LS Ingegneria Informatica per la Gestione d'Azienda

Network Security

Gianluca Dini

Dipartimento di Ingegneria dell'Informazione: Elettronica, Informatica, Telecomunicazioni - University of Pisa

gianluca.dini@ing.unipi.it

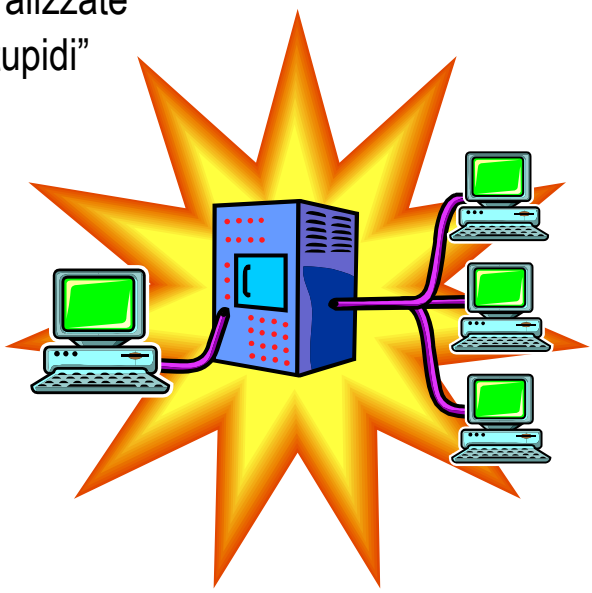
Introduzione

Cosa vedremo

- Problemi
- Principali primitive crittografiche
  - Algoritmi simmetrici o a chiave segreta (DES)
  - Algoritmi Asimmetrici o a chiave pubblica (RSA)
  - Funzioni hash (MD5)
- Principali servizi
  - Segretezza, autenticazione, integrità, non-ripudio
- Esempi: IPSEC

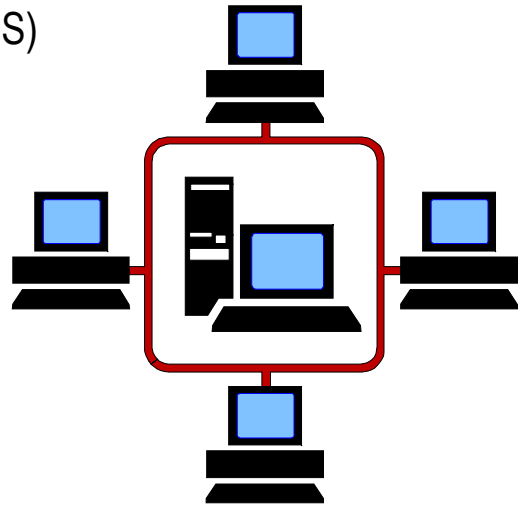
Sistema Centralizzato

- CARATTERISTICHE
  - Informazioni ed elaborazione centralizzate
  - Accesso per mezzo di terminali “stupidi”
  - Unico dominio amministrativo
  - Protezione fisica
  - Utenti noti e fidati
  - Sviluppo SW, simulazioni
- OBIETTIVO
  - Garantire l’integrità del sistema
- MECCANISMI
  - Sistema Operativo
  - Meccanismi HW



Local Area Network

- CARATTERISTICHE
  - Condivisione di risorse (file, stampanti, CPU)
  - Workstation (Windows, Unix, MS-DOS)
  - Unico dominio amministrativo
  - La rete è sicura
  - Sviluppo SW, simulazioni
- OBIETTIVO
  - Garantire l'integrità del sistema
- MECCANISMI
  - Autenticazione basata sull'indirizzo (address-based authentication)



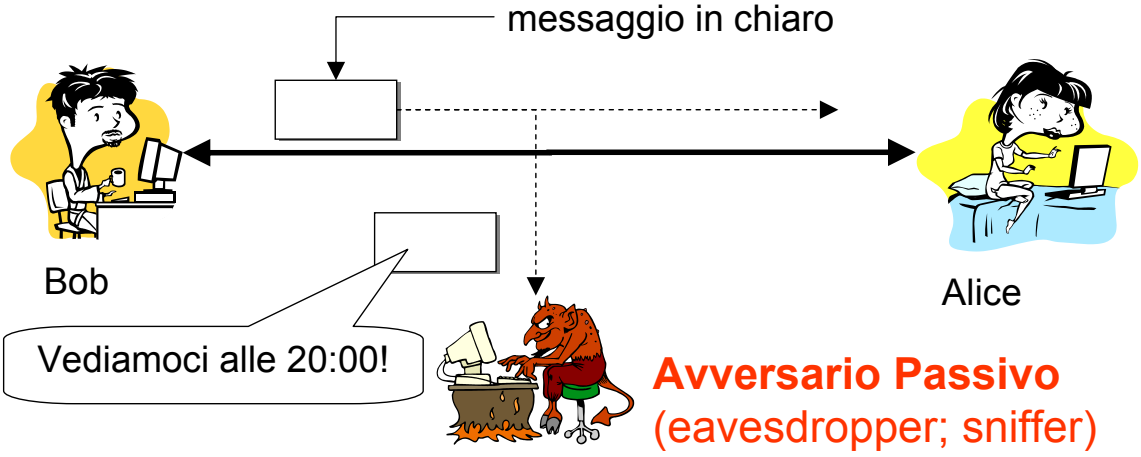
Wide Area Network

- CARATTERISTICHE
  - Domini applicativi differenti
  - Utenti sconosciuti o inaffidabili
  - La rete è insicura
  - Applicazioni: e-Commerce, telemedicina,…
- OBIETTIVO
  - Confidenzialità, Autenticazione, Integrità, Non-ripudio
- MECCANISMI
  - Cifratura, Firma digitale, Funzioni hash

Attacchi alla sicurezza

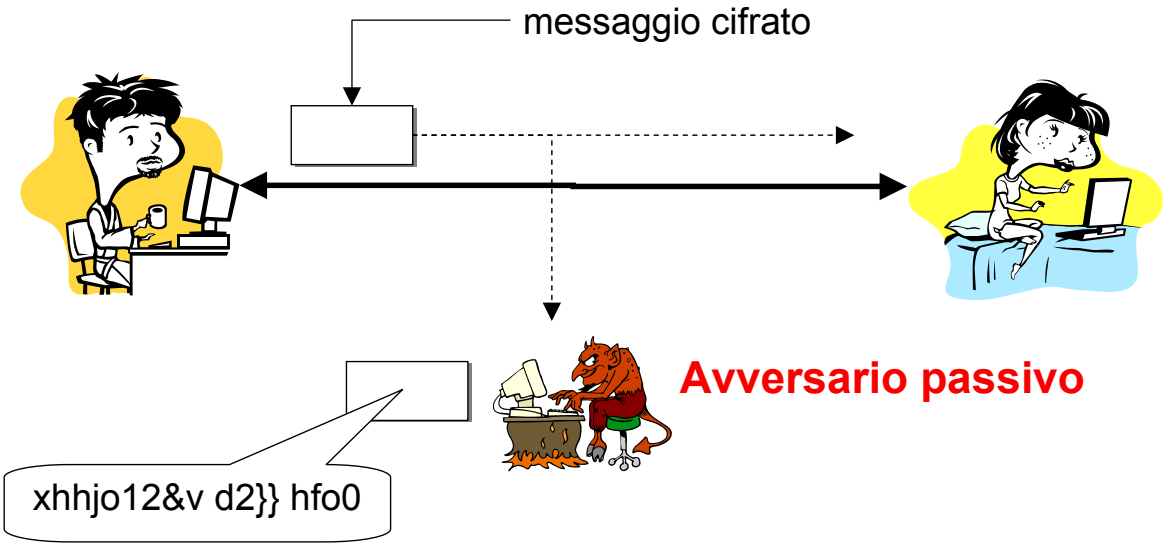
- Rilascio del contenuto dei messaggi
- Analisi del traffico
- Impersonificazione (masquerade)
- Modifica dei messaggi
- Cancellazione dei messaggi
- Replicazione dei messaggi (replay)

Rilascio del contenuto dei messaggi



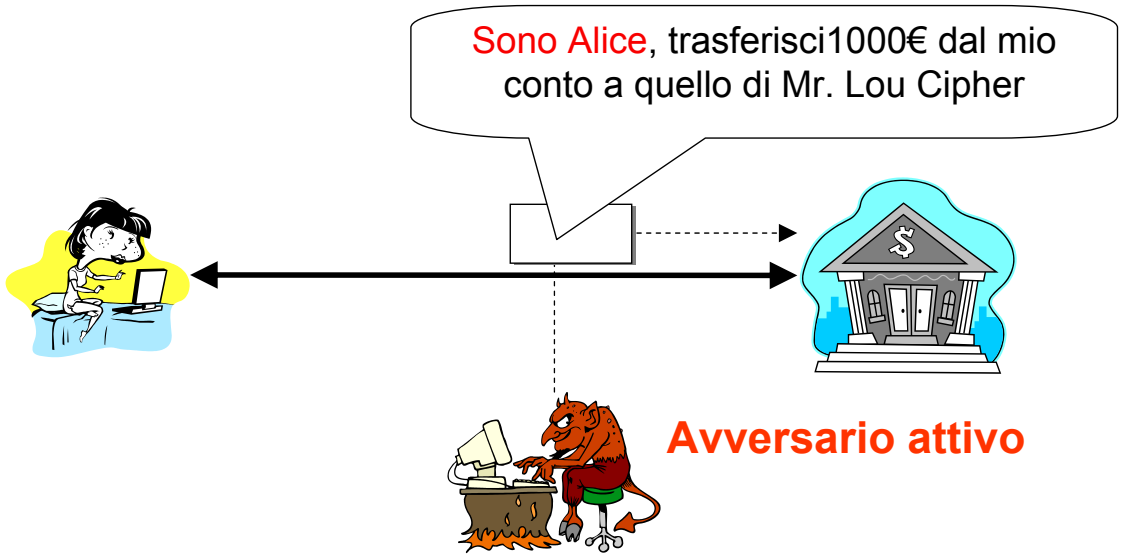
L'avversario ricava il contenuto delle trasmissioni

Analisi del traffico



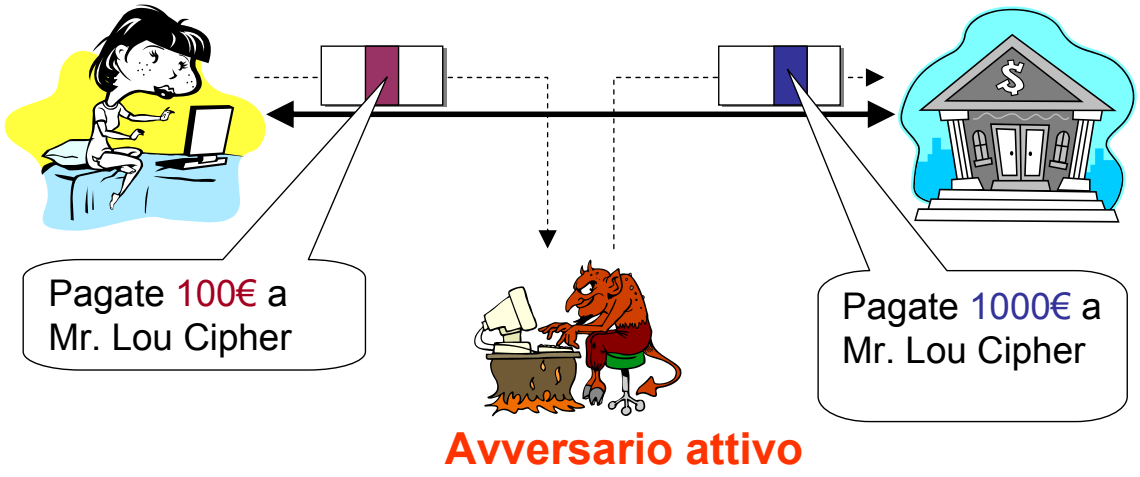
L'avversario determina l'identità degli host comunicanti, la frequenza e la dimensione dei messaggi

Impersonificazione (masquerade)



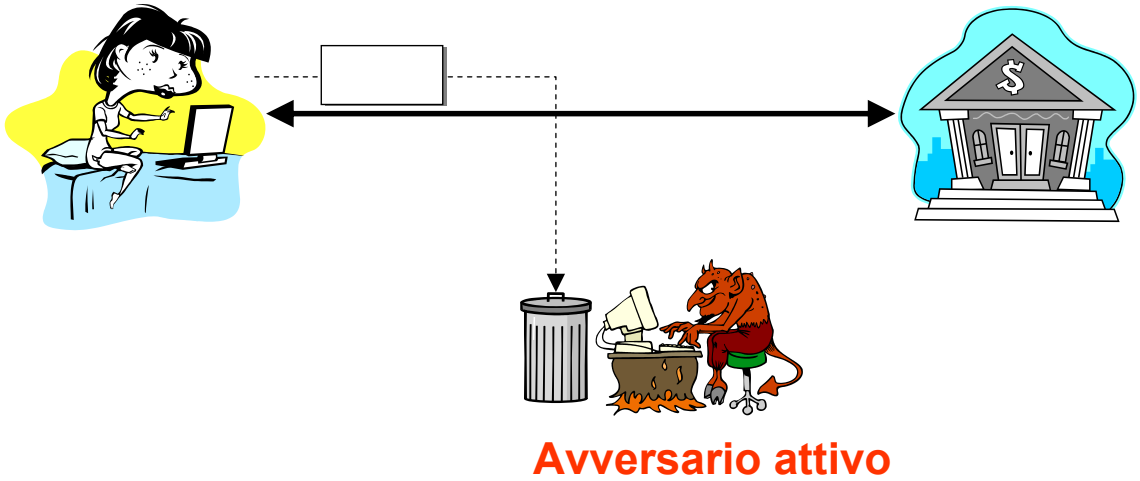
L'avversario finge di essere un entità (host, user) diversa.

Modifica dei messaggi

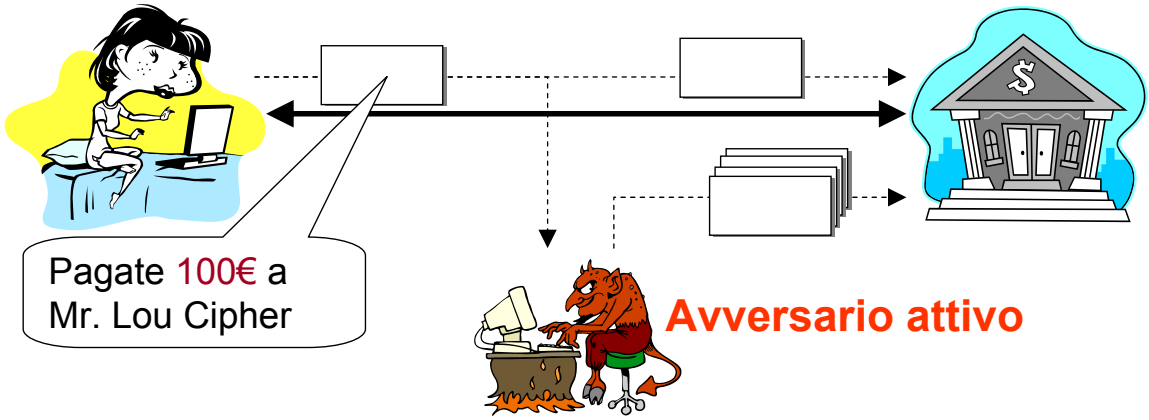


L'avversario modifica porzioni di un messaggio

Cancellazione dei messaggi



Replicazione dei messaggi (replay)



L'avversario intercetta un messaggio e lo ritrasmette una o più volte

ATTACCHI ALLA SICUREZZA

Tipo	Attacco	Contromisure
PASSIVO	Rilascio messaggi Analisi del flusso	Difficili da rilevare; facili da controbattere
ATTIVO	Masquerade Modifica messaggi Replay Negazione del servizio	Facili da rilevare; difficili da controbattere

Servizi di sicurezza

- Confidenzialità (privacy, secrecy)
- Integrità dei dati
- Autenticazione
  - Identificazione
  - Provenienza dei messaggi
- Non-ripudio

Crittografia

- La *crittografia* è la disciplina che studia le tecniche matematiche relative alla sicurezza (confidenzialità, integrità dei dati, autenticazione e non-ripudio)
- Le *primitive crittografiche* sono:
  - *cifrari*
  - *funzioni hash*
  - *firma digitale*

Regole fondamentali

- Inventare un algoritmo crittografico è compito dei matematici, ma utilizzarlo correttamente è compito di un ingegnere informatico
- Crittografia non è sinonimo di sicurezza

"*Whenever anyone says that a problem is easily solved by cryptography, it shows that he doesn't understand it*"

(Roger Needham/Butler Lampson)

Symmetric Encryption

Symmetric Encryption Scheme

$\mathcal{M}$: message space

$\mathcal{C}$: ciphertext space

$\mathcal{K}$: keyspace

$E : \mathcal{P} \times \mathcal{K} \rightarrow \mathcal{C}$ encryption transformation

$D : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{P}$ decryption transformation

Two properties

⇒ $\forall m \in \mathcal{M}, \forall e \in \mathcal{K}, \exists d \in \mathcal{K}: m = D_d(E_e(m))$

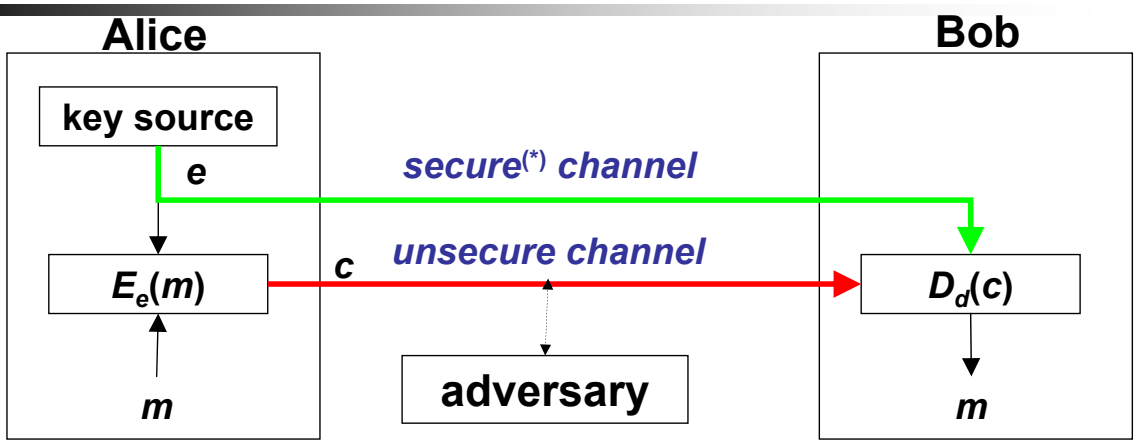
⇒ It is *computationally* "*easy*" to compute *d* knowing *e*, and viceversa

In most practical symmetric encryption scheme *e* = *d*

Security of a symmetric cipher (intuition)

- Let $c = E_e(m)$ and $m = D_e(c)$
- **The symmetric cipher (E, D) is secure iff**
  - Given ***c*** is **difficult** to determine ***m*** without knowing ***e***, and viceversa
  - Given ***c*** and m is **difficult** to determine ***e***, unless it is used just once

2-party comm with symmetric encryption



- Alice and Bob know ***E*** and ***D***
- Alice and Bob trust each other
- key ***e*** is a shared secret between Alice and Bob

(*) the channel is not ***physically*** accessible to the adversary and ensures both confidentiality and integrity

Discussion

- How can Bob be sure that $m = D_e(c)$ is good?
  - Bob knows ***m*** in advance
  - Bob knows a part of ***m*** in advance (e.g., email)
  - Bob knows that ***m*** has certain structural redundancies (e.g., ASCII)

Discussion

EXAMPLE (DES-CBC)

- Bob receives

  c = `f3 9e 8a 73 fc 76 2d 0f`
  `59 43 bd 85 c3 c9 89 d2`
  `bf 96 b6 4f 34 b8 51 dd`

- Bob deciphers ***c*** with

  k = `0x3dd04b6d14a437a9`

- Bob obtains
  - ***m*** = “`Ci vediamo alle 20!`”

Discussion

What is the effect of a “small” change in the ciphertext?

