

# Reti di Calcolatori



**Prof. Roberto Canonico**

**Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione**

**Corso di Laurea in Ingegneria Informatica**

Sicurezza nella comunicazione in rete:  
integrità dei messaggi,  
firma elettronica,  
protocolli di autenticazione

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

### Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonico, Giorgio Ventre



# Funzioni hash crittografiche

- Una funzione hash  $H$  è una funzione non invertibile che mappa una stringa  $m$  di lunghezza arbitraria in una stringa  $h$  di lunghezza minore prefissata (*digest*)
$$h = H(m)$$
- Le funzioni hash crittografiche sono particolari funzioni hash che godono di alcune proprietà che le rendono adatte per l'uso nella crittografia
  - **resistenza alla pre-immagine**: dato un valore di hash  $h$ , sia computazionalmente intrattabile la ricerca di una stringa  $m$  tale che:
$$h = H(m)$$
  - **resistenza alla seconda pre-immagine**: data una stringa  $m_1$ , sia computazionalmente intrattabile la ricerca di una stringa  $m_2$  tale che:
$$H(m_1) = H(m_2)$$
  - **resistenza alle collisioni**: sia computazionalmente intrattabile la ricerca di una coppia di stringhe  $m_1$  ed  $m_2$  tali che:  $H(m_1) = H(m_2)$



# Esempio di hash crittografico

- La figura mostra i 20 byte (160 bit) restituiti dalla funzione di hash crittografico SHA-1 per cinque diversi messaggi
  - Una piccola modifica del testo produce valori di hash drasticamente differenti (*effetto valanga*)

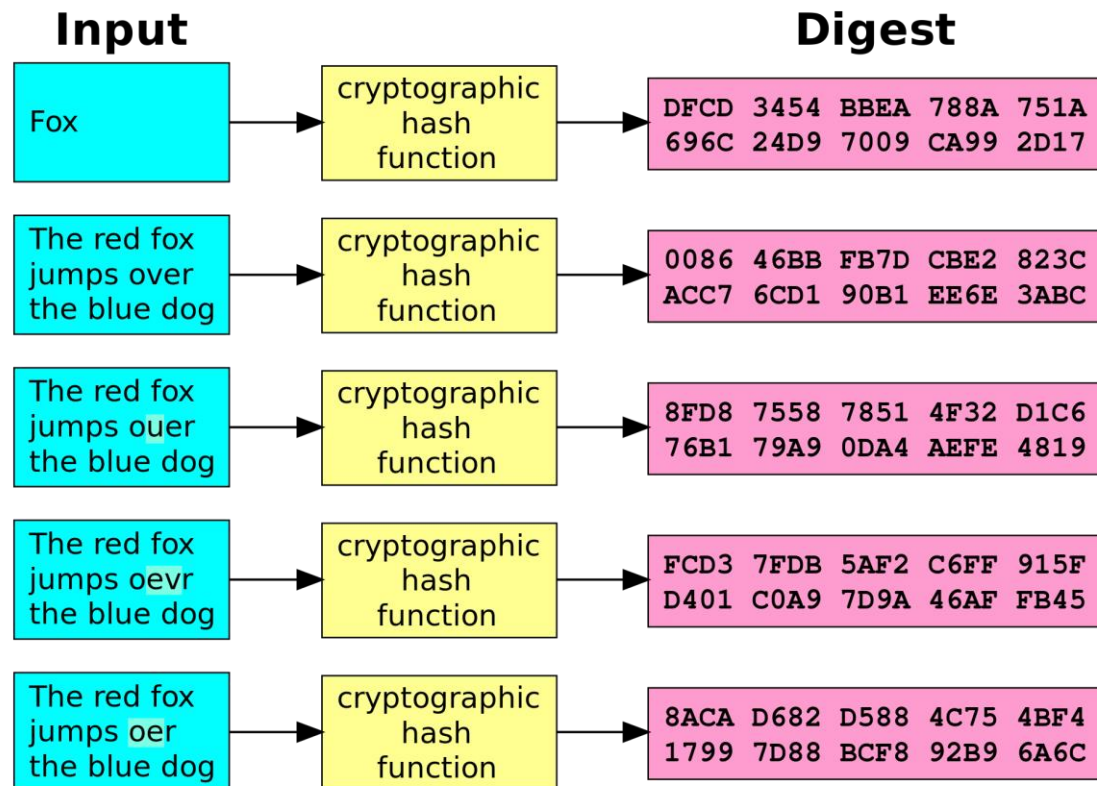


Immagine da: [https://en.wikipedia.org/wiki/File:Cryptographic\\_Hash\\_Function.svg](https://en.wikipedia.org/wiki/File:Cryptographic_Hash_Function.svg)



# Funzioni hash crittografiche: collisioni

- Data una funzione di hash  $H()$ , si dice che *si produce una **collisione*** quando due documenti diversi  $\text{Doc1} \neq \text{Doc2}$  hanno lo stesso valore di hash  $H(\text{Doc1}) = H(\text{Doc2})$

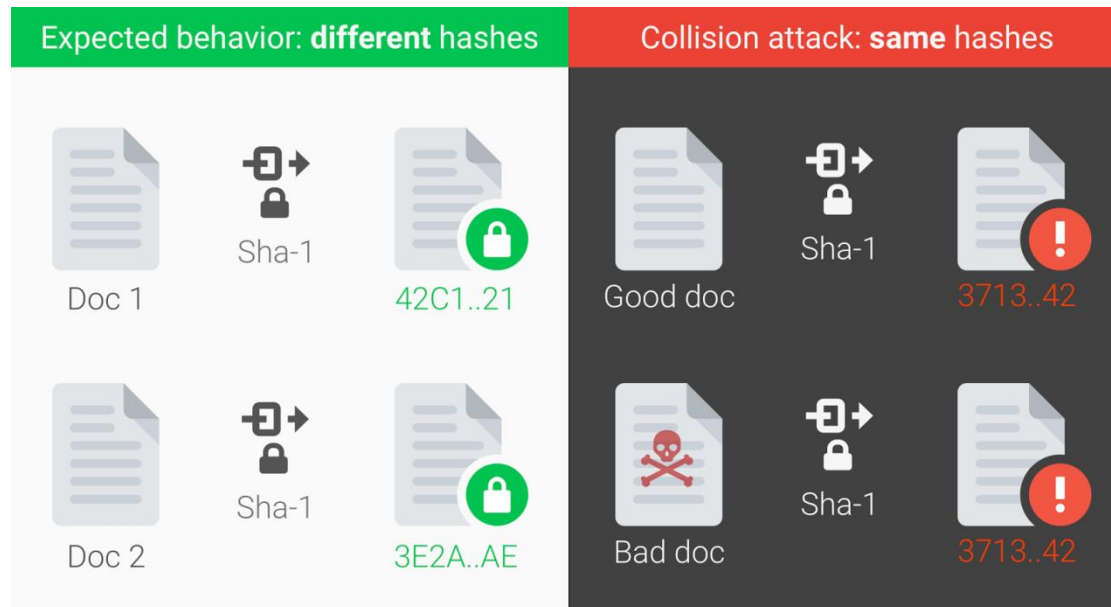


Immagine da: <https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html>



# Perchè la checksum non è una funzione hash crittografica

- La Internet checksum (usata da TCP ed UDP per verificare l'integrità di una PDU arrivata a destinazione) produce un *digest* di lunghezza fissa (16-bit) da un messaggio di lunghezza arbitraria ma **non** è una funzione di hash crittografica

Infatti, dato un messaggio con un dato valore di checksum, è facile trovare un altro messaggio con lo stesso valore di checksum:

<u>messaggio</u>	<u>ASCII format</u>
I O U 1	49 4F 55 <b>31</b>
0 0 . 9	30 30 2E <b>39</b>
9 B O B	39 42 4F 42
	<hr/>
	B2 C1 D2 AC

<u>messaggio</u>	<u>ASCII format</u>
I O U 9	49 4F 55 <b>39</b>
0 0 . 1	30 30 2E <b>31</b>
9 B O B	39 42 4F 42
	<hr/>
	B2 C1 D2 AC

Messaggi diversi  
ma stessa checksum!



