Introduzione alla Crittografia per Sistemi

Chiavi Pubbliche e Private
Chiavi Segrete
MAC e Firma Digitale
GPG/PGP
Firma per Distribuzione packages

Sistemi Operativi
Virtualizzazione e Integrazione di sistemi

Crittografia - elenco argomenti

- Codifica e Decodifica.
  - Crittografia a chiave segreta (simmetrica)
    - DES, Triple DES, AES.
  - Crittografia a chiave pubblica (asimmetrica)
    - RSA
- Funzioni Hash crittografiche.
  - MD5 , SHA-1, SHA-2 (SHA512), SHA-3
- Applicazioni
  - Tecnica MAC per Integrità del messaggio
  - Firma Digitale
  - Firma GPG per distribuzione packages apt
- Certificazione della chiave pubblica.

Crittografia

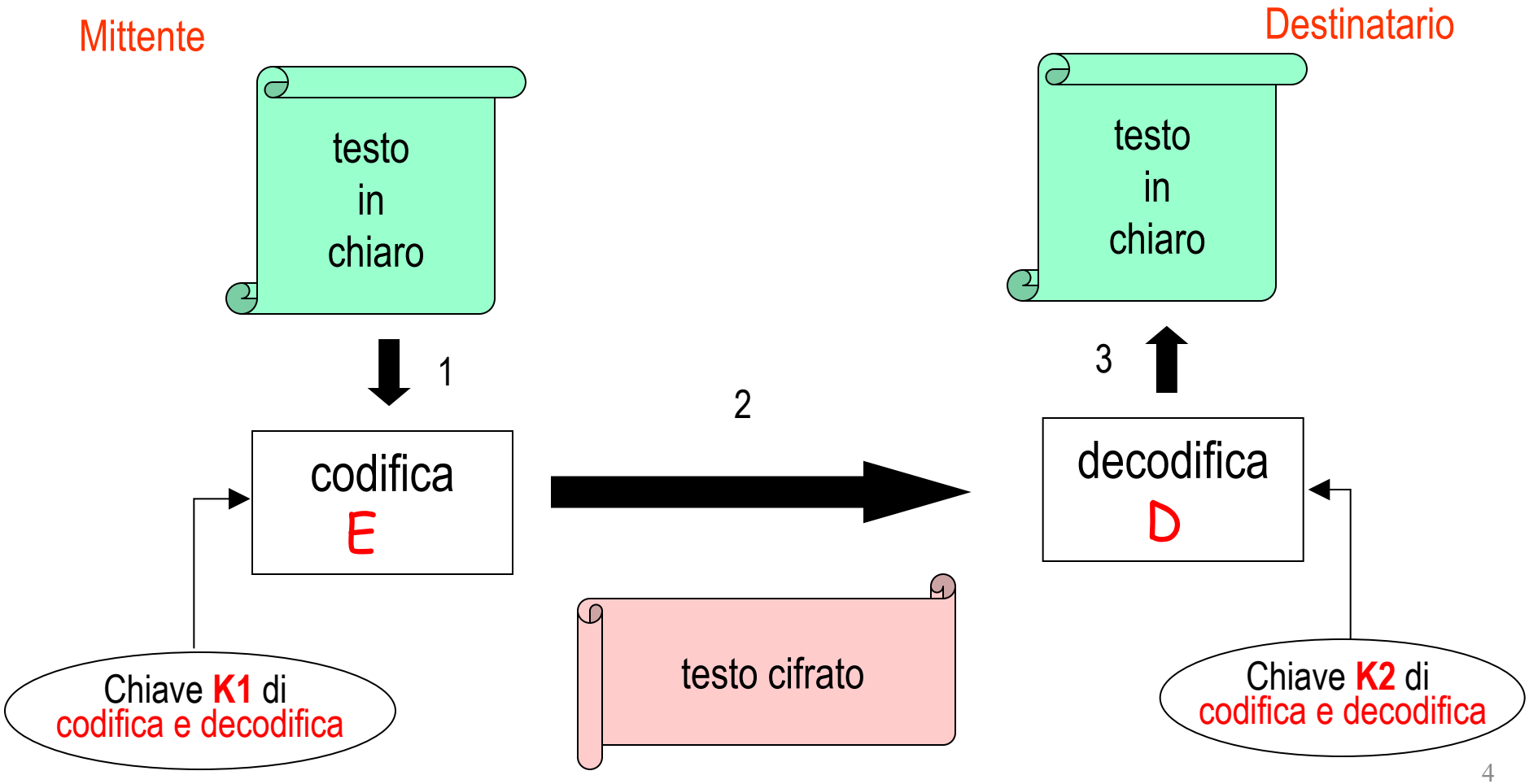
- Scienza antichissima: codificare e decodificare informazione. Tracce risalenti all'epoca di Sparta

- Codificare: testo in chiaro → testo codificato
- Decodificare: testo codificato → testo in chiaro
- Ambedue basate su: **algoritmo** e **chiave**
  - *Es: "Shiftare" di k una stringa*
- Algoritmo pubblico!
- Sicurezza data da:
  1. segretezza della chiave
  2. robustezza dell'algoritmo

Codifica e decodifica

Le funzioni di codifica e decodifica sono noti, la chiave di cifratura no

E_{K1} = encoding con chiave K1
D_{K2} = decoding con chiave K2
t.c D_{K2} (E_{K1} (testo)) == testo



Crittografia a chiave segreta (simmetrica)

- **Medesima chiave per codifica e decodifica.**
- **Richiede che solo mittente e destinatario conoscano la chiave segreta.**

Le funzioni di codifica e decodifica sono note, la chiave è nota solo ai 2 che comunicano

- Segretezza, autenticazione, integrità, ma non "Non repudiabilità".
- Proprietà dipendono dalla segretezza della chiave (che è nota solo a 2).

\+ Di solito DES (Data Encryption Standard), **Triple-DES**, **AES** (Advanced Encryption Standard, noto come Algoritmo di Rijndael) e altri.