- Single bit change
  - $c[0]_7 = \sim c[0]_7$ (`73 9e 8a 73 fc ...`)
  - ***m′***=“`e8¢biö=}o alle 20:00!`”
- Single byte change
  - c[c.lenght() – 1] = 0x00 (`... 34 b8 51 00`)
  - ***m′***=“`Ci vediamo alle "}2gÀlõ`”

Discussion

- Upon *seeing* ***m***, Bob *believes* that:
  - only Alice saw message ***m*** (privacy)
  - message ***m*** comes from Alice (?provenience?)
  - message ***m*** has not been modified (?integrity?)

On trust

What does "Alice and Bob trust each other" mean?

- Alice (Bob) believes that Bob (Alice) does not reveal ***m***
- Alice (Bob) believes that Bob (Alice) keeps key ***e*** secret, i.e.,
  - Alice (Bob) believes that Bob (Alice) is competent to do key management
  - Alice (Bob) believes that Bob (Alice) does not reveal the key

Esiste un cifrario perfetto?

- Con un **cifrario perfetto**, il crittoanalista, esaminando un crittogramma *c* non acquisisce sul messaggio *m* alcuna conoscenza di cui non disponesse già prima
  - Shannon (1949) formalizzò questa proprietà in termini di processi stocastici come segue.
    - Sia *M* una variabile aleatoria che assume valori nello spazio $\mathcal{M}$ dei messaggi
    - Sia *C* una cariabile aleatoria che assume valori nello spazio $\mathcal{C}$ dei crittogrammi
    - **DEFINIZIONE**. Un cifrario è perfetto se per ogni $m \in \mathcal{M}$ e per ogni $c \in \mathcal{C}$, vale la relazione $\mathcal{P}$**(M = m | C = c) =** $\mathcal{P}$**(M = m)**
- **TEOREMA.** In un cifrario perfetto, il numero delle chiavi deve essere maggiore o uguale al numero dei messaggi possibili

One-time Pad (Vernam, 1917)

- **Assumptions**
- Let ***m*** be a ***t***-bit message

  Let ***e*** be a sequence of ***t*** randomly chosen bits
- **Encryption and decryption functions**

  Encryption: $c_i = m_i \oplus e_i$, $0 \leq i \leq t$

  Decryption: $m_i = c_i \oplus e_i$, $0 \leq i \leq t$
- **Esempio**
  - *m* = 01010101, *e* = 01001110, *c* = 00011011 (si noti che *m* è periodico ma *c* no)

One-Time Pad è un cifrario perfetto

- **TEOREMA**. One-Time Pad è un cifrario perfetto se la chiave è scelta in modo perfettamente random per ogni messaggio e
  1. tutti i messaggi hanno la stessa lunghezza ***t***
  2. tutte le sequenze di ***t*** bit sono messaggi possibili

One-Time Pad

- One-Time Pad is vulnerable to a **known-plaintext attack**
  - key *k* can be easily obtained from *m* and *c*: $e_i = m_i \oplus c_i$
- **The key must be used only once.**

  Let us suppose that a key **e is used twice**, that is $c = m \oplus e$ and $c' = m' \oplus e$.

  It follows that $c \oplus c' = m \oplus m'$.

  This provides important information to a cryptanalyst who has both *c* and *c'*. For instance, a sequence of zeros in $c \oplus c'$ corresponds to equal sequences in *m* and *m'*
- One-Time Pad requires to generate a key of **many random bits:** this problem is not trivial!

Unconditional security

- **Unconditional security** (**perfect secrecy**)
  - An adversary is assumed to have **unlimited computational resources**
  - The uncertainty in the plaintext after observing the ciphertext must be equal to the a priori uncertainty about the plaintext
  - Observation of the ciphertext provides no information whatsoever to an adversary
  - A **necessary condition** for a symmetric-key encryption scheme to be unconditionally secure is that the key bits are chosen randomly and independently and the key is at least as long as the message

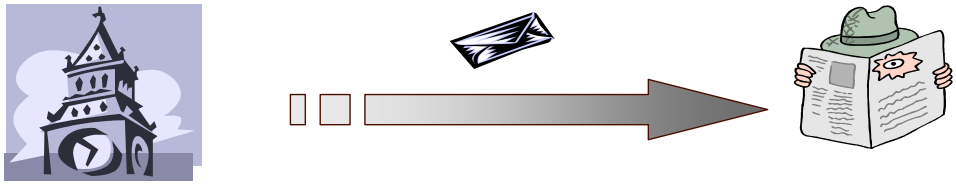
Security of one-time pad

- One-time padding is **unconditionally secure** against ciphertext-only attack
  - Any t-bit plaintext message m^* can be recovered from a t-bit ciphertext c by using a proper key $e^* = m^* \oplus c$
- The fact that the key should be at least as long as the plaintext complicates key distribution and key management
  - For this reason, in practice, stream ciphers are used where the key stream is *pseudo randomly* generated from a smaller secret key
  - These ciphers are not unconditionally secure but, hopefully, practically secure

One-time pad

- c[i] = m[i] + e[i] mod 26
- m = “SUPPORT JAMES BOND”

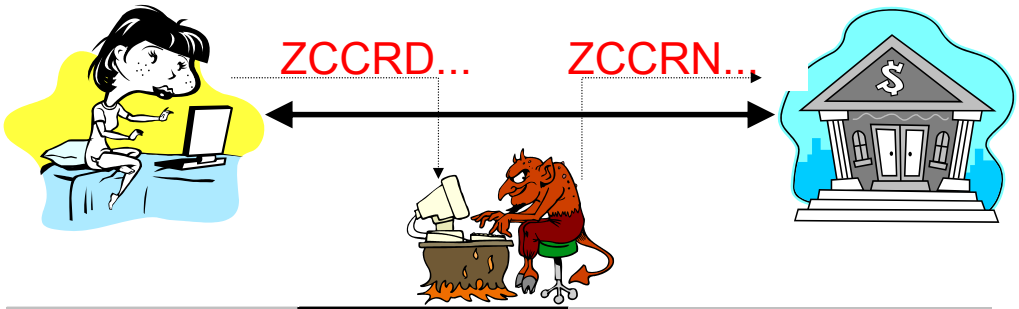
m =	S	U	P	P	O	R	T	J	A	M	E	S	B	O	N	D
k =	W	C	L	N	B	T	D	E	F	J	A	Z	G	U	I	R
c =	O	W	A	C	P	K	W	N	F	V	E	R	H	I	V	U



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

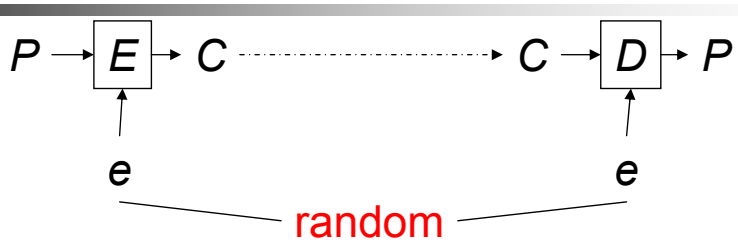
An insecure systems made by secure components

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

Block cipher



$|P| = |C| = n$ bits (block lenght)

$|e| = k$ bits (key lenght)

$e \in \mathcal{K} \subseteq V_k$

$P \in \mathcal{P} \subseteq V_n$

$C \in \mathcal{C} \subseteq V_n$

V_i set of i-bits vectors

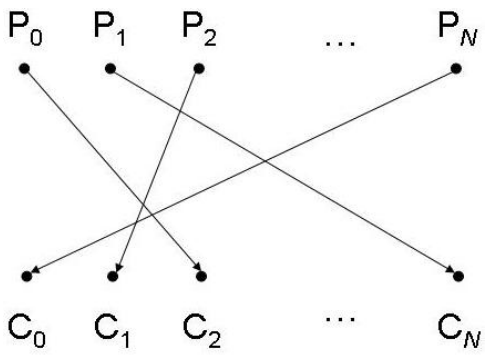
For any key **e**,

- $E_e(P)$ must be an *invertible* mapping from V_n to V_n and
- $D_e(P)$ is the *inverse function*

True random cipher

For any key **e**, E_e defines a particular substitution (permutation)

$N = 2^n$



- All the possible substitutions are $2^n!$
- Therefore, in a true random cipher, the key lenght $k = \lg(2^n!) \approx (n - 1.44)\, 2^n$ i.e., the **key lenght is 2^n times the block lenght**
- This key lenght is impractical

***In practice*, the encryption function corresponding to a randomly chosen key *should appear* a randomly chosen invertible function**

Standard assumptions

- The objective of the adversary is to recover the plaintext from the ciphertext (partial break) or even the key (total break)
- **Standard assumptions**. An adversary
  1. has access to all data transmitted over the ciphertext channel;
  2. knows all details of the encryption function except the secret key (Kerckhoff's assumption)

Classification of attacks

- Attacks are classified according to what information an adversary has access to
  - **ciphertext-only attack**
  - **known-plaintext attack**
  - **chosen-plaintext attack**

stronger

- A cipher secure against chosen-plaintext attacks is also secure against ciphertext-only and known-plaintext attack
- It is customary to use ciphers resistant to a chosen-plaintext attack even when mounting that attack is not practically feasible

Computational (practical) 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***
- **The adversary is assumed to have a limited computation power**

Attack complexity

- **Attack complexity** is the dominant of:
  - **data complexity** — expected number of input data units required
    - Ex.: exhaustive data analysis is $O(2^n)$
  - **storage complexity** — expected number of storage units required
  - **processing complexity** — expected number of operations required to processing input data and/or fill storage with data
    - Ex.: exhaustive key search is $O(2^k)$

Attack complexity

- A block cipher is **computationally secure** if
  - ***n*** is sufficiently large to preclude exhaustive data analysis
  - ***k*** is sufficiently large to preclude exhaustive key search
  - ***no known attack*** has data complexity and processing complexity significantly less than, respectively, 2^n and 2^k

Exhaustive key search

- Number of processors necessary to break a key
- Every processor performs 10^6 encryption/second

Key size (bit)	1 Year	1 Month	1 Week	1 Day
56	2,300	28,000	120,000	830,000
64	590,000	7,100,000	3.1×10^7	2.1×10^8
128	$1,1\times10^{25}$	$1,3\times10^{26}$	$5,6\times10^{26}$	$3,9\times10^{27}$

Exhaustive key search

- Cost of a year-2005 hardware cracker

1 Year	1 Month	1 Week	1 Day
56 bit			
$2000	$24,000	$100,000	$730,000
64 bit			
$510,000	$6.2M	$27M	$190M
128 bit			
9.4×10^{24}	1.2×10^{26}	4.9×10^{26}	3.3×10^{27}

Exhaustive key search

- Exhaustive key search is a known-plaintext attack
- Exhaustive key search may be a ciphertext-only attack if the plaintext has known redundancy
- Exhaustive key search has widespread applicability since cipher operations (including decryption) are generally designed to be computationally efficient
- Given $\lceil (k+4)/n \rceil$ pairs of plaintext-ciphertext, a key can be recovered by exhaustive key search in an expected time $O(2^{k-1})$
  - Exhaustive DES key search requires 2^{55} decryptions and one plaintext-ciphertext pair

Exhaustive data analysis

- A dictionary attack is a known-plaintext attack
- A dictionary attack requires to assemble plaintext-ciphertext pairs for a fixed key
- A complete dictionary requires at most 2^n pairs

Cryptoanalysis: an historical example

Monoalphabetic 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

- **The key is a permutation of the alphabet**
- **Encryption algorithm**: every cleartext character having position *p* in the alphabet is *substituted* by the character having the same position *p* in the key
- **Decryption algorithm**: every ciphertext character having position *p* in the key is *substituted* by the character having the same position *p* in the cleartext
- **Number of keys** = $26! - 1 \approx 4 \times 10^{26}$ ($\geq$ number of seconds since universe birth)

Cryptoanalysis: an historical example

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

("Romeo and Juliet", Shakespeare)

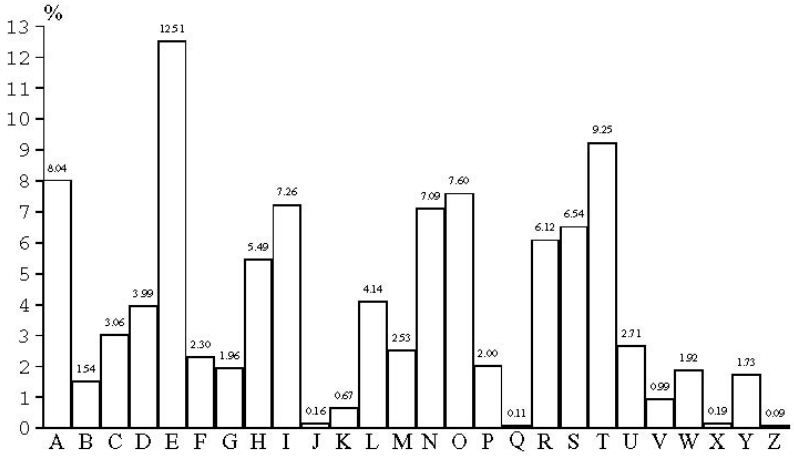
P′ = "TWOHO USEHO LDSBO THALI KEIND IGNIT
YINFA IRVER ONAWH EREWE LAYOU RSCEN E"

C = "HNZEZ KGSEZ WIGUZ HEJWR VSRYI RAYRH
PRYCJ RFMSF ZYJNE SFSNS WJPZK FGLSY S"

Cryptoanalysis: an historical example

- The monoalphabetic-substitution cipher maintains the redundancy that is present in the cleartext
- It can be "easily" cryptoanalized with a ciphertext-only attack based on language statistics

Frequency of single characters in English text



Security 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

- Linear cryptanalysis is a known-plaintext attack
- Differential cryptanalysis is primarily a chosen-plaintext attack

Linear/differential cryptoanalysis

- **Linear cryptonalysis**
  - è una tecnica di crittoanalisi per cifrari a blocchi ed a caratteri
  - Attribuita a Mitsuru Matsui (1992)
- **Differential cryptoanalysis**
  - è una tecnica di crittoanalisi principalmente concepita per cifrari a blocchi ma che può essere applicata anche ai cifrari a caratteri
  - Attribuita a to Eli Biham and Adi Shamir verso la fine degli anni `80

Cryptoanalysis of DES

- **Linear cryptonalysis**
  - A known-plaintext attack has $O(2^{43})$ data complexity and $O(2^{43})$ computation complexity.
    - With a chosen-plaintext attack, data complexity can be reduced by a factor of 4.
- **Differential cryptoanalysis**
  - Known-plaintext attack has $O(2^{55})$ data complexity and $O(2^{55})$ computation complexity
  - Chosen-plaintext attack has $O(2^{47})$ data complexity and $O(2^{47})$ computation complexity
  - DES is "surprisingly" resilient to DC.
- LC is the "best" analytical attack but is considered unpractical

Encryption modes and Multiple encryption

Electronic CodeBook

Cipher Block Chaining

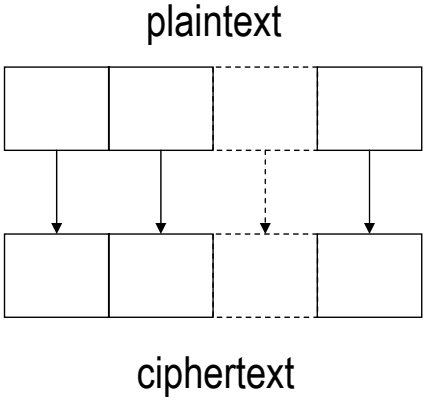
3DES (EDE, EEE)

Encryption modes

- A block cipher encrypts plaintext in fixed-size *n*-bit blocks
- When the plaintext exceeds *n* bit, there exist several methods to use a block
  - **Electronic codebook (ECB)**
  - **Cipher-block Chaining (CBC)**
  - Cipher-feedback (CFB)
  - Output feedback (OFB)