# Funzioni hash usate in pratica (1)

- **MD5 (RFC 1321)**

- ideato da Ron Rivest
- calcola un valore hash da 128 bit in 4 passi
- recentemente (2005) sono state trovate delle tecniche di attacco per MD5 che sono in grado di trovare due messaggi con lo stesso valore di hash (*collisioni*) quindi MD5 non dovrebbe più essere usato

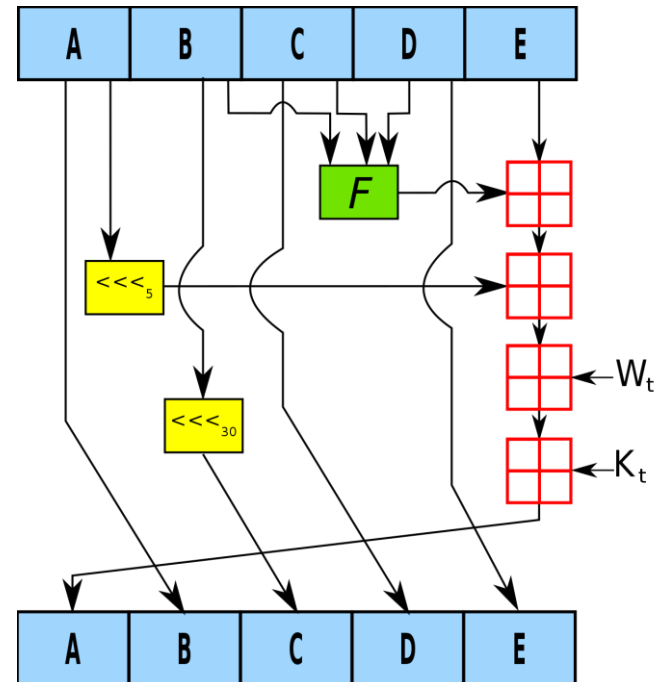
- **SHA-1**

- calcola un valore hash da 160 bit con un algoritmo basato su 80 iterazioni
- SHA-1 è usata da diversi protocolli ed applicazioni, tra i quali TLS ed SSL, PGP, SSH, S/MIME, ed IPsec
- nel 2005, un team di ricercatori della Shandong University in China ha annunciato di aver trovato una tecnica in grado di creare collisioni SHA-1 con un numero di operazioni dell'ordine di  $2^{69}$  pertanto oggi SHA-1 non è più considerato sicuro per applicazioni che richiedono elevata sicurezza
- nel 2017, un team di ricercatori di Google ha annunciato di aver realizzato con successo due diversi file PDF aventi lo stesso hash in un numero di tentativi di circa  $2^{63}$



# Come funziona SHA-1

- SHA-1 calcola un digest di 160 bit (20 byte) con un algoritmo che consiste di vari passi
- Il fulcro dell'algoritmo SHA-1 è chiamato *compression function* ed è formato da 4 cicli di 20 iterazioni ciascuno
- La figura mostra una singola iterazione:  
A, B, C, D ed E sono parole di 32 bit;  
F è una funzione non lineare che varia ad ogni ciclo;  
left\_shift\_n denota una rotazione dei bit di sinistra di n posti; n varia per ogni ciclo
- Il blocco + denota l'addizione modulo  $2^{32}$
- $K_t$  è una costante





# Funzioni hash usate in pratica (2)

- **SHA-2**

- il nome SHA-2 indica in realtà quattro diversi algoritmi (SHA-224, SHA-256, SHA-384 e SHA-512) che producono digest di lunghezza in bit pari al numero indicato nella loro sigla
- gli algoritmi usati da SHA-2 sono simili a quello usato da SHA-1
- gli algoritmi SHA-256 e SHA-512 lavorano, rispettivamente, con word di 32 e 64 bit: utilizzano un numero differente di rotazioni e di costanti addizionali, ma la loro struttura è sostanzialmente identica
- gli algoritmi SHA-224 e SHA-384 sono semplicemente versioni troncate dei precedenti due, con hash calcolati con differenti valori iniziali
- SHA-2 è oggi usato al posto di SHA-1 in diversi protocolli ed applicazioni, tra i quali TLS ed SSL, PGP, SSH, S/MIME, ed Ipsec

- **SHA-3**

- SHA-3 (Secure Hash Algorithm 3) è l'ultimo membro della famiglia di standard Secure Hash Algorithm, rilasciato dal NIST il 5 agosto 2015
  - il codice sorgente dell'implementazione di riferimento è di pubblico dominio
  - SHA-3 è un sottoinsieme della famiglia di primitive crittografiche Keccak
-

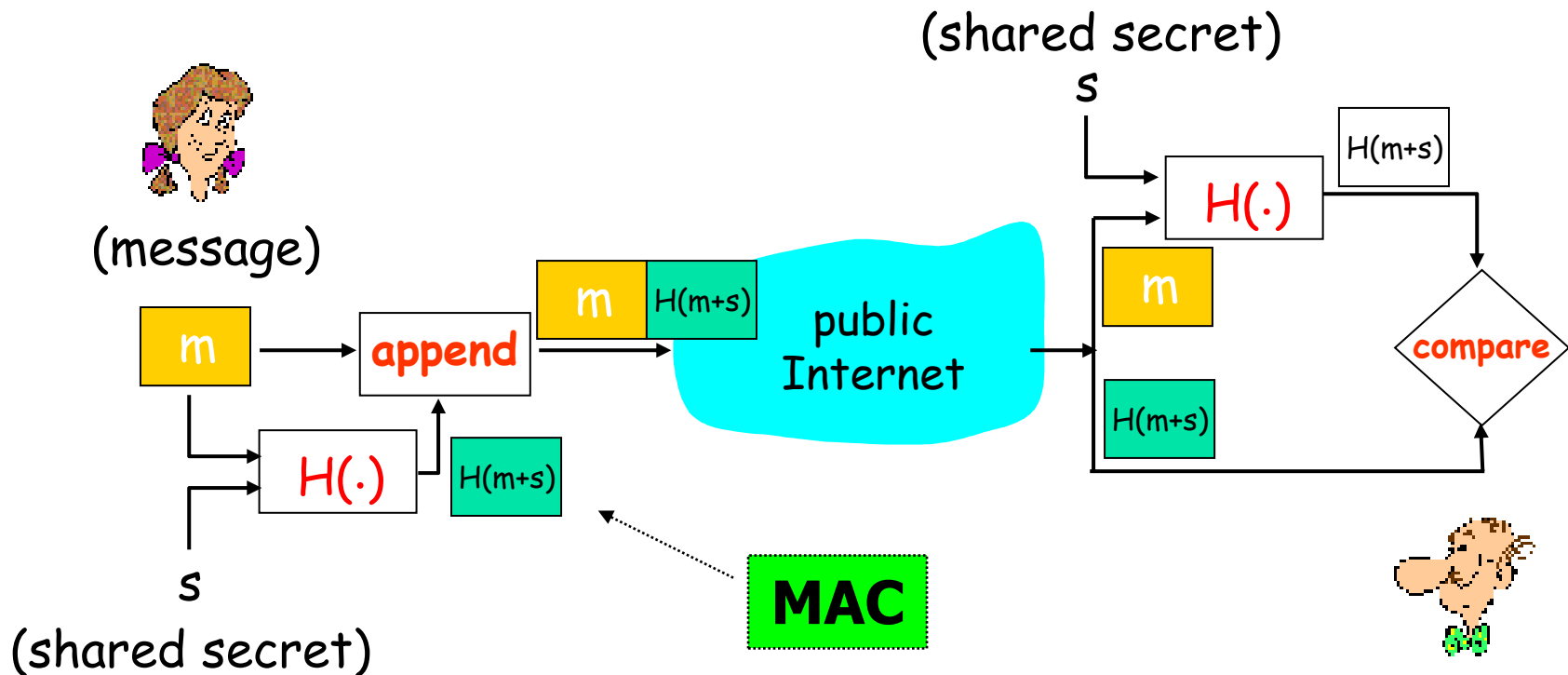


# Integrità dei messaggi

- Bob riceve un messaggio da Alice, e desidera assicurarsi che:
  - il messaggio non è stato alterato da alcuno dopo che Alice lo ha inviato
- Prima soluzione: **funzioni hash crittografiche**
  - Alice trasmette il messaggio insieme al valore di hash calcolato sul messaggio originario
  - Se il messaggio arriva a Bob alterato da Trudy, il valore di hash calcolato da Bob sarà completamente differente rispetto a quello calcolato da Alice sul testo originale, rivelando la tentata modifica
  - Non è sufficiente: se Trudy facesse pervenire a Bob l'hash ricalcolato sul messaggio alterato, Bob non ha modo di scoprire la manomissione
    - Occorre un segreto condiviso esclusivamente da Alice e Bob !



