Operazione Rif. PA 2023-19410/RER approvata con DGR 1317/2023 del 31/07/2023 finanziata con risorse del Programma Fondo sociale europeo Plus 2021-2027 della Regione Emilia –Romagna.

Progetto n. 1 - Edizione n. 1

TECNICO PER LA PROGETTAZIONE E LO SVILUPPO DI APPLICAZIONI INFORMATICHE

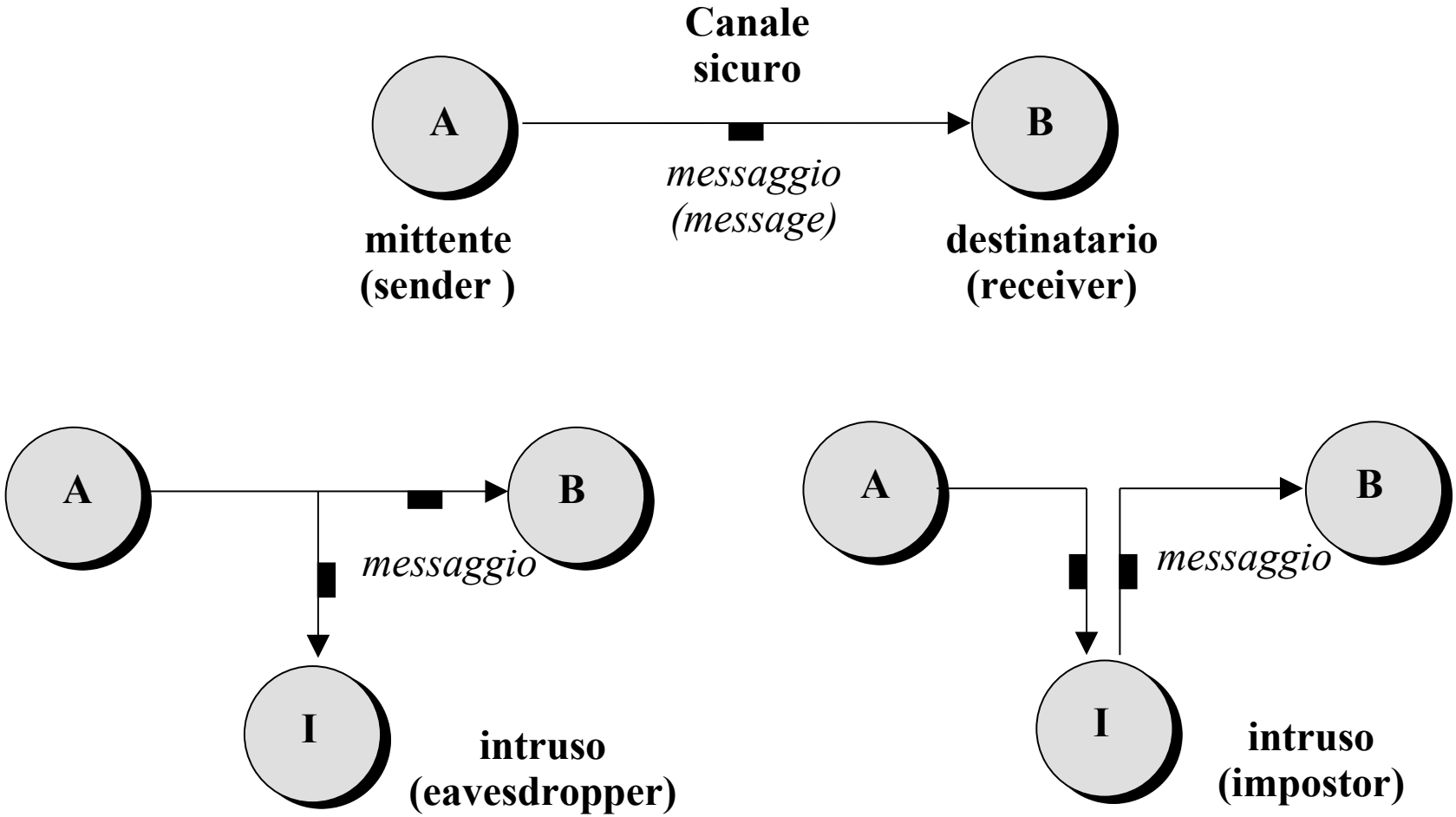
MODULO: N. 5 Titolo: SICUREZZA DEI SISTEMI INFORMATICI E DISPIEGO DELLE APPLICAZIONI
DURATA: 21 ORE DOCENTE: MARCO PRANDINI

CRITTOGRAFIA
SICUREZZA DELLE RETI

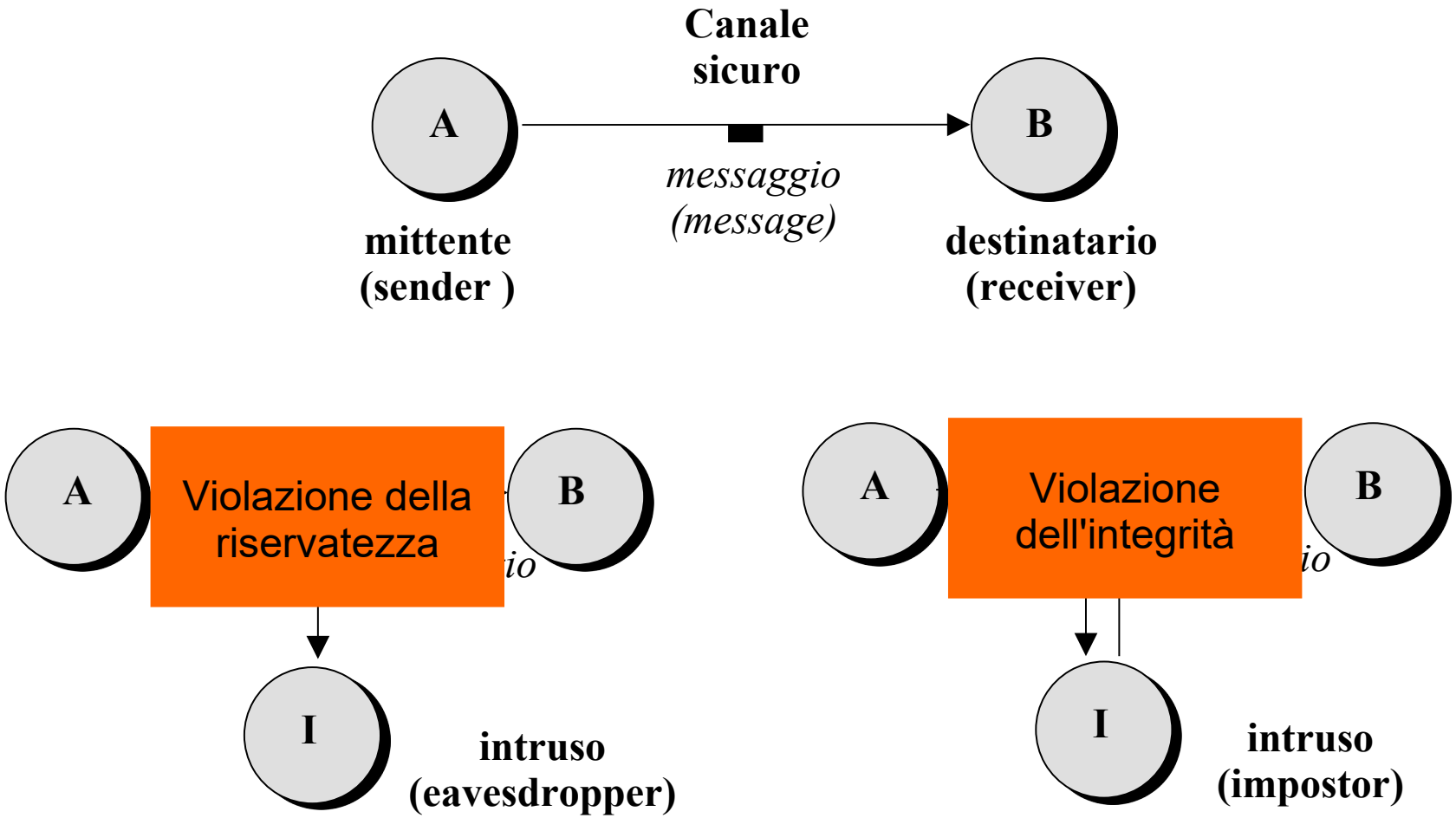
Ricordiamo le sfaccettature della sicurezza delle informazioni