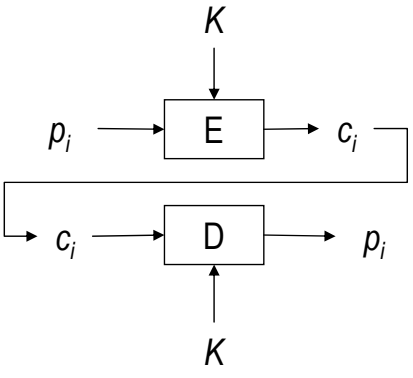
Encryption modes: ECB

- **Electronic Codebook (ECB)**



plaintext blocks are encrypted separately

$$\forall 1 \le i \le t, c_i \leftarrow E_k(p_i)$$
$$\forall 1 \le i \le t, p_i \leftarrow D_k(c_i)$$



Encryption modes: ECB

Properties

- **Identical plaintext results in identical ciphertext**
  - ECB doesn't hide data patterns
- **No chaining dependencies: blocks are enciphered independently of other blocks**
  - ECB allows block reordering and substitution
- **Error propagation: one or more bit errors in a single ciphertext block affects decipherment of that block only**

Encryption modes: ECB

AN EXAMPLE OF BLOCK REPLAY

- **A bank transaction 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
- **Format of a credit message**
  - Src bank: *M* (12 byte)
  - Rcv banck: *R* (12 byte)
  - Client: *C* (48 byte)
  - Bank account: *N* (16 byte)
  - Amount of money: *D* (8 byte)
- **Cipher (n = 64 bit; modalità ECB)**

Encryption modes: ECB
block replay

- Mr. Lou Cipher is a client of the banks and wants to make a fraud.
- Lou Cipher is an active adversary and wants to replay a Bank B1's message "credit 100$ to Lou Cipher" many times
- Attack strategy
  - The adversary 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

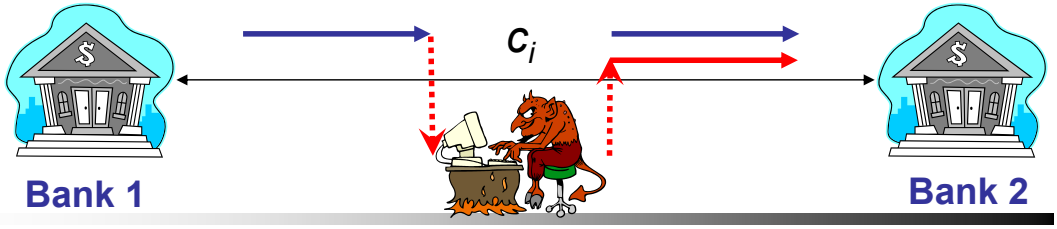
Encryption modes: ECB

AN EXAMPLE OF BLOCK REPLAY

1. The adversary performs **k** equal transfers
   - credit 100$ to Lou Cipher $\Rightarrow c_1$
   - credit 100$ to Lou Cipher $\Rightarrow c_2$
   - ...
   - credit 100$ to Lou Cipher $\Rightarrow c_k$

   $c_1 = c_2 = \ldots = c_k$

   COMMENT. ***k*** is large enough to allow the adversary to identify the cryptograms corresponding to its transfers
2. The adversary searches "his own" cryptograms over the network
3. The adversary replies one of these cryptograms



Encryption modes: ECB

AN EXAMPLE OF BLOCK REPLAY

- An 8-byte timestamp field ***T*** is added to the message to prevent replay attacks

block no.	1	2	3	4	5	6	7	8	9	10	11	12	13
	T	M	R	C							N	D	

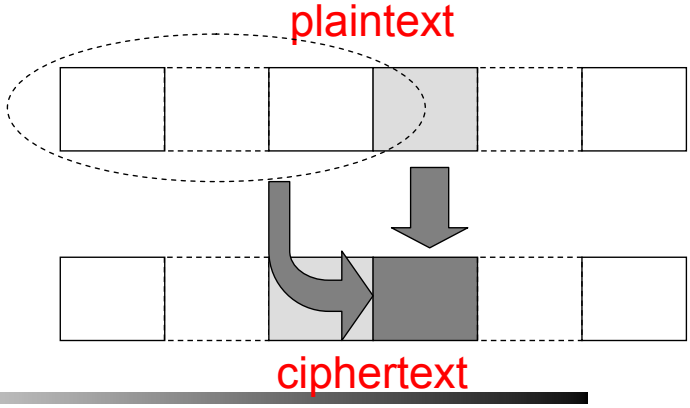
However, the adversary can

1. identify “his own” cryptograms as before by inspecting blocks 2–13;
2. intercept any “fresh” cryptogram;
3. substitute block 1 of “his own” cryptogram with block 1 of the “fresh” cryptogram

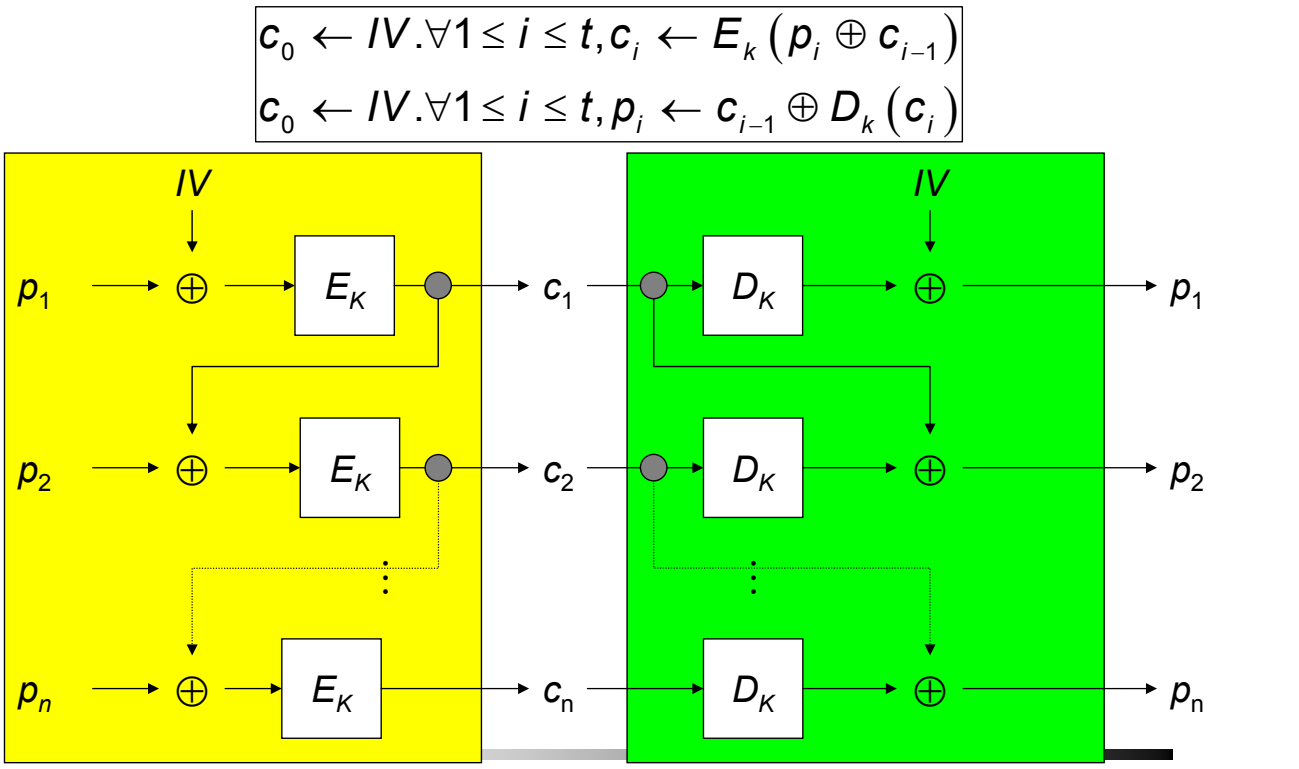
Encryption modes: Cipher Block Chaining

- CBC segue il **principio di diffusione** di Shannon introducendo una **dipendenza di posizione** tra il blocco in elaborazione e quelli precedenti
- CBC è un cifrario a blocchi in cui blocchi identici del messaggio vengono cifrati in modo **diverso** eliminando ogni periodicità

c_i depends on p_i and all preceding plaintext blocks



CBC



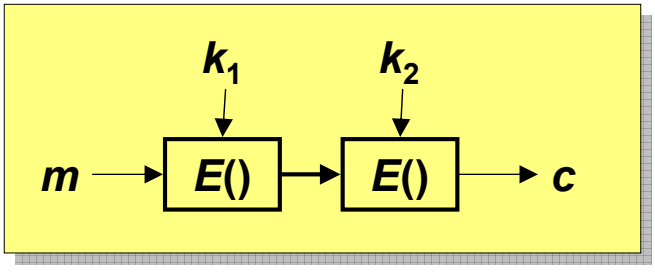
CBC: properties

- Identical ciphertext result from the same plaintext under the same key and IV
- IV can be sent in the clear; its integrity must be guaranteed
- Chaining dependencies: c_i depends on p_i and all preceding plaintext blocks
  - Ciphertext block reordering affects decryption
- Error propagation: bit errors in c_i affect decryption of c_i and c_{i+1}
- Error recovery: CBC is self-synchronizing or ciphertext autokey
- Framing errors: CBC does not tolerate “lost” bits

Multiple encryption

- If a cipher is subject to exhaustive key search, encipherment of a message more than once **may** increase security
- Multiple encryption may be extended to messages exceeding one block by using standard modes of operation
- Cascade cipher is the concatenation of L ≥ 2 ciphers, each with independent keys
- Multiple encryption is similar to a cascade cipher but the ciphers are identical (either *E* or *D*) and the keys need not be independent

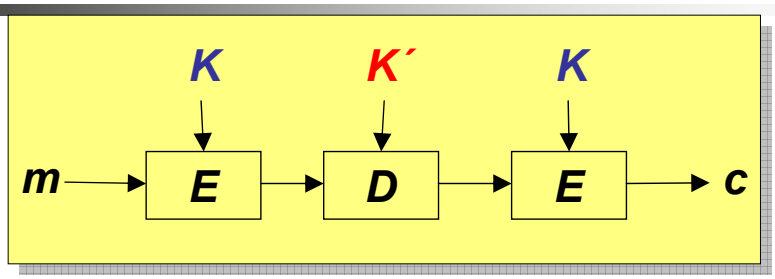
Double encryption



- Double encryption is subject to a known-plaintext attack called "meet-in-the-middle" attack which requires 2^k operations and 2^k storage units

Triple encryption

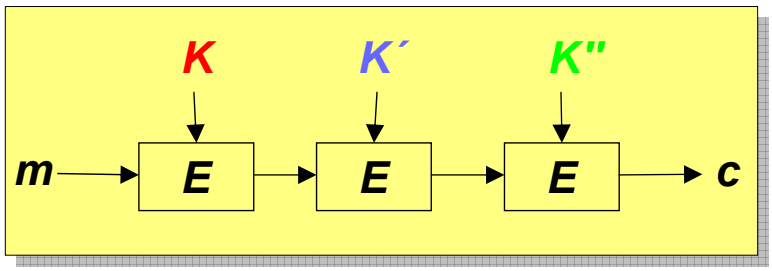
EDE



- Used in financial applications
- ANSI X9.17 e ISO 8732
- A chosen-plaintext attack which requires 2^k operations, 2^k data inputs and 2^k storage units
- A known-plaintext attack requires p data inputs, $2^{k+n}/p$ operations, and $O(p)$ storage units
- Backward compatibility: E when $K = K'$

Triple encryption

EEE



- A known-plaintext attack similar to meet-in-the-middle, which requires 2^{2k} operations and 2^k units of storage
- With DES, $k = 56$ (DES), the cipher is practically secure

Key Distribution

Distribuzione delle chiavi

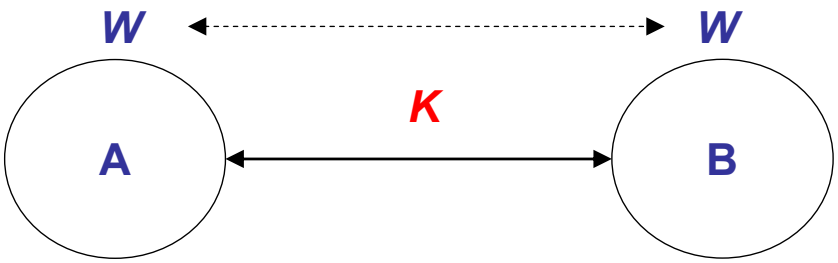
- Alice e Bob devono condividere un segreto, la chiave *K*…
- …ma come si fa a distribuire la chiave *K* ad Alice ed a Bob?
- La chiave *K* non può essere trasmessa in chiaro sulla rete perché la rete è insicura

Distribuzione delle chiavi

- Possibili soluzioni
  - Faccia a faccia
  - Corriere fidato
  - Chiave in tanti pezzetti ed invio di ciascun pezzetto attraverso un diverso canale di comunicazione (telefono, email, piccione,…)
- Non sempre queste soluzioni sono possibili e/o economiche e/o efficienti
- Si vorrebbe utilizzare la rete per distribuire le chiavi

Point-to-point key management

point-to-point



- Parties know each other
  e.g., a client *A* has an account on server *B*
- *A* and *B* *a priori* share a *key W*
- *A* and *B* wants to establish a *session key K*

Point-to-point key management

one-pass

$M1 \quad A \rightarrow B: \quad E_W(t_A, B, K)$

- **t_A is a timestamp (a "fresh" quantity) requires synchronized clocks**

with challenge-response

$M1 \quad A \leftarrow B: \quad n_B$

$M2 \quad A \rightarrow B: \quad E_W(n_B, B, K)$

- **n_B is a nonce (a "fresh" quantity)**

both parties contribute to the session key

$M1 \quad A \leftarrow B: \quad n_B$

$M2 \quad A \rightarrow B: \quad E_W(K_A, n_B, n_A, B)$

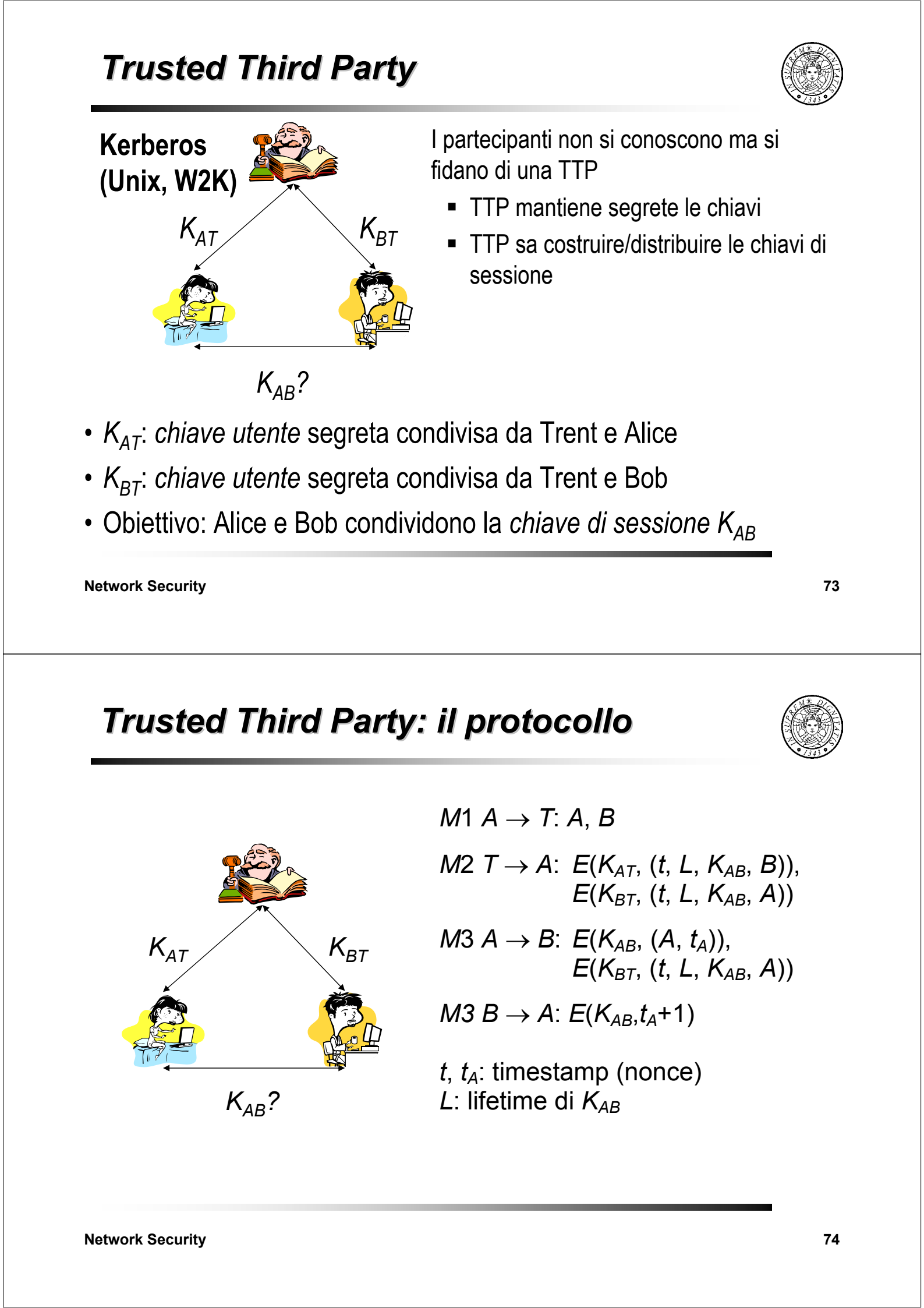
$M3 \quad A \leftarrow B \quad E_W(K_B, n_A, n_B, A)$

- **n_A and n_B are nonces**
- **K_A and K_B are keying materiale**
- **$K = f(K_A, K_B)$**

Three Way Handshake

- Il protocollo 3WH richiede comunque un ***segreto condiviso a priori*** (la chiave W)
- Come si distribuisce questa chiave?
  - Offline
  - Spesso *W* è ottenuta da una *password P*
- Questa gestione delle chiavi non scala
  - *n* utenti $\Rightarrow$ $n \times (n\text{-}1)/2$ chiavi di handshake $\Rightarrow$ ***O*(n^2) *segreti condivisi a priori***