# Message Authentication Code (MAC)



Consente di controllare l'integrità di un messaggio attraverso l'uso di una funzione hash ed una chiave segreta  $s$  (*chiave di autenticazione*) nota ad entrambi (*shared secret*)



# Firma digitale

- La firma digitale è una tecnica crittografica che ha gli stessi scopi della firma fatta a mano sui documenti cartacei
- Il mittente di un messaggio (Bob) appone la sua firma digitale al fine di stabilire che egli è il creatore/autore del testo in esso contenuto
- Come la firma tradizionale, anche quella digitale deve essere **verificabile** e **non falsificabile**: il destinatario del messaggio (Alice) deve poter provare ad un terzo che è stato Bob, e nessun altro (inclusa la stessa Alice) ad apporre la firma al documento




# Firma digitale

Una semplice tecnica di firma digitale:

- Bob “firma” il messaggio  $m$  crittografandolo con la sua chiave privata  $K_B^-$ , creando un messaggio firmato  $K_B^-(m)$

Bob's message,  $m$

Dear Alice  
Oh, how I have missed  
you. I think of you all the  
time! ... (blah blah blah)  
Bob

  $K_B^-$  Bob's private  
key

public key  
encryption  
algorithm

$K_B^-(m)$

Bob's message,  
 $m$ , signed  
(encrypted) with  
his private key

# Firma digitale (continua)



- Alice riceve il messaggio  $m$ , con la firma digitale  $K_B^-(m)$
- Alice verifica che  $m$  sia stato effettivamente firmato da Bob decifrando con la chiave pubblica di Bob  $K_B^+$  il testo cifrato ricevuto  $K_B^-(m)$  e verifica che sia  $K_B^+(K_B^-(m)) = m$
- se  $K_B^+(K_B^-(m)) = m$ , chiunque abbia firmato  $m$  deve possedere la chiave privata di Bob

Alice così verifica che:

- Bob ha firmato  $m$
- Nessun altro ha firmato  $m$
- Bob ha firmato  $m$  e non  $m'$

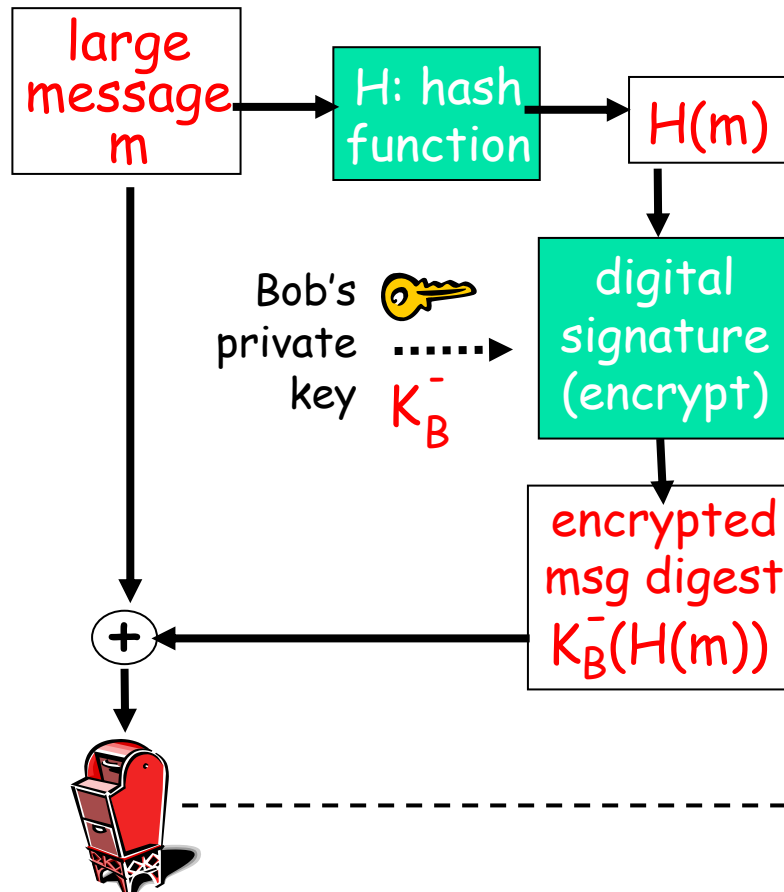
non-ripudio:

- ✓ Alice può portare il documento  $m$ , e la relativa firma  $K_B^-(m)$  da un giudice e provare che Bob ha firmato  $m$

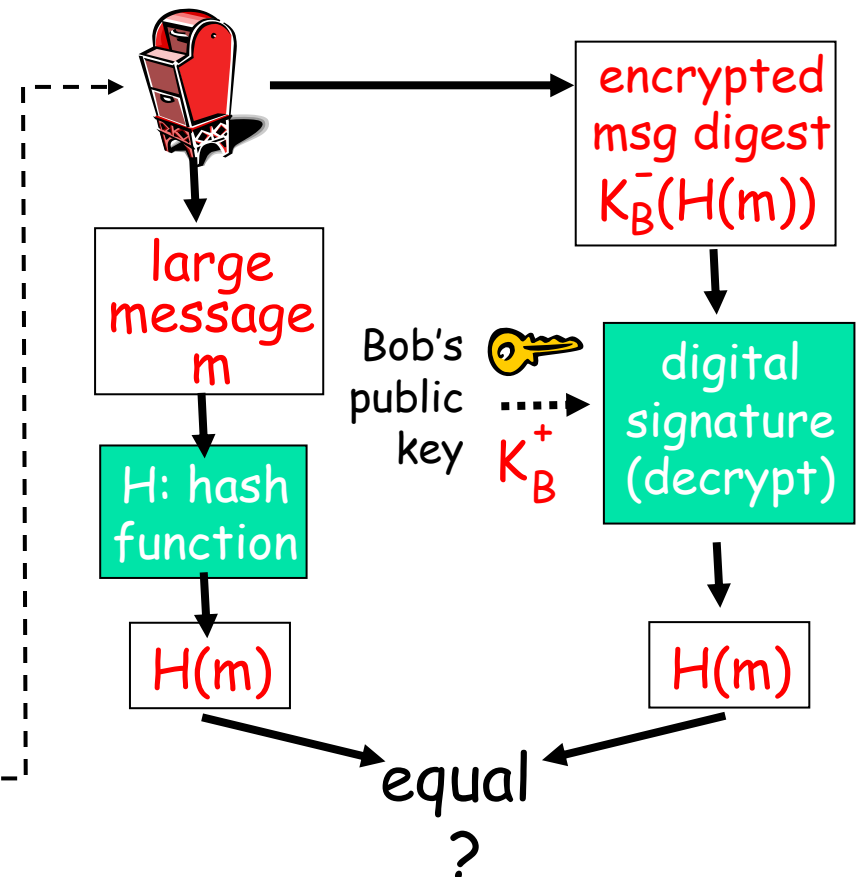
# Firma digitale = firma del digest



Bob invia un messaggio firmato digitalmente:



Alice verifica la firma e l'integrità del messaggio ricevuto:





# Firma digitale in Italia

- I presupposti giuridici che rendono possibili transazioni legali fatte grazie alla tecnologia della Firma Digitale si fondano soprattutto sull'articolo 15 comma 2 della legge 15 marzo 1997 n. 59, la cosiddetta "Bassanini 1", che recita:
  - *"Gli atti, dati e documenti formati dalla pubblica amministrazione e dai privati con strumenti informatici o telematici, i contratti stipulati nelle medesime forme, nonché la loro archiviazione e trasmissione con strumenti informatici sono validi e rilevanti a tutti gli effetti di legge..."*
- L'attuale Codice dell'Amministrazione Digitale, D. Lgs. 7 marzo 2005, n. 82 dispone inoltre, all'art. 21, comma 2, che "il documento informatico, sottoscritto con firma digitale o con un altro tipo di firma elettronica qualificata, ha l'efficacia [della forma scritta] prevista dall'articolo 2702 del codice civile. L'utilizzo del dispositivo di firma si presume riconducibile al titolare, salvo che sia data prova contraria."

# Certificazione della chiave pubblica



## Problema di distribuzione affidabile delle chiavi pubbliche:

- Quando Alice ottiene la chiave pubblica di Bob (da un sito web, una email, un supporto di registrazione, ecc...), come fa ad essere sicura che sia realmente la chiave pubblica di Bob e non di un altro (es. Trudy) ?