\+ DES non più sicuro.

\+ usano chiavi di varie dimensioni 64,128,168, 256 bit, ...

\+ al crescere della dimensione aumenta la sicurezza.

\+ abbastanza veloci.

\- Occorre distribuire una chiave a ciascuna coppia di utenti.

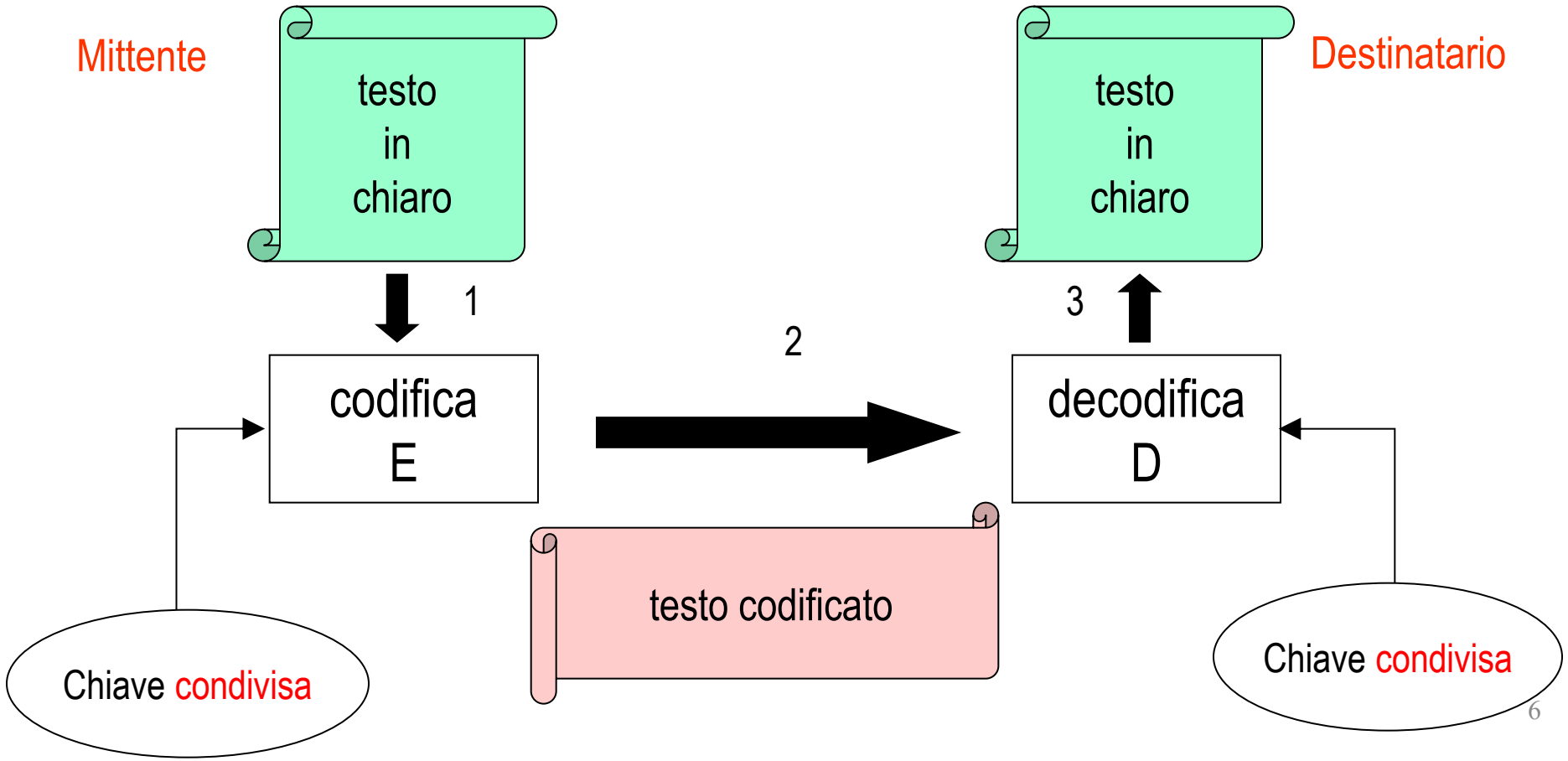
\- Per n utenti servono n*(n-1)/2 chiavi diverse

Proprietà codifica a chiave segreta

Segretezza, Autenticazione, Integrità

E_K = encoding con chiave K
D_K = decoding con chiave K
t.c D_K (E_K (testo)) == testo

manca Non Repudiabilità



Pro e contro della Crittografia a chiave segreta

PRO

- Gli algoritmi (Triplo DES, AES e altri) possono essere implementati in modo efficiente

CONTRO

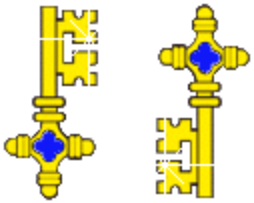
- Le parti che comunicano devono prima scambiarsi la chiave in modo sicuro
- Questo è un punto critico per il quale non esistono al momento soluzioni generali affidabili

Crittografia a chiave pubblica (asimmetrica)

Le funzioni di codifica/decodifica sono note, la chiave pubblica è nota, la chiave privata è segreta

- Una chiave per codifica, un'altra per decodifica
- Mittente e destinatario NON condividono una chiave segreta.

- **Ogni utente ha una coppia di chiavi**
  - **chiave privata**: segreto personale da custodire
  - **chiave pubblica**: informazione da diffondere a tutti
- **Entrambe usabili per codificare o decodificare**, e soprattutto
- **il testo codificato con una chiave può essere decodificato solo con l'altra chiave.**



- Di solito algoritmo **RSA** (*dagli autori Rivest, Shamir e Adelson*)
- Usano chiavi di 1024-2048 bit
- Sotto 2048 non consigliabili (meno sicuri)
- Sono un po' lenti
+ Segretezza, autenticazione, integrità…

Crittografia asimmetrica
Proprietà funzioni crittografiche e chiavi

La funzione di cifratura è denotate con il simbolo K seguito da un apice + o - a indicare rispettivamente cifratura con chiave pubblica o privata.

