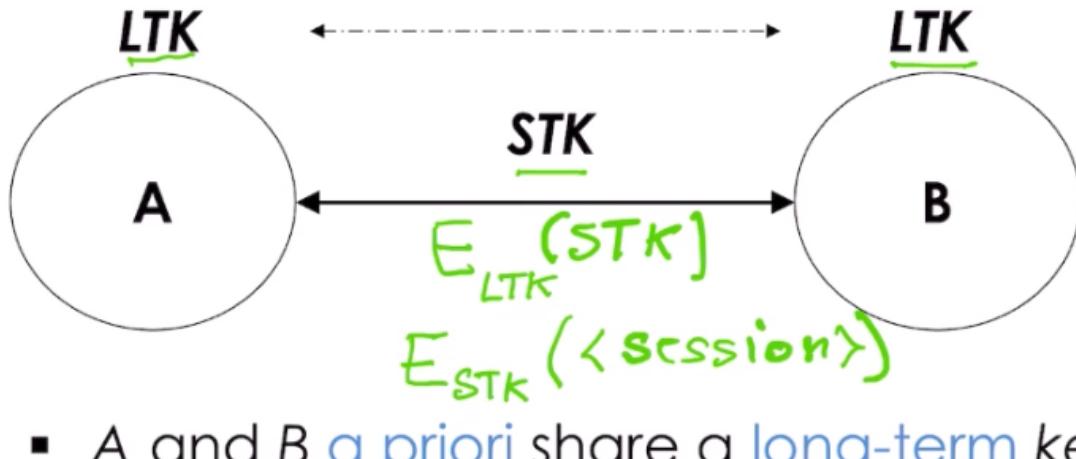
Keys Establishment

Per introdurre la cifratura a chiave pubblica, partiamo da uno dei problemi principali che diede l'impulso alla invenzione della chiave pubblica. Fin ora abbiamo ipotizzato che Alice e Bob hanno stabilito **offline** la chiave simmetrica da usare, usando un canale sicuro e diverso dalla rete sulla quale l'attaccante è in ascolto.

Più avanti invece vedremo come poter scambiare delle chiavi usando la rete.

Introduciamo il concetto di **chiave di sessione**:

Session key (1)



- A and B **a priori share a long-term key LTK**
- Whenever A and B want to communicate they establish a **short-term session key STK**



in cui ipotizzo che Alice e Bob abbiano a disposizione una **chiave a lungo termine LTK** che supponiamo se la siano scambiata incontrandosi al bar. Normalmente non si usa direttamente questa chiave per cifrare i messaggi, ma se ne usa un'altra indicata con **STK** chiamata **chiave di sessione**. Supponiamo che la STK non ci sia, dunque A e B usano solo la LTK per cifrare. Scambiandosi nel tempo i dati cifrati, l'attaccante può provare a fare degli attacchi analitici, in quanto forniscono materiale cifrato all'attaccante che è in ascolto. Avendo abbandonato la sicurezza perfetta, vuol dire che ci sono degli attacchi analitici che possono essere fatti contro gli algoritmi *ma che tendenzialmente non sono convenienti* (troppo tempo, troppo storage, troppi input come il materiale cifrato da analizzare). Ma sono fattibili, dunque avendo molto materiale cifrato, è più probabile riuscire nell'attacco. Dunque la LTK sarebbe bene non usarla per cifrare i dati, in quanto più uso quella chiave più diventa di valore per l'attaccante che cercherà il più possibile di rubarla. **La LTK viene quindi solo usata per distribuire le STK**. Posso dunque usare la LTK per cifrare la STK, in questo modo la STK va sul canale in modo cifrato, dunque anche se l'avversario sta ascoltando vede passare un messaggio cifrato e non riesce a

ricavarsi la STK. Successivamente useranno Alice e Bob la STK per cifrare la sessione. La STK cambierà ad ogni sessione rendendo così la vita difficile all'attaccante in caso di attacco analitico. La dimensione di STK saranno i soliti 128 bit che però vengono usati solo per un breve periodo di tempo (dunque viene usata per la bulk encryption ma viene usata solo per un periodo limitato di tempo):



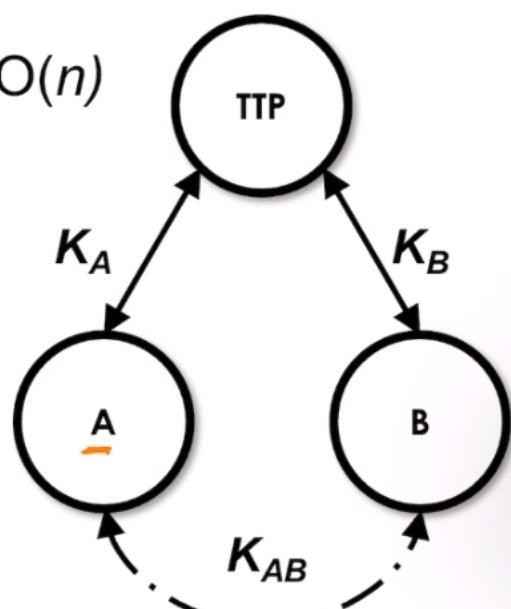
Session key (3)

- Long-term Key
 - Used for long time
 - Encrypt small amount of data (kex proto)
- Short-term Key
 - Session key
 - Used for a short time (a session)
 - Encrypt large amount of data (bulk encryption)

Il meccanismo di seguito presentato viene usato in molte organizzazioni:

(Online) Trusted Third Party

- Each user shares **a long-term key** with TTP
- Every user stores one key
- The overall number of keys is $O(n)$
- TTP is a single point of failure
 - TTP must be always online
 - TTP knows all the keys
 - TTP can read all msg
 - TTP can impersonate any party



Bob potrebbe essere un server aziendale, Alice un utente aziendale e TTP una terza parte fidata.

Ipotizziamo che TTP condivida una chiave a lungo termine con ciascun soggetto (K_A e K_B). Ad

esempio la TTP potrebbe essere il computer dell'amministratore di sistema.

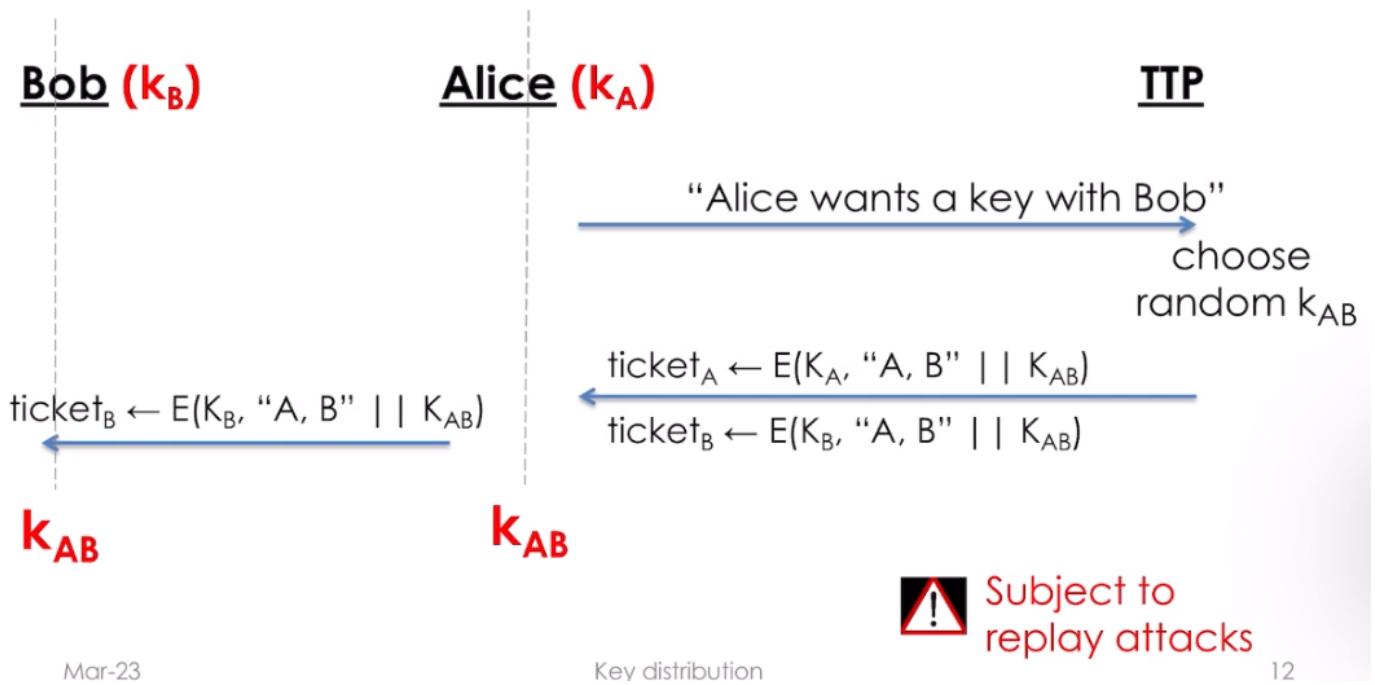
Quando un nuovo impiegato viene assunto in fase di enrollment(A), si presenterà dall'amministratore di sistema (TTP) il quale gli fornirà una chiave. Stesso discorso per il server (B) sul quale al momento dell'installazione viene configurata dall'amministratore una chiave K_B . Se Alice vuole usare il server B in modo sicuro, deve stabilire la chiave di sessione K_{AB} con il server B usando le chiavi K_A e K_B .

Quando la sessione tra l'utente e il server scadrà, la K_{AB} non verrà più usata.

⚠ Il protocollo di seguito presentato non può essere usato nella realtà in quanto vulnerabile a replay attack!

Key distribution: a toy protocol

Alice wants a shared key with Bob.



⚠ Subject to replay attacks

Mar-23

Key distribution

12

Supponiamo che Alice che ha la chiave K_A e voglia parlare con Bob, la terza entità fidata genera una chiave di sessione random e crea due **ticket**, il ticket_A e il ticket_B . Il ticket_A è cifrato con chiave K_A dunque solo Alice è in grado di decifrarlo e tramite cui la TTP comunica ad Alice la chiave di sessione K_{AB} . Ugualmente per il ticket_B il quale può essere decifrato solo da Bob, al cui interno si trova la stessa chiave di sessione K_{AB} inviata ad Alice.

! Dunque la distribuzione della chiave di sessione K_{AB} sembrerebbe essere avvenuta correttamente e senza problemi...

Naturalmente è importante che questo protocollo sia **efficiente** e che la **TTP** sia realmente **fidata**, inoltre deve sempre essere online in quanto se non dovesse essere più disponibile nessuno riuscirebbe più a

parlare, deve essere sicura in quanto se venisse penetrata le chiavi di tutti verrebbero ottenute dall'attaccante.