- Confidentiality (riservatezza)
- Integrity (integrità)
  - Authenticity (paternità)
- Availability (disponibilità)

Mondi ideali e reali



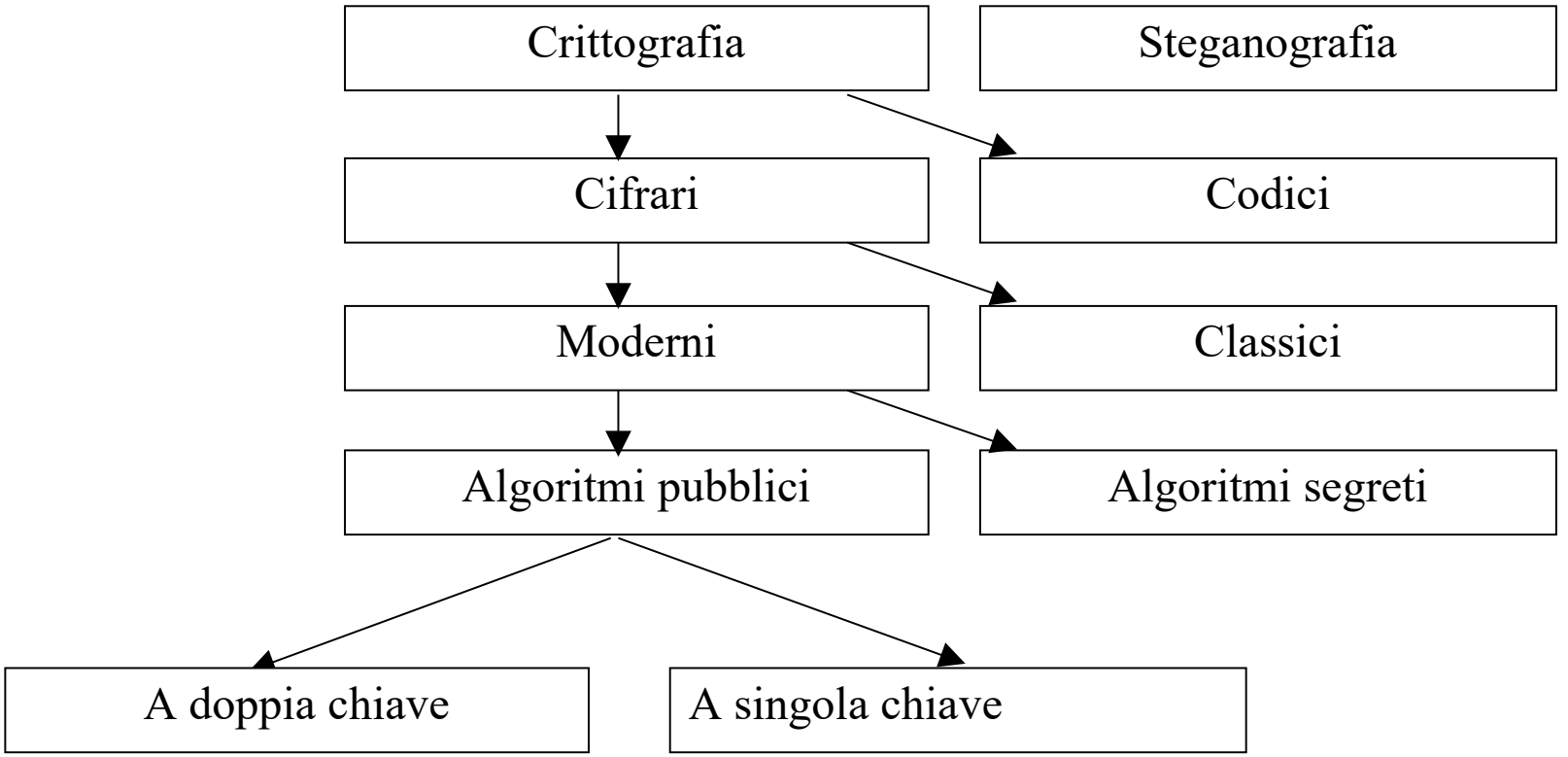
Mondi ideali e reali



Soluzione: crittografia

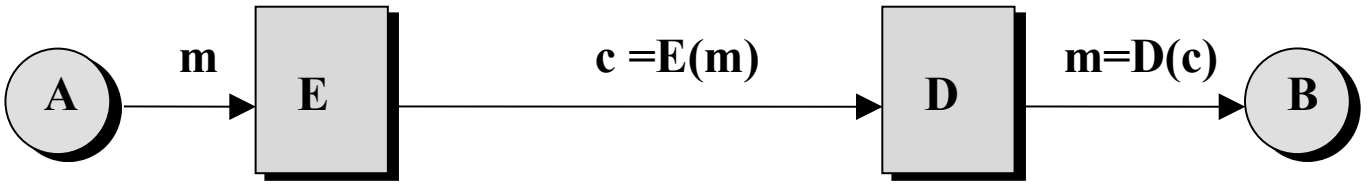
- Un'elaborazione matematica e algoritmica della codifica delle informazioni
- Prevenire la violazione della riservatezza (una rilevazione a posteriori sarebbe inefficace!):
  → alterare il codice in modo da renderlo incomprensibile a chi non ha diritto di apprendere le informazioni
- Rilevare la violazione dell'integrità e autenticità (non può essere prevenuta!)
  → aggiungere al codice elementi che permettano la verifica delle informazioni ricevute

Tecniche per la realizzazione delle primitive



Cifrari per la riservatezza

- Due operazioni
  - Cifratura
    converte il testo in chiaro in testo cifrato
  - Decifrazione
    converte il testo cifrato in testo in chiaro



I principi di Kerckhoffs (1883)

1) Le système doit être matériellement, sinon mathématiquement, indéchiffrable ;
   - Sicurezza *computazionale* o *assoluta*
2) Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ;
   - ~~segreto=algoritmo~~ segreto=chiave!
3) La clef doit pouvoir en être communiquée et etenue sans le secours de notes écrites, et être changée ou modifiée au gré des correspondants;
   - Cifratura = ricordare un segreto semplice per poter scambiare molti segreti arbitrari

Metodi della crittoanalisi

- A seconda del materiale a disposizione del crittanalista si possono avere diverse opportunità di attacco

- Forza bruta
  - Si tira ad indovinare D (o il suo particolare segreto) e si decifra il testo intercettato: se non ha alcun senso si ripete il procedimento

- Solo testo cifrato
  - Si eseguono analisi statistiche su una grande quantità di materiale cifrato e se ne usano le indicazioni per individuare quale p probabilmente corrisponde ad un dato c

- Testo in chiaro noto
  - Ci si procura in qualche modo sia dei testi cifrati, sia i corrispondenti testi in chiaro e si cerca di dedurre D analizzando le varie coppie

- Testo scelto
  - Si può scegliere testo da cifrare o da far decifrare per ottimizzare il procedimento di deduzione della chiave

- Rubber hose
  - Si minaccia, ricatta o tortura qualcuno finché non cede la chiave.