Indichiamo con K^+(testo) la cifratura di un testo mediante chiave pubblica di un utente.
Indichiamo con K^-(testo) la cifratura mediante chiave privata di un utente.

Allora

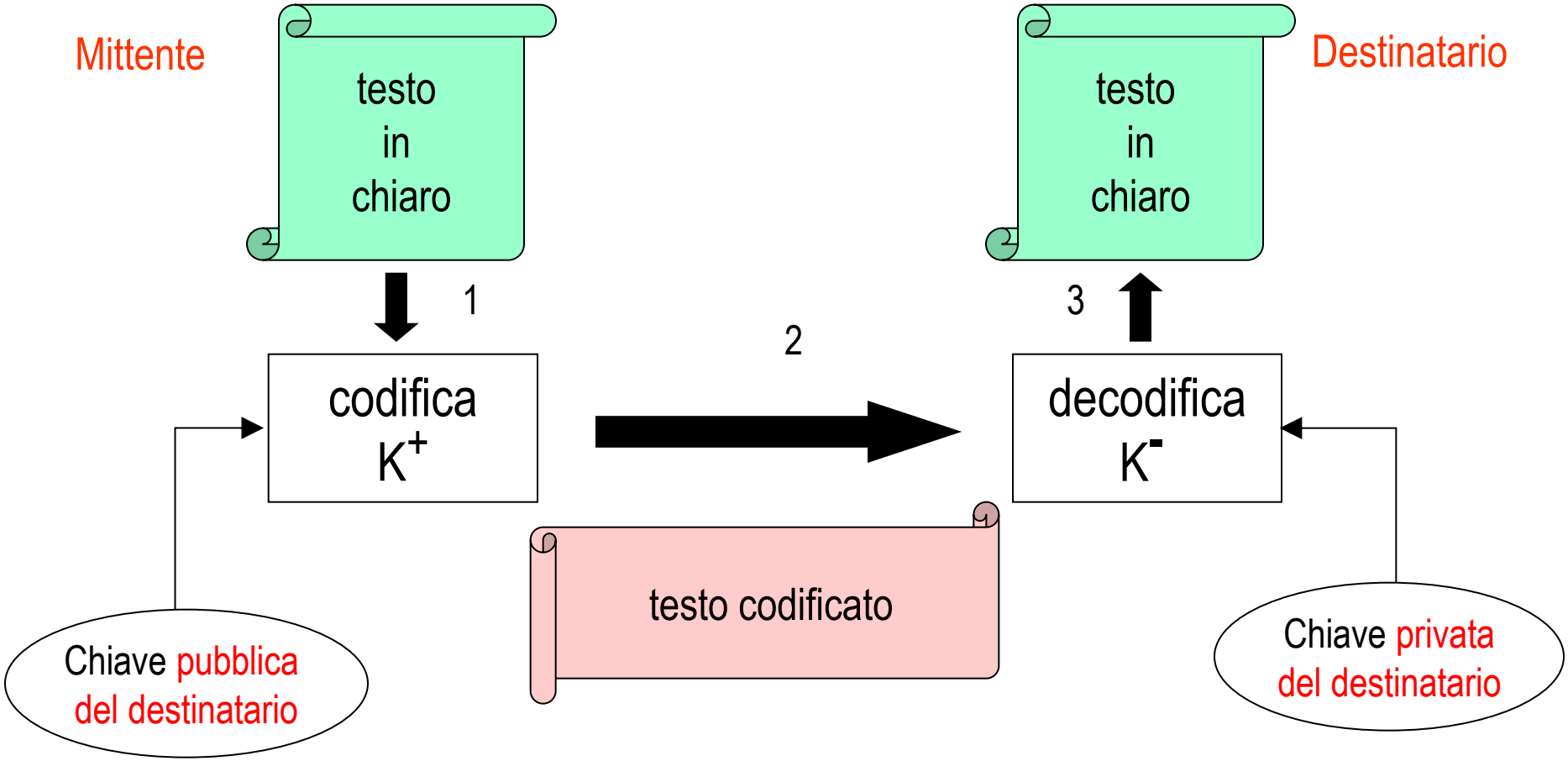
$$K^+ (K^- (\text{testo})) == \text{testo}$$
$$K^- (K^+ (\text{testo})) == \text{testo}$$

Dalla chiave pubblica K^+, deve essere impossibile calcolare la chiave privata K^-.

Dal testo codificato deve essere impossibile trovare il testo in chiaro senza conoscere la chiave.