Key question

- Can we generate shared keys without an online TTP?
 - Answer: YES!
 - Starting point of public-key cryptography
 - Merkle (1974)
 - Diffie-Hellman (1976) ←
 - RSA (1977) ←
 - More recently: ID-based encryption (2001), functional encryption (2011)...

Negli anni 70 qualcuno si pose il problema di riuscire a [ottenere chiavi condivise senza avere una TTP](#) online.

Questo fu il punto di partenza che portò all'invenzione e al consolidamento della cifratura a chiave pubblica.

Diffie-Hellman (1976)

Public key distribution system

- A **public key distribution system** allows two users to securely **exchange a key** over an **insecure channel**
- Most influential paper
 - + Whitfield Diffie and Martin Hellman, **New directions in cryptography**, *IEEE Transactions of Information Theory*, 22(6), pp. 644-654

In questo articolo si pongono di stabilire una chiave condivisa senza terze parti fidate.

Per farlo è necessario introdurre la aritmetica modulare:



Facts on modular arithmetic

- Multiplication is commutative
 $(\underline{a} \times \underline{b}) = (\underline{b} \times \underline{a}) \text{ mod } n$
- Power of power is commutative
 $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b \text{ mod } n$

E' possibile ridefinire l'**esponenziazione**:

Facts on modular arithmetic

- Parameters
 - Let p be prime
 - Let $g \in (1, p)$ be a generator (primitive root), i.e.,
 $\forall 1 \leq x < p, \exists t \text{ s.t. } g^t \bmod p = x$
- DISCRETE EXPONENTIATION
 - Given g, p and x , to compute $y = g^x \bmod p$ is computationally «easy»
- DISCRETE LOGARITHM
 - Given $g, 1 \leq y \leq p-1$, it is computationally «difficult» to determine x ($1 \leq x \leq p - 1$) s.t. $y = g^x \bmod p$

p è un numero primo, ovvero divisibile solo per 1 e per se stesso. Posso inoltre definire l'operazione inversa ovvero il **logaritmo**. L'operazione di esponenziazione con numeri primi molto grandi rimane semplice computazionalmente da fare **ma lo stesso discorso non vale per il logaritmo!!!**.

! ! ! Non esistono infatti ad oggi algoritmi efficienti per calcolare il logaritmo in modo efficiente. Questo è il trucco di base usato in tutta la crittografia a chiave pubblica. ! ! !

Il caso d'uso prevede che Alice e Bob non si siano mai incontrati e non esiste una TTP fidata. Alice e Bob devono solamente conoscere **g** e **p** le quali però non è necessario che siano segrete, possono essere note anche all'avversario !:

Alice pensa un numero naturale $a < p$ e lo tiene segreto.

Bob pensa un numero naturale $b < p$ e lo tiene segreto.

Alice si calcola $Y_A = g^a \bmod p$ e la spedisce a Bob.

Bob calcola $Y_B = g^b \bmod p$ e la spedisce ad Alice.

Quando Bob riceve Y_A calcola $Y_A^b \bmod p$.

Quando Alice riceve Y_B calcola $Y_B^a \bmod p$.

$$Y_A^b = g^a \bmod p = g^{ab} \bmod p$$

$$Y_B^a = g^b \bmod p = g^{ba} \bmod p = g^{ab} \bmod p$$

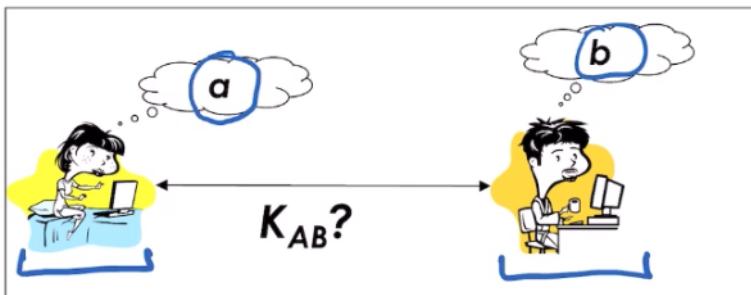


(per le proprietà delle potenze)

Entrambi ottengono esattamente la stessa quantità, **la quale viene usata come chiave**:



The Diffie-Hellman protocol



- Let p be a large prime (600 digits, 2000 bits)
- Let $1 < g < p$
- Let p, g publicly known

Alice chooses a random number a
Bob chooses a random number b

$$M1 \ A \rightarrow B: A, Y_A = g^a \text{ mod } p$$

$$M2 \ B \rightarrow A: B, Y_B = g^b \text{ mod } p$$

Alice computes $K_{AB} = (Y_B)^a \text{ mod } p = g^{ab} \text{ mod } p$

Bob computes $K_{AB} = (Y_A)^b \text{ mod } p = g^{ab} \text{ mod } p$

$$a < b$$

$$\begin{aligned} Y_A &\rightarrow g^a \text{ mod } p \\ Y_B &= g^b \text{ mod } p \end{aligned}$$

$$\begin{aligned} Y_B &\rightarrow g^b \text{ mod } p \\ Y_A &= (Y_B)^a \text{ mod } p \end{aligned}$$

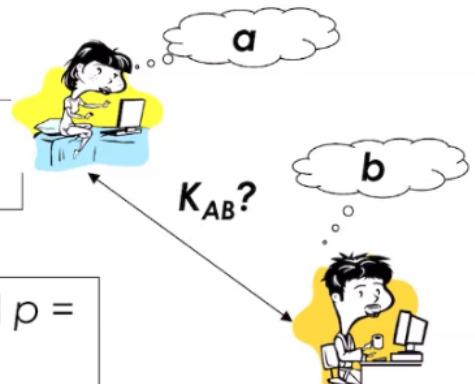
Mar-23

Un esempio con cifre piccole:

Let $p = 11, g = 7$

NUMBERS

Alice chooses $a = 3$ and computes $Y_A = g^a \text{ mod } p = 7^3 \text{ mod } 11 = 343 \text{ mod } 11 = 2$



Bob chooses $b = 6$ and computes $Y_B = g^b \text{ mod } p = 7^6 \text{ mod } 11 = 117649 \text{ mod } 11 = 4$

$A \rightarrow B: 2$

$B \rightarrow A: 4$

Alice receives 4 and computes $K_{AB} = (Y_B)^a \text{ mod } p = 4^3 \text{ mod } 11 = 9$

Bob receives 2 and computes $K_{AB} = (Y_A)^b \text{ mod } p = 2^6 \text{ mod } 11 = 9$

Security of Diffie-Hellman

- Intuition. Eavesdropper sees p , g , Y_A and Y_B and wants to compute K_{AB}
- The Diffie-Hellman Problem
 - Given p , g , $Y_A = g^a \pmod{p}$ and $Y_B = g^b \pmod{p}$, compute $g^{ab} \pmod{p}$
 - How hard is this problem?

L'avversario è in agguato e potrebbe spiare i 2 valori Y_A e Y_B , poi noi sappiamo che l'avversario conosce \boxed{p} e \boxed{g} in quanto li conoscono tutti. Il suo obiettivo è calcolare $g^{ab} \pmod{p}$.

? Quanto è difficile risolvere questo problema?

Si pensa che sia l'unico modo anche se nessuno è riuscito a dimostrarlo matematicamente, ma ad oggi il modo migliore per risolvere questo problema è che:

l'avversario conosce $Y_A = g^a \pmod{p}$ in quanto la prende dalla rete, e sa che è il risultato di \boxed{g} alla qualcosa, non conosce il qualcosa ma conosce \boxed{g} , deve dunque fare il **logaritmo** che però è difficult molto più difficult quanto più grosso è il valore di \boxed{p} .

Non riesce a calcolarsi \boxed{a} in quanto ci sono in gioco numeri troppo grandi dunque l'esponenziazione che sono le operazioni che fanno Alice e Bob sono easy mentre il logaritmo è difficult.

Security of Diffie-Hellman

- If logs (\pmod{p}) are easily computed, then DHP can be easily solved
- There is no proof of the converse, i.e., if logs (\pmod{p}) are difficult then DH is secure **MAI PROVATO**
- We don't see any way to compute K_{AB} from Y_A and Y_B without first obtaining either a or b

Fin tanto che sarà difficile computazionalmente parlando calcolare il logaritmo, Diffie-Hellman come algoritmo sarà sicuro.

Per dare un'idea di quanto sia difficile come problema:

How hard is DHP

- Let p be a n -bit prime ($p < 2^n$)
- Exponentiation takes at most $(2 \times \log_2 p) < 2n$ multiplications (mod p)
 - #mult is linear in the exponent n ; complexity of a multiplication is $O(n^2)$ → complexity of exponentiation is $O(n^3)$
- Taking logs (mod p) requires $p^{1/2} = 2^{n/2}$ operations
- Example $n = 512$
 - Exponentiation requires at most 1024 multiplications
 - Taking logs mod p requires $2^{256} = 10^{77}$ operations

per fare un'esponenziazione, con p su 1024 bit o 2048 bit ed è dell'ordine di 2^n , ci vorranno $2n$ operazioni, dunque se n è uguale a 1024 ci vorranno 2048 operazioni di moltiplicazione che per un computer non sono tante, se invece voglio fare il logaritmo discreto le operazioni saranno $2^{n/2}$ ovvero 10^{77} operazioni!.

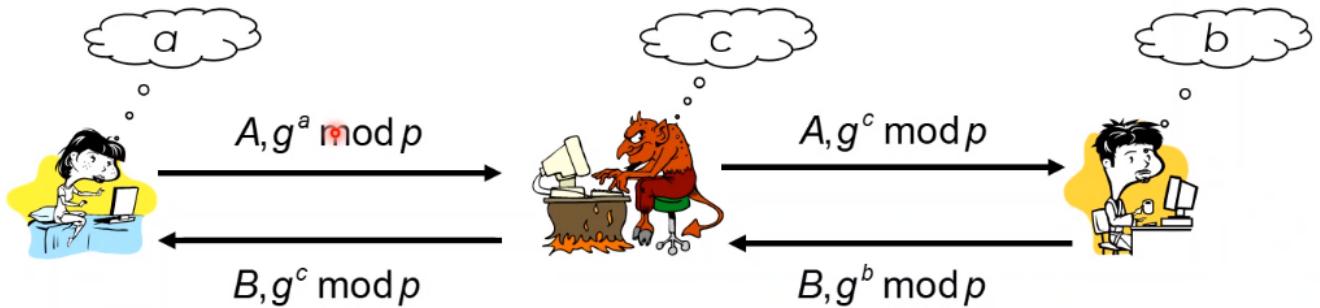
⚠ Con Diffie-Hellman dunque si riesce a stabilire una chiave di sessione senza che A e B si siano mai incontrati e senza che ci sia una terza entità fidata.