Trusted Third Party



Kerberos (Unix, W2K)

I partecipanti non si conoscono ma si fidano di una TTP

- TTP mantiene segrete le chiavi
- TTP sa costruire/distribuire le chiavi di sessione

- K_{AT}: *chiave utente* segreta condivisa da Trent e Alice
- K_{BT}: *chiave utente* segreta condivisa da Trent e Bob
- Obiettivo: Alice e Bob condividono la *chiave di sessione* K_{AB}

Trusted Third Party: il protocollo

$M1\ A \rightarrow T$: A, B

$M2\ T \rightarrow A$: $E(K_{AT}, (t, L, K_{AB}, B))$,
$E(K_{BT}, (t, L, K_{AB}, A))$

$M3\ A \rightarrow B$: $E(K_{AB}, (A, t_A))$,
$E(K_{BT}, (t, L, K_{AB}, A))$

$M3\ B \rightarrow A$: $E(K_{AB}, t_A+1)$

t, t_A: timestamp (nonce)
L: lifetime di K_{AB}

Trusted Third Party: svantaggi (1)

- IN GENERALE:
  - TTP può diventare un “collo di bottiglia”
    - prestazioni
    - security
  - TTP deve memorizzare chiavi a lungo termine
  - TTP deve essere *incondizionatamente* sicuro
- QUESTO PROTOCOLLO IN PARTICOLARE:
  - Clock sincronizzati

Trusted Third Party: svantaggi (2)

IN GENERALE

- Il protocollo TTP richiede **segreti condivisi a priori** (K_{AT} e K_{BT})
  - Rispetto a 3WH, cambia la scala del problema
  - *n* utenti $\Rightarrow$ *n* chiavi utente condivise con TTP $\Rightarrow$ ***O(n) segreti condivisi a priori***
- Il problema tuttavia resta

Protocollo Diffie-Hellman

- OBIETTIVO

  Effettuare la distribuzione delle chiavi senza bisogno di segreti condivisi a priori

- Il protocollo DIFFIE-HELLMAN è un primo passo verso la soluzione

Il problema del logaritmo discreto

- Sia p un ***numero primo grande***

  Sia g, $1 \leq g < p$, ***un generatore*** **di** $\mathbb{Z}_p^*$**,** ovvero tale che $\forall\ 1 \leq n < p, \exists$ un intero i per cui $g^i \bmod p = n$

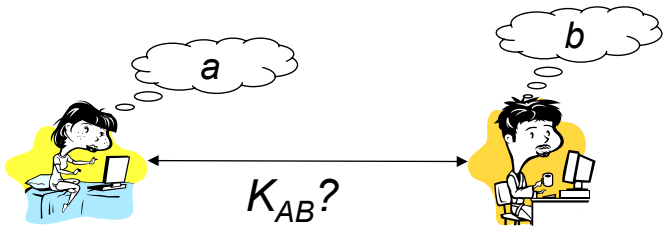
- *ESPONENZIALE DISCRETO*

  Dati g, x, è facile calcolare $y = g^x \bmod p$

- *LOGARITMO DISCRETO*

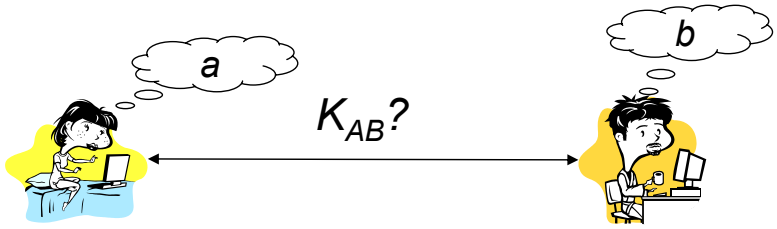
  Dati g, $1 \leq y \leq p$-1, è difficile determinare x ($0 \leq y \leq p$-2) tale che $y = g^x \bmod p$

Protocollo Diffie-Hellman: scenario



- Sia p un numero primo grande,
- Sia $1 \leq g < p$
- I numeri p e g sono noti a tutti

Protocollo Diffie-Hellman: protocollo



Alice sceglie un numero random a
Bob sceglie un numero random b

M1 A $\rightarrow$ B: A, $g^a \bmod p$
M2 B $\rightarrow$ A: B, $g^b \bmod p$

Alice calcola
$K_{AB} = (g^b)^a \bmod p = g^{ab} \bmod p$

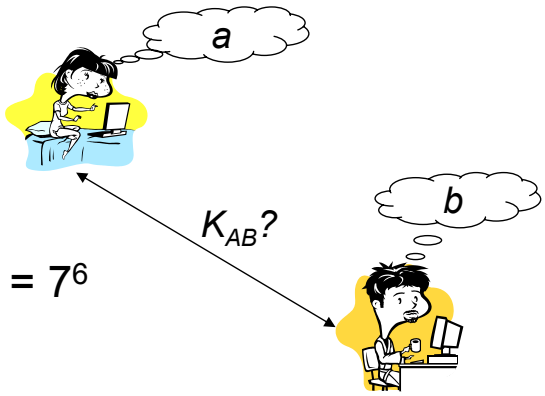
Bob calcola
$K_{AB} = (g^a)^b \bmod p = g^{ab} \bmod p$

Protocollo Diffie-Hellman: un esempio

Siano p = 11, g = 7

Alice pensa $a = 3$ e calcola $g^a \bmod p = 7^3 \bmod 11 = 343 \bmod 11 = 2$

Bob pensa $b = 6$ e calcola $g^b \bmod p = 7^6 \bmod 11 = 117649 \bmod 11 = 4$



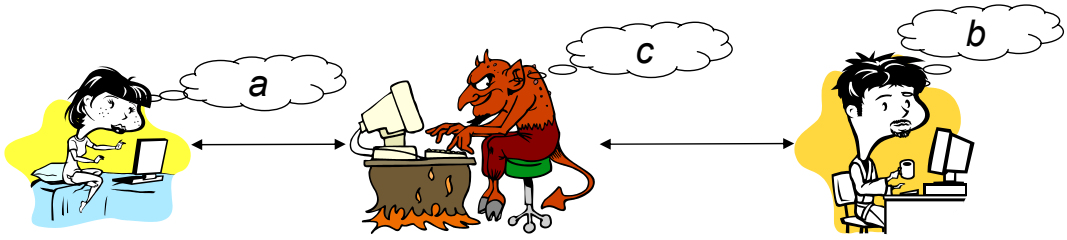
$A \rightarrow B$: 2
$B \rightarrow A$: 4

Alice riceve 4 e calcola $K_{AB} = (g^b)^a \bmod p = 4^3 \bmod 11 = 9$

Bob riceve 2 e calcola $K_{AB} = (g^a)^b \bmod p = 2^6 \bmod 11 = 9$

L'UOMO NEL MEZZO

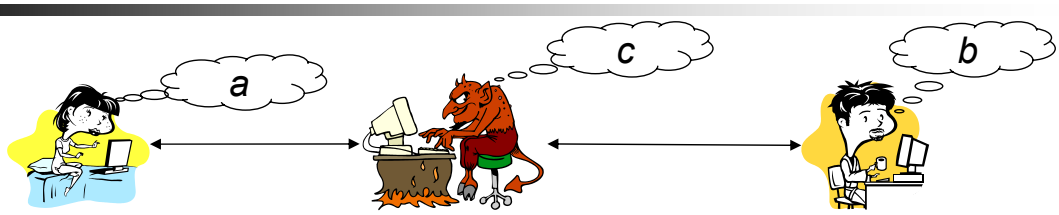
Alice non ha nessuna garanzia che sta effettivamente interagendo con Bob e viceversa



MAN-IN-THE-MIDDLE ATTACK

- L'uomo-nel-mezzo (l'avversario) può leggere e modificare tutti i messaggi tra Alice e Bob senza che questi se ne accorgano
- Alice e Bob si illudono di comunicare su di un canale sicuro

L'UOMO NEL MEZZO



$A \rightarrow B[M] : A, g^a \bmod p$

$M \rightarrow B : A, g^c \bmod p$

$B \rightarrow A[M] : B, g^b \bmod p$

$M \rightarrow A : B, g^c \bmod p$

Alice calcola $K_{AM} = \left(g^c\right)^a \bmod p = g^{ac} \bmod p$

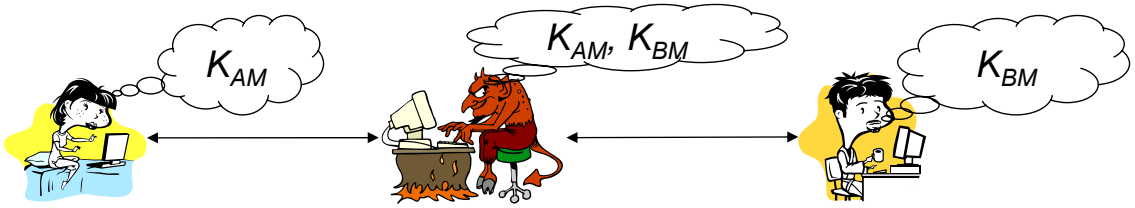
Bob calcola $K_{BM} = \left(g^c\right)^b \bmod p = g^{bc} \bmod p$

Avversario calcola

$K_{AM} = \left(g^a\right)^c \bmod p = g^{ac} \bmod p$, e

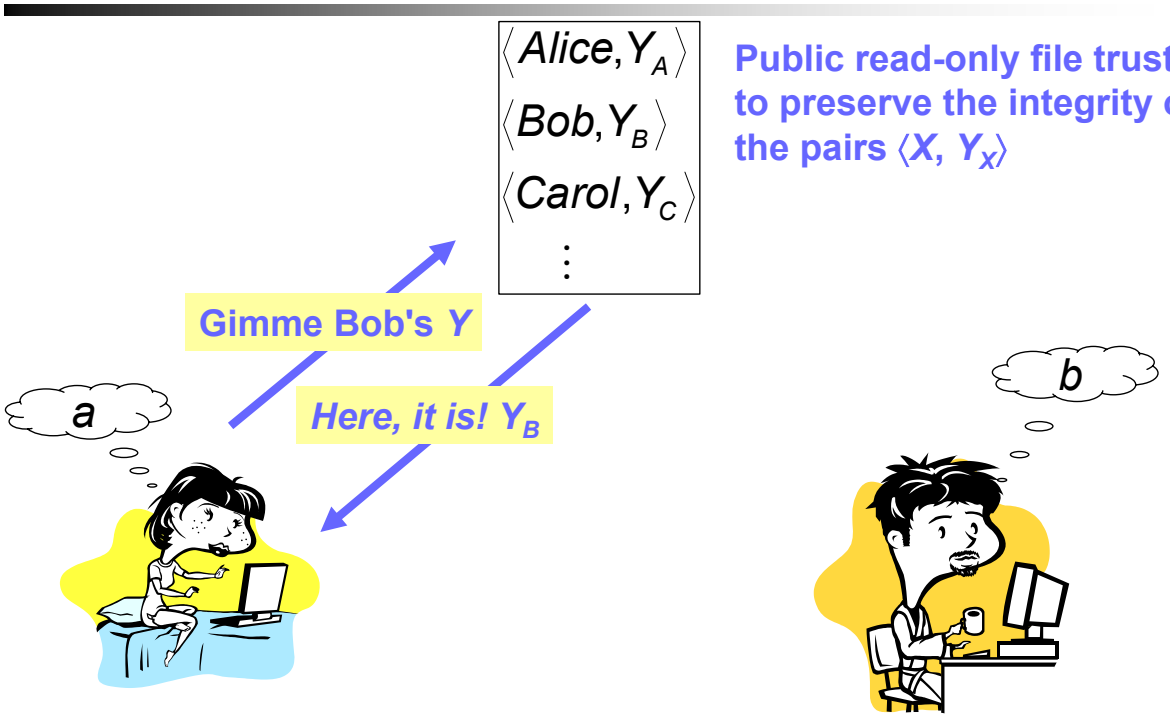
$K_{BM} = \left(g^b\right)^c \bmod p = g^{bc} \bmod p$

L'UOMO NEL MEZZO

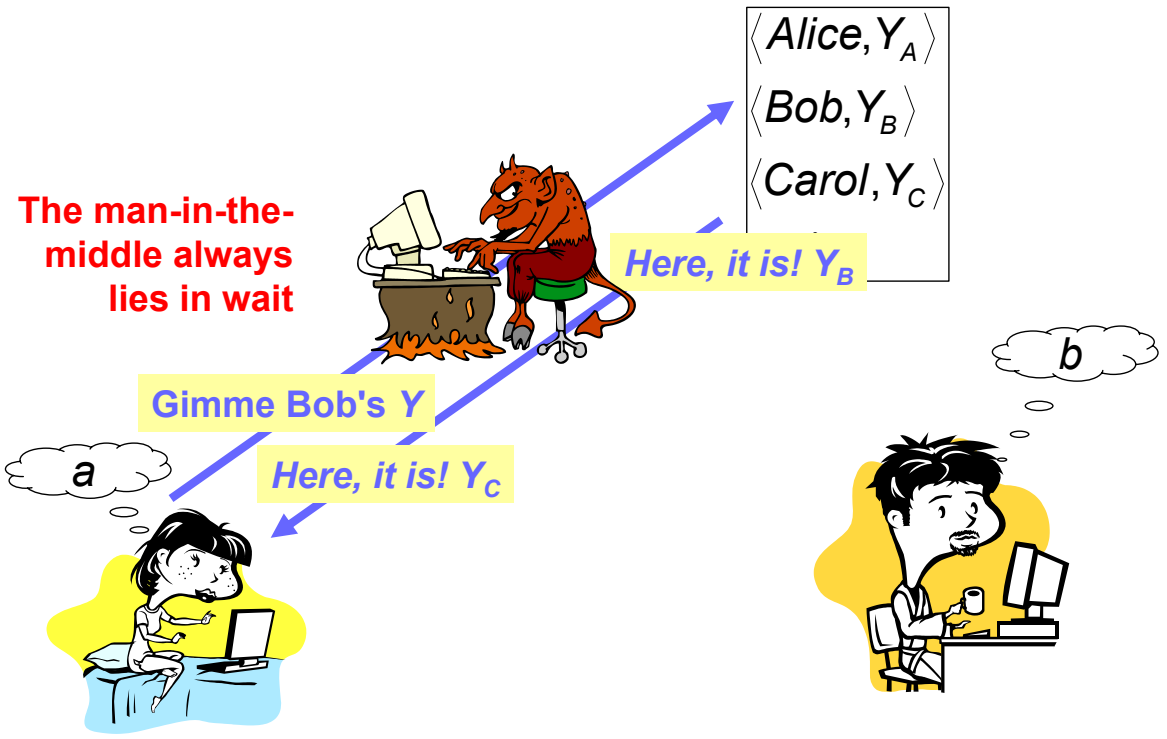


- Alice crede di comunicare con Bob per mezzo di K_{AM}
- Bob crede di comunicare con Alice per mezzo di K_{BM}
- L'avversario può
  - leggere tutte le trasmissioni tra Alice e Bob
  - impersonare ciascuno dei due

Diffie-Hellman protocol



Diffie-Hellman protocol

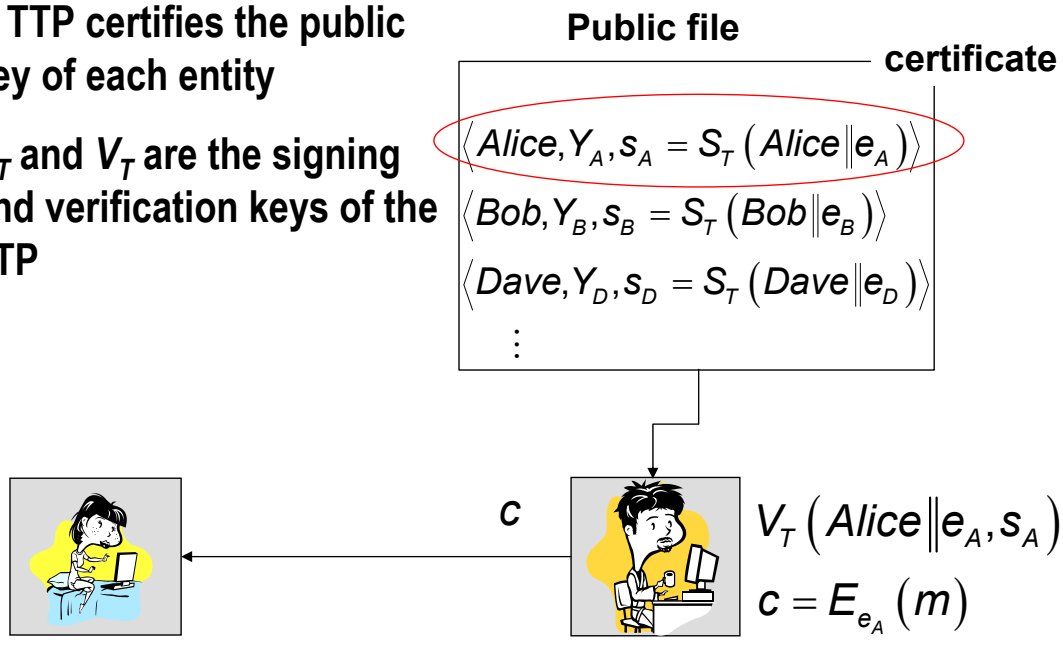


Key distribution with public encryption

- **Advantages**
  - No TTP is required
  - The public file could reside with each entity
  - Only *n* public keys need to be stored to allow secure communications between any pair of entities, assuming that the only attack is that by a passive adversary
- **Disadvantages**
  - Key management becomes more difficult in the presence of an active adversary

Key distribution with public keys

- **A TTP certifies the public key of each entity**
- **S_T and V_T are the signing and verification keys of the TTP**



Key distribution with certificates

- Advantages
  - It prevents and active adversary from impersonation on the network
  - The TTP cannot monitor communications
  - Entities need to trust the TTP only to bind identities to public keys properly
  - Certicates can be stored locally so eliminating per-communication interaction with the public file
- Disadvantages
  - if the signing key of TTP is compromised, all communications become insecure
  - All trust is placed with one entity