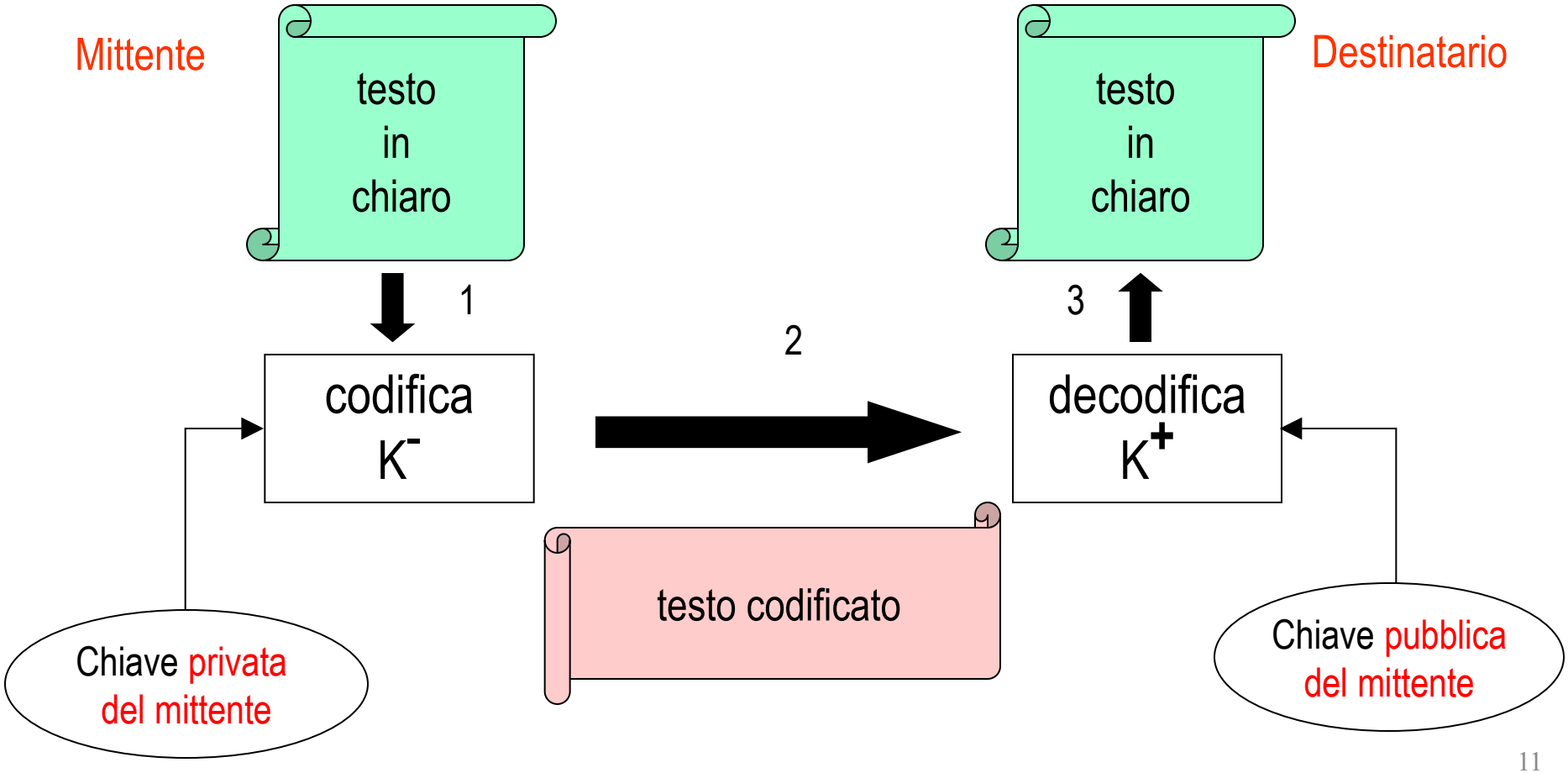
Proprietà codifica a chiavi asimmetriche

Segretezza



Proprietà codifica a chiavi asimmetriche

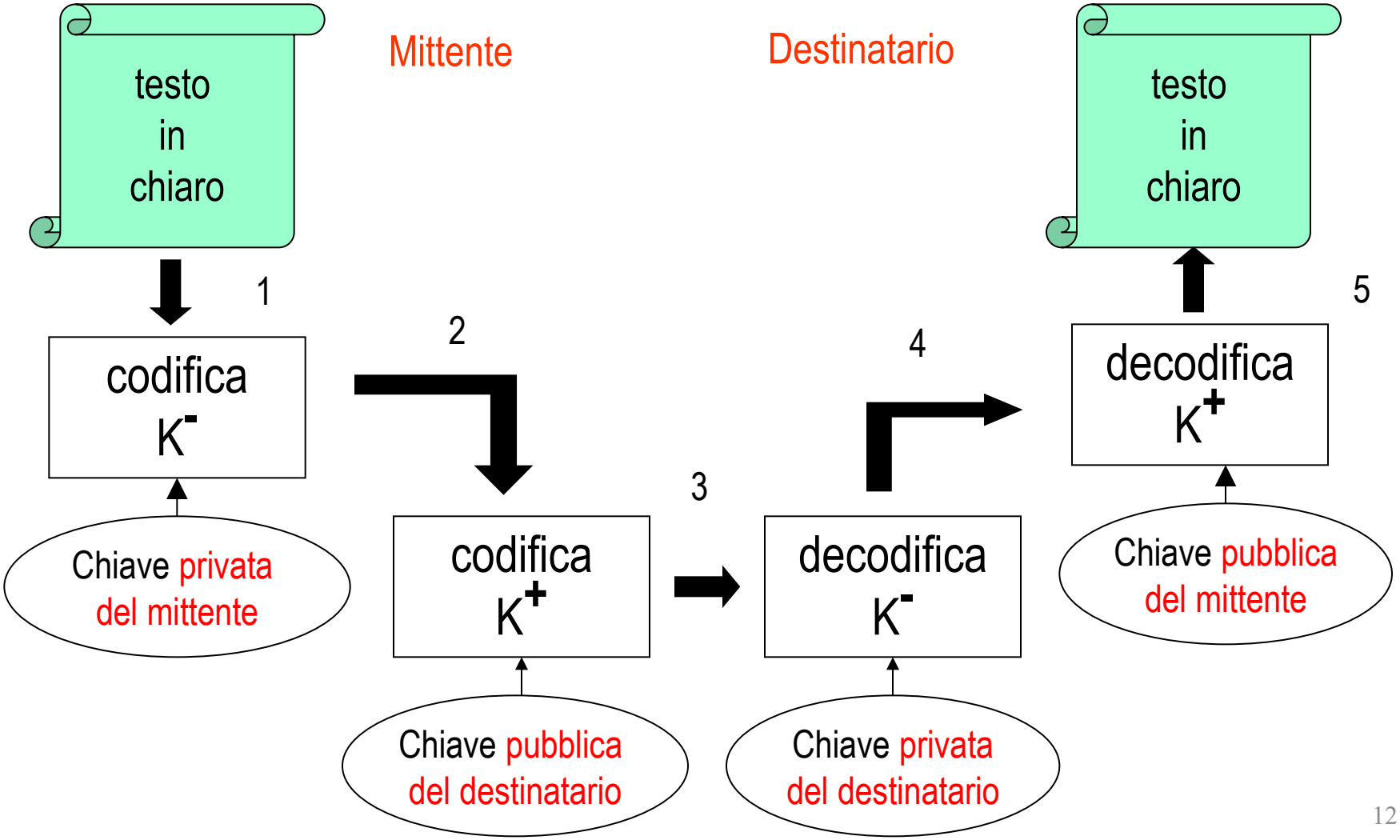
Autenticazione, integrità e Non Repudiabilità