Man In The Middle

Lo schema proposto è sicuro fin tanto che l'avversario si limita solamente a leggere le comunicazioni. Nel caso in cui l'avversario è in grado anche di introdurre sulla rete dei nuovi messaggi la situazione si complica.

Infatti in caso di avversario attivo:

Man-in-the-middle



$$K_{AM} = g^{ac} \text{ mod } p$$

$$K_{AM} = g^{ac} \text{ mod } p, \text{ e}$$

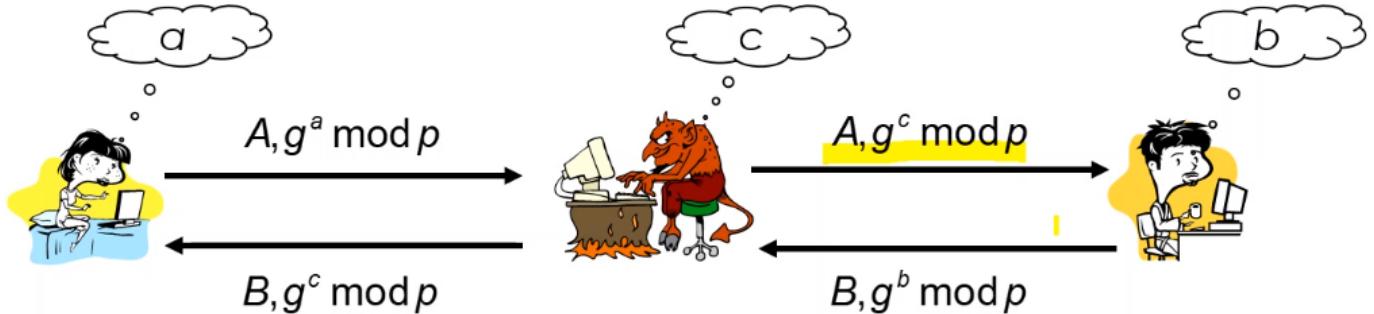
$$K_{BM} = g^{bc} \text{ mod } p$$

$$K_{BM} = g^{bc} \text{ mod } p$$

l'avversario sostituisce Y_A con $Y_C = g^c \text{ mod } p$. Dunque stesso discorso lo farà per Bob, dove al posto di Y_B ci mette Y_C . Dunque Alice ricaverà la chiave K_{AM} , Bob si ricava K_{BM} , e l'avversario è in grado di ricavarsi tutte e due le chiavi. Dunque ci sono 2 canali in quanto l'attaccante è nel mezzo anzichè un unico canale tra A e B.

Vorremmo trovare una soluzione a questo problema indipendentemente dalla rete su cui ci troviamo.

⚠ Il problema è che quando Bob riceve il messaggio in giallo, questo messaggio non porta alcuna prova di provenienza da Alice:



$$K_{AM} = g^{ac} \text{ mod } p$$

$$K_{AM} = g^{ac} \text{ mod } p, \text{ e}$$

$$K_{BM} = g^{bc} \text{ mod } p$$

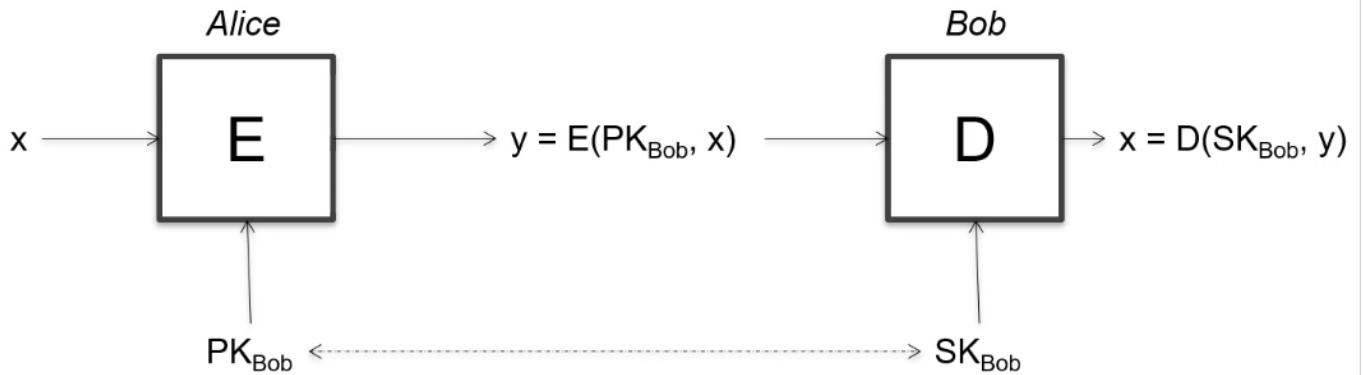
$$K_{BM} = g^{bc} \text{ mod } p$$

Da un punto di vista crittografico non esiste nulla che leggi in modo indissolubile Alice con il suo parametro pubblico ovvero con la sua chiave pubblica (Y_A è la sua chiave pubblica mentre a è la sua chiave privata). In modo equivalente il discorso vale per Bob.

💡 Questo problema si risolve con le firme digitali.

Sicuramente abbiamo risolto una parte del problema in quanto in presenza di avversario passivo riusciamo a scambiare una chiave senza che esista una TTP o si siano mai incontrati prima. Ma se l'avversario è attivo è necessario ricorrere ai certificati.

Public key encryption



- PK_{Bob} : public key
- SK_{Bob} : private key
- Alice knows Bob's public key PK_{Bob}
- Bob keeps secret his own private key SK_{Bob}

Se Alice vuole inviare un messaggio in modo confidenziale a Bob, prende la chiave pubblica di Bob, la usa come chiave di cifratura del messaggio x , il crittogramma y viene inviato a Bob il quale lo decifra usando la sua chiave privata riottenendo il testo in chiaro x . Questo è un tipo di comunicazione molti a uno (N:1) in quanto tutti coloro che conoscono la chiave pubblica di Bob possono inviargli un messaggio segreto.

Public key encryption

- [DEF] A public key encryption scheme is a triple of algorithms (G, E, D) s.t.
- G is a randomized algorithm for key generation (pk, sk)
- $c = E(pk, m)$ is a *randomized* algorithm that takes $m \in M$ and outputs $c \in C$
- $m = D(sk, c)$ is deterministic algorithm that takes $c \in C$ and outputs $m \in M$
- They fulfill the consistency property:

$$\forall (pk, sk), \forall x \in M, D(sk, E(pk, x)) = x$$

L'algoritmo di generazione delle chiavi quando si parlava di crittografia a chiave simmetrica era sostanzialmente una generazione di bit random, nel caso di Diffie-Hellman invece l'algoritmo di generazione delle chiavi è più complesso in quanto la chiave viene generata in modo random ma la chiave pubblica viene generata da un calcolo matematico avente una stretta dipendenza con la chiave privata:

 chiave privata

$Y_A = g^a \text{ mod } p$  chiave pubblica

Security properties of PKE (informal)

- Security property 1
 - Known $pk \in K$ and $y \in C$, it is computationally infeasible to find the message $x \in M$ such that $y = E(pk, x)$
- Security property 2
 - Known the public key $pk \in K$, it is computationally infeasible to determine the corresponding secret key $sk \in K$

1. Se conosco la chiave pubblica pk e il crittogramma y , deve essere computazionalmente difficile ricavare x .
2. Se conosco la chiave pubblica pk deve essere molto difficile ricavarmi la corrispondente chiave segreta sk in quanto dovrei risolvere il logaritmo discreto, che risulta molto difficile da risolvere. Vedremo che non si parla più di impossible ma di difficult ovvero computazionalmente difficile.

Observations

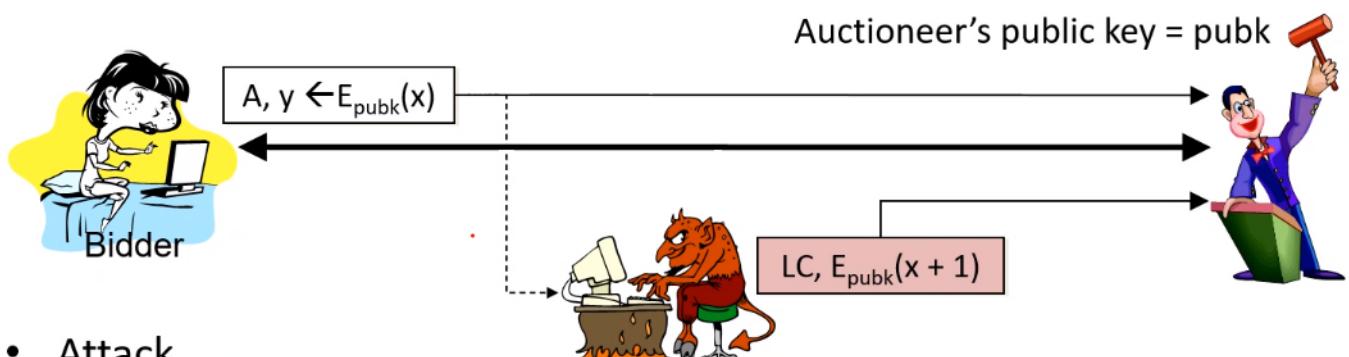
- Any PK encryption scheme cannot be perfect
- PK encryption algorithms generally rely on hard problems from number theory and algebra

Si può dimostrare che un cifrario a chiave pubblico non può essere perfetto e quindi tutti i cifrari a chiave pubblica sfruttano dei problemi complicati.

Supponiamo che Alice voglia fare un offerta (bid) ad un asta. Per farla deve conoscere la chiave pubblica dell'asta. Alice cifra l'offerta con la chiave pubblica dell'asta in quanto non vuole che eventuali concorrenti vedano la sua offerta.



Attack against small plaintext space



- Attack
 - The adversary tries all possible values v for the bid (e.g., 2^{32}) until he finds a value v^* such that $y == E(\text{pubK}, v^*) \Rightarrow x = v^*$. Then, the adversary sends a bid $b = x + 1$
 - It's a consequence of its being non-perfect
- Solution: **salting (randomization)**:
 - $r \leftarrow \text{random}(); y \leftarrow E_{\text{pubk}}(r || x)$

L'avversario vede passare Y che è cifrato con la chiave pubblica dell'asta. L'avversario però può prendere tutti i possibili valori dell'offerta (bid) tale per cui può provare a cifrare con la chiave pubblica tutti i possibili valori che l'offerta può assumere, ad esempio 2^{32} per computer a 32 bit, dopo di che può verificare se il valore che ha prodotto è uguale a quello inviato da Alice e letto dalla rete. Se i valori sono uguali con alta probabilità avrà trovato l'offerta che Alice ha fatto (v^*) e dunque gli basterà fare un offerta di poco più alta*.