Asymmetric Encryption and Digital signatures

CIFRATURA ASIMMETRICA

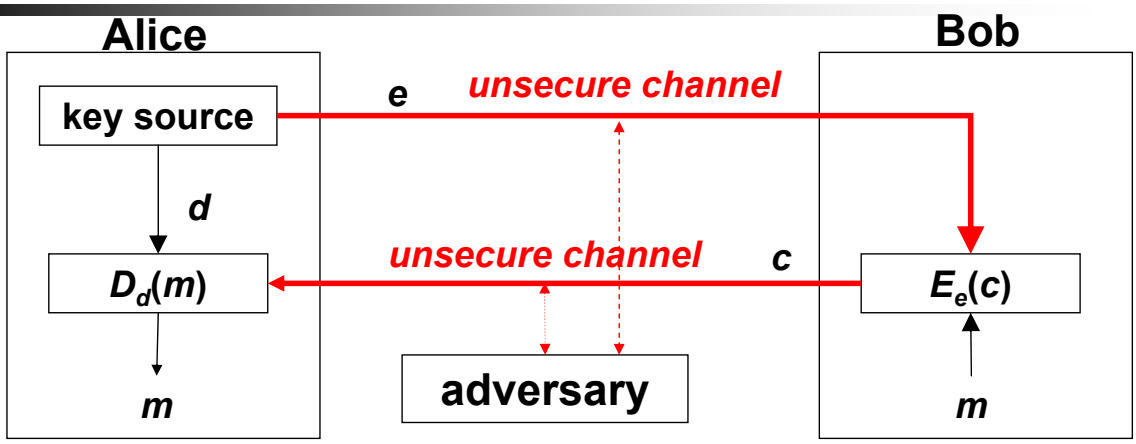
- Algoritmo di Cifratura $E()$
  - $c = E(e, m)$ — la cifratura del messaggio in chiaro m con la chiave e produce il testo cifrato c
- Algoritmo di Decifratura $D()$
  - $m = D(d, c) = D(d, E(e, m))$ — la decifratura del messaggio cifrato c con la chiave d ($d \neq e$) produce il testo in chiaro m
- Proprietà di $E()$ e $D()$
  I. Dato c è molto difficile ricavare m se non si conosce d
  II. Dati m e c è molto difficile ricavare d, a meno che d non sia utilizzata una sola volta
  III. Anche se si conosce e è molto difficile ricavare d e viceversa

Chiave Pubblica e Privata

Ogni utente ha una coppia di chiavi (e, d)

- una la mantiene segreta,
- l'altra la rende pubblica
- Ad esempio:
  - e_A: chiave pubblica di Alice
  - d_A: chiave privata di Alice

2-party comm with asymmetric encryption



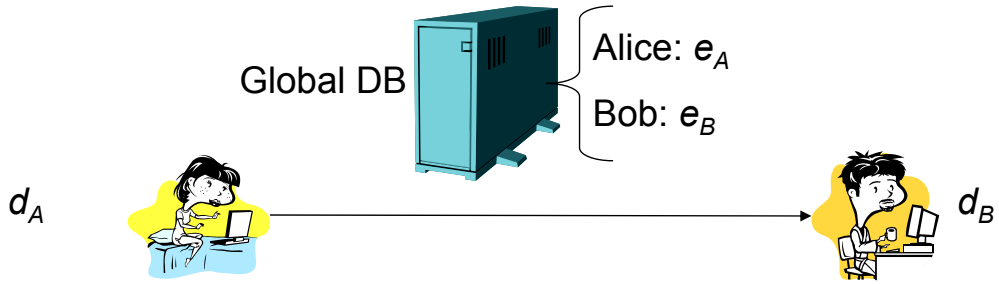
- The encryption key *e* can be sent on the ***same*** channel on which the ciphertext *c* is being transmitted
- It is necessary to ***authenticate*** public keys to achieve ***data origin authentication*** of the public keys themselves

Algoritmi più noti

- RSA (1978) – probabilmente il più diffuso; la sua sicurezza non è stata provata
- Knapsack (1978) – violato più volte, non è considerato sicuro
- Rabin (1979)
- ElGamal (1985)
- Schnorr (1991)

Segretezza

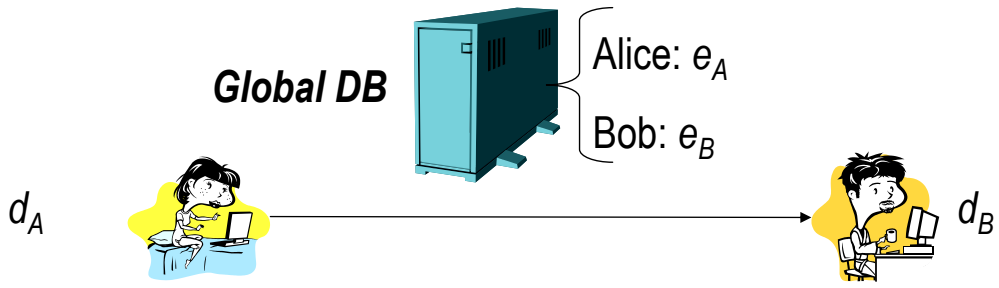
Alice vuole inviare un messaggio segreto *m* a Bob



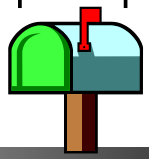
1. Alice si procura e_B, la chiave pubblica di Bob
2. Alice calcola $c = E_{eB}(m)$
3. Alice invia *C* a Bob
4. Bob calcola $m = D_{dB}(c)$

Metafora: cassetta postale

Alice vuole inviare un messaggio segreto *M* a Bob



Chiunque può inserire un messaggio nella cassetta postale, ma solo chi ha la chiave (privata) può aprire la cassetta e prelevare il messaggio



Firma Digitale

- ***Algoritmo di firma S()***
  - $s = S(d_A, m)$ costituisce la firma digitale del messaggio m con la chiave d_A
- ***Algoritmo di verifica V()***
  - $V(e_A, [m], s)$ restituisce VERO se s è la firma digitale di m con d_A, FALSO altrimenti
  - Alcuni algoritmi non richiedono m che, in tali casi, è uno dei risultati della verifica
- ***Proprietà di S() e V()***

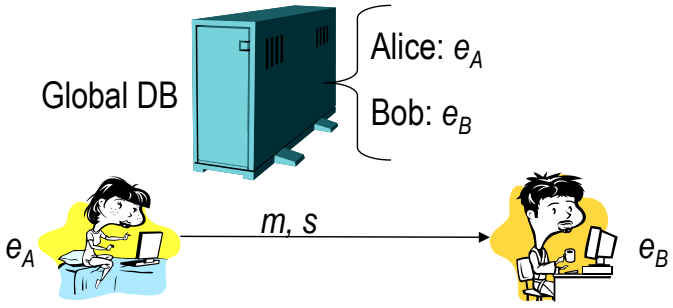
I. Dato m è molto difficile produrre s se non si conosce d_A

II. Dati m e s è molto difficile ricavare d_A, a meno che non sia utilizzata una sola volta

III. Anche se si conosce e_A è molto difficile ricavare d_A e viceversa

Provenienza di un messaggio

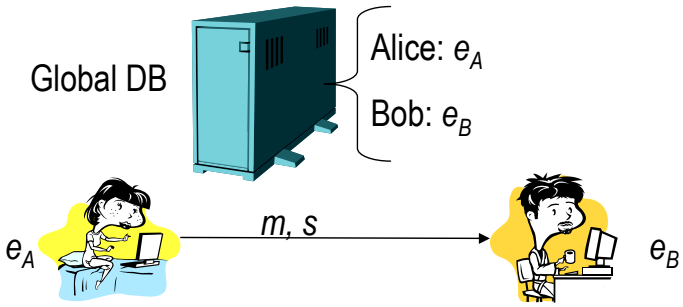
Alice non vuole inviare un messaggio segreto a Bob ma vuole fornirgli una prova di provenienza



1. Alice firma m, cioè calcola $s = S(d_A, m)$
2. Alice invia (m, s) a Bob
3. Bob si procura e_A e verifica s, cioè
4. calcola $v = V(e_A, m, s)$ e verifica che $v \equiv$ TRUE

Metafora: firma

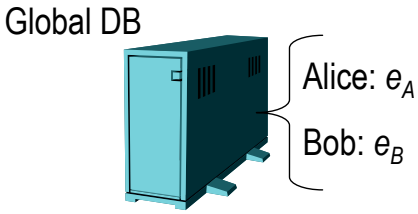
Alice non vuole inviare un messaggio segreto a Bob ma vuole fornirgli una prova di provenienza



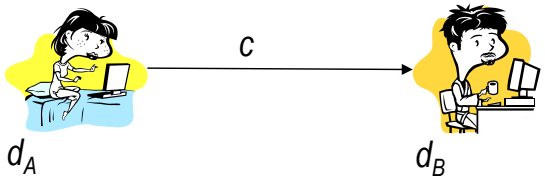
- Solo chi ha la chiave privata può firmare un documento
- Tutti gli altri possono verificare la firma con la chiave pubblica

Segretezza + Provenienza

Alice vuole inviare un messaggio segreto m a Bob, fornendogli anche una prova di provenienza

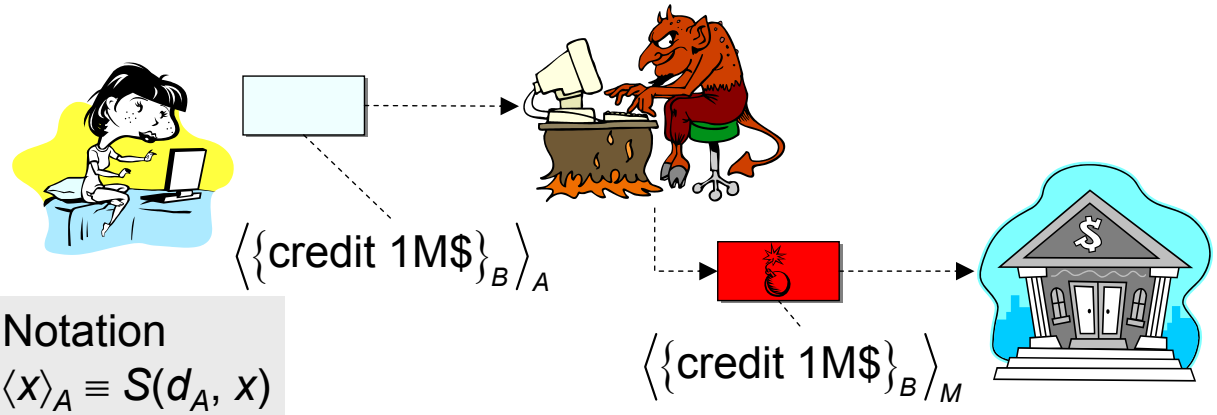


1. Alice calcola $s = S(d_A, m)$
2. Alice si procura e_B e calcola $c = E(e_B, (m, s))$
3. Alice invia c a Bob



1. Bob calcola $(m, s) = D(d_B, c)$
2. Bob si procura e_A, e
3. verifica che $D(e_A, m, s) = \text{TRUE}$

Sign and then encrypt



Notation
$\langle x \rangle_A \equiv S(d_A, x)$
$\{x\}_B \equiv E(e_B, x)$

The adversary can remove Alice's signature and replace it with his own

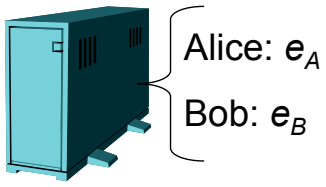
This is not possible any longer if signature and encryption operations are performed in a reverse order

Algoritmi di Firma più Diffusi

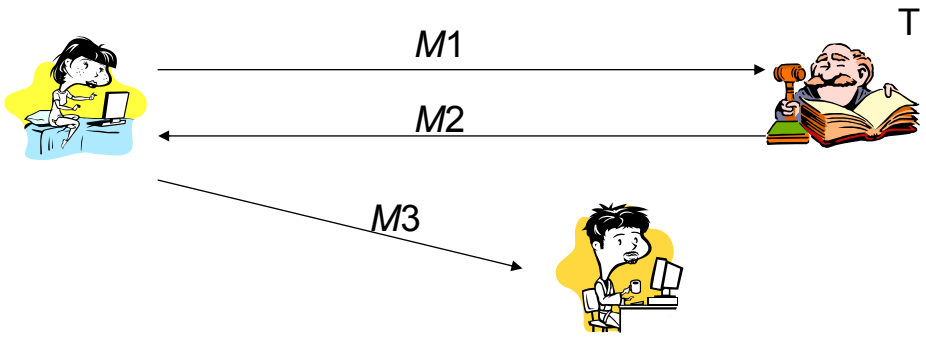
- **RSA**
  - Lo stesso algoritmo si utilizza sia per firmare sia per cifrare (E ≡ S; D ≡ V)
  - La verifica della firma restituisce il messaggio
- **ElGamal**
  - Per firmare e cifrare si usano algoritmi diversi (E ≠ S; D ≠ V)
- **DSS**
  - Algoritmo per la sola firma digitale

Centro Distribuzione Chiavi

- La cifratura asimmetrica non richiede segreti condivisi
- Quando necessario, basta solo (!!!) procurarsi la chiave pubblica del partner…
- ... ma dove si va a prenderla?...
- Presso il ***Centro distribuzione dati***, una ***terza entità fidata*** che assicura ***l'integrità*** del collegamento
  utente ↔ chiave pubblica dell'utente



Distribuzione delle Chiavi



Alice vuole conoscere la chiave pubblica e_B di Bob
M1 $A \rightarrow T$: A, B
M2 $T \rightarrow A$: T, B, e_B
Alice ritiene che e_B sia la chiave pubblica di Bob

Alice invia un messaggio segreto m a Bob
M3 $A \rightarrow B$: A, $E(e_B, m)$

…sembra tutto OK, ma…

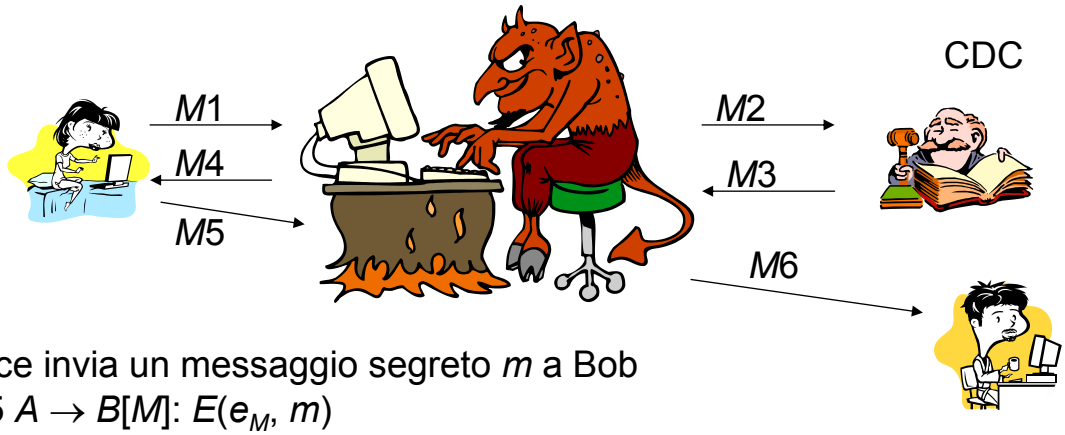
L'uomo nel mezzo è sempre in agguato

Alice vuol conoscere la chiave pubblica e_B di Bob

M1 $A \rightarrow T[M]$: A, B
M2 $M[A] \rightarrow T$: A, B
M3 $T \rightarrow A[M]$: T, B, e_B
M4 $M[T] \rightarrow A$: T, B, e_M

Alice ritiene che e_M sia la chiave pubblica di Bob

L'uomo nel mezzo è sempre in agguato



Alice invia un messaggio segreto m a Bob
M5 $A \rightarrow B[M]$: $E(e_M, m)$
M6 $M \rightarrow B$: $E(e_B, m)$
L'avversario (l'uomo-nel-mezzo) può leggere tutti i messaggi segreti che Alice invia a Bob

Problema analogo quando Alice vuole procurarsi la chiave pubblica di Bob per verificare una firma digitale

Il problema

- Il problema nascono dal messaggio M4

$$\text{M4 } M[T] \rightarrow A\text{: } T, B, e_M$$

- Alice attribuisce (erroneamente) M4 a *T* (CDC) ma
- non c'è nessuna prova che quel messaggio sia effettivamente di *T*
- È un problema di autenticità e non di segretezza:
- M4 non trasporta alcuna informazione segreta

Certificati

LA SOLUZIONE

- Il centro distribuzione chiavi *T* deve rilasciare un ***certificato***:

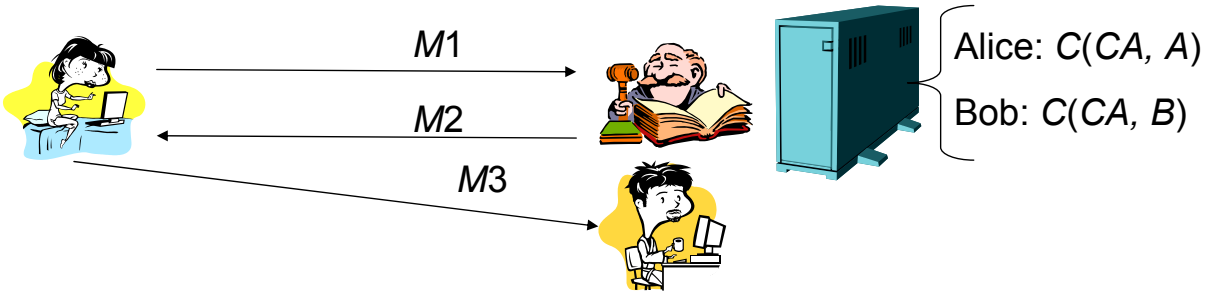
  $\boldsymbol{C(A) = \text{Alice}, e_A, S_{dCA}(\text{A}, e_A) = \text{Alice}, e_A, \langle A, e_A \rangle_{CA}}$

  un documento/messaggio ***firmato*** da una *Certification Authority* (CA) che stabilisce il collegamento (*utente* $\leftrightarrow$ *chiave pubblica dell'utente*)

Certification Authority

- CA è una ***terza entità fidata*** che garantisce
- ***l'integrita*** del collegamento (*utente* ↔ *chiave pubblica dell'utente* e
- la ***corrispondenza*** tra una chiave pubblica ed il soggetto possessore della chiave pubblica (certificazione)
  - ***identificazione del soggetto***
  - ***autenticazione della chiave***

Distribuzione Chiavi con Certificato



Alice vuole sapere la chiave pubblica e_B di Bob

M1 $A \rightarrow T$: B
$M2$ $T \rightarrow A$: $C(CA, B)$

❖ Alice verifica la firma della CA e si convince che e_B è la chiave pubblica di Bob

$C(CA, B) = B, e_B, S(d_{CA}, (B, e_B))$

Chi certifica una CA?