Proprietà codifica a chiavi asimmetriche

Le quattro insieme

Segretezza, Autenticazione, integrità e Non Repudiabilità



Pro e contro della Crittografia a chiave pubblica

PRO

- Non occorre scambiarsi chiavi segrete: le uniche chiavi che è necessario comunicare sono quelle pubbliche (che per definizione sono pubbliche!)

CONTRO

- Gli algoritmi di crittografia asimmetrica sono più lenti di quelli per crittografia simmetrica
- Come facciamo ad essere certi che la chiave pubblica, ad es., di Alice è *veramente di Alice?*

Funzioni Hash Crittografiche

Sono anche dette **one-way hash** o anche **funzioni digest** cioè funzioni riassunto.

Def: una **Funzione hash crittografica MD( )**, message digest, è una funzione che:

1. prende in input una stringa (**messaggio**) **m** , a lunghezza variabile
2. e produce una stringa (**digest**) a lunghezza fissa MD(m) che possiamo pensare come un riassunto digitale del messaggio.
3. e che soddisfa la seguente proprietà di **Non reversibilità**
   - Deve essere computazionalmente impossibile trovare due messaggi x e y tali che MD(x) = MD(y)

     Tale proprietà può essere espressa anche così:
     - dato y = MD(x), con x sconosciuta, allora è impossibile determinare x.
4. Se due messaggi m1 e m2 differiscono di poco, i due digest dovrebbero essere assai diversi.
   - Non garantisce che 2 messaggi molto diversi abbiano sempre digest diversi.

NB: le funzioni hash crittografiche non richiedono chiavi di cifrature.

Differenze

- Le funzioni **Hash crittografiche** differiscono dalle funzioni di **codifica e decodifica** in quanto
  i. le hash non consentono una decodifica, cioè non sono invertibili.
  ii. inoltre le Hash non prevedono l'uso di una chiave.
  iii. Infine, le funzioni di codifica e decodifica non restituiscono messaggi a lunghezza fissa.

- Le funzioni **Hash crittografiche** differiscono dal **checksum di Internet** poiché
  - le checksum Internet soddisfano le proprietà 1, 2 ma non la 3 (cioè la non reversibilità).
  - Le checksum di Internet sono funzioni hash ma non sono crittografiche nel senso che per favorire la velocità di calcolo sono semplici e sarebbe perciò possible trovare due messaggi che hanno stesso checksum.

Funzioni Hash crittografiche più importanti

- **MD5** (Message Digest 5)
  - Sviluppato da Ronald Rivest nel 1991
  - Genera un digest lungo 128 bit
  - **Oggi non è ritenuto abbastanza sicuro**
- **SHA** (Secure Hash Algorithm)
  - Famiglia di funzioni hash crittografiche sviluppate dalla NSA a partire dal 1993.
  - **SHA-1** genera un digest lungo 160 bit ma **è poco sicuro**.
  - **SHA-2** indica genericamente le successive versioni, che si distinguono per numero di bit del digest.
    - SHA-224, SHA-256, SHA-384, **SHA-512**
    - SHA-x genera un digest lungo x bit.
    - Ad esempio, SHA-512 genera un digest lungo 512 bit.
    - La sicurezza e (purtroppo) il tempo necessario a calcolare il digest, crescono con la dimensione del digest.
    - Si preferisce usare SHA-256 o **SHA-512**
  - **SHA-3 (Secure Hash Algorithm 3)**
    - recente, sviluppato da NIST, ulteriormente sicuro. SHA3-512.

Integrità di messaggi e Firma Digitale

tecnica MAC - Message Authentication Code

- La tecnica detta **Message Autentication Code** consente di
  - **autenticare** (parzialmente) **l'origine** di un messaggio (riconoscere il mittente)
  - e **garantire l'integrità** del messaggio stesso
- facendo uso conbinato della crittografia (a chiave pubblica o a chiave segreta) e delle funzioni hash crittografiche.

- **Se tra mittente e destinatario esiste una chiave segreta per crittografia simmetrica**,
  - allora si può usare funzione Hash e crittografia simmetrica.
  - **Questa è la tecnica MAC vera e propria.**

- **Se invece il mittente è dotato di propria chiave pubblica** (coppia di chiavi pubblica e privata),
  - allora si può usare funzione Hash e crittografia asimmetrica.
  - **Questa è la tecnica di Firma Digitale.**

MAC -Message Autentication Code
Spedizione

- Supponiamo di dover spedire un messaggio m.
- La tecnica della Firma Digitale del messaggio mediante MAC, stabilisce di eseguire le seguenti operazioni:

In spedizione del messaggio **m**, occorre

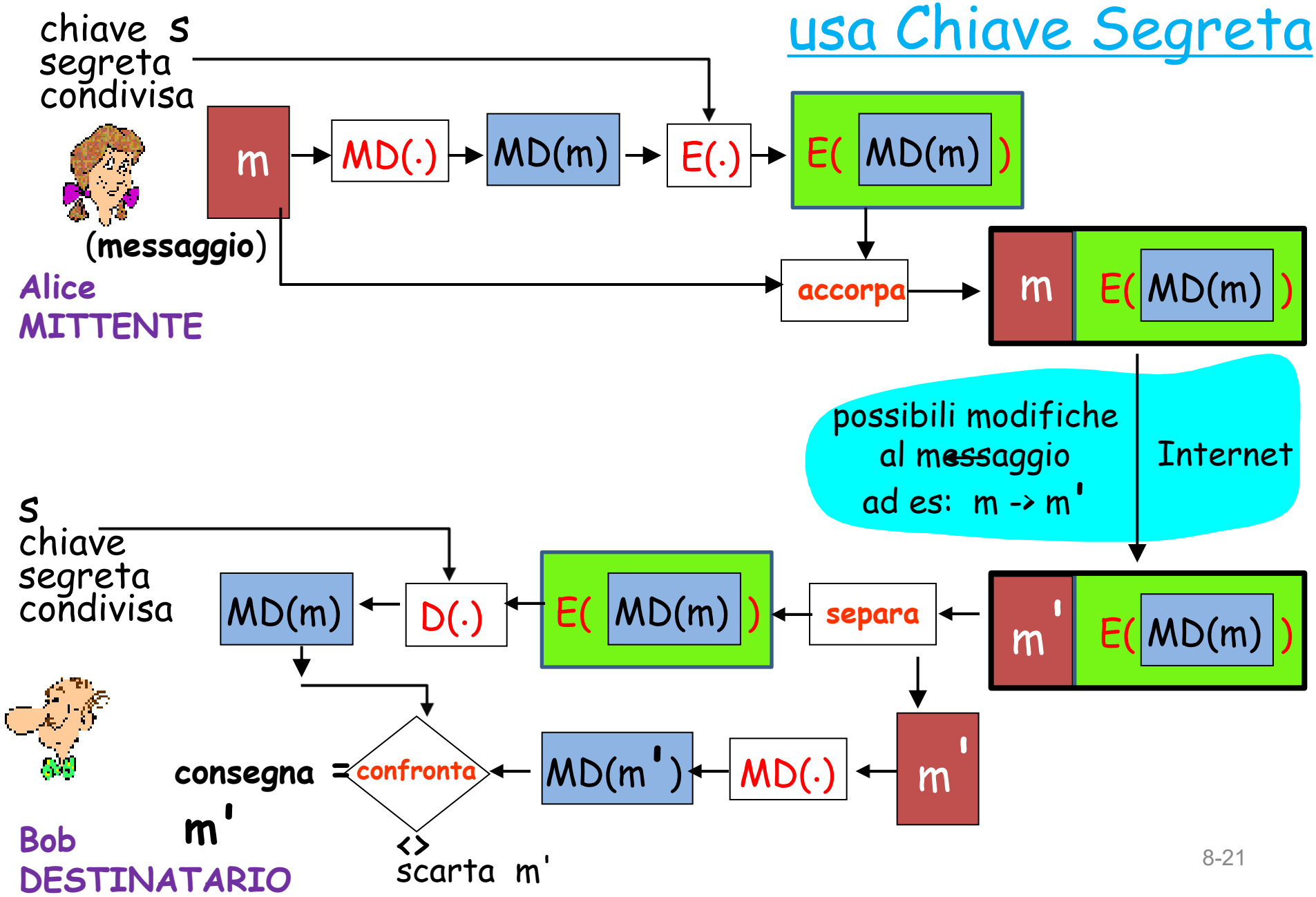
- applicare al messaggio una Funzione Hash Crittografica per creare una stringa che sia un riassunto (digest) del messaggio stesso. Otteniamo MD(m)
- Codificare il riassunto (digest) del messaggio
  - o con la codifica a chiave segreta, usando la chiave segreta K condivisa tra mittente e destinatario. Otteniamo $MAC=E_k(MD(m))$
  - oppure con la codifica a chiave pubblica, utilizzando la chiave privata del mittente. Otteniamo $MAC=\mathbf{K}^-(MD(m))$
- Il riassunto così codificato viene detto Codice di Autenticazione del Messaggio (Message Authentication Code, MAC).
- Il MAC del messaggio viene concatenato al messaggio stesso.
- Viene spedito il nuovo messaggio formato dal messaggio unito al suo MAC **(m,MAC)** cioè $(m,\mathbf{K}^-(MD(m)))$ oppure $(m, E_k(MD(m)))$.