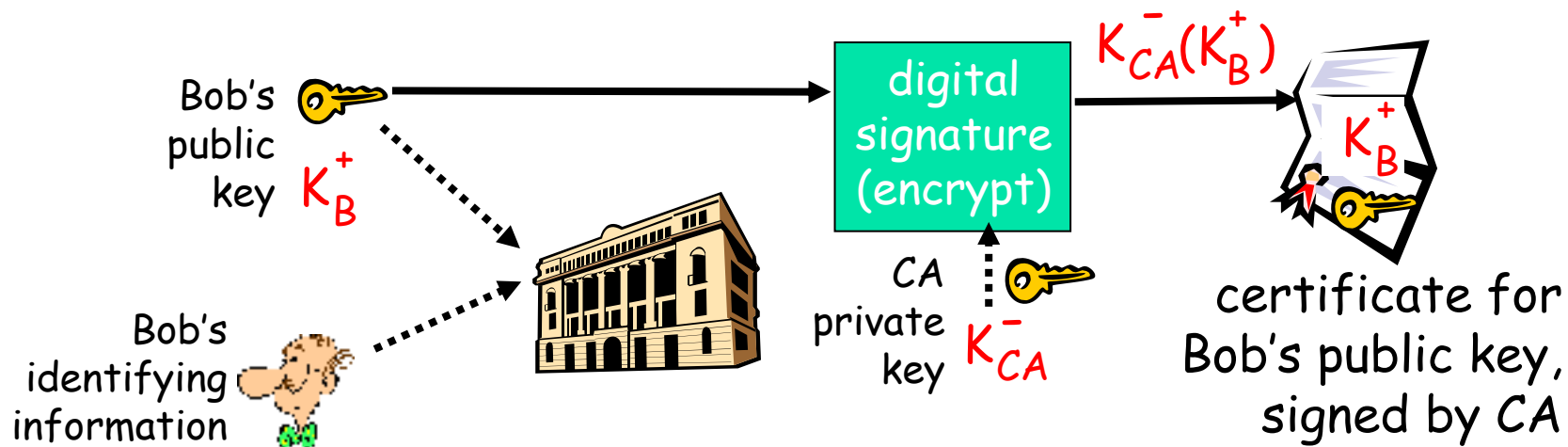
## Soluzione:

- trusted certification authority (CA)
- Una CA è un ente (*trusted third party*), pubblico o privato, abilitato a rilasciare un certificato digitale tramite procedura di certificazione che segue standard internazionali e conforme alla normativa europea e nazionale in materia

# Certification Authorities



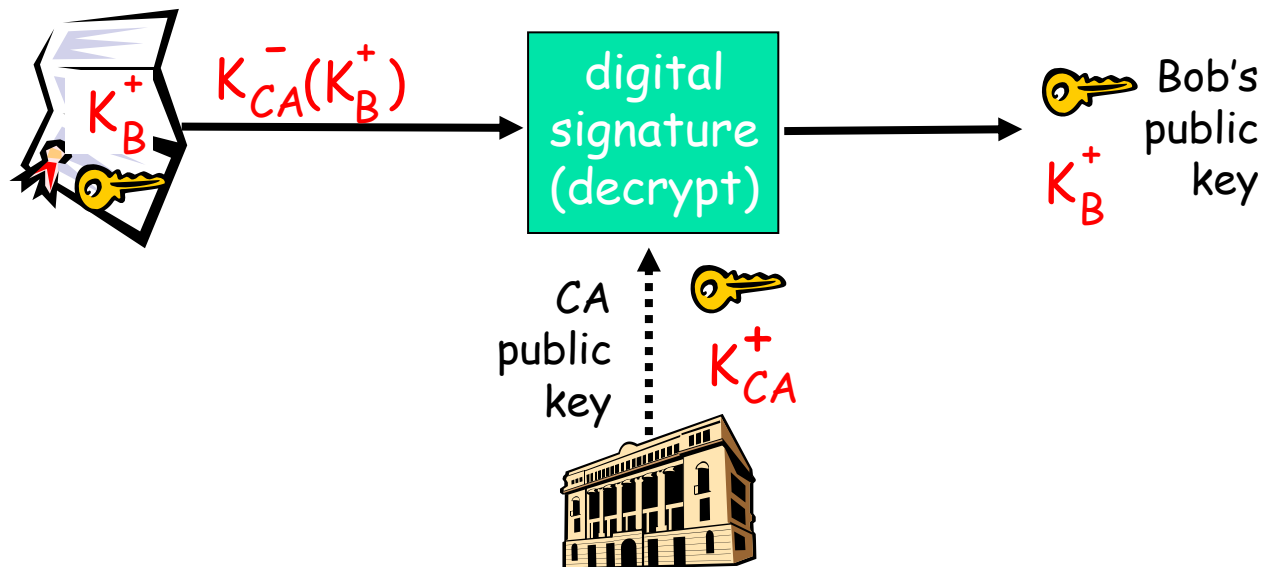
- Una CA associa (in modo sicuro) una chiave pubblica ad una particolare entità E
- E registra la sua chiave pubblica presso la CA
  - A tal fine, E fornisce la sua “prova di identità” alla CA
  - CA crea un “certificato” che collega E alla sua chiave pubblica
  - Un certificato contiene la chiave pubblica di E firmata digitalmente dalla CA: CA afferma che “Questa è la chiave pubblica di E”



# Certification Authorities



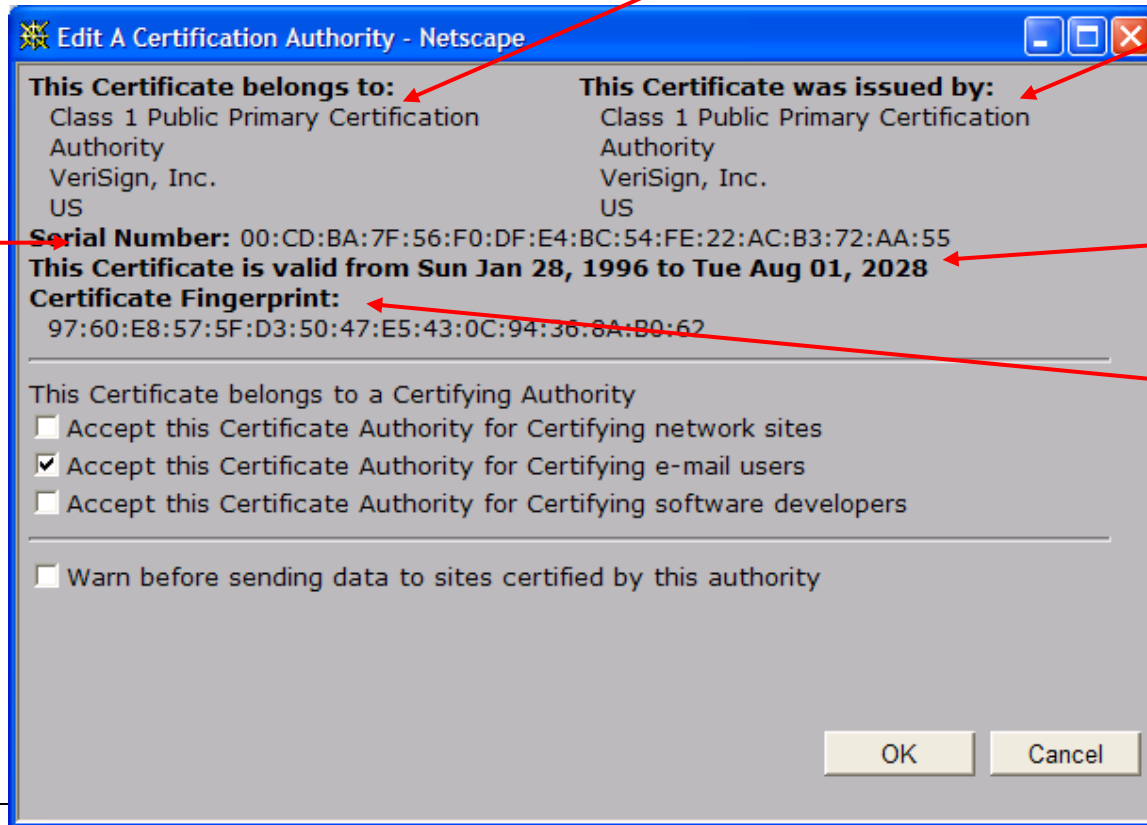
- quando Alice desidera la chiave pubblica di Bob:
  - ottiene il certificato di Bob (da Bob o da altri)
  - applica la chiave pubblica della CA al certificato di Bob, ed ottiene la chiave pubblica di Bob





# Un certificato contiene:

- Un numero di serie (unico tra quelli emessi dalla CA)
- Informazioni riguardo al possessore del certificato



- info su chi ha emesso il certificato
- date di validità
- firma digitale di chi ha emesso il certificato