In sintesi: crittoanalisi e crittografia

- Di fronte a un testo cifrato con algoritmo noto, cosa può sempre fare un crittoanalista?
  - Analizzare le proprietà statistiche del testo
    → robustezza = capacità dell'algoritmo di occultare le proprietà del testo in chiaro
  - Cercare la chiave tra tutte quelle possibili
    → sicurezza assoluta = rendere totalmente indistinguibile la chiave giusta dalle altre
    → sicurezza computazionale = rendere troppo oneroso il processo di ricerca della chiave

Cifrari a blocchi

- La parte difficile è implementare E e D “modularmente” per poter lavorare liberamente sul numero di round
- Cifrari di Feistel
- Standard storico: DES (National Bureau of Standards degli U.S.A in collaborazione con IBM, pubblicato nel 1977, blocchi di 64 bit, chiave di 56 bit)

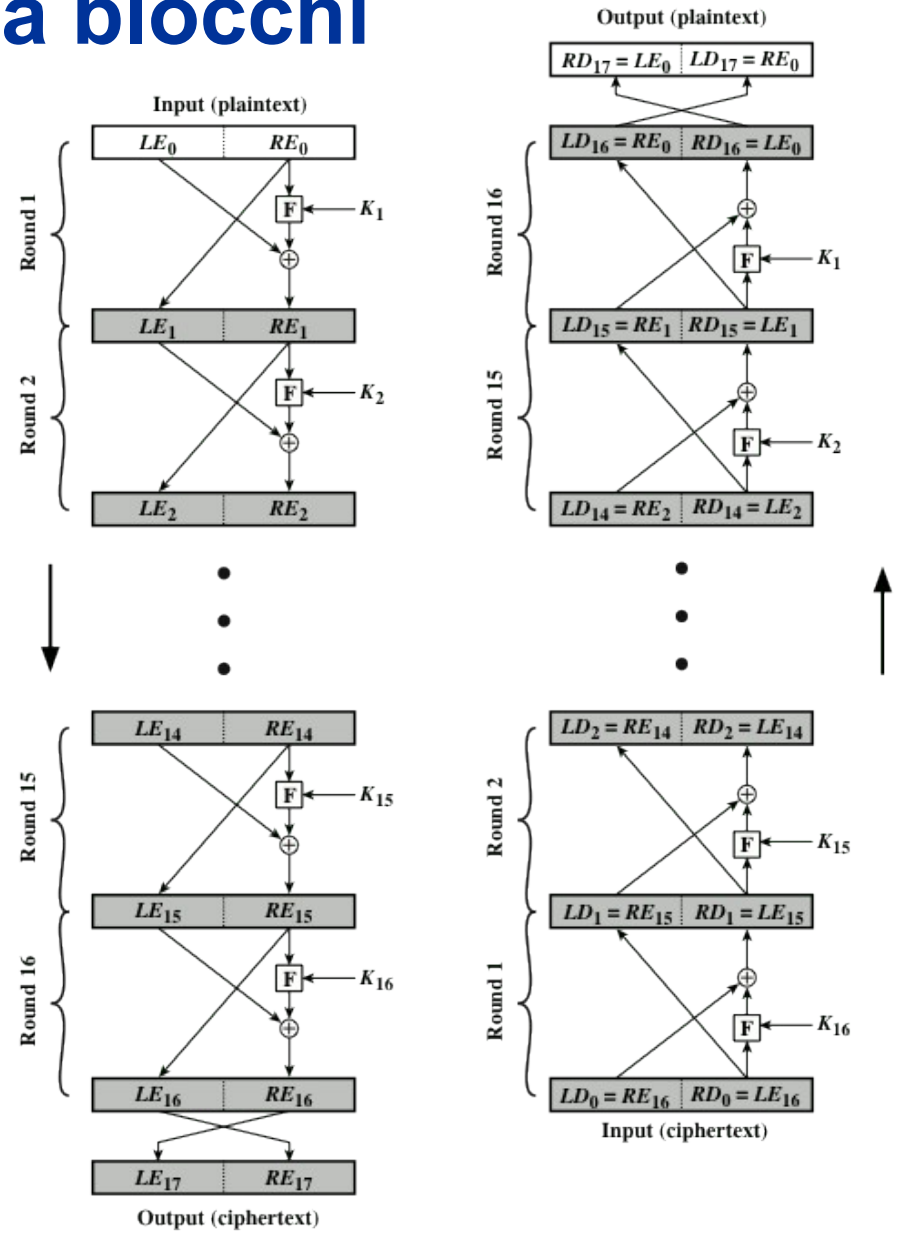


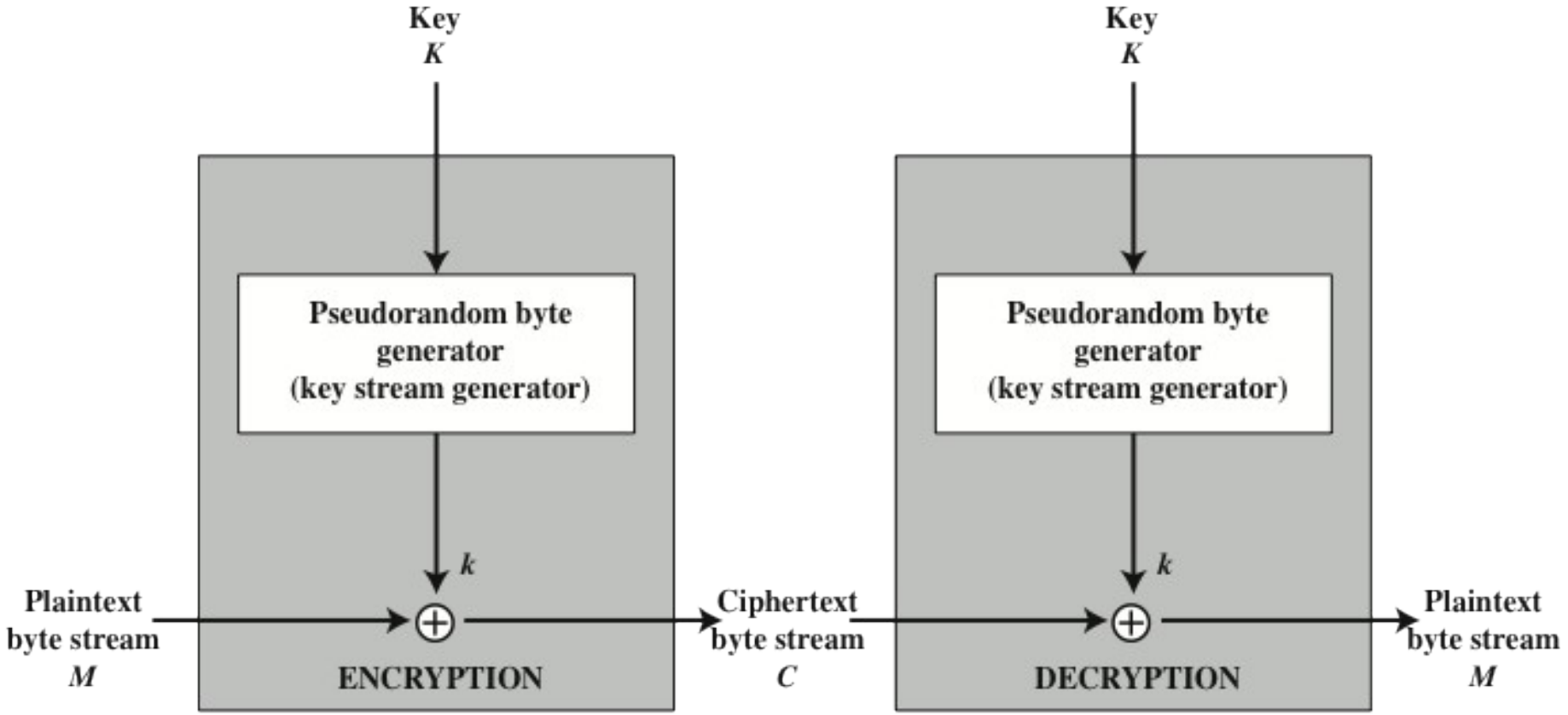
Figure 3.3 Feistel Encryption and Decryption (16 rounds)

AES

- Standard attuale: FIPS 197 “Advanced Encryption Standard” (Rijndael)
  https://csrc.nist.gov/publications/detail/fips/197/final
  - Interessante il processo di selezione
    https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development
- Non usa la struttura di Feistel ma l’aritmetica dei campi finiti
- Blocchi di 128 bit
- Può utilizzare chiavi di lunghezza diversa
  - 128 bit
  - 192 bit
  - 256 bit

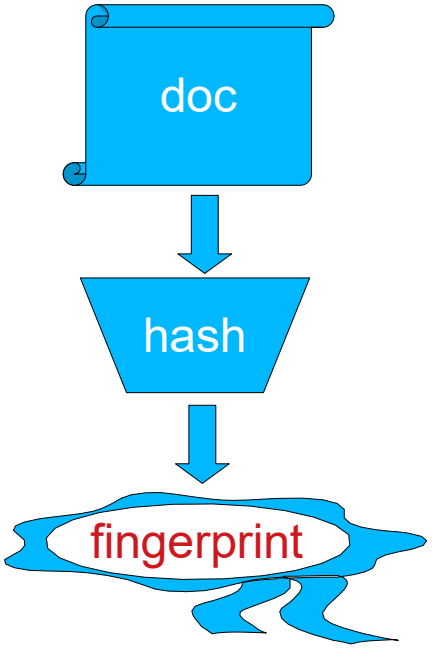
Cifrari a flusso

- One-Time Pad con alfabeto insignificante: solo 0 e 1 → Analisi statistica delle frequenze inapplicabile
- *Flusso di chiave* = generazione sequenza casuale
  - Seme = chiave condivisa



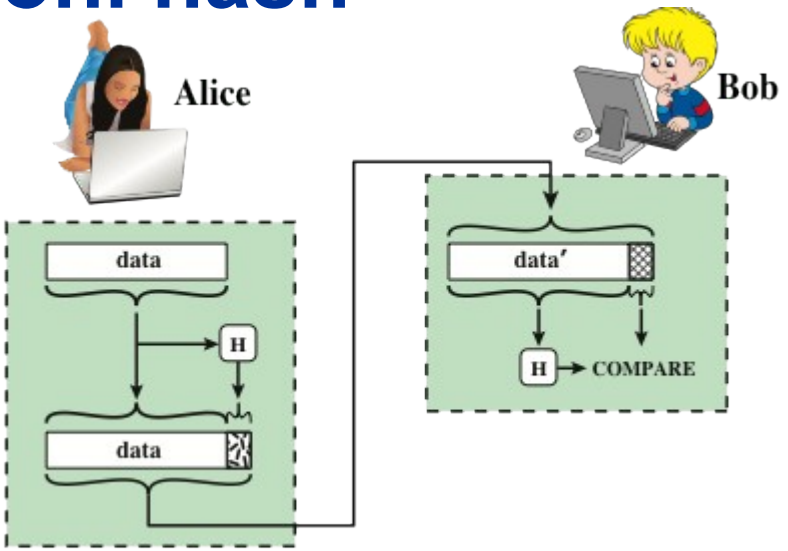
Funzioni hash

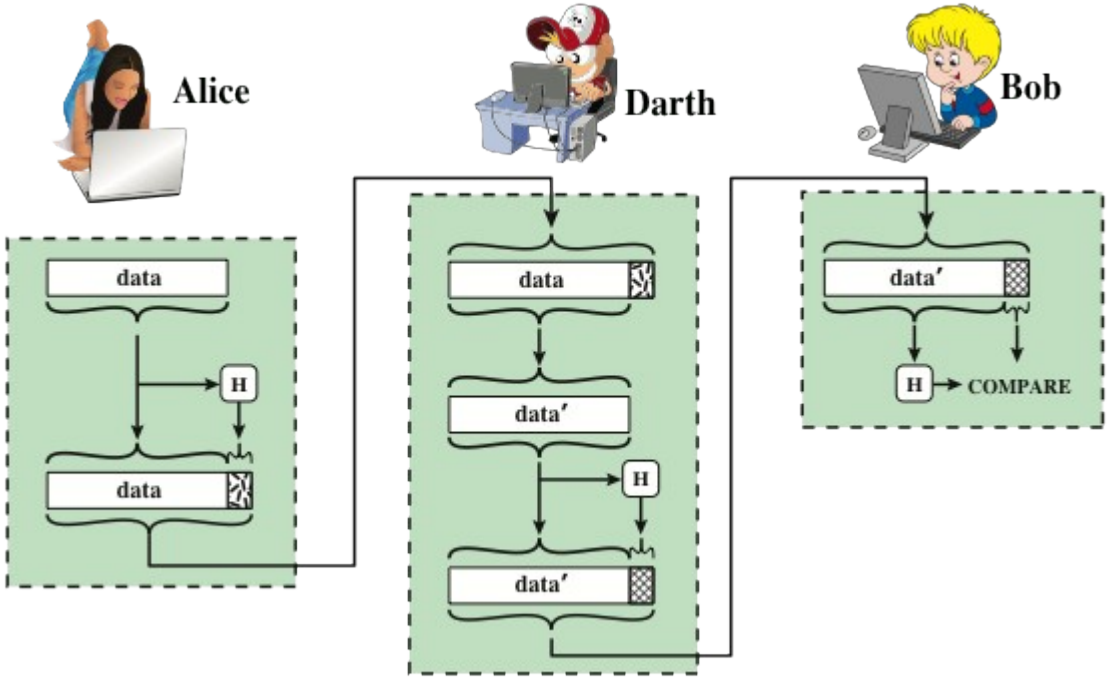
- Gli stessi principi dei cifrari a blocchi possono essere usati senza chiave per ottenere "impronte digitali" compatte di documenti di dimensione arbitraria
- Fingerprint:
  - dimensione fissa (f. non biunivoca)
  - f. pubblica, senza chiave
- Funzioni hash crittografiche – robuste se:
  1) Non si può trovare un documento che abbia un fingerprint prefissato (proprietà di unidirezionalità, o **one-way**)
  2) Non si può trovare una coppia di documenti con lo stesso fingerprint (proprietà di assenza di collisioni, o **collision-free**)



Utilità delle funzioni hash

- Integrità (checksum)
  - ✔ Ok contro alterazioni accidentali





⊖ Man-in-the-middle attack!

- Protezione necessaria
  - Canale sicuro
  - Altro?
- Autenticazione?
  - Manca elemento univoco dell'autore

Problemi difficili e trabocchetti

- Funzioni *pseudo-unidirezionali*
  - Operazioni facili in un verso e (speriamo) computazionalmente infattibili nell'altro
  - A meno di conoscere un segreto
- Fattorizzazione di grandi numeri
- Molte operazioni in aritmetica modulare
  - Numeri interi
  - Come risultato di un'operazione si prende il resto della divisione per un *modulo* fisso

Crittografia asimmetrica: RSA (1977)

- Generazione delle chiavi:
  1. si scelgono due numeri primi **p** e **q**
  2. il modulo viene calcolato come $n = p \cdot q$
  3. si sceglie a caso un numero **d** e si calcola un numero **e** tale che $e \cdot d \bmod (p-1)(q-1) = 1$
     - Facile solo conoscendo p e q, che vengono poi dimenticati
- La chiave **pubblica** è **(e, n)**, la chiave **privata (d, n)**
- Cifratura: $c = m^e \bmod n$
- Decifrazione $m = c^d \bmod n$

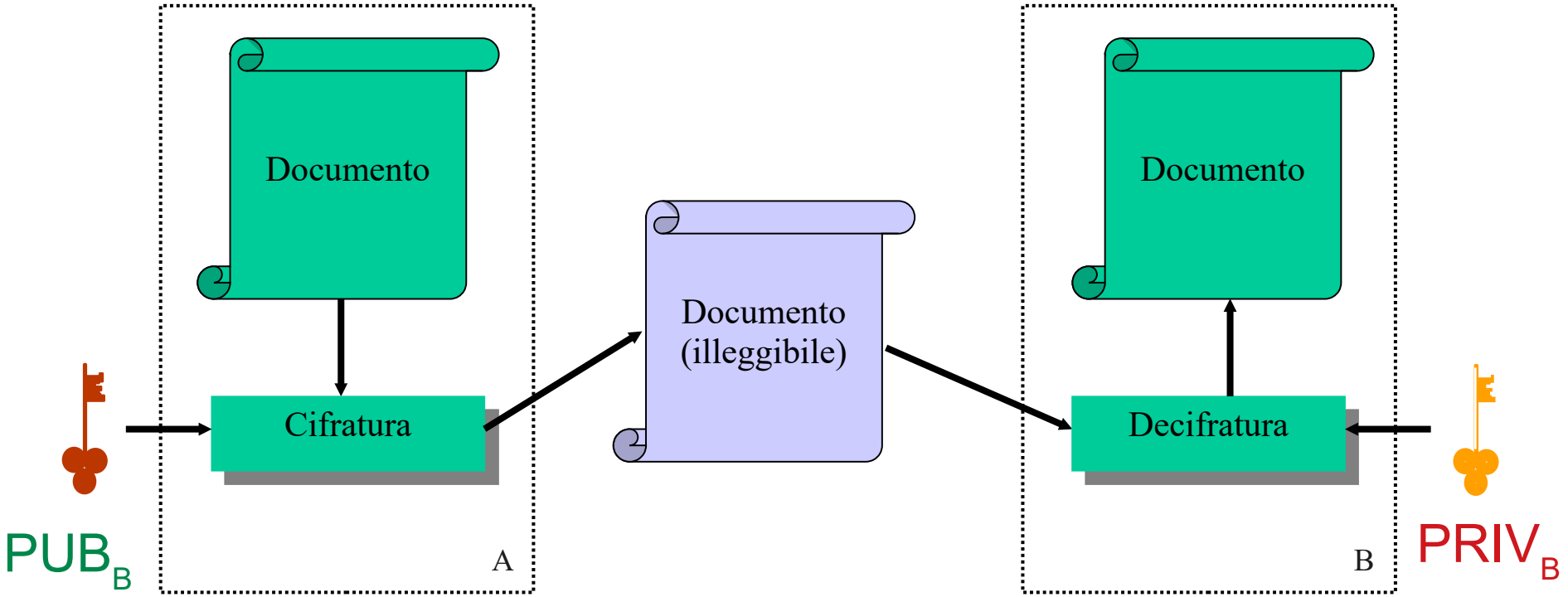
Robustezza

- Non ci sono modi efficienti noti di invertire l'esponenziale modulare
  - Complessità assimilabile a forza bruta
- Ci sono algoritmi "quasi efficienti" per fattorizzare il modulo
  - General Number Field Sieve, sub-esponenziale
  - Contromisura: moduli grandi (oltre 2048 bit)
- Trappole
  - Non è dimostrabile che non esistano algoritmi classici efficienti (ma nessuno ha idea di come trovarli)
  - Quantum computing
  - Implementazioni troppo efficienti
    - Spesso si sceglie *e* con pochi "1" (es. 3, 17, 65537)
    - Se troppo piccolo, m^e non "trabocca" da n!

Vantaggi della c. asimmetrica

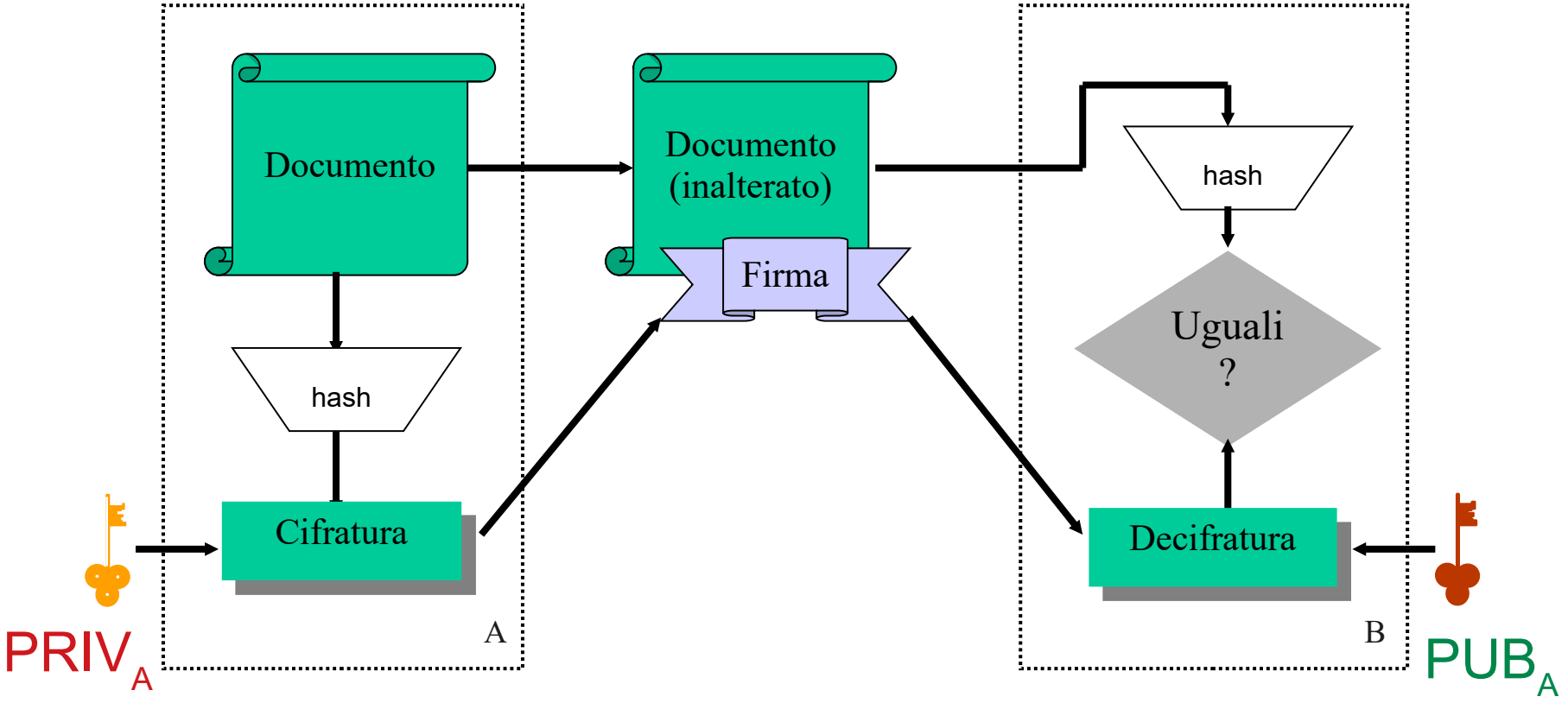
- Per la riservatezza
  - È un cifrario a blocchi, di sostituzione, con dimensioni enormi
    - No forza bruta
    - No analisi statistica dell'"alfabeto" (salvo casi particolari)
  - Le chiavi usate per cifrare e decifrare sono diverse e dalla chiave pubblica non è derivabile la chiave privata
    - La chiave pubblica può essere distribuita
    - Chiunque può usarla per cifrare
    - La chiave privata corrispondente è l'unica che può decifrare
- La chiave privata è specifica di un solo utente quindi utile anche per ***autenticare***

C. asimmetrica per la riservatezza



C. asimmetrica per l'integrità e l'autenticità

- Soluzione del problema dell'uomo nel mezzo visto per gli hash: cifrare il fingerprint con la chiave privata
  - Verifica corretta solo se tutto inalterato → integrità
  - Verifica corretta con PUB_A solo se la firma era stata prodotta con $PRIV_A$ → autenticità (**se posso fidarmi che PUB_A sia davvero di A!**)

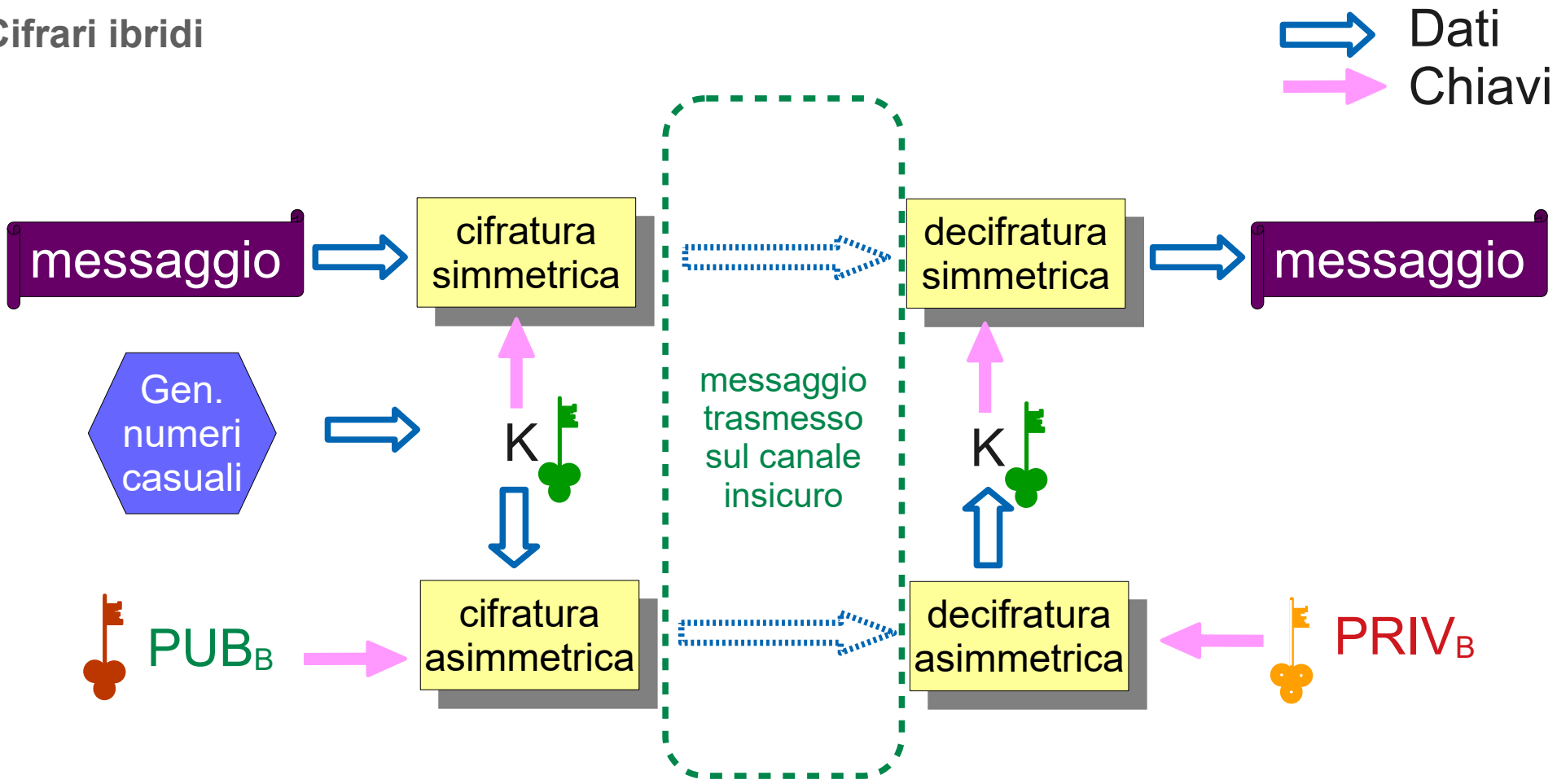


C. asimmetrica - pregi e difetti

- Grandi vantaggi:
  - distribuzione delle chiavi
  - utilità per tutte le proprietà di sicurezza
- Punti deboli:
  - Prestazioni (5-10 volte più lento di AES)
    - Sistemi ibridi
  - alcuni attacchi specifici (known plaintext)

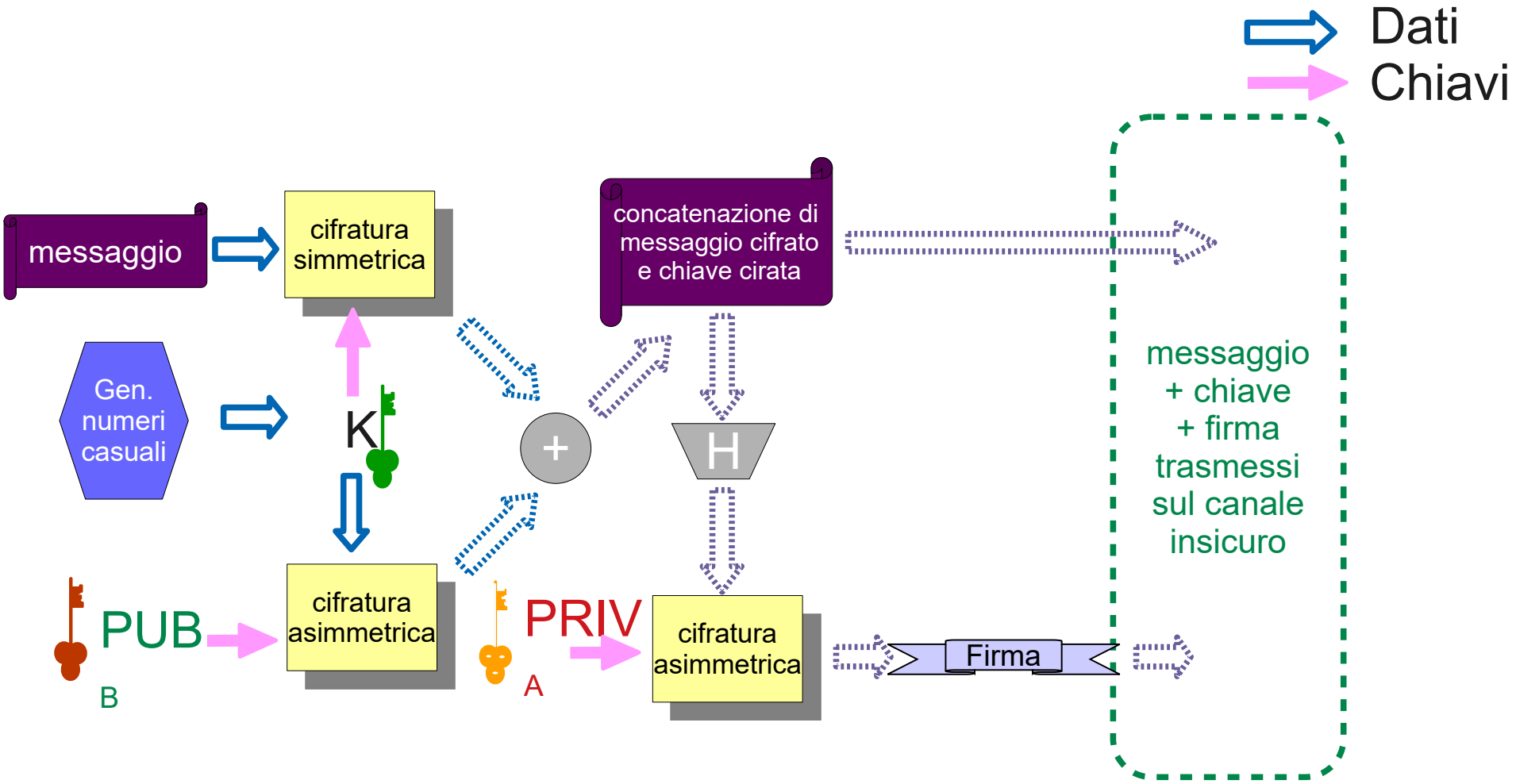
Aggiungere prestazioni e flessibilità

Cifrari ibridi

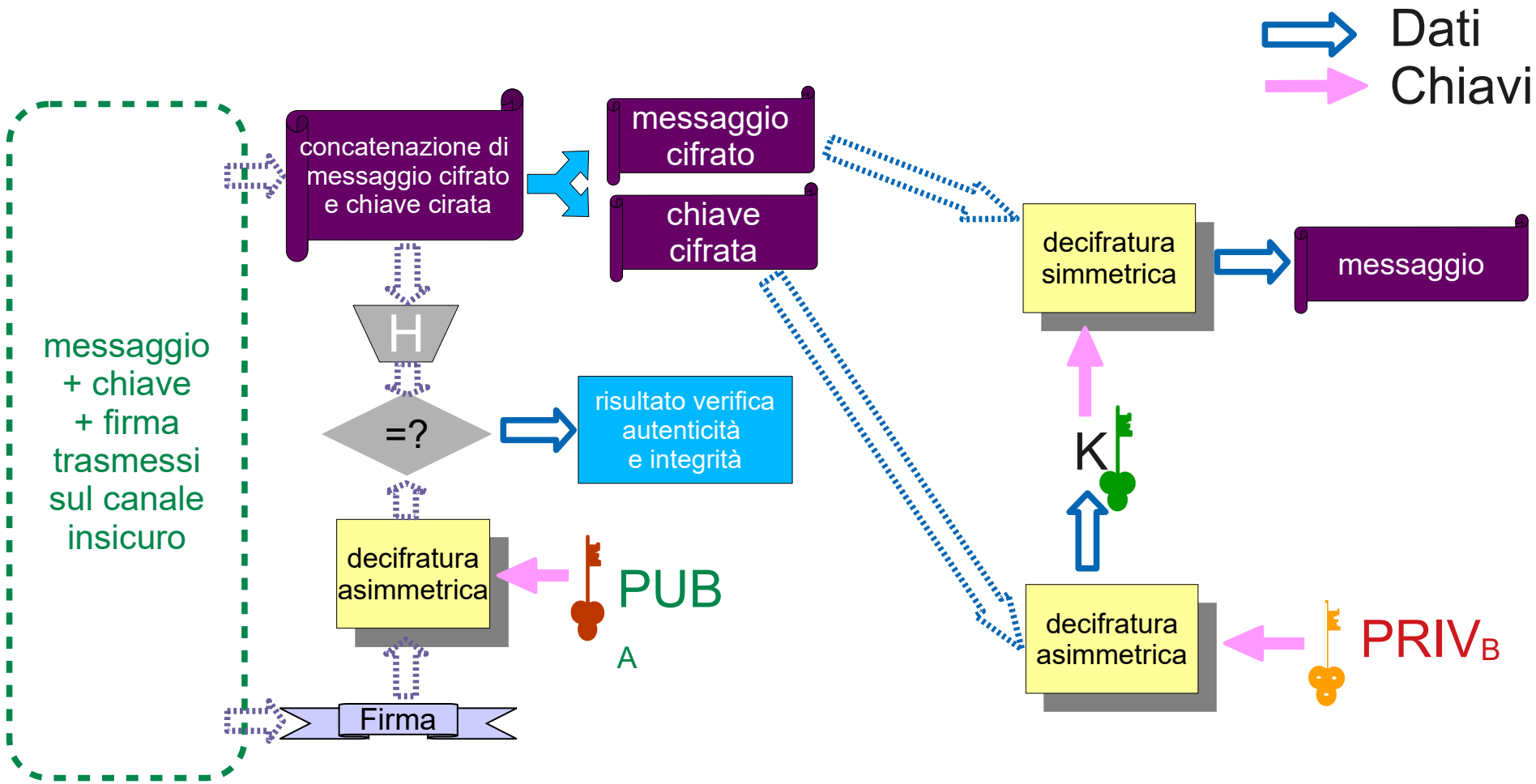


Più destinatari = un solo messaggio cifrato & più copie di K cifrate con la chiave pubblica di ognuno

Tutto insieme (schema di principio) – mittente A



Tutto insieme (schema di principio) – destinatario B



Gestione delle chiavi

- Le chiavi crittografiche sono l'elemento fondamentale per la sicurezza delle operazioni crittografiche
  - gli algoritmi, salvo errori di implementazione, sono molto robusti
- Bisogna considerare tutte le fasi del ciclo di vita
  - Generazione
  - Memorizzazione
  - Distribuzione

Generazione delle chiavi

- Per le chiavi simmetriche, i nonce, i vettori di inizializzazione, i padding, …
  - basta un buon generatore di numeri casuali
- Per le chiavi asimmetriche
  - servono numeri primi
  - si parte da numeri random
  - si applica un test di primalità

    https://en.wikipedia.org/wiki/Primality_test

PRNG

- La casualità gioca un ruolo fondamentale per la generazione delle chiavi e la randomizzazione dei protocolli crittografici
- Che proprietà devono avere i numeri casuali?

Randomness

Unpredictability

Uniform distribution

→ The frequency of occurrence of ones and zeros should be approximately equal

Independence

→ No one subsequence in the sequence can be inferred from the others

TRUE random sequences

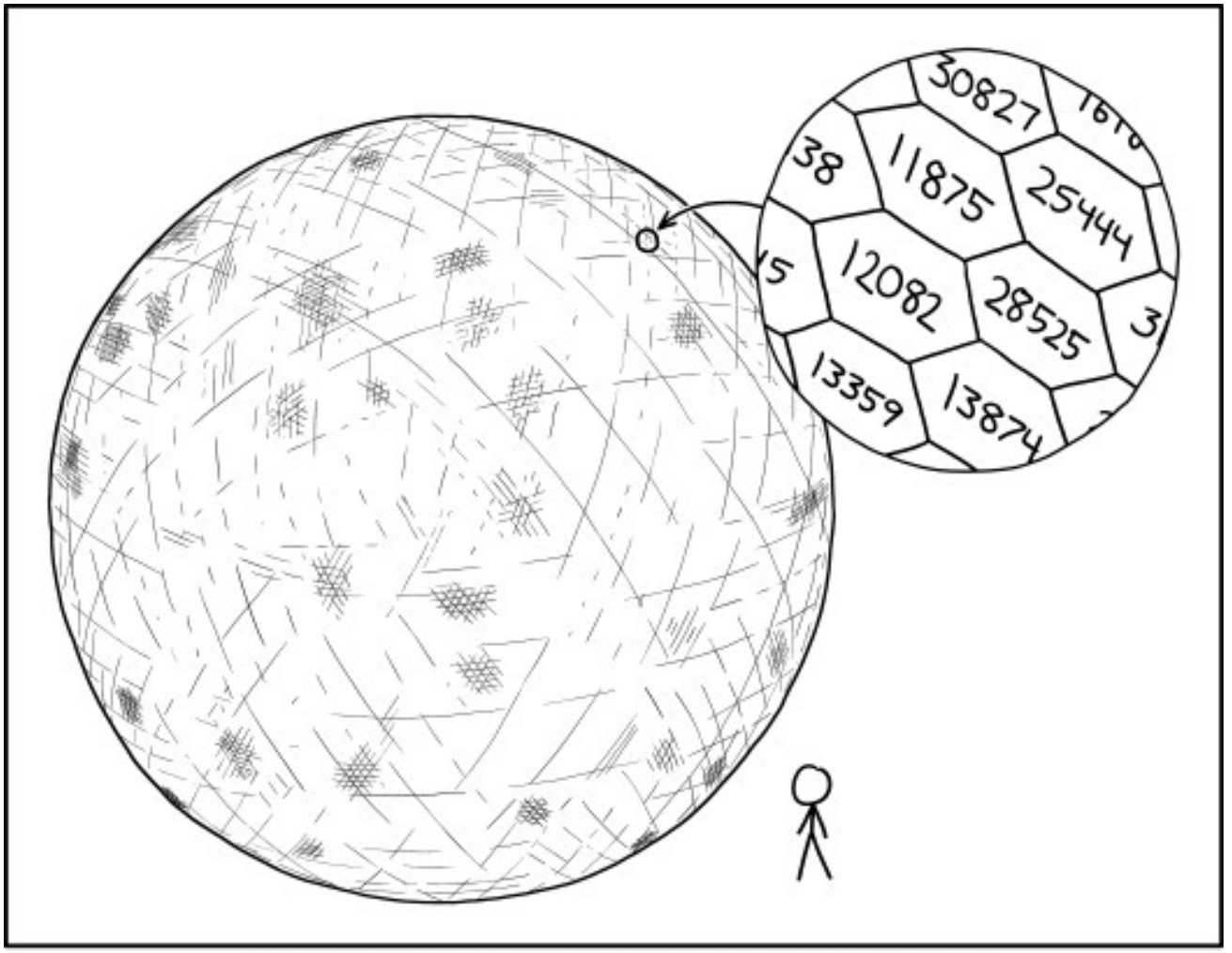
→ perfect independence guarantees unpredictability

HARD AND INEFFICIENT

PSEUDO random sequences

Algorithmic → must take extra care to make elements of the sequence hard to predict knowing earlier ones

True Random Number Generation



THE HARDEST PART OF SECURELY GENERATING RANDOM 16-BIT NUMBERS IS ROLLING THE D65536.

https://xkcd.com/2626/

True Random Number Generation

- Sorgenti fisiche di *entropia*
  - Elementi ad hoc, es. rumore termico, processi dinamici caotici https://blog.cloudflare.com/randomness-101-lavarand-in-production/
  - Eventi “imprevedibili” nel calcolatore, es. intervalli di arrivo degli interrupt dai dispositivi
- Elaborazione
  - Conversione A/D
  - Condizionamento (rimozione bias)

Pseudo Random Number Generation

- Algoritmo → Determinismo!
- Se il risultato supera i test statistici, accettabile come PRNG
  - Ma sempre attenzione all'imprevedibilità
- Tipicamente input = *seme* (*seed*) prodotto da TRNG
  - Noto il seme → nota la sequenza generata!
  - Se algoritmo robusto, una sequenza di valori intermedi non permette di risalire al seme o di ipotizzare il valori futuri

Resistenza alla forza bruta

- **Tempo di test dello spazio delle chiavi DES/AES con tecnologie recenti:**

	Lunghezza della chiave in bit		
Budget	56	128	256
1 K€ (individuo)	16 anni	10^{22} anni	10^{61} anni
1 M€ (impresa)	6 giorni	10^{19} anni	10^{58} anni
1 G€ (NSA)	8 minuti	10^{16} anni	10^{55} anni

- **Attenzione alle ricerche con tempo di calcolo gratis (lotteria cinese, virus) e alla sfortuna!**
- **Anche se la legge di Moore proseguisse, c'è un limite invalicabile: la