MAC -Message Autentication Code
Ricezione

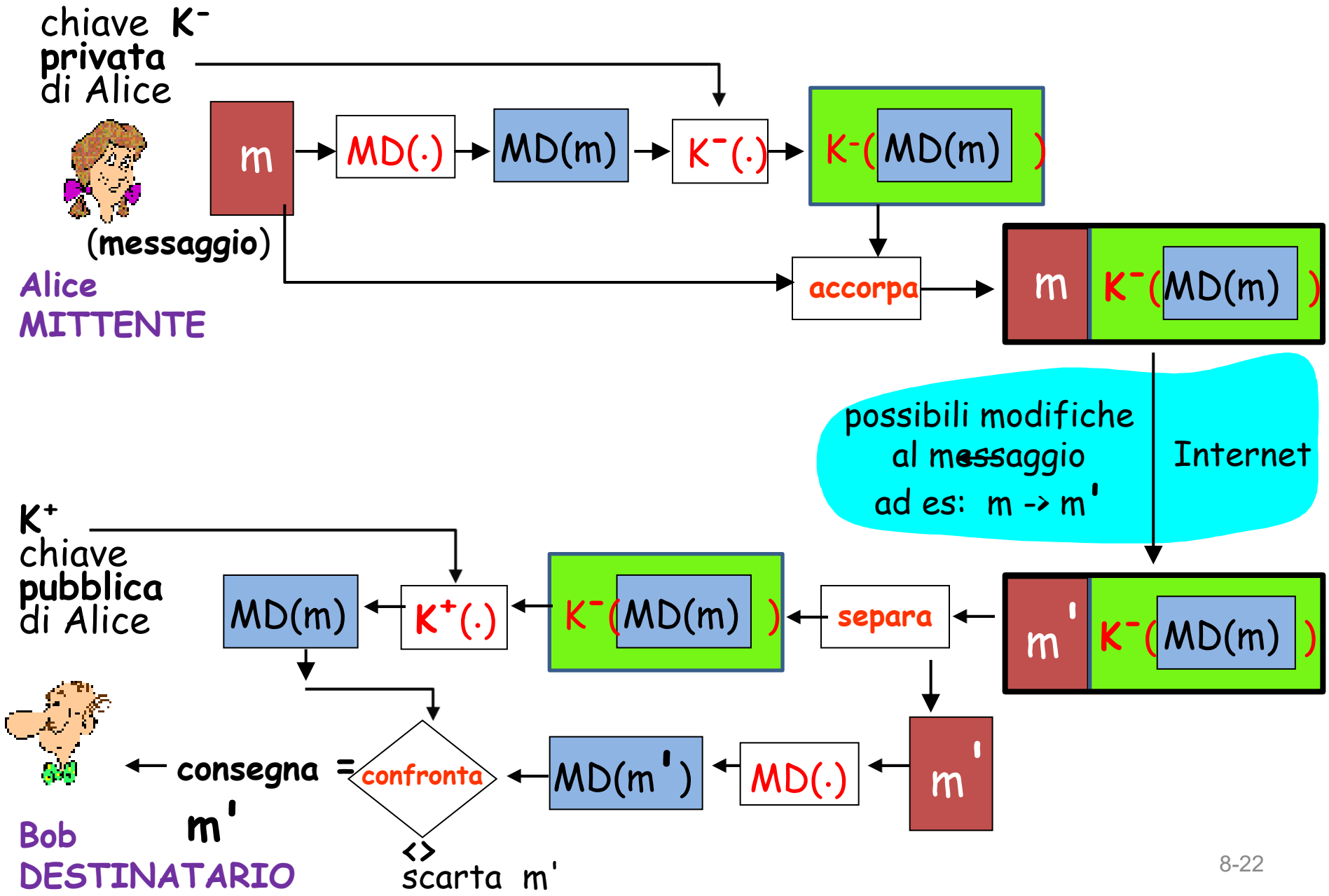
Alla **ricezione del messaggio con il MAC** (m',MAC), occorre:

- separare il messaggio m' in chiaro dal MAC del messaggio.
- Decodificare il MAC
  - o con la codifica a chiave pubblica, utilizzando la chiave pubblica del mittente. Ottengo $K^+(K^-(MD(m)))=MD(m)$.
  - oppure con la codifica a chiave segreta utilizzando la chiave segreta condivisa tra mittente e destinatario. Ottengo $D_k(E_k(MD(m))) = MD(m)$.
- **Se la decodifica produce errori** significa che il mittente non è quello, dichiarato nel messaggio, di cui conosciamo la chiave. **Fine**.
- Invece, se la decodifica va a buon fine, quello che otteniamo è il digest del messaggio originale **MD(m)**.
- Ora calcoliamo il digest del messaggio m' ricevuto **MD(m')**
- Infine confrontiamo i due digest,
  - quello ricavato decodificando il MAC **MD(m)**
  - e quello ricalcolato dal messaggio in chiaro ricevuto **MD(m')**
- Se i due digest sono uguali **MD(m)== MD(m')** allora il messaggio ricevuto m' è l'originale ed è stato mandato dal mittente dichiarato.

Codice di autenticazione dei messaggi - MAC usa Chiave Segreta



Firma Digitale - usa Chiave Pubblica



Differenze tra MAC, Firma Digitale e Codifica/Decodifica

- La tecnica **MAC** è applicabile solo se mittente e destinatario conoscono una chiave segreta.
- Quindi il destinatario sa che, se riesce a decodificare il messaggio, questo proviene dall'altro (o da sé stesso).
- Però, il destinatario non può dimostrare, ad osservatori esterni, che l'altro ha spedito il messaggio, potrebbe essere stato lui stesso.

=> **Integrità, autenticazione, mancano segretezza e non repudiabilità.**

- La **firma digitale** è applicabile solo se il mittente ha una chiave pubblica/privata.
- **Chiunque può leggere il messaggio e riconoscere chi è stato a spedirlo e verificarne l'integrità**.

$\Rightarrow$ **Integrità, autenticazione, non repudiabilità, manca segretezza**, c'é broadcast.

- MAC e Firma Digitale sono **più rapide della codifica del documento intero** poiché **NON DEVONO codificare tutto il documento ma solo il digest**.

GPG o PGP signature (firma GPG/PGP)

- La **GPG o PGP signature** (firma GPG) è un meccanismo di firma digitale creata utilizzando il sistema **PGP (Pretty Good Privacy)** che è proprietario. In ambiente Linux si usa il suo equivalente open-source **GPG (GNU Privacy Guard): si dice PGP ma in realtà si intende GPG.**
- Il suo **scopo principale** non è nascondere un messaggio, ma **garantire l'autenticità e l'integrità di un file o pacchetto software.**
- Scenario: il maintainer di un repository vuole distribuire software, ad esempio packages di una distribuzione Linux. Ma ci sono anche dei repository "copie".
- Obiettivo: il maintainer vuole garantire ai suoi utenti che il software che scaricano da un qualche repository è esattamente quello rilasciato da lui e non è stato modificato da altri. In altri termini:
  - Integrità: il software non è stato modificato da altri.
  - Non repudiabiità: il software è stato pubblicato dal mantainer.
- Requisiti: il maintainer possiede una coppia di chiavi Pubblica/Privata.
- **Creazione firma**: Per ciascun package da pubblicare, il maintainer usa la propria chiave private e crea la firma del package [ed altro], e distribuisce insieme package e firma del package.
- **Verifica firma**: L'utente possiede già o scarica la chiave pubblica del maintainer, poi scarica il package e la firma e usa la chiave pubblica per verificare la firma.
- In realtà, per risparmiare CPU, il maintainer non firma ciascun singolo package .deb **bensì firma il file di indice del repository** che contiene, per ciascun package, la versione e il checksum (o meglio message digest fatto con SHA256).