PROBLEMA

- Per verificare la firma sul certificato,
  Alice si deve procurare e_{CA}, la chiave pubblica di CA…
- Come fa Alice ad essere sicura che e_{CA} è proprio la chiave pubblica di CA?
- Chi rilascia un certificato alla CA
- Dove si trova questo certificato?

Chi certifica una CA?

SOLUZIONE

- CA pubblica e_{CA} sui quotidiani più importanti
- La chiave di CA è certificata da un'altra CA
  - un'altra chiave pubblica, un altro certificato e così via…
  - Certification Hierarchy (X.509), Certification Web (PGP)

Come si ragiona

Alice riceve $C(B)$ e lo verifica per mezzo di e_{CA}

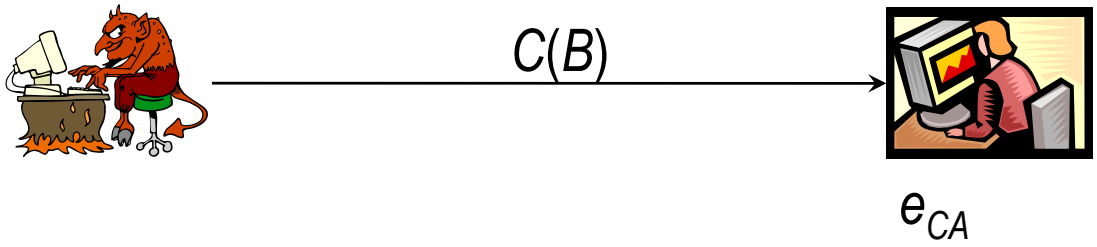
1. Siccome Alice crede che "e_{CA} appartiene a CA" allora è portata a credere che "CA ha detto che e_B appartiene a Bob"
2. Siccome Alice si fida di CA allora è portata a credere che "e_B appartiene a Bob"

Come si ragiona

Alice si fida di CA, cioè Alice ritiene che

- CA gestisca correttamente le proprie chiavi
- CA faccia bene il proprio mestiere di certificatore
  - CA ha verificato con certezza l'identità di Bob
  - CA ha verificato che e_B non sia già in uso (difficile!!)
  - CA ha verificato il possesso di d_B da parte di Bob
  - …

Significato di un certificato



Dopo aver ricevuto $C(CA, B)$ Alice

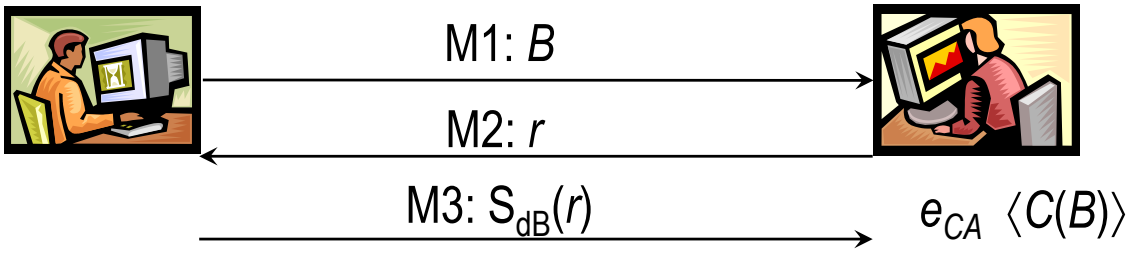
- ***può*** concludere che e_B appartiene a Bob
- ***non può*** concludere che sta parlando con Bob

Un certificato in sè non dá alcuna garanzia sull'identità del partner della comunicazione

Identificazione

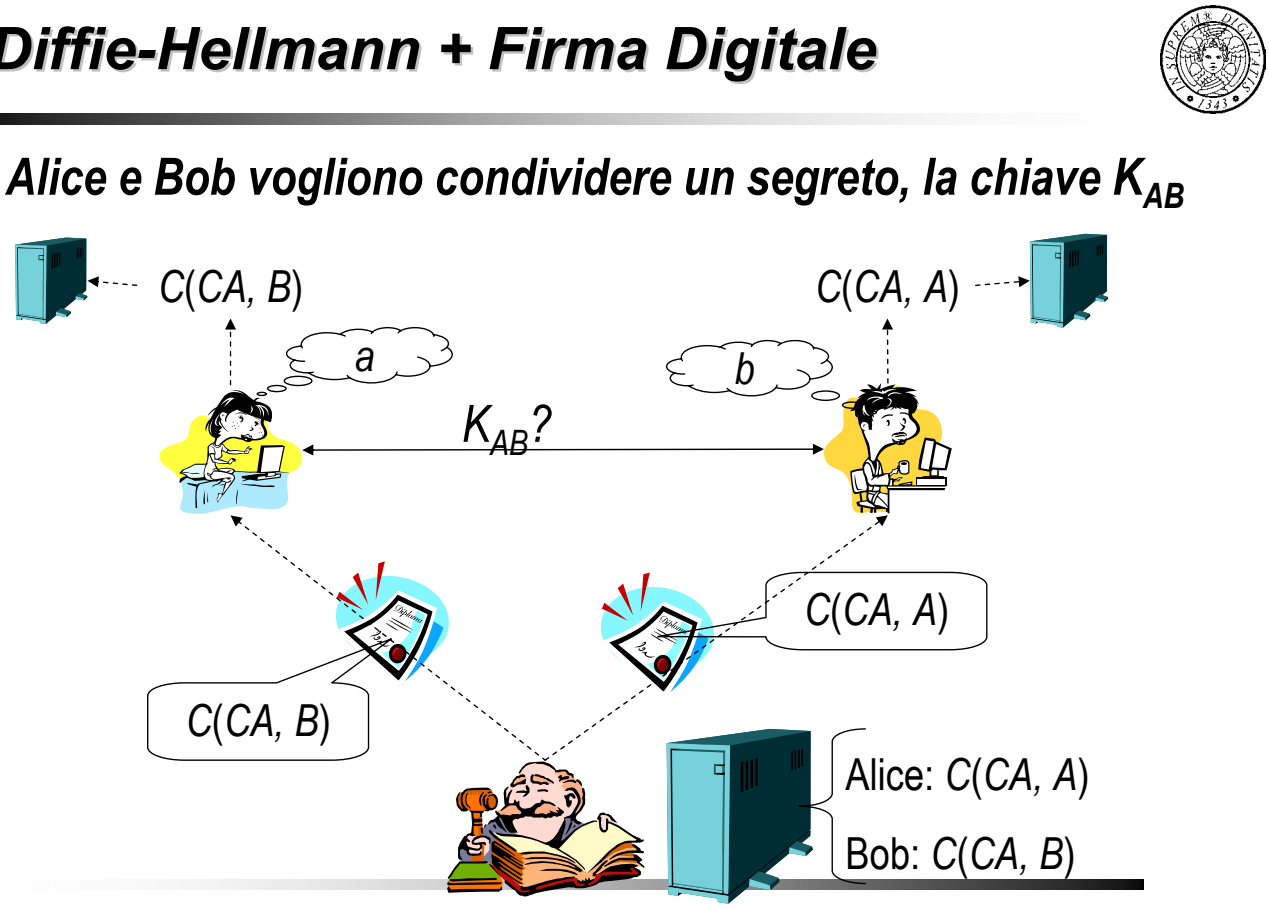
Alice vuol convincersi di comunicare effettivamente con Bob

1. Alice invia un messaggio (*challenge*) a Bob che lo deve restituire firmato con la chiave d_B



2. Dopo aver ricevuto M3 e verificato $E(d_B, r)$ Alice può concludere che sta parlando con Bob perché solo Bob può aver firmato M2 perché solo lui conosce d_B

Diffie-Hellmann + Firma Digitale



Alice e Bob vogliono condividere un segreto, la chiave K_{AB}

Diffie-Hellmann + Firma Digitale

Alice sceglie un numero random a (segreto)

- M1 $A \rightarrow B$: $g^a \bmod p$

Bob sceglie un numero random b (segreto),
calcola $K_{AB} = (g^a)^b \bmod p = g^{ab} \bmod p$

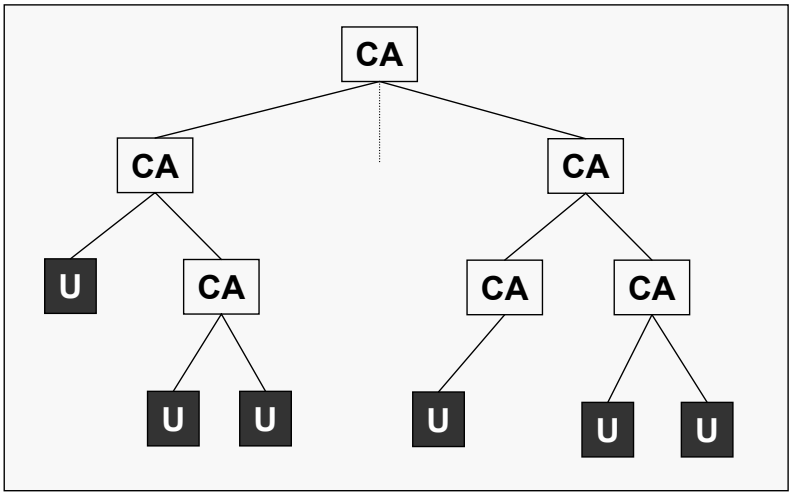
- M2 $B \rightarrow A$: $g^b \bmod p$, $E(K_{AB}, S(PRIV_B, (g^b \bmod p, g^a \bmod p)))$

Alice calcola $K_{AB} = (g^a)^b \bmod p = g^{ab} \bmod p$

- M3 $A \rightarrow B$: $E(K_{AB}, S(PRIV_A, (g^b \bmod p, g^a \bmod p)))$

Alice (Bob) conclude che K_{AB} è la chiave per comunicare ora con Bob (Alice)

Organizzazione Gerarchica



CA = Certification Autority

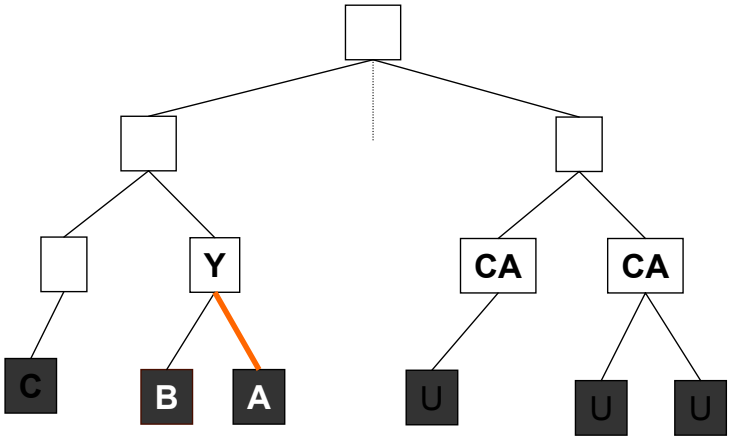
U = User

SCALABILITÀ

DELEGA dell'AUTORITÀ e della FIDUCIA

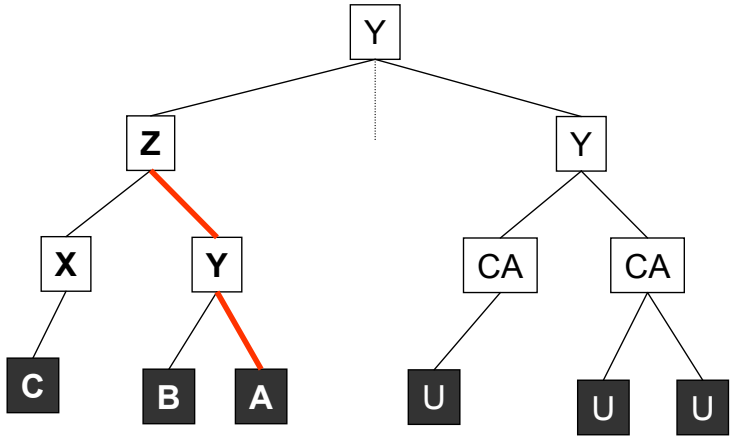
Privacy Enhanced Internet Mail (PEM)
RFC 1421–1424

Organizzazione Gerarchica



- Bob vuole procurarsi la chiave pubblica e_A di Alice, allora
- Bob preleva il certificato $\boldsymbol{CA(Y, A) = Alice, e_A, S(d_Y, (A, e_A))}$ da Y

Organizzazione Gerarchica



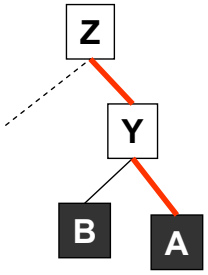
- Carol vuole procurarsi la chiave pubblica e_A di Alice, allora necessita della *catena di certificati* $C(Y, A)$, $C(Z, Y)$
- Delega di Autorità e di Fiducia: Z delega la certificazione a Y; Z deve controllare che Y svolga bene il proprio compito; Z pubblica la politica che segue per delegare l'autorità

Delega di Fiducia

Carol si fida di Z di cui conosce la chiave pubblica e_Z

Quando Carol riceve $C(Y, A)$, $C(Z, Y)$,

- $C(Z, Y)$ porta Carol a credere che e_Y è la chiave pubblica di Y
- $C(Y, A)$ porta Carol a credere che "Y ha detto che e_A è la chiave di Alice"
- Siccome Carol si fida di Z e Z certifica Y, allora
  Carol si fida anche di Y e quindi
  Carol si fida che "e_A è la chiave pubblica di Alice"



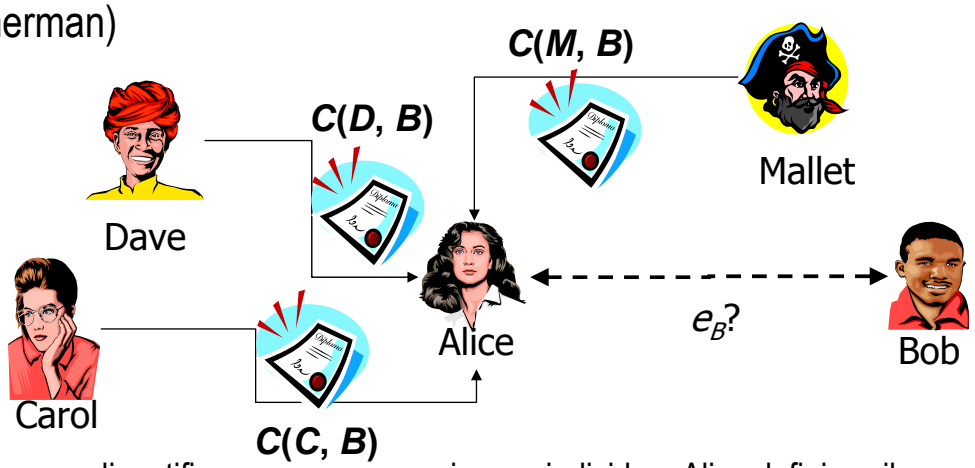
In generale tra una CA e l'utente possono esserci una o più CA

L'utente si deve fidare di tutte

Pretty Good Privacy (PGP)

È il singolo utente che decide quanta fiducia riporre in un certificato

"*PGP is for people who prefer to pack their own parachutes*" (P. Zimmerman)



In base al numero di certificati e la fiducia in ciascun individuo, Alice definisce il **proprio** livello di fiducia in e_B

FORMATO DI UN CERTIFICATO

STANDARD X.509 (RFC 2459)

- PRINCIPALI COMPONENTI
  - Il nome dell'entità certificata
  - La chiave pubblica di tale entità
  - Il nome della CA
  - La firma digitale
- ALTRE COMPONENTI
  - Algoritmi usati per la firma digitale
  - Periodo di validità del certificato

X.509: ESEMPIO (continua)

BASIC INFORMATION

Signature Algorithm: sha1WithRSAEncryption
Issuer: C=US, O=American Express Company, Inc.,
OU=American Express Technologies,
CN=American Express Global Certificate Authority

del subject

Validity
Not Before: Aug 14 19:06:00 1998 GMT
Not After : Aug 14 23:59:00 2013 GMT

X.500 (ITU-T)

Subject: C=US, O=American Express Company, Inc.,
OU=American Express Technologies,
CN=American Express Global Certificate Authority

Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (2048 bit)
Modulus (2048 bit):
f0:24:26:66:2e:fb:eb:4a:73:71:53:89:47:cb:26:
...
Exponent: 65537 (0x10001)

chiave del subject

CERTIFICATO X.509 (continua)

Signature Algorithm: sha1WithRSAEncryption
c7:61:45:a8:8a:71:b9:be:34:e9:21:7b:21:cd:56:13:98:d5:
...