# Autenticazione: protocollo ap1.0



obiettivo: Bob desidera che Alice provi la sua identità

Protocollo ap1.0: Alice dice "Io sono Alice"

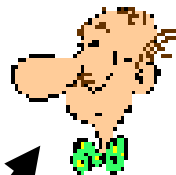




# Protocollo ap1.0: fallimento

Fallimento: in una rete Bob non può "vedere" Alice, perciò Trudy può facilmente fingere di essere Alice

Protocollo ap1.0: Alice dice "Io sono Alice"



"Io sono Alice"



# Autenticazione: protocollo ap2.0

Protocollo ap2.0: Alice dice "Io sono Alice" in un pacchetto IP contenente il proprio indirizzo IP come indirizzo IP sorgente

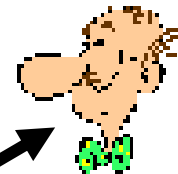




# Protocollo ap2.0: fallimento

Fallimento: Trudy può creare un pacchetto IP facendo lo "spoofing" dell'indirizzo IP di Alice

Protocollo ap2.0: Alice dice "Io sono Alice" in un pacchetto IP contenente il proprio indirizzo IP come indirizzo IP sorgente

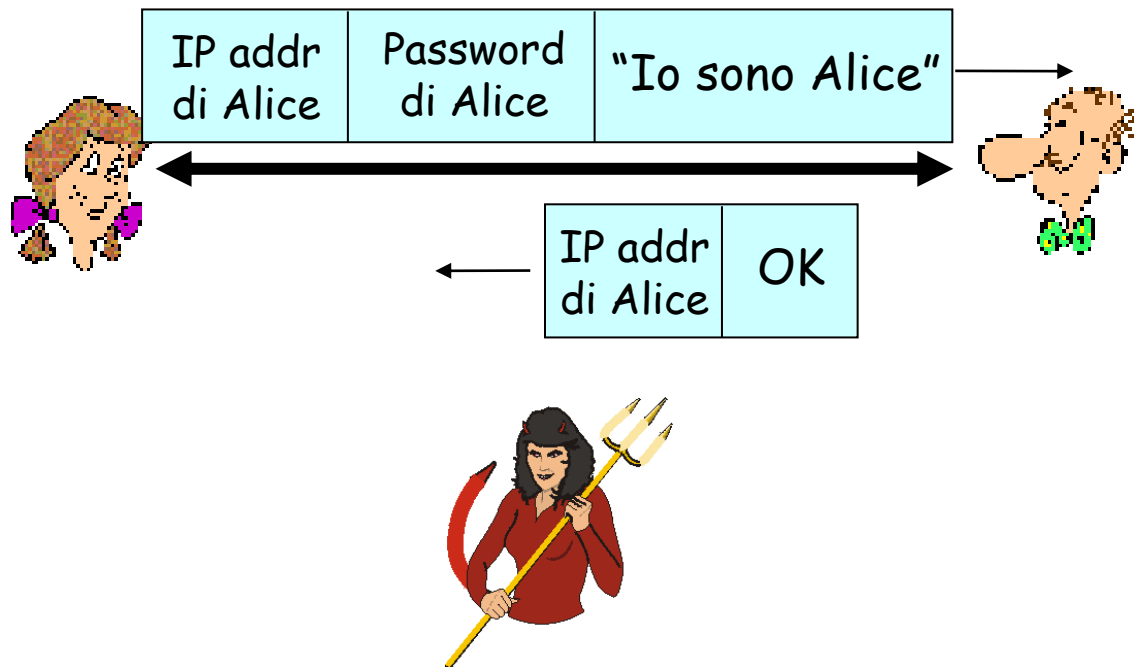


Indirizzo IP di Alice	"Io sono Alice"
-----------------------	-----------------



# Autenticazione: protocollo ap3.0

Protocollo ap3.0: Alice dice "Io sono Alice" e manda la sua password segreta per provare la sua affermazione

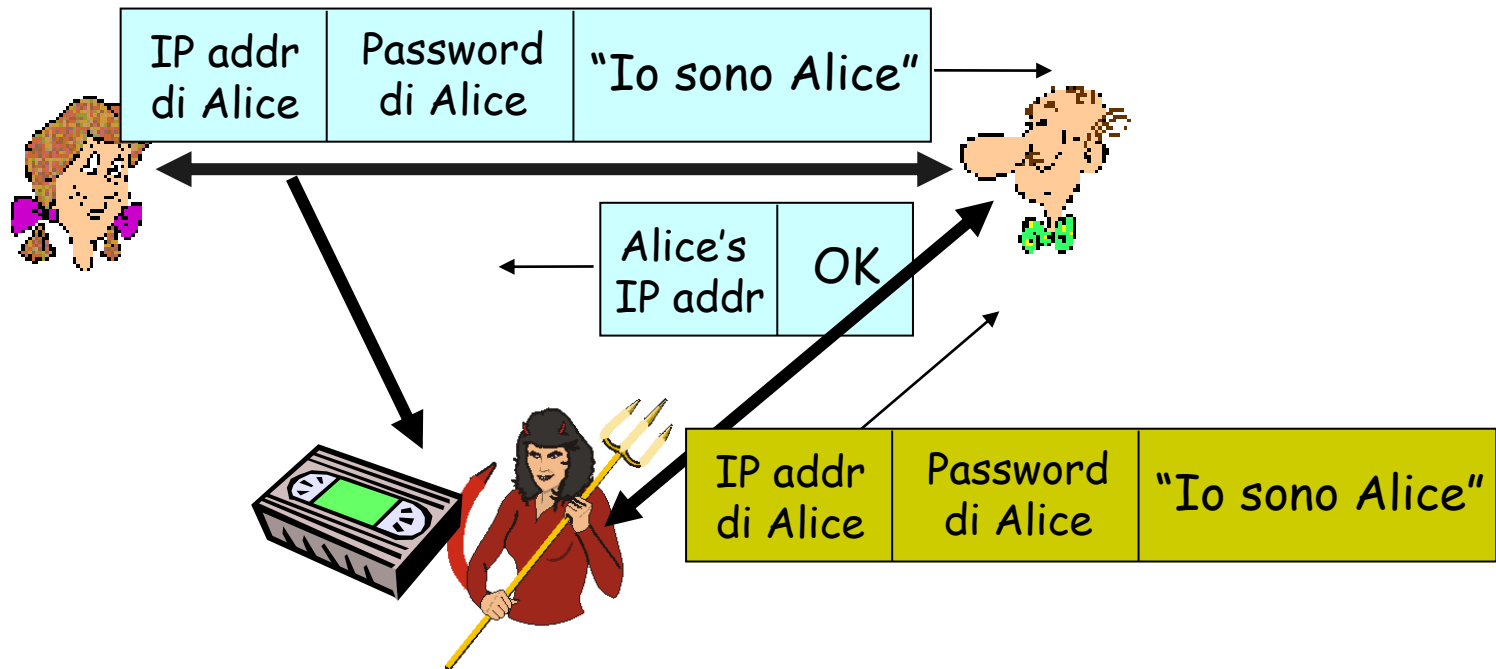




# Protocollo ap3.0: fallimento

Protocollo ap3.0: Alice dice "Io sono Alice" e manda la sua password segreta per provare la sua affermazione

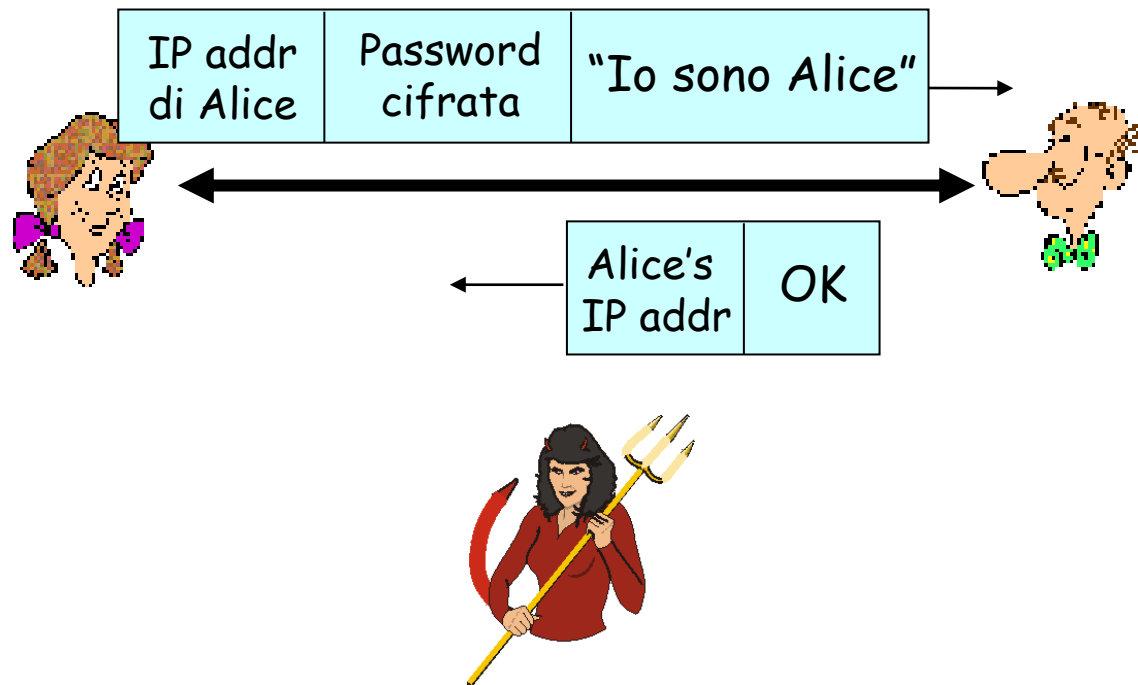
*Attacco playback:* Trudy registra il pacchetto di autenticazione di Alice e successivamente lo rimanda a Bob