L'aspetto cruciale che permette questo tipo di attacco è che l'avversario ha accesso alla chiave pubblica esattamente come tutti gli altri utenti che comunicano sulla rete.

Per contrastare questo problema si introduce una randomizzazione e si prende il messaggio x e gli si appicca un numero random r gonfiando in modo artificiale il messaggio x . Ecco perchè è necessario il **salting**.

Aggiungendo ad esempio un salt da 64 bit, l'attaccante dovrà fare $2^{(64+32)}$ tentativi.

Families of PK Cryptography

- Integer factorization schemes (mid 70s)
 - Most prominent scheme: RSA
- Discrete Logarithm Schemes (mid 70s)
 - Most prominent schemes: DHKE, ElGamal, DSA
- Elliptic Curves Schemes (mid 80s)
 - Generalization of the Discrete Logarithm
 - Most prominent schemes: ECDH, ECDSA

Alcuni algoritmi usano come problema difficile da risolvere la **fattorizzazione dei numeri primi** altri la risoluzione del logaritmo discreto altri ancora la risoluzione di curve ellittiche.

Families of PK Cryptography

- Other schemes
 - Multivariate Quadratic, Lattice
 - They lack maturity
 - Poor performance characteristics
 - Hyperelliptic curve cryptosystems
 - Secure and efficient
 - They have not gained widespread adoption
 - Lattice-based
 - Post Quantum Cryptography

Se vogliamo avere una sicurezza ad 80 bit, ovvero per poter rompere l'algoritmo l'avversario deve strutturare un algoritmo che abbia 2^{80} cicli, si dice che abbiamo una sicurezza di 80 bit. Vuol dire che questi algoritmi a chiave pubblica sono molto lenti guardando questa tabella, perché più grande è la chiave e più conti bisogna fare:

Key Lengths and Security Level

Algorithm Family	Cryptosystem	Security Level			
		80	128	192	256
Integer Factorization	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Discrete Logarithm	DH, DSA, ElGamal	1024 bit	3072 bit	7680 bit	15360 bit
Elliptic curves	ECDH, ECDSA	160 bit	256 bit	384	512 bit
Symmetric key	AES, 3DES	80 bit	128 bit	192 bit	256 bit



RULE OF THUMB - The computational complexity of the three pk algorithm families grows roughly with the cube bit length

2^{80}

Per avere una sicurezza a 256 bit ho bisogno per diffie hellman di circa 15000 bit.

Per le curve ellitiche possiamo ridurre l'ordine di grandezza di tanto cambiando l'algoritmo. Dunque la curva ellitica pur essendo crittografia a chiave pubblica sono più veloci.

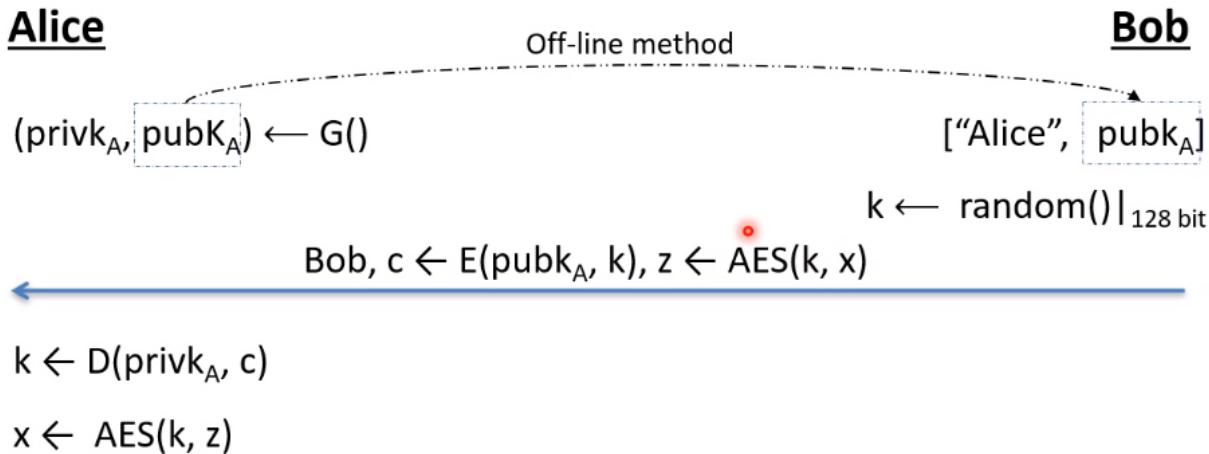
Key transportation

La crittografia a chiave pubblica è estremamente lenta.

Supponiamo che Bob consca la chiave pubblica di Alice (pubK_A), Bob si inventa una chiave simmetrica

(k) con AES ad esempio, la cifra con la chiave pubblica di Alice, che corrisponde a una cifratura lenta ma la chiave privata è piccola, 128 bit, dunque anche se lenta è tollerabile in termini di tempo. Poi i dati grossi (x), come ad esempio un film, li cifra con un algoritmo simmetrico usando una chiave simmetrica. Il vantaggio è che con la cifratura a chiave pubblica non mi è richiesta la TTP e non è

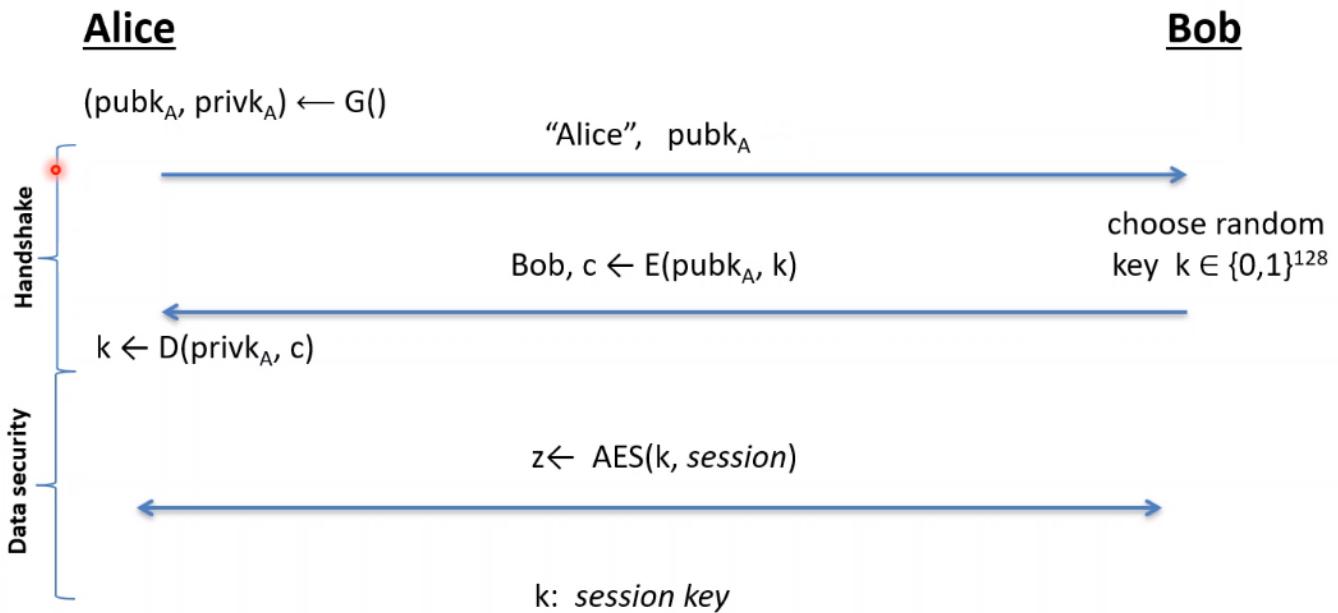
Digital envelope



Quando Alice riceve il messaggio, per prima cosa decifra c con la chiave privata e ottiene la chiave k simmetrica, poi usa la chiave simmetrica per decifrare i dati. Questo schema viene usato ad esempio dalle PEC.

Una variante di questa è usata nella comunicazione tra client e server:

Basic key transport protocol



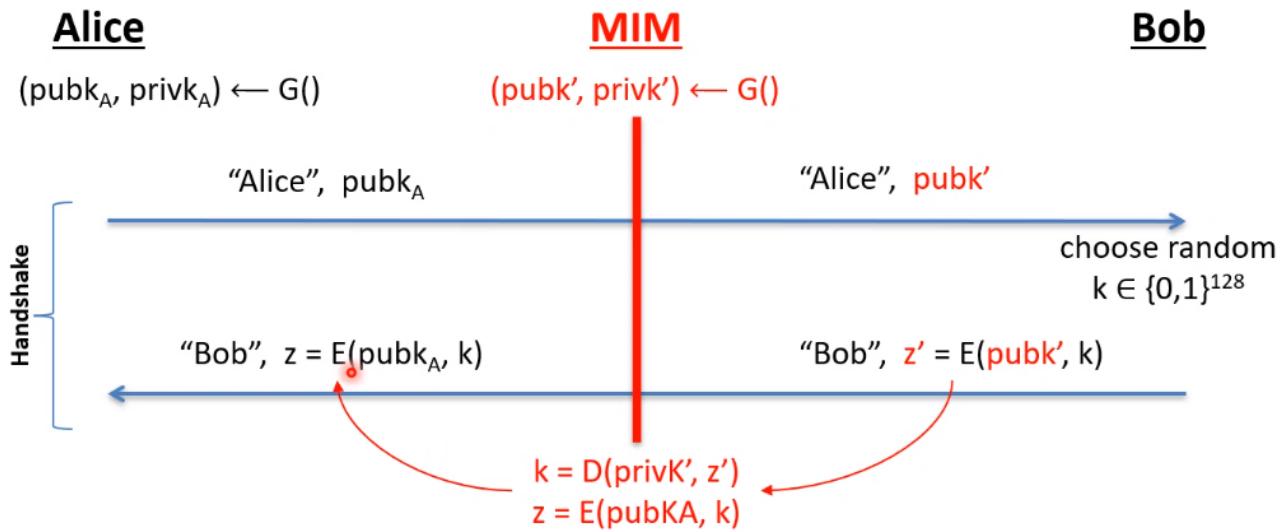
Disclaimer: it's a toy protocol

L'uomo nel mezzo può però cercare di intromettersi nell'handshake: quindi quando Alice spedisce a

Bob il messaggio, l'attaccante potrebbe sostituire la chiave di Alice con la sua e Bob penserebbe di parlare direttamente con Alice anche se in realtà sta parlando con l'attaccante:

Man-in-the-middle (MIM) attack

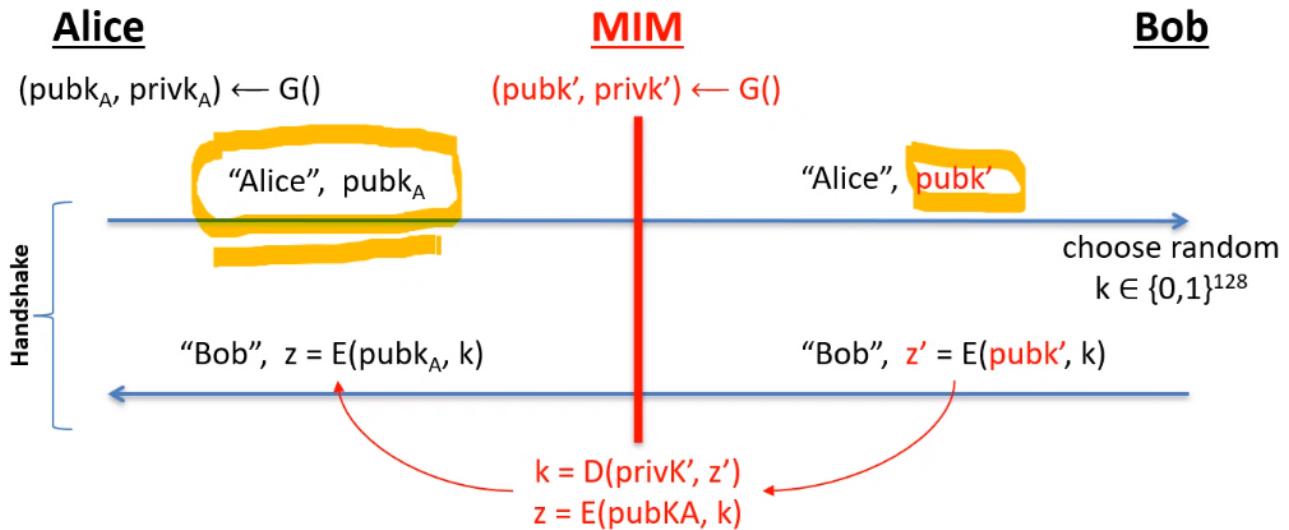
The protocol is insecure against **active** attacks



Il problema del MITM è un problema di tutti gli schemi di cifratura a chiave pubblica. Il problema sta nel fatto che non esiste nulla che leghi l'identificatore di Alice ad Alice stessa. Questo non è un problema di confidenzialità ma di **AUTENTICITA'** !

Man-in-the-middle (MIM) attack

The protocol is insecure against **active** attacks



MIM - comments

- MIM attack is an **active** attack
- Lack of key authentication makes MIM possible
- Certificates are a solution

Key Establishment:

2. Point-to-point key pre-distribution *

- suffers from scalability problems ✓
- suffers from security problems
- suffers from both

✓ Correct 1/1 Points

3. Key establishment with Key Distribution Center *

- avoids pre-distribution of long-term keys
- requires pre-distribution of long-term keys ✓
- the one or the other, according to the setting

✗ Incorrect 0/1 Points

4. The Key Distribution Center is *

- a performance point-of-failure only
- a security point-of-failure only
- is both a performance and security point-of-failure ✓

✗ Incorrect 0/1 Points

5. Select the correct sentence, where DH = Diffie-Hellman and DLP = Discrete Logarithm
Problem. *

- If DH is secure then DLP is hard
- If DLP is hard then DH is secure
- If DLP is easy, then DH is insecure ✓

✓ Correct 1/1 Points

6. Select the correct sentence. *

- Plain DH is secure against a MIM-attack
- Plain DH is insecure against a MIM-attack ✓
- Plain used to be insecure against a MIM-attack

Public Key Encryption:

2. A public key cipher

- can be perfect
- cannot be perfect ✓
- may be perfect depending on the length of the secret key

✓ Correct 1/1 Points

3. In a public key scheme, private and public keys are

- independent of each other
- randomly generated
- algorithmically related ✓

✗ Incorrect 0/1 Points

4. In a public key encryption scheme, randomization at encryption time is

- necessary ✓
- useless
- recommended

✓ Correct 1/1 Points

5. Public key encryption is typically used for

- bulk encryption
- key exchange/transport ✓
- both

✓ Correct 1/1 Points

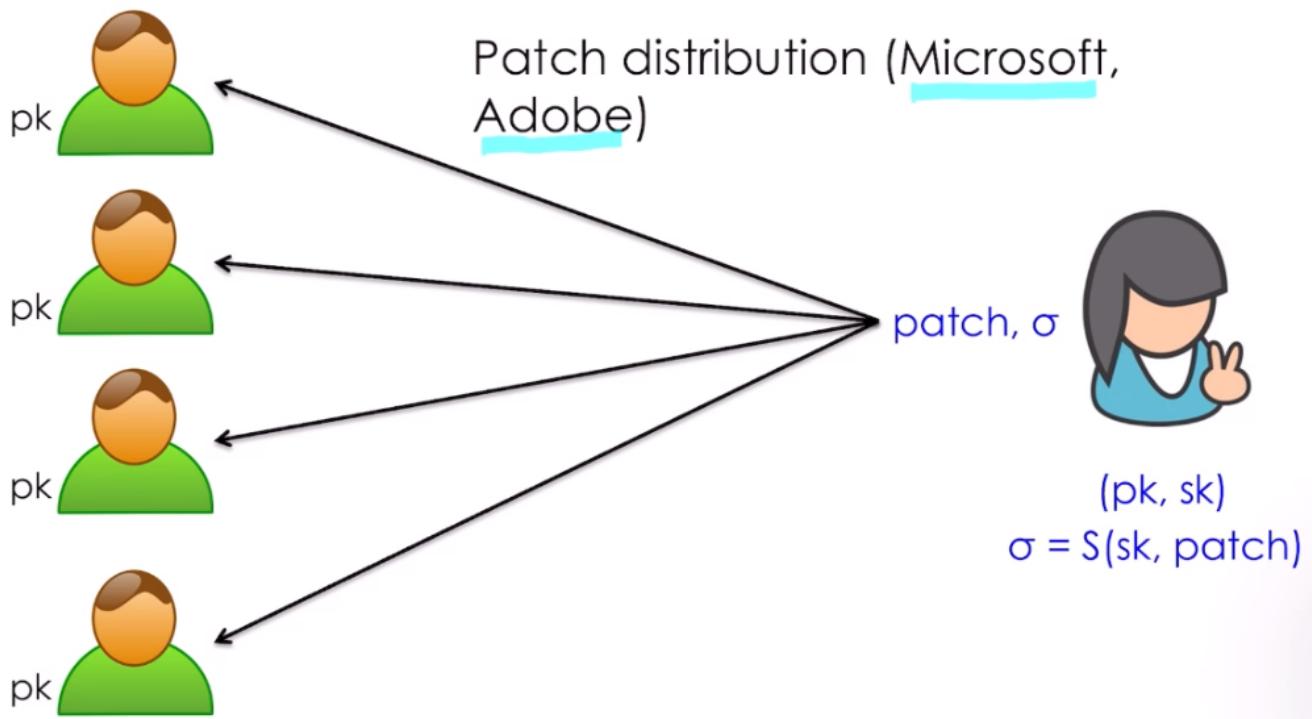
6. Without public key authentication,

- certain public key cryptosystems suffer from MIM attack
- any public key cryptosystem suffers from MIM attack ✓
- old public key cryptosystems suffer from MIM attack

Digital signatures

...