chiave di CA

-----BEGIN CERTIFICATE-----

MIIEBDCCAuygAwIBAgICAIUwDQYJKoZIhvcNAQEFBQAwgZYxCzAJBgNVBAYTAlVT
MScwJQYDVQQKEx5BbWVyaWNhbiBFeHByZXNzIENvbXBhbnksIEluYy4xJjAkBgNV
BAsTHUFtZXJpY2FuIEV4cHJlc3MgVGVjaG5vbG9naWVzMTYwNAYDVQQDEy1BbWVy
aWNhbiBFeHByZXNzIEdsb2JhbCBDZXJ0aWZpY2F0ZSBBdXRob3JpdHkwHhcNOTgw
ODE0MTkwNjAwWhcNMTMwODE0MjM1OTAwWjCBljELMAkGA1UEBhMCVVMxJzAlBgNV
BAoTHkFtZXJpY2FuIEV4cHJlc3MgQ29tcGFueSwgSW5jLjEmMCQGA1UECxMdQW1l
cmljYW4gRXhwcmVzcyBUZWNobm9sb2dpZXMxNjA0BgNVBAMTLUFtZXJpY2FuIEV4
cHJlc3MgR2xvYmFsIENlcnRpZmljYXRlIEF1dGhvcml0eTCCASIwDQYJKoZIhvcN
AQEBBQADggEPADCCAQoCggEBAPAkJmYu++tKc3FTiUfLJjxTkpRMysKFtQ34w1e9
Lyofahi3V68MABb6oLaQpvcaoS5mJsdoo4qTaWa1RlYtHYLqkAwKIsKJUI0F89Sr
c0HwzxKsKLRvFJSWWUuekHWG3+JH6+HpT0N+h8onGGaetcFAZX38YW+tm3LPqV7Y
8/nabpEQ+ky16n4g3qk5L/WI5IpvNcYgnCuGRjMK/DFVpWusFkDpzTVZbzIEw3u1
D3t3cPNIuypSgs6vKW3xEW9t5gcAAe+a8yYNpnkTZ6/4qxx1rJG1a75AsN6cDLFp
hRlxkRNFyt/R/eayypaDedvFuKpbepALeFY+xteflEgR9a0CAwEAAaNaMFgwEgYD
VR0TAQH/BAgwBgEB/wIBBTAOBgNVHQ8BAf8EBAMCAQYwFwYDVR0gBBAwDjAMBgoq
hkiG+Q8KAQUBMBkGA1UdDgQSBBBXRzV7NicRqAj8L0Yl6yRpMA0GCSqGSIb3DQEB
BQUAA4IBAQDHYUWoinG5vjTpIXshzVYTmNUwY+kYqkuSFb8LHbvskmnFLsNhi+gw
RcsQRsFzOFyLGdIr80DrfHKzLh4n43WVihybLsSVBYZy0FX0oZJSeVzb9Pjc5dcS
sUDHPIbkMWVKyjfG3nZXGWlMRmn8Kq0WN3qTrPchSy3766lQy8HRQAjaA2mHpzde
VcHF7cTjjgwml5tcV0ty4/IDBdACOyYDQJCevgtbSQx48dVMVSng9v1MA6lUAjLR
V1qFrEPtWzsWX6C/NdtLnnvo/+cNPDuom0lBRvVzTv+SZSGDE1Vx60k8f4gawhIo
JaFGS0E3l3/sjvHUoZbCILZerakcHhGg

-----END CERTIFICATE-----

La CA crea il certificato firmando la basic information con la propria chiave privata

L'algoritmo RSA
(Rivest-Shamir-Adleman)

Rivest Shamir Adleman (RSA, 1978)

- CREAZIONE DELLE CHIAVI
  Selezionare ***p***, ***q*** due numeri primi grandi (100÷200 cifre decimali)
  Sia $n = p \times q$ ed $\phi(n)=(p-1)(q-1)$
  Scegliere un numero random ***e*** tale che $GCD(e, \phi(n)) = 1$
  Calcolare ***d*** tale che $e \times d = 1 \bmod \phi(n)$
  CHIAVE K_1 (d): ***(d, n)***
  CHIAVE K_2 (e): ***(e, n)***

  A questo punto i numeri ***p*** e ***q*** possono essere cancellati ma non devono ***mai*** essere rivelati

Algoritmo RSA

- CIFRATURA

  Sia m un intero $1 \leq m < n$, $c = m^e \bmod n$

- DECIFRATURA

  $c^d \equiv m \bmod n$

  Queste due operazioni sono "pesanti" dal punto di vista computazionale

Example with artificially small numbers

Key generation

- Let $p = 47$ e $q = 71$
  $n = p \times q = 3337$
  $\phi = (p-1) \times (q-1) = 46 \times 70 = 3220$
- Let $e = 79$
  $ed \equiv 1 \bmod \phi(n)$
  $79 \times d \equiv 1 \bmod 3220$
  $d = 1019$

Encryption

Let $m = 9666683$

Divide m into blocks $m_i < n$

$m_1 = 966$; $m_2 = 668$; $m_3 = 3$

Compute

$c_1 = 966^{79} \bmod 3337 = 2276$

$c_2 = 668^{79} \bmod 3337 = 2423$

$c_3 = 3^{79} \bmod 3337 = 158$

$c = c_1 c_2 c_3 = 2276\ 2423\ 158$

Decryption

$m_1 = 2276^{1019} \bmod 3337 = 966$

$m_2 = 2423^{1019} \bmod 3337 = 668$

$m_3 = 158^{1019} \bmod 3337 = 3$

$m = 966\ 668\ 3$

The RSA Problem (RSAP)

- **DEFINITION. The RSA Problem** (**RSAP**): recovering plaintext *m* from ciphertext *c*, given the public information (*n*, *e*)
- **FACT. RSAP $\leq_P$ FACTORING**
  - FACTORING is at least as difficult as RSAP or, equivalently,
  - RSAP is not harder than FACTORING
- It is widely believed that the RSA and the integer factorization problems are computationally equivalent, although no proof of this is known.

RSAP from another viewpoint

- A possible way to decrypt $\boldsymbol{c = m^e \bmod n}$ is to compute the ***e*-th root** of *c*
  - Computing the ***e***-th root is a computationally easy problem iff ***n*** is prime
  - If ***n*** is not prime the problem of computing the ***e***-th root is *equivalent* to factoring

Security of RSA: relation to factoring (1)

La sicurezza di RSA è basata sulla difficoltà della scomposizione in fattori primi di numeri interi grandi, ma finora non è mai stato provato che RSA è equivalente alla fattorizzazione

Security of RSA: relation to factoring (2)

- ***La fattorizzazione è facile $\Rightarrow$ violare RSA è facile***
  - Sia **(*e*, *n*)** la chiave pubblica.
  - Si scompone ***n***, si ricava ***p*** e ***q*** e ϕ**(n)** e quindi si può risolvere **e**$\times$**d = 1 mod** ϕ**(n)** ricavando **d**

- ***Violare RSA è difficile $\Rightarrow$ la scomposizione è difficile***
  - ***Questa implicazione non è mai stata provata.***
  - Finora non è mai stato trovato un metodo più efficiente per violare RSA che non sia quello di scomporre in fattori il modulo **n**

Security of RSA: relation to factoring (3)

- The problem of computing the RSA decryption exponent **d** from the public key **(n, e)** and the problem of factoring **n** are computationally equivalent
  - If the adversary could somehow factor **n**, then he can subsequently compute the private key **d** efficiently
  - If the adversary could somehow compute **d**, then it could subsequently factor **n** efficiently

Security of RSA: relation to factoring (4)

RELATIONSHIP BETWEEN FACTORING AND TOTALLY BREAKING RSA

- **A possible way to completely break RSA is to discover $\Phi(n)$**
- **Computing $\Phi(n)$ is computationally equivalent to factoring *n***
  - Given **p** and **q**, s.t. ***n* =*pq***, computing **Φ(n)** is immediate.
  - Let $\Phi(n)$ be given.

    Let $\Phi(n) = (p-1)(q-1) = n - (p+q) + 1$, then, $\mathbf{x_1 = (p+q)}$.

    Let $(p - q)^2 = (p + q)^2 - 4n$, i.e., $(p - q)(p+q) = (p + q)^2 - 4n$, then,

    $\mathbf{x_2 = (p - q)}$.

    Finally, **p = (x1 + x2)/2** and **q = (x1 – x2)/2.**

Security of RSA: brute force attack

- A possible way to completely break RSA is an exhaustive attack to the private key ***d***
- This attack could be more difficult than factoring because, according to the choice for ***e***, ***d*** can be much greater than ***p*** and ***q***.

Factoring algorithms

- **Brute Force**
  - Unfeasible if *n* large and $|p| = |q|$
- **General purpose**
  - the running times depend solely on the size of *n*
  - Quadratic sieve
  - General number field sieve
- **Special purpose**
  - the running times depend on certain properties of *n* (lead to the introduction of **strong primes**)
  - Trial division
  - Pollard's rho algorithm
  - Pollard's $p-1$ algorithm
- **Elliptic curve algorithm**

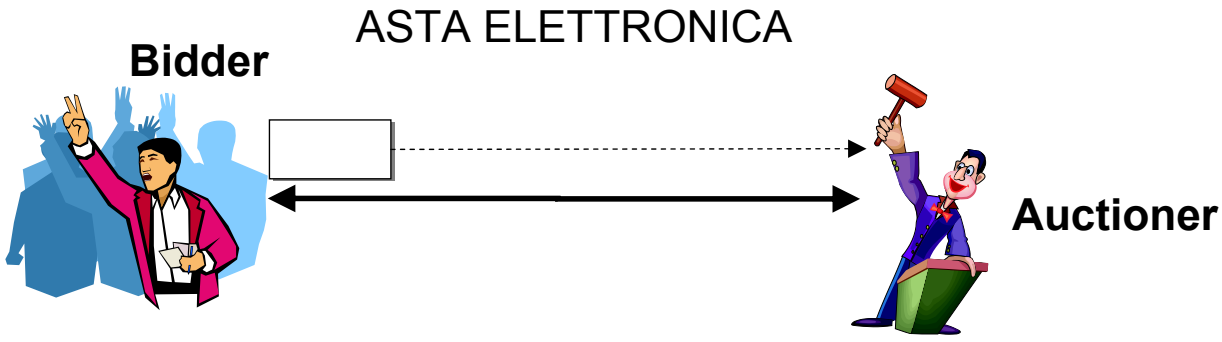
Running times

Trial division: $O\left(\sqrt{n}\right)$

Quadratic sieve: $O\left(e^{\left(\sqrt{\ln(n)\bullet\ln\ln(n)}\right)}\right)$

General number field sieve: $O\left(e^{\left(1.923\times\sqrt[3]{\ln(n)\bullet(\ln\ln(n))^2}\right)}\right)$

Un sistema insicuro con componenti sicure



Formato di un'offerta: ⟨*bidder*, *bid*⟩, con *bid* su 32 bit

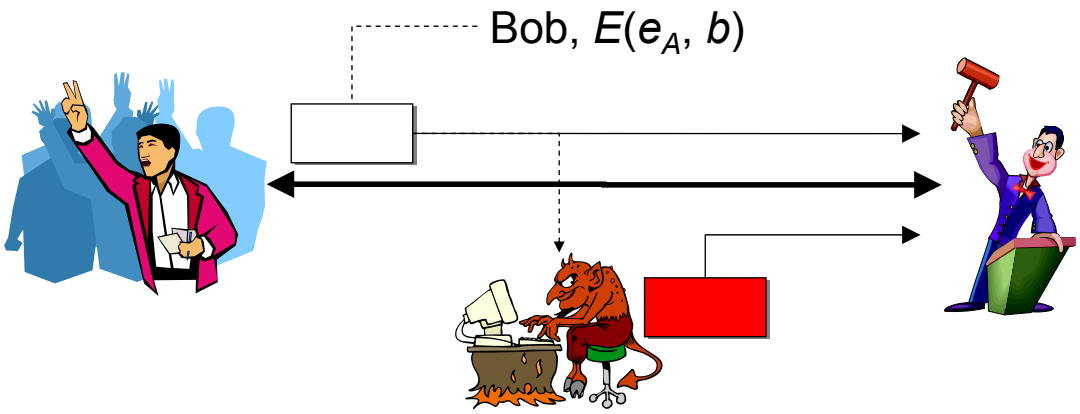
Formato di un messaggio d'offerta m = *Bob*, $E(e_A, b)$

Per privacy, l'offerta è cifrata con e_A,la chiave pubblica dell'auctioner

Bob dispone di un certificato relativo a e_{A0}

Un sistema insicuro con componenti sicure
dictionary attack

ASTA ELETTRONICA



L'avversario prova tutte le possibili (2^{32}) offerte finché ne trova una b', tale che $E(e_A, b') \equiv E(e_A, b)$, allora $b' \equiv b$

L'avversario fa quindi la minima offerta $b'+1$

Un sistema insicuro con componenti sicure
chosen-ciphertext attack

ASTA ELETTRONICA

- Sia $d_A = (d, n)$ e $e_A = (e, n)$
- Sia $c = b^e \bmod n$, con b bid
- L'avversario M si procura c e chiede all'auctioner di decifrare c e restituire il testo in chiaro corrispondente
- L'auctioner rifiuta perché, ad esempio, si "accorge" che b è un bid
- L'avversario allora sferra un ***chosen-ciphertext attack***

(continua)

Proprietà moltiplicative di RSA

- **Il chosen-plaintext attack sfrutta il fatto che RSA È UN OMOMORFISMO**

$$Se\ m = m_1 \times m_2 \Rightarrow c = c_1 \times c_2$$

Un sistema insicuro con componenti sicure

chosen-ciphertext attack

CHOSEN-CIPHERTEXT ATTACK

- L'avversario *M* si procura ***c***, seleziona un valore ***x*** (opportuno), calcola $a = x^e \times c \bmod n$ e chiede all'auctioner di decifrare ***a***
- Questa volta l'auctioner ritorna all'avversario *M* il valore $t = a^d = x^{ed} \times c^d = xb_A^{de} = xb \bmod n$ perchè non lo riconosce come bid
- L'avversario *M* ricava il bid $b = tx^{-1} \bmod n$

Un sistema insicuro con componenti sicure

chosen-ciphertext attack

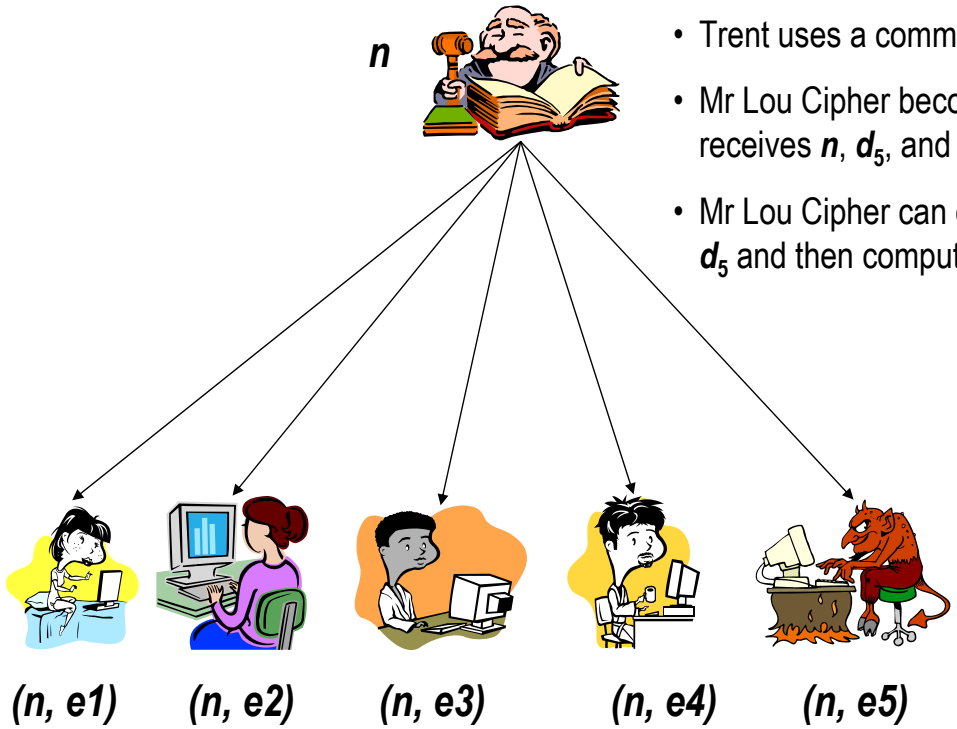
- **UNA POSSIBILE CONTROMISURA AL CHOSEN-PLAINTEXT ATTACK**

imporre una struttura ridondante al testo in chiaro

$$b' = b \,||\, b$$

Un sistema insicuro con componenti sicure

Common modulus attack

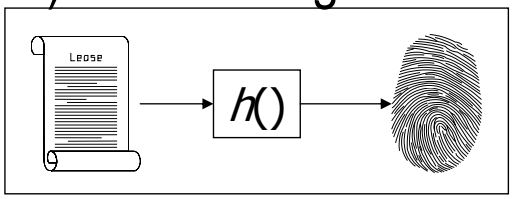


- Trent uses a common modulus ***n***.
- Mr Lou Cipher becomes a user and receives ***n***, d_5, and e_5.
- Mr Lou Cipher can efficiently factor ***n*** from d_5 and then compute all d_i.