# Autenticazione: protocollo ap3.1

Protocollo ap3.1: Alice dice "Io sono Alice" e manda la sua password segreta **cifrata** per provare la sua affermazione

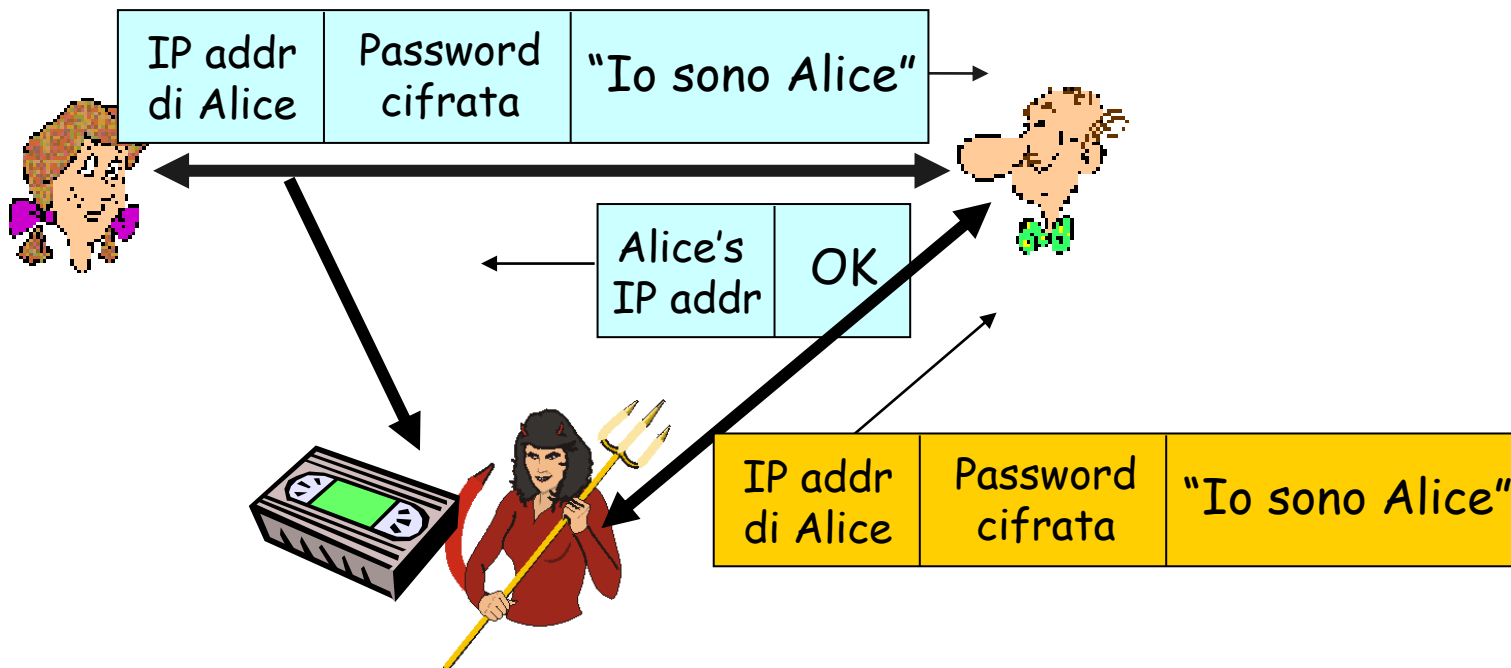




# Protocollo ap3.1: fallimento

Protocollo ap3.1: Alice dice "Io sono Alice" e manda la sua password segreta **cifrata** per provare la sua affermazione

**Attacco playback**: Trudy registra il pacchetto di autenticazione di Alice e successivamente lo rimanda a Bob (**funziona ancora!**)



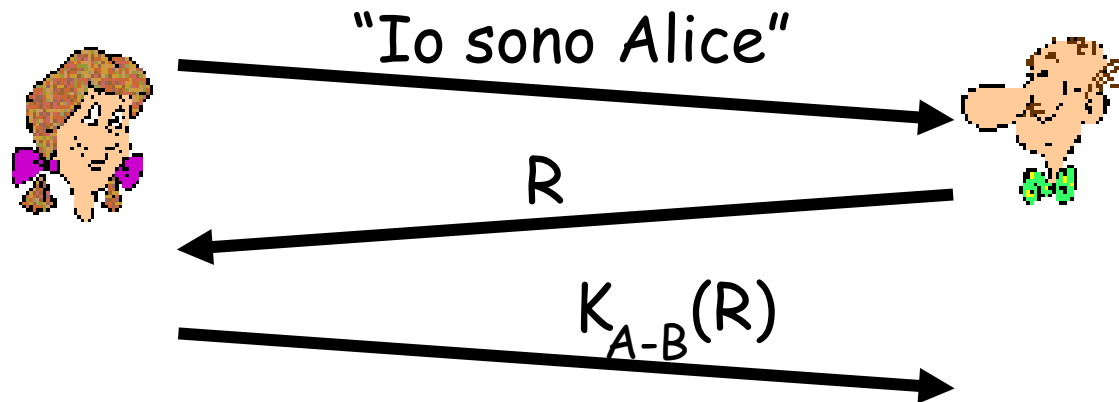


# Autenticazione: protocollo ap4.0

Obiettivo: impedire l'attacco playback

Soluzione: Nonce = numero  $R$  usato *una sola volta*

ap4.0: Bob invia ad Alice un **nonce**,  $R$   
Alice deve restituire  $R$ , cifrato con la sua chiave segreta



Problema da risolvere: quello della crittografia a chiave simmetrica,  
cioè lo scambio delle chiavi



# Autenticazione: protocollo ap5.0

ap5.0: Bob invia ad Alice un **nonce**,  $R$   
Alice deve restituire  $R$ , cifrato con la sua chiave privata