Prototypical application



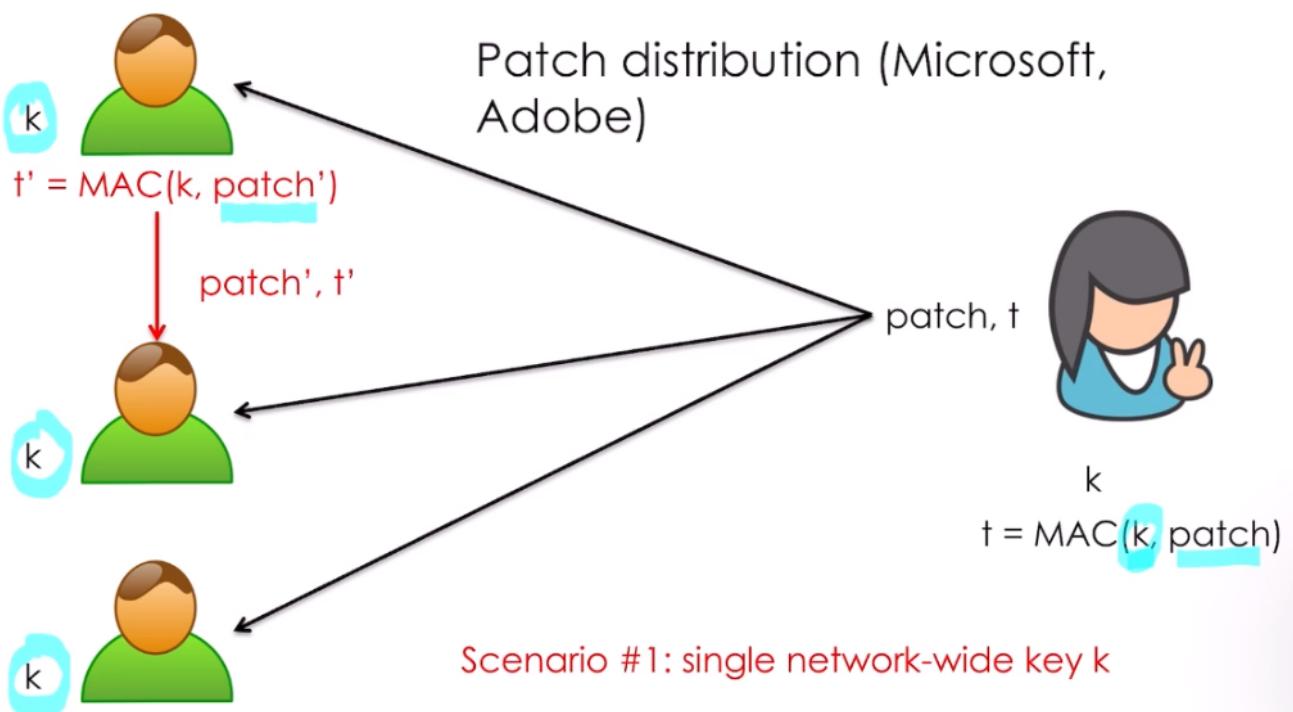
Mar-23

Digital signatures

4

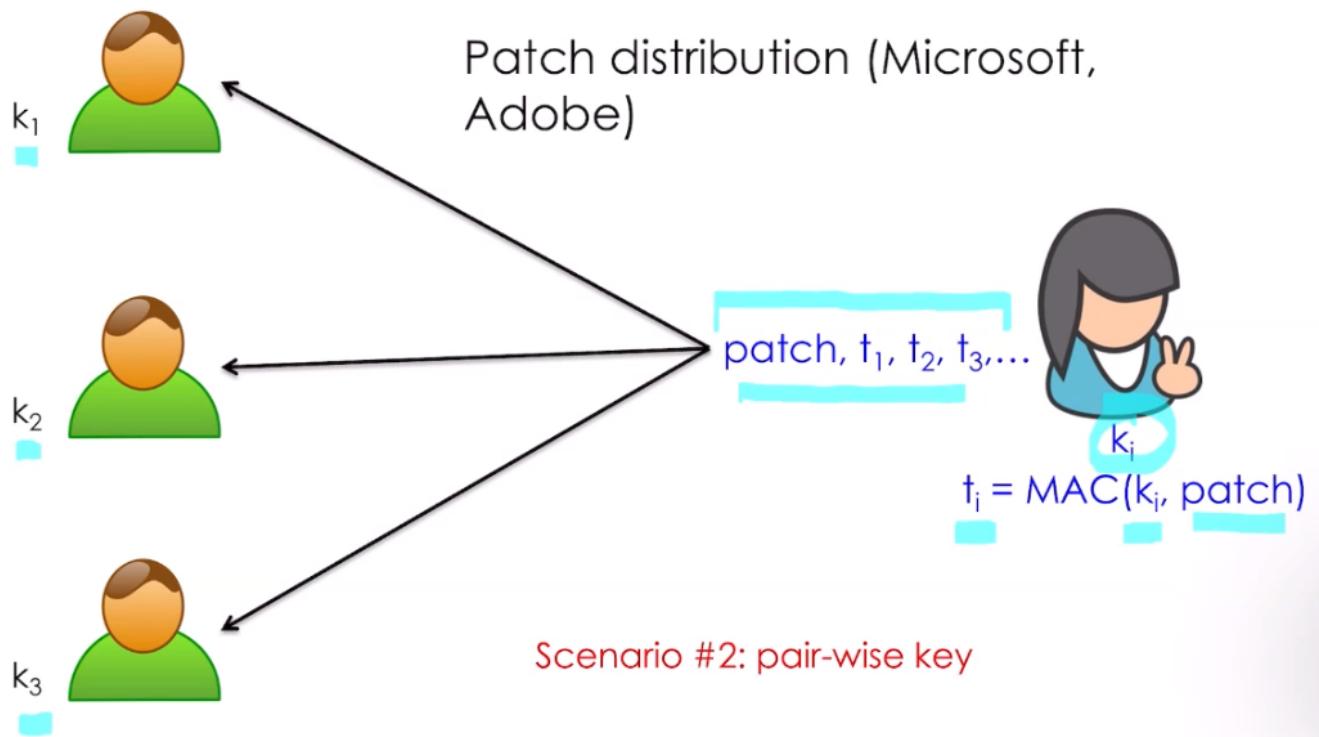
L'aggiornamento viene firmato con la chiave privata del produttore (di Microsoft ad esempio) e la patch con la firma viene inviata ad ogni computer. Ogni dispositivo che riceve la patch, verifica la firma digitale ovvero verifica che provenga effettivamente da Microsoft e che non sia stata modificata durante la trasmissione ed infine la installa. Questa è una forma di comunicazione 1:N. Se la stessa cosa volessi farla con il MAC sarebbe difficile in quanto la chiave K dovrei darla a tutti:

Comparison to MACs



e comunque ogni utente potrebbe modificare la patch e ridistribuirla impersonando Microsoft.
Allora l'alternativa potrebbe essere avere tante chiavi quanti sono gli utenti:

Comparison to MACs



da un punto di vista commerciale significa che dentro ogni copia di Windows dovrei avere una copia diversa della chiave.

Il MAC si usa nelle comunicazioni punto-punto non nelle comunicazioni 1:N, in questi casi si usa la firma digitale.

Comparison to MACs

- Public verifiability
 - DS: anyone can verify the signature
 - MAC: Only a holder of the key can verify a MAC tag
 - Transferability
 - DS can forward a signature to someone else
 - MAC cannot
 - Non-repudiability
 - DS is not repudiable
 - MAC is repudiable
-
- Il vantaggio della firma digitale è che chiunque può verificare la firma mentre con il MAC solo chi è in possesso della chiave può verificare il tag del MAC.
 - La **NON repudiabilità**: Alice usa la propria chiave privata che è un segreto dunque mentre per verificare la firma può farlo chiunque mentre per generarla ci vuole la chiave privata che essendo un segreto può essere stata generata solo da Alice, in quanto solo Alice conosce la chiave privata.

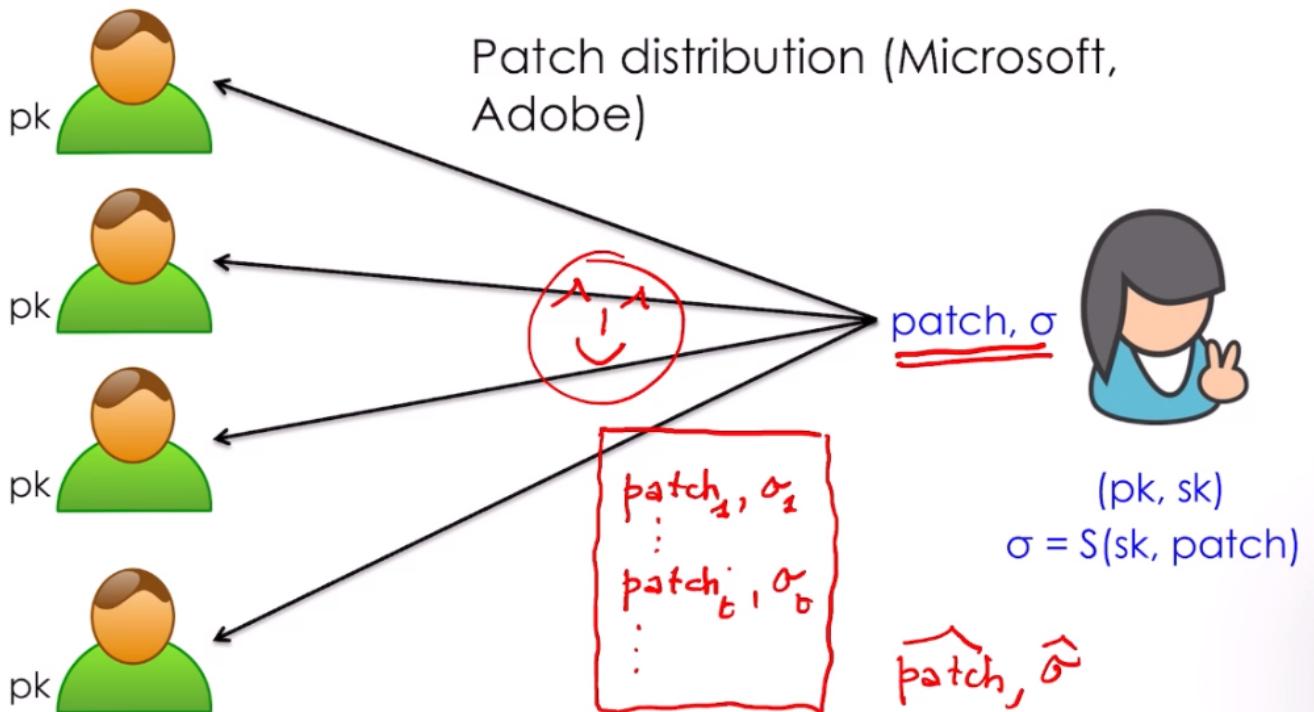
Lo schema:

Digital signature scheme

- A *signature scheme* is defined by three algorithms (G, S, V):
 - **Key generation algorithm** G takes as input 1^n and outputs (pk, sk)
 - **Signature generation algorithm** S takes as input a private key sk and a message m and outputs a signature $\sigma = S(sk, m)$
 - **Signature verification algorithm** V takes as input a public key pk , a signature σ and (optionally) a message m and outputs True or False: $V(pk, \sigma, [m]) \rightarrow \text{true} \mid \text{false}$
 - **Consistency.** For all m and (pk, sk) , $V(pk, [m], S(sk, m)) = \text{TRUE}$

L'avversario potrebbe registrarsi tutte le coppie che vede passare:

Prototypical application



una firma digitale è considerata sicura se l'avversario non riesce in modo efficiente a calcolare la firma

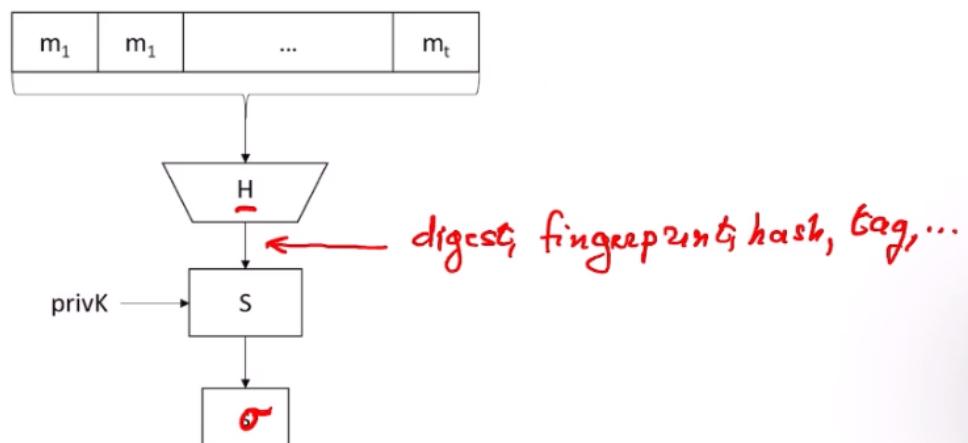
Security model

- Threat model
 - Secure w.r.t. adaptive chosen-message attack
 - The attacker knows the public key
 - The attacker can induce the sender to sign *messages of the attacker's choice*
- Security goal
 - ~~Existential unforgeability~~
 - Attacker should be *unable* to forge valid signature on **any** message not signed by the sender

Quello che si fa tendenzialmente quando abbiamo un messaggio da firmare, il messaggio viene prima compresso e fatto passare attraverso una funzione hash e poi si va a farne la firma digitale:

Hash functions vs Digital Signatures

- Hash functions
 - Allow efficient and secure digital signatures of messages of any length



questo perchè la firma digitale è un algoritmo matematico che richiede tanti conti!