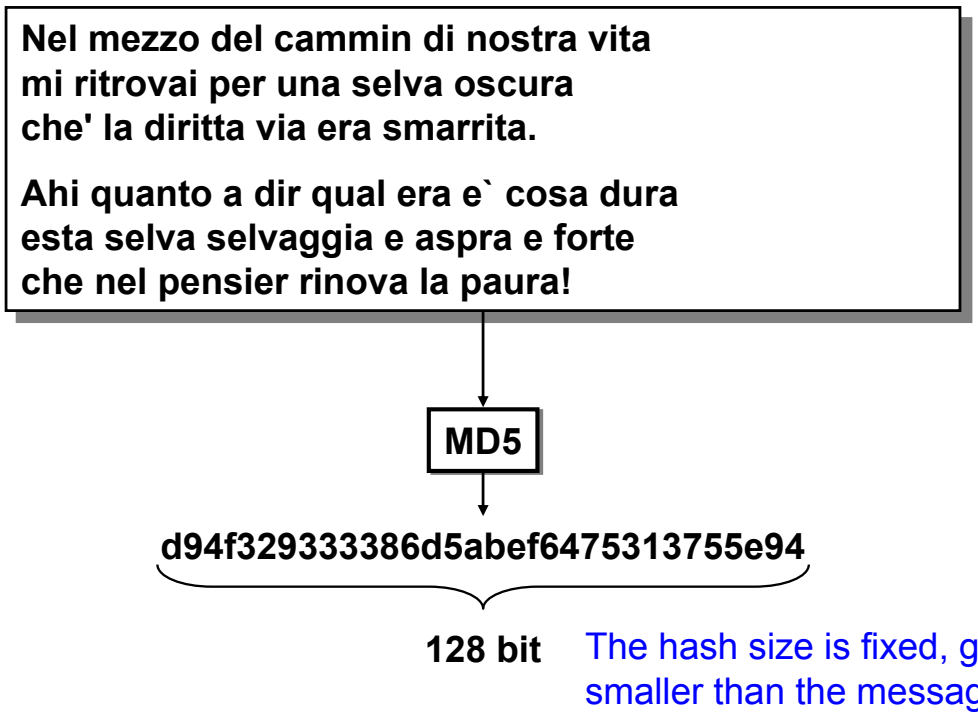
Hash functions

Hash function

- The hash (fingerprint, digest) of a message must be
  - "easy" to compute
  - "unique"
  - "difficult" to invert
  - The hash of a message can be used to
  - guarantee the integrity and authentication of a message
  - "uniquely" represent the message



Hash function



Basic properties

- A hash function maps bitstrings of arbitrary, finite length into bitstrings of fixed size

$$h : \{0,1\}^* \rightarrow \{0,1\}^m$$

- A hash function is a function *h* which has, as minumum, the following properties
  - **Compression** – *h* maps an input *x* of arbitrary finite lenth to an output *h*(*x*) of fixed bitlength *n*
  - **Ease of computation** – given an input *x*, *h*(*x*) is easy to compute
- A hash function is **many-to-one** and thus implies **collisions**

Additional security properties (MDC)

A **hash function** may have one or more of the following additional 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 computationallyinfeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that $h(x) = h(x')$

Motivation of properties

- *Preimage resistance*
  - Digital signature scheme based on RSA:
    - (n, d) is a private key; (n, e) is a public key
    - A digital signature s for m is $s = (h(m))^d \bmod n$
  - If h is not preimage resistance an adversary can
    - select $z < n$, compute $y = z^e \bmod n$ and find m' such that $h(m') = y$;
    - claim that z is a digital signature for m' (existential forgery)

Motivation of properties

- *2nd-preimage resistance*
  - Digital signature with appendix (*S*, *V*)
    - $s = S(h(m))$ is the digital signature for m
  - A trusted third party chooses a message m that Alice signs producing $s = S_A(h(m))$
  - If h is not 2nd-preimage resistant, an adversary (e.g. Alice herself) can
    - determine a 2nd-preimage m' such that h(m') = h(m) and
    - claim that Alice has signed m' instead of m

Motivation of properties

- *Collision resistance*
  - Digital signature with appendix (S, V)
    - s = S(h(m)) is the digital signature for m
  - If h is not collision resistant, Alice (an untrusted party) can
  - choose m and m' so that h(m) = h(m')
  - compute $s = S_A(h(m))$
  - issue ⟨m, s⟩ to Bob
  - later claim that she actually issued ⟨m', s⟩

Relationship between properties

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

MDC classification

- A **one-way hash function** (**OWHF**) is a hash function with the following properties: **preimage resistance**, **2-nd preimage resistance**
- A **collision resistant hash function** (**CRHF**) is a hash function with the following properties: **preimage resistance**, **collision resistance**

Objective of adversaries vs MDC

- **Attack to OWHF**
  - given an hash value y, find a preimage x such that y = h(x); or
  - given a pair (x, h(x)), find a second preimage x' such that h(x) = h(x')
- **Attack to CRHF**
  - find any two inputs x. x', such that h(x) = h(x')

CRHF must be designed to withstand standard birthday attacks

Hash type	Design goal	Ideal strength
OWHF	preimage resistance	2^m
	2nd-premage resistance	2^m
CRHF	collisione resistance	$2^{m/2}$

Upper bounds of strength

Hash Function	*n*	*m*	Preimage	Collision	Comments
Matyas-Meyer-Oseas	*n*	*m*	2^n	$2^{n/2}$	cifrario
MDC-2 (con DES)	64	128	2×2^{82}	2×2^{54}	cifrario
MDC-4 (con DES)	64	128	2^{109}	2×2^{54}	cifrario
Merkle (con DES)	106	128	2^{112}	2^{56}	cifrario
MD4*	512	128	2^{128}	2^{20}	ad-hoc
MD5	512	128	2^{128}	2^{64}	ad-hoc
RIPEMD-128	512	128	2^{128}	2^{64}	ad-hoc
SHA-1, RIPEMD-160	512	160	2^{160}	2^{80}	ad-hoc

block size: *n*
output size: *m*

bitsize for practical security

OWHF: m ≥ 80; CRHF: m ≥ 160

An example

Alice wants to be able to proof that, at a given time *t*, she held a document *m* without revealing it

d = *h*(*m*)

Notary

⟨*Alice*, *d*⟩

t = *clock*()

s = *S*(*PRIV*$_N$, (*d*, *t*))

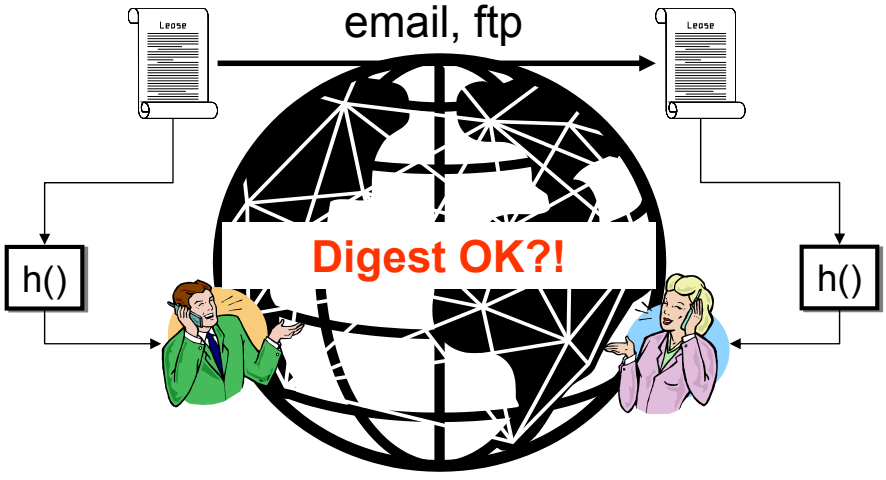
⟨*Notary*, *t*, *s*⟩

Digital signature indissolubly links d to *t*

Alice can exhibit *m*, *t*, *s*

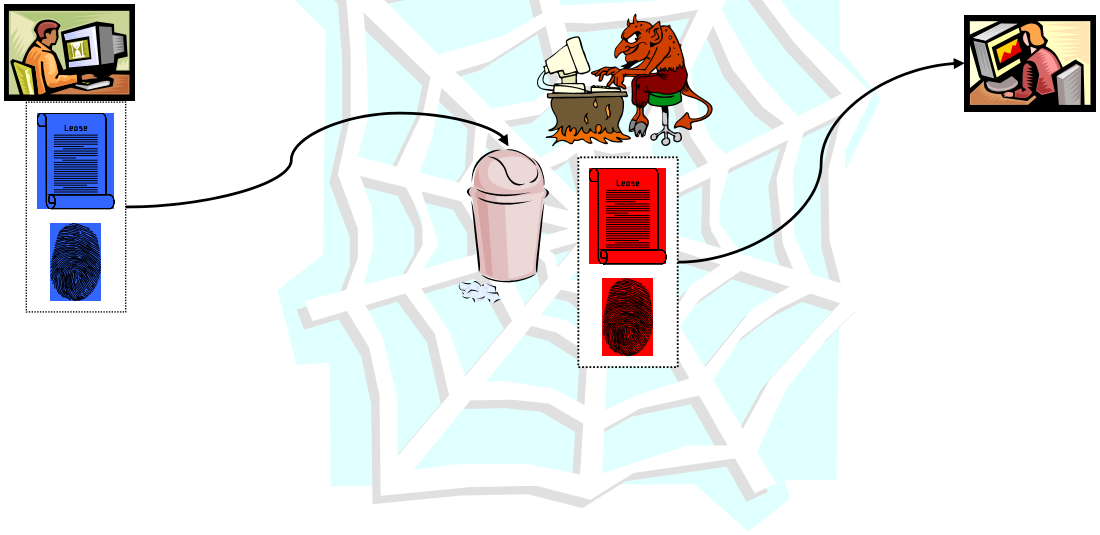
Example

*The purpose of **MDC**, **in conjunction with other mechanisms** (authentic channel, encryption, digital signature), is to provide **message integrity***



Example

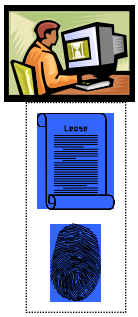
An insecure system made of secure components



MDC alone is not sufficient to provide data integrity

Example: mdc + cipher

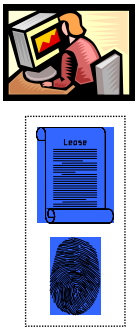
secret shared key e



m

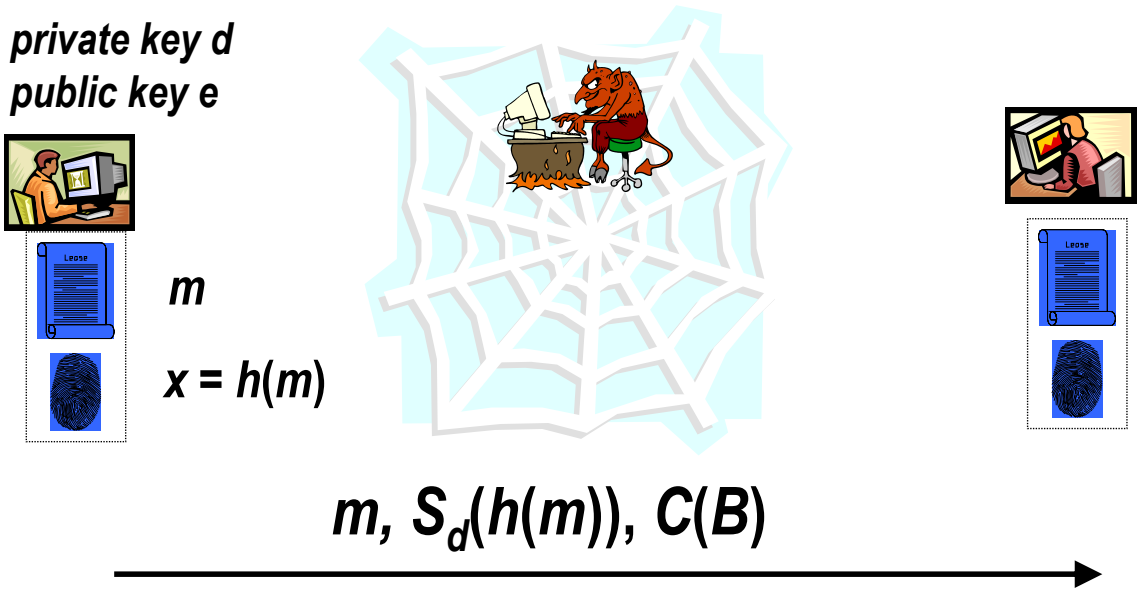
x = h(m)

e



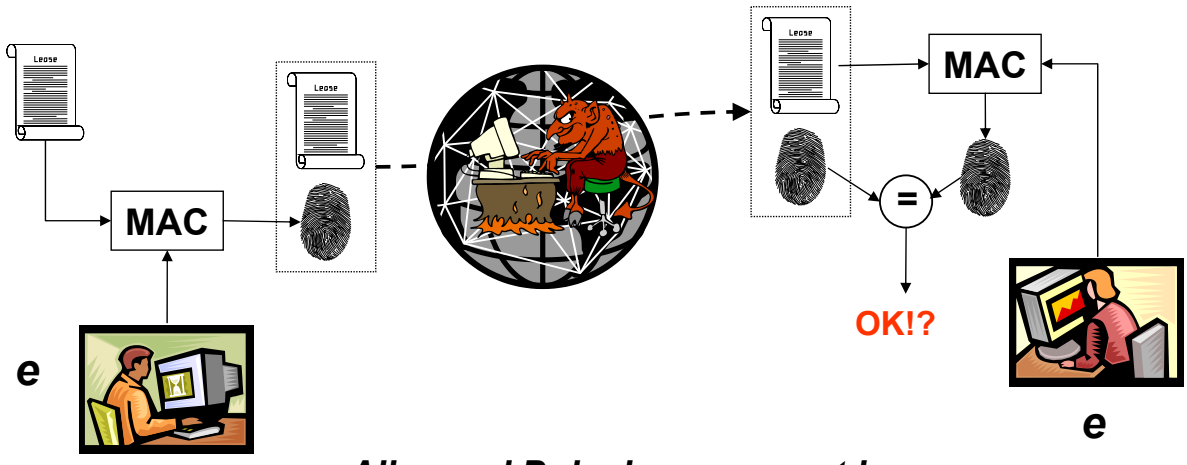
$$c = E_e(m, h(m))$$

Example: mdc + digital signature



Message Authentication Code

*The purpose of **MAC** is to provide **message authentication by symmetric techniques** (without the use of any additional mechanism)*



Alice and Bob share a secret key e

Message Authentication Code

Definition. A MAC algorithm is a famility of functions h_k, parametrized by a **secret** key k, with the following properties:

- **ease of computation** – Given a function $\boldsymbol{h_k}$, a key $\boldsymbol{k}$ and an input $\boldsymbol{x}$, $\boldsymbol{h_k(x)}$ is ***easy to compute***
- **compression** – $\boldsymbol{h_k}$ maps an input $\boldsymbol{x}$ of arbitrary finite bitlength into an output $\boldsymbol{h_k(x)}$ of fixed length $\boldsymbol{n}$.
- **computation-resistance** – for each key $\boldsymbol{k}$, given zero o more $\boldsymbol{(x_i, h_k(x_i))}$ pairs, it is **computationally infeasible** to compute $\boldsymbol{(x, h_k(x))}$ for any new input $\boldsymbol{x \neq x_i}$ (including possible $\boldsymbol{h_k(x) = h_k(x_i)}$ for some $\boldsymbol{i}$).

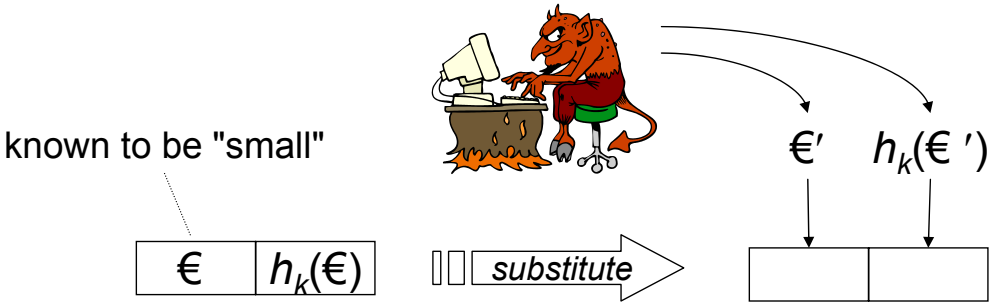
Message Authentication Code

- ***MAC forgery*** occurs if computation-resistance does not hold
- **Computation resistance implies key non-recovery** (but not vice versa)
- **MAC definition says nothing about preimage and 2nd-preimage for parties knowing *k***
- **For an adversary not knowing *k***
  - **h_k must be 2nd-preimage and collision resistant;**
  - **h_k must be preimage resistant w.r.t. a chosen-text attack;**

Types of forgery

- Forgery allows an adversary to have a forged text accepted as authentic
- Classification of forgeries
  - ***Selective forgeries***: an adversary is able to produce text-MAC pairs of text of his choice
  - ***Existential*** *forgeries*: an adversary is able to produce text-MAC pairs, but with no control over the value of that text
- Comments
  - Key recovery allows both selective and existential forgery
  - Even an existential forgery may have severe consequences

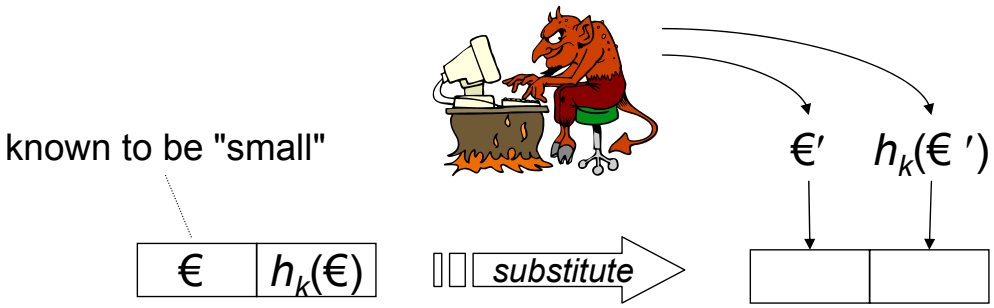
An example of existential forgery



Mr. Lou Cipher

- knows that € is a small number
- esistentially forges a pair (€′, $h_k(€')$) with €′ uniformly distributed in [0, $2^{32} - 1$] ($P_{forgery} = 1 - €/2^{32}$)
- substitutes (€, $h_k(€)$) with (€′, $h_k(€')$)

An example of existential forgery



Countermeasure

Messages whose integrity or authenticity has to be verified are constrained to have pre-determined structure or a high degree of verifiable redundancy

For example: change € into € || €

Security objectives

Let h_k be a MAC algorithm with a k-bit key and an m-bit output

Design Goal	Ideal strength	Adversary's Goal
key non-recovery	2^k	deduce *key*
computational resistance	$P_f = \max(2^{-k}, 2^{-m})$	produce new (text, MAC)

P_f is the probability of forgery by correctly guessing a MAC

bitsize for practical security

- **m ≥ 64 bit**
- **k ≥ 64 ÷ 80 bit**

Implementation

- ***MAC based on block-cipher***
  - CBC-based MAC
- ***MAC based on MDC***
  - The MAC key should be involved at both the start and the end of the MAC computation

$$h_k(x) = h\left(k \| p \| x \| k\right) \qquad \textit{envelope method with padding}$$

$$h_k(x) = h\left(k \| p_1 \| h\left(k \| p_2 \| x\right)\right) \qquad \textit{hash-based MAC}$$