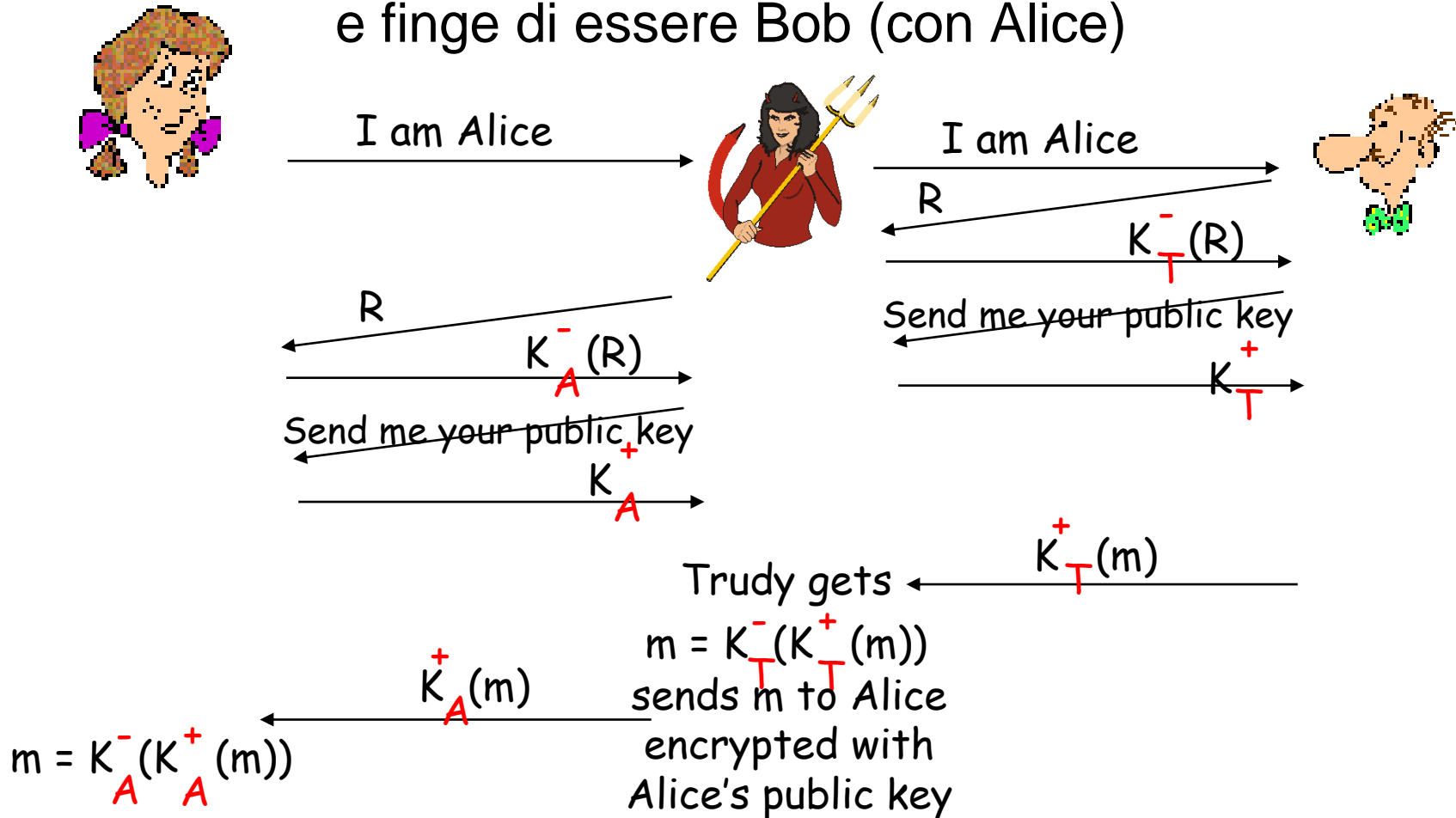
Bob calcola  
 $K_A^+ (K_A^-(R))$   
e sa che solo Alice  
possiede la chiave  
privata tale che  
 $K_A^+ (K_A^-(R)) = R$



# Protocollo ap5.0: difetto di sicurezza

## Attacco "man in the middle":

Trudy finge di essere Alice (con Bob)  
e finge di essere Bob (con Alice)

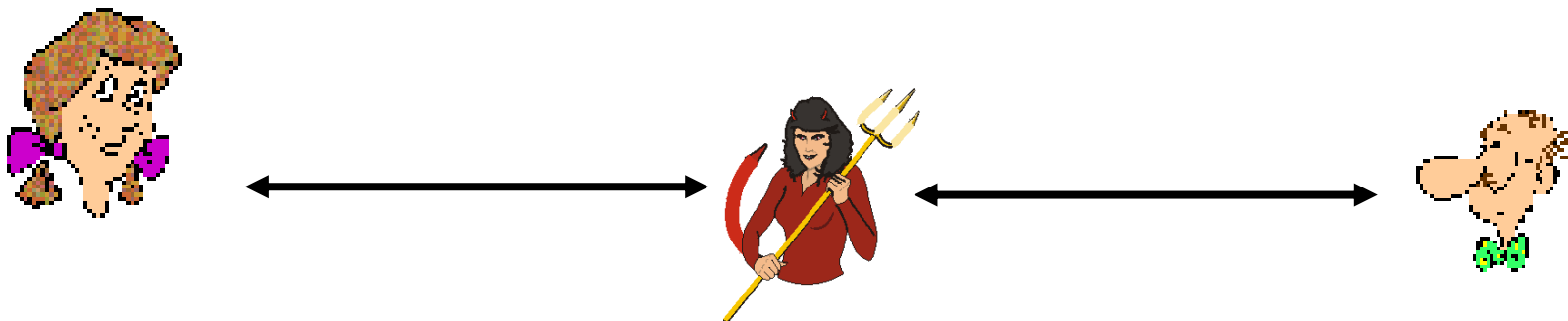


## Protocollo ap5.0: difetto di sicurezza (2)



### Attacco “man in the middle”:

Trudy finge di essere Alice (con Bob)  
e finge di essere Bob (con Alice)



Difficile da rilevare:

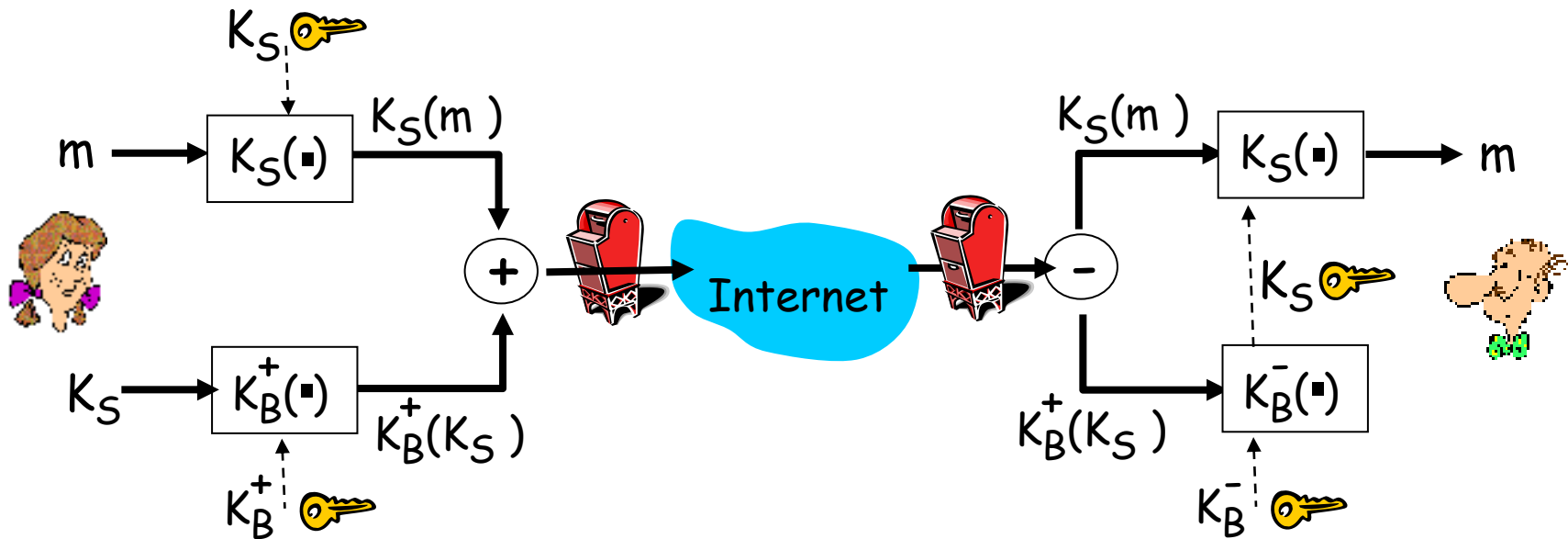
- ❑ Bob riceve tutto quello che Alice gli manda, e viceversa
- ❑ il problema è che anche Trudy riceve tutti i messaggi!