Hash functions vs digital signatures

- Recall Hash functions properties
 - Pre-image resistance
 - Second pre-image resistance
 - Collision resistance
- These properties are crucial for digital signatures security



Hash functions vs digital signatures

- Preimage resistance
 - Digital signature scheme based on (school-book) RSA
 - (n, d) is Alice's private key;
 - (n, e) is Alice's public key
 - $s = (H(m))^d \pmod{n}$
 - THM. If $H()$ is not pre-image resistant => existential forgery
 - Proof
 - Select $s < n$
 - Compute $y = s^e \pmod{n}$
 - Find m such that $H(m) = y (\Leftarrow)$
 - Claim that s is the digital signature of m

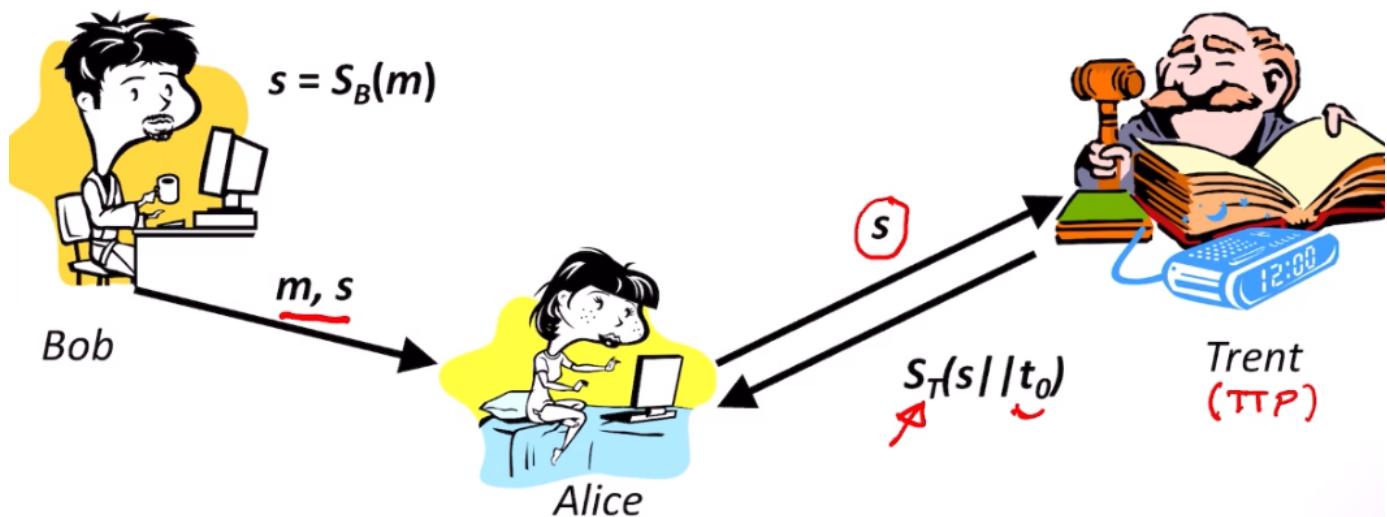
Le firme digitali usate ad oggi come ad esempio RSA è stata inventata nel 1978 oppure la DSA/DSS inventata nel 1985.

Dig sig vs non-repudiation

- Data origin authentication as provided by a digital signature is valid only while the secrecy of the signer's private key is maintained
- A threat that must be addressed is a signer who intentionally discloses his private key, and thereafter claims that a previously valid signature was forged
- How to address this threat
 - Prevent direct access to the key
 - Use of a trusted timestamp agent
 - Use of a trusted notary agent

Potrebbe essere possibile che un attaccante riesca a rubare la chiave privata di un altro utente e riesca a produrre una firma digitale

Trusted timestamping service



- Trent certifies that digital signature s **exists** at time t_0
- If Bob's priv-key is compromised at $t_1 > t_0$, then s is **valid**

La TTP afferma che all'istante t_0 la firma digitale S esisteva. A questo punto Bob non può più ripudiare la firma digitale e se dovesse accadere.

Certificate

Certificatea and Certification Authority

- A *Certificate* is a data structure that cryptographically links the identifier of a subject to its public key (and other stuff)
$$\text{Cert}_A = A, \text{pk}_A, L_A, S_{CA}(A|\text{pk}_A|L_A)$$
 - With A : user's identifier; pk_A : user's public key; and L_A : validity interval
 - The link is represented by the digital signature of Certification Authority (trusted third party)
- A *Certification Authority* is TTP that attests the authenticity of a public key

Un certificato contiene alcune componenti fondamentali. Contiene:

- A : l'identificatore del soggetto
- pk_A : la chiave pubblica
- L_A : periodo di validità
- $S_{CA}(A|\text{pk}_A|L_A)$: firma digitale su queste 3 quantità concatenate fatta dalla certification authority con la propria chiave privata.

CA's obligations

- CA must be reliable
 - CA must verify that the name (Alice) goes along with the key (privKA)
 - CA must verify that the owner of (pK, pK) pair is really entitled to use that name
 - CA establishes rules/policies to verify that a person has rights to the name
- CA's certificate must be (immediately) available
 - Ex. CA's certificate is embedded in a browser installation package
 - Ex. CA's certificate is released at user registration time

La CA deve saper verificare che il nome Alice vada insieme alla chiave e che il possessore delle chiavi sia effettivamente autorizzato a usare quel nome.

Dato che un certificato ha un periodo di validità, tutte le firme digitali fatte al di fuori di questo periodo sono tecnicamente legale ma da un punto di vista "normativo" non deve considerarsi valida.

All'interno del periodo di validità in cui posso usare la firma digitale, può avvenire che il dispositivo di firma venga trafugato o smarrito, a quel punto il certificato è sempre valido ma avendo perduto la chiave privata ogni certificato emesso usando quella chiave deve essere **revocato!**

Verifica del certificato



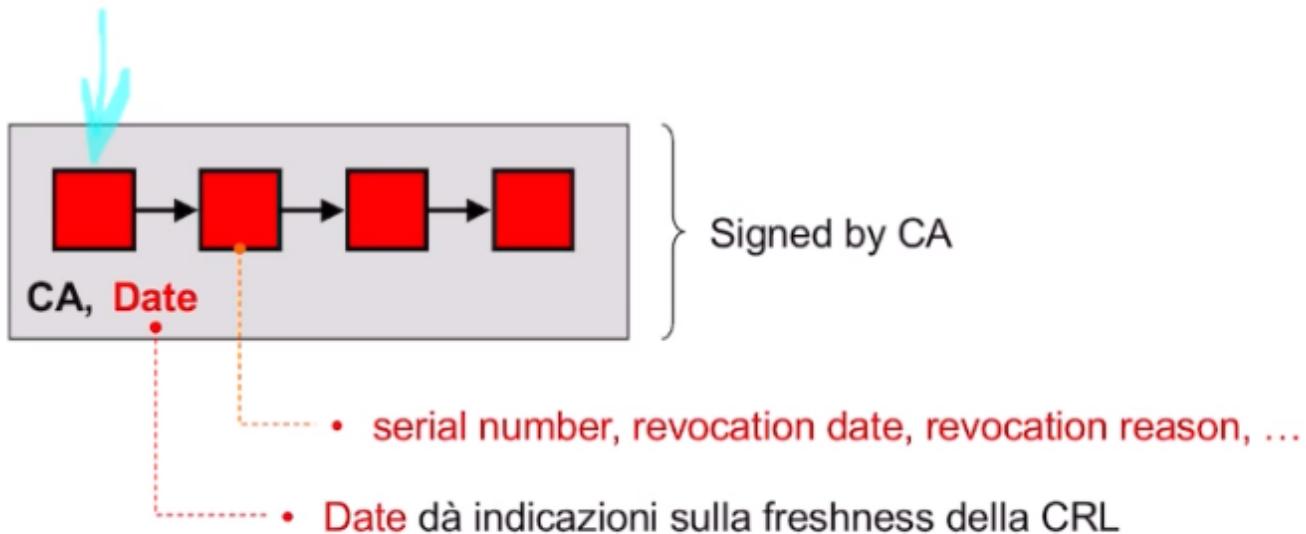
How to verify a certificate

- Bob verifies authenticity of pk_A using $Cert_A$
 - a) Verify validity of CA's public key
 - a) Bob obtains CA's public key pk_T [once at set-up]
 - b) Verify the digital signature in $Cert_A$ by using CA's public key
 - c) Verify that certificate $Cert_A$ is valid (within L_A)
 - d) Verify that certificate $Cert_A$ is not revoked
 - e) If all these checks are successful, then Bob accepts pk_A as authentic key of Alice

- b) verificare la firma digitale usando la chiave pubblica della CA
- c) verificare che il certificato sia valido
- d) verificare se il certificato è stato revocato
- e) a quel punto Bob accetta la chiave pk_A come chiave autentica di Alice

CRL

- A revoked certificate lies in CRL until expiration



An example: X.509

A data structure with several fields

- | | |
|--------------------------------------|--------------------------------------|
| 1. Version | 7. Subject public key information |
| 2. Serial number | 8. Issuer unique identifier (v=2,3) |
| 3. Signature algorithm identifier | 9. Subject unique identifier (v=2,3) |
| 4. Issuer distinguished name | 10. Extensions (v=3) |
| 5. Validity interval | 11. Signature |
| 6. Subject <u>distinguished name</u> | |

X.509 uses the Abstract Syntax Notation, ASN.1, (RFC 1422)

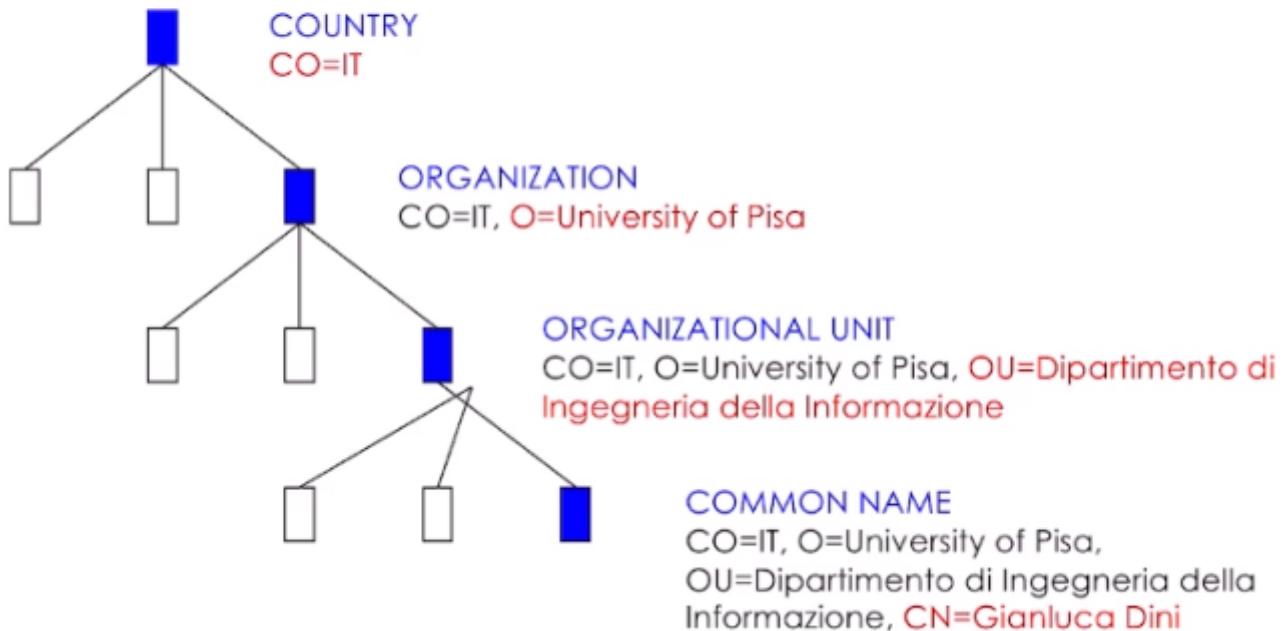
X.509 has been conceived for X.400 mail standard

X.509 uses Distinguished Names

Lo standard X.500 mi permette di sapere come sono fatti gli indicatori presenti all'interno del certificato

Distinguished names

X. 500



La certification authority che ha rilasciato questo certificato è in questo esempio VeriSign Trust Network:

Example: <https://www.mps.it>

Certificate name

www.mps.it

Consorzio Operativo Gruppo MPS

Terms of use at www.verisign.com/rpa (c)00

Florence

Italy, IT

Issuer

VeriSign Trust Network

www.verisign.com/CPS Incorp. by Ref. LIABILITY LTD.(c)97 VeriSign

Details

Certificate version: 3

Serial number: 0x652D0F8ADAB4C7B168A27BBD1C3E9D9D

Not valid before: Mar 2 00:00:00 2004 GMT

Not valid after: Mar 2 23:59:59 2005 GMT

Fingerprint: (MD5) CA CA 88 08 EC D0 8E 49 A6 9A 66 C4 69 31 E0 AE

Fingerprint: (SHA-1) 82 64 CB 69 F0 43 86 43 FF B4 55 D4 25 EF 51 60 65 46 D3 87

il certificato ha un numero seriale che lo identifica in modo univoco in tutto il mondo, presenta inoltre un periodo di validità. L'immagine di seguito racchiude la chiave pubblica di Alice composta da 2 numeri

(e,n) e la firma digitale della CA:

Example: <https://www.mps.it>

Public key algorithm: rsaEncryption

Public-Key (1024 bit):

Modulus:

```
00: E1 80 74 5E E7 E5 54 8B DF 6D 00 95 B5 96 27 AC  
10: 66 93 E0 49 B9 6F 5B 73 53 1C BE 1C EB 47 64 B2  
20: 12 95 70 E6 CD 50 67 02 88 E3 EE 9D B1 91 49 C8  
30: 8D 58 19 4B 86 8F C0 2E 65 E8 F2 D4 82 CC 55 DB  
40: 43 BC 66 DA 44 2F 53 B3 48 4B 37 15 F3 AB 67 C1  
50: 69 B4 53 23 19 30 1A 19 23 7F 28 E0 E3 C0 6B 18  
60: FF 84 C4 AC A9 74 28 DB FF E9 48 CA 75 D5 35 D6  
70: 46 FB 7D D4 A7 3F A1 4B 00 60 14 DC D5 00 CF C7
```

Exponent:

```
01 00 01
```

Public key algorithm: sha1WithRSAEncryption

```
00: 23 A6 FE 90 E3 D9 BB 30 69 CF 43 2C FD 4B CF 67  
10: D7 3C 46 22 9A 08 DB 05 1D 45 DC 07 F3 1E 4D 1F  
20: 4B 11 23 5B 42 91 14 95 25 88 1F BD 60 E5 6F 84  
30: 44 70 7A 95 EC 30 E4 46 4F 37 87 F1 B2 FA 45 04  
40: 6F 7C BE 97 25 C7 20 E7 F3 90 55 51 99 3A 72 35  
50: 40 F2 E8 E3 36 3A 7D 58 61 9C 91 D6 AC 34 E7 E8  
60: 09 27 64 4F 2C 4C C2 D2 A3 32 DB 2B 7E F0 B6 F3  
70: 69 96 E4 2B C3 2B 42 ED CA 2C 3C C8 F5 AA E6 71
```

RSA

(e, n)

pk_A

$Z^{16} + 1$

S_{CA}

contd

Mar-23

PKIs

20

Contiene inoltre ulteriori informazioni come ad esempio dove si trova in rete la CRL:

Example: <https://www.mps.it>

Extensions:

X509v3 Basic Constraints: CA:FALSE

X509v3 Key Usage: Digital Signature, Key Encipherment

X509v3 CRL Distribution Points:

URI:<http://crl.verisign.com/Class3InternationalServer.crl>

X509v3 Certificate Policies:

Policy: 2.16.840.1.113733.1.7.23.3

CPS: <https://www.verisign.com/rpa>

X509v3 Extended Key Usage: Netscape Server Gated Crypto, Microsoft Server Gated Crypto, TLS Web Server Authentication, TLS Web Client Authentication

Authority Information Access:

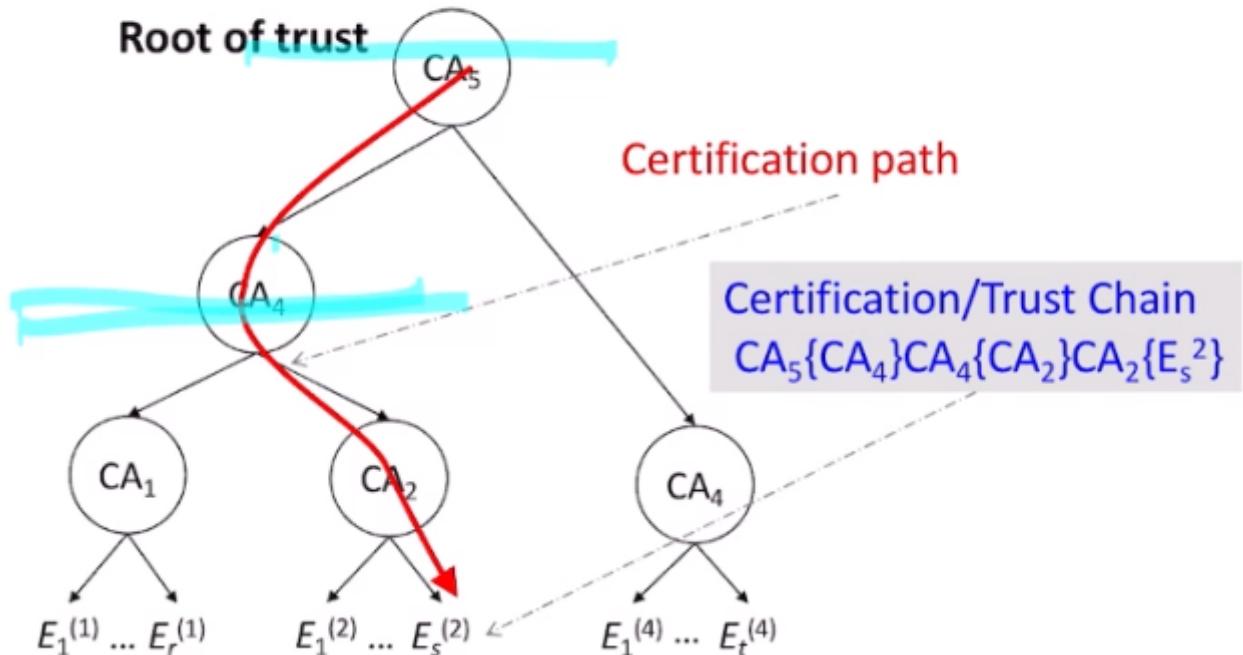
OCSP - URI:<http://ocsp.verisign.com>

Unknown extension object ID 1 3 6 1 5 5 7 1 12:

0_].[0Y0WOU..image/gif0!0.0...+.....k..j.H.,{..0%.#<http://logo.verisign.com/vslogo.gif>

In genere si ha una certification authority di primo livello che rilascia dei certificati ad altre CA di secondo/terzo/etc.. livello. Esistono infatti delle CA radice che permettono di centralizzare l'informazione riguardo la catena di certificazione dei certificati rilasciati a livello mondiale:

Centralized Trust Model



Per poter rilasciare certificati una CA deve possedere un certificato con il campo **CA:TRUE**



Esempio: <https://www.mps.it>

Extensions:

X509v3 Basic Constraints: **CA:TRUE, pathlen:0**

X509v3 Certificate Policies:

Policy: 2.16.840.1.113733.1.7.1.1

CPS: <https://www.verisign.com/CPS>

X509v3 Extended Key Usage: TLS Web Server Authentication, TLS Web Client

Authentication, Netscape Server Gated Crypto, 2.16.840.1.113733.1.8.1

X509v3 Key Usage: Certificate Sign, CRL Sign

Netscape Cert Type: SSL CA, S/MIME CA

X509v3 CRL Distribution Points:

URL:<http://crl.verisign.com/pca3.crl>



Firma digitale:

✓ Correct

2. Digital signatures make it possible to achieve □₄₈

- confidentiality
- authenticity and integrity only
- authenticity, integrity and non-repudiation ✓

✓ Correct

3. MAC make it possible to achieve □₄₉

- confidentiality
- authenticity and integrity only ✓
- authenticity, integrity and non-repudiation

✓ Correct

4. In a situation of conflicting interests (e.g., e-commerce), the best choice is □₅₀

- MAC
- Digital Signatures ✓
- Either MAC or Digital Signatures, interchangeably

✗ Incorrect

5. In a digital signature scheme □₅₁

- the hash function must be collision resistant ✓
- the hash function may be collision resistant
- the hash function must be one-way

Certificati:

2. A certificate

- indissolubly links an identifier to a public key ✓
- asserts a subject's trustworthiness
- specifies key's uses

✓ Correct 1/1 Points

3. Which one of the following is a good certificate?

- A, pubK_A, L_A, S_Ca(pubK_A | L_A)
- A, pubK_A, L_A, S_Ca(A | L_A)
- A, pubK_A, L_A, S_Ca(A | pubK_A | L_A) ✓

✓ Correct 1/1 Points

4. If a (private) key is compromised, then

- the corresponding certificate expires
- the corresponding certificate becomes invalid
- the corresponding certificate has to be revoked. ✓

✓ Correct 1/1 Points

5. The tasks of a Certification Authority are

- to identify the subject and attest authenticity of the public key ✓
- to attest both the morality of the subject and the authenticity of the public key
- to attest authenticity of the public key

✓ Correct 1/1 Points

6. A CRL contains

- all revoked certificates
- all expired certificates
- all revoked certificates that have not expired yet ✓