Il maintainer firma con GPG il repository apt 1/3

- Per risparmiare CPU, il maintainer non firma ciascun singolo package .deb bensì firma il file di indice del repository che contiene, per ciascun file di elenco e per ciascun file package, la versione e il checksum (o meglio message digest fatto con SHA512). (Vedere struttura files del repository di Ubuntu 24.04 noble numbat in **https://archive.ubuntu.com/ubuntu/dists/noble/**)

- Il package iputils-ping è di tipo open source, sta nella sezione main del repo.
- **Supponiamo che il maintainer abbia modificato il package iputils-ping** e lo abbia ricompilato per Ubuntu con processore AMD a 64 bit **creando un file**

  pool/main/i/iputils/iputils-ping_20240117-1build1_amd64.deb
- **Cosa deve fare il maintainer per firmare il repository modificato?**

- Calcolare il digest SHA512 del file .deb
- Inserire il nome del file e il suo digest SHA512 nel file elenco dei package di ubuntu noble mail amd64 (file **/ubuntu/dists/noble/main/binary-amd64/Packages**).

- Calcolare il digest SHA512 del file Packages
- Inserire il percorso del file Packages e il suo digest SHA512 nel file elenco degli elenchi di ubuntu noble **/ubuntu/dists/noble/Release**

- Calcolare il digest SHA512 del file Packages **e poi cifrarlo con la chiave privata del maintainer generando così la firma GPG** e mettendo questa firma nel file **/ubuntu/dists/noble/Release.gpg**

Verifica firma GPG e integrità dei files scaricati 1/2

- L'utente vuole scaricare l'ultima versione del package iputils-ping per Ubuntu 24.04 (cioè versione noble) per Amd 64 bit. Sa già che sta in sez. main del repo.
- L'utente già possiede o ha scaricato la chiave pubblica del maintainer. (**come ?**)

1. Download elenco degli elenchi del repository e verifica firma e digest:
   - L'utente scarica i due files da https://archive.ubuntu.com/
   - Elenco degli elenchi **/ubuntu/dists/noble/Release**
   - Firma GPG dell'elenco **/ubuntu/dists/noble/Release.gpg**
   - Verifica la firma GPG decifrandola con la chiave pubblica del maintainer. Se decifra ottiene il message digest SHA512 del file Release originale. Chiamiamolo md0.
   - Calcolo il digest SHA512 del file Release scaricato: se è uguale a md0 il file scaricato Release è originale. OK
2. Download elenco dei packages e verifica digest
   - Cerca nell'elenco Release, ci trova una riga con nome e posizione del file elenco dei packages per sezione main, compilati per architettura AMd64, è **/ubuntu/dists/noble/main/binary-amd64/Packages** e anche il checksum (message digest SHA512) dell'elenco.
   - Scarico il file Packages (se è gzippato lo gunzippo), calcolo il message digest SHA512 del file, lo confronto con quello che c'era nella riga del primo elenco Release. Se i due digest sono uguali allora il file Packages è l'originale. OK

Verifica firma GPG e integrità dei files scaricati 2/2

3. Download package cercato iputils-ping e verifica digest.
   - Cerco nell'elenco Packages il package iputils-ping, trovo alcune righe che lo descrivono, c'è il nome, il percorso da cui scaricarlo

     Package: **iputils-ping**
     Architecture: amd64
     Version: 3:20240117-1build1
     Provides: ping
     Depends: libcap2-bin, libc6 (>= 2.38), libcap2 (>= 1:2.10), libidn2-0 (>= 0.6)
     Filename: **pool/main/i/iputils/iputils-ping_20240117-1build1_amd64.deb**
     Size: 44254
     SHA512:
     2bca340dd0cd67c08f1454d7807703b80f15778dd1ce5f34925aef8a22a85dd8d04dca8bacab957dc430452aed327c3753678eca4e431deb8884e22e037ad8ae
     Description: Tools to test the reachability of network hosts

   - Scarico il file dal percorso, cioè https://archive.ubuntu.com/ubuntu**/pool/main/i/iputils/iputils-ping_20240117-1build1_amd64.deb**
   - Calcolo il digest SHA512 del file .deb scaricato, lo confronto con quello trovato nella descrizione del package, se sono uguali il file .deb è originale. OK
   - Posso installare il package iputils-ping

- **NB: Non devo fare tutto ciò a mano, il tool apt-get lo fa al posto mio**

Reperire chiave pubblica del maintainer del repo.

Ma come fa l'utente che vuole installate un package a sapere quale è la chiave pubblica del maintainer del repository?

Chi gli garantisce che gli venga data proprio la chiave giusta ?

1. **Chiavi Preinstallate** (La Fiducia Iniziale)

   L'utente si fida del produttore del sistema operativo (Canonical per Ubuntu, Debian Project per Debian) che ha verificato e incluso la chiave corretta nell'immagine ISO originale. Se l'immagine ISO è autentica, anche la chiave lo è.

   Durante l'installazione del s.o. la chiave corretta è installata nelle directory di sistema opportune (/etc/apt/trusted.gpg.d/ o /usr/share/keyrings/)

2. Chiavi di Terze Parti (Il Problema di Fiducia)

   Il problema diventa serio quando si aggiunge un repository esterno (terze parti, come Google, Docker, Microsoft).

   L'utente è responsabile di verificare che gli venga data la chiave giusta.

   L'utente sa quale chiave usare perché:

   A. **Segue Istruzioni Ufficiali dal Fornitore**. Il fornitore istruisce l'utente a scaricare la chiave da un dominio sotto il suo controllo e a salvarla.

      Esempio: **wget** ... https://packages.microsoft.com/keys/microsoft.asc | gpg ...

   B. Verifica la chiave mediante un digest (fingerprint) che gli viene dato con un media di comunicazione diverso da qello da cui ha ottenuto la chiave.