Il difetto di sicurezza di ap5.0 è legato alla distribuzione delle chiavi pubbliche



# E-mail sicura: caso 1 – lato Alice

Alice vuole inviare un messaggio confidenziale  $m$  a Bob



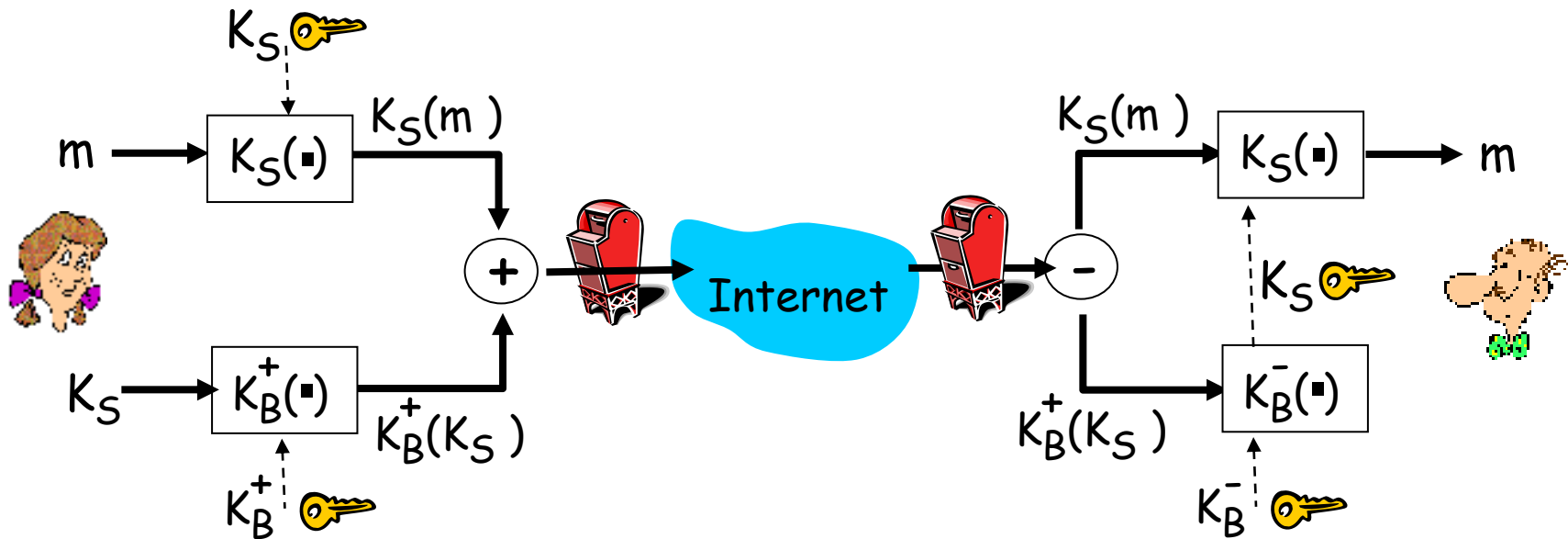
**Alice:**

- genera una chiave *simmetrica* privata  $K_S$
- cifra il messaggio  $m$  con  $K_S$
- cifra  $K_S$  con la chiave pubblica di Bob
- invia sia  $K_S(m)$  che  $K_B^+(K_S)$  a Bob



# E-mail sicura: caso 1 – lato Bob

Alice vuole inviare un messaggio confidenziale  $m$  a Bob



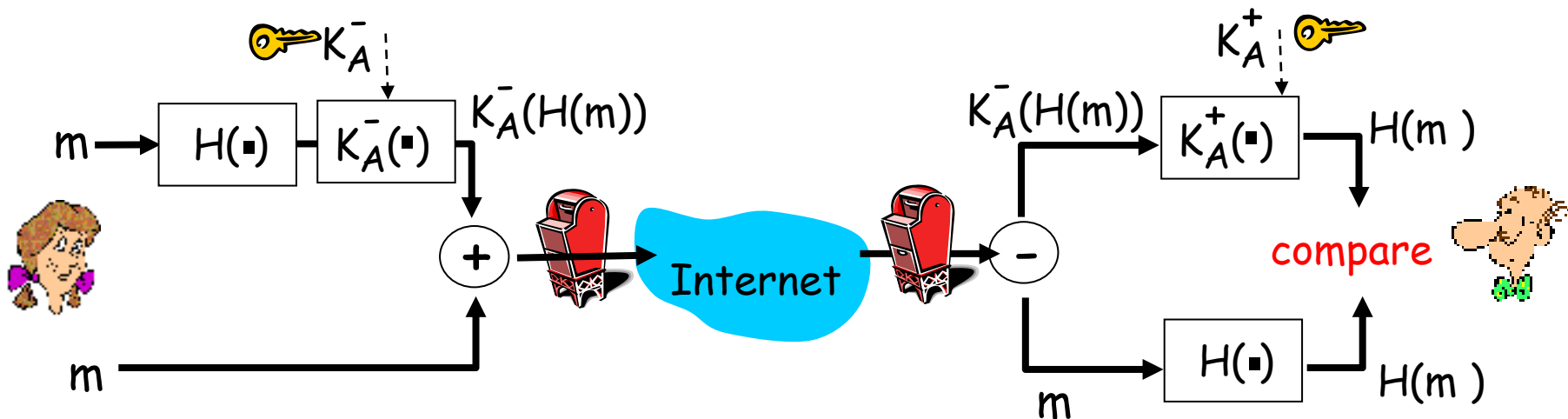
**Bob:**

- usa la sua chiave privata per decifrare e recuperare  $K_S$
- usa  $K_S$  per decifrare  $K_S(m)$  per recuperare  $m$



# E-mail sicura: caso 2

Alice vuole che Bob possa essere sicuro della identità del mittente e della integrità del messaggio ricevuto

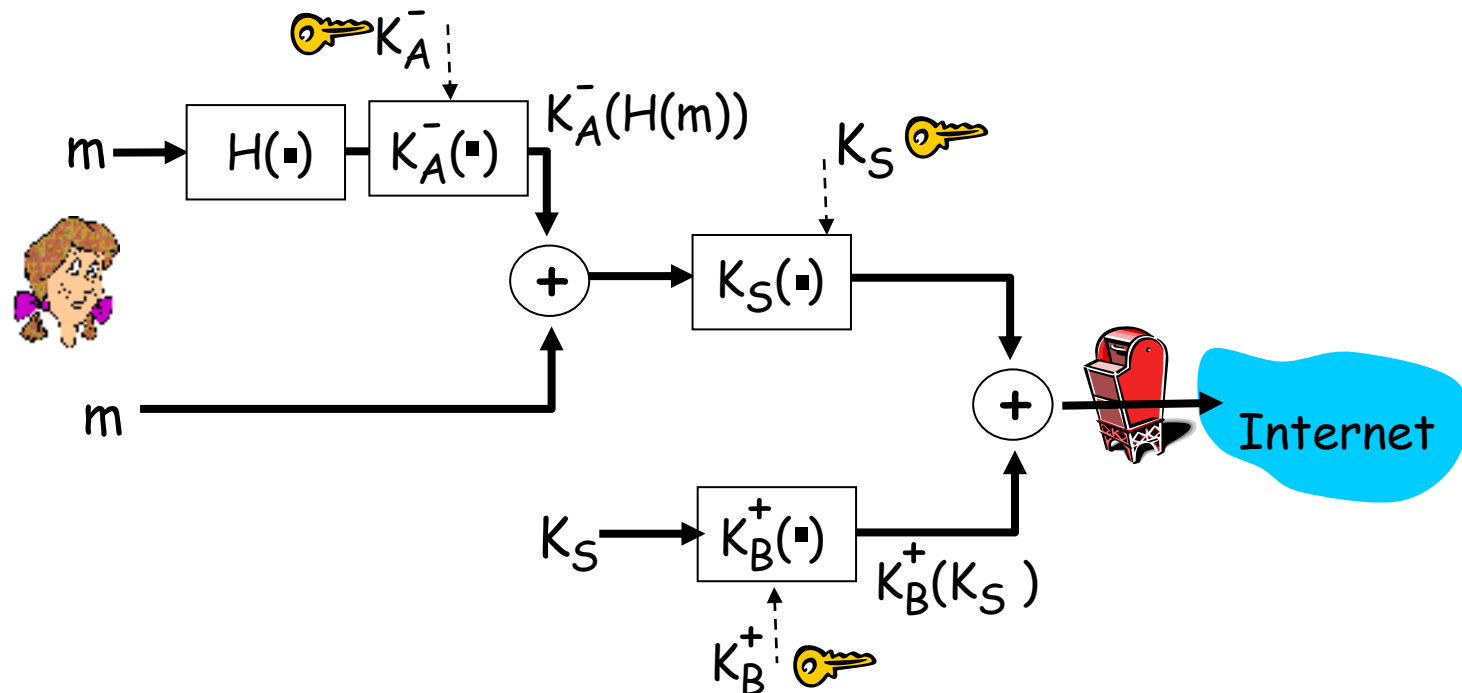


- Alice appone la sua firma digitale al messaggio
- invia il messaggio (in chiaro) e la firma digitale



# E-mail sicura: caso 3

Alice vuole inviare un messaggio confidenziale a Bob e vuole che Bob possa essere sicuro della identità del mittente e della integrità del messaggio ricevuto



**Alice usa 3 chiavi:** la sua chiave privata, la chiave pubblica di Bob, la chiave simmetrica appena generata  $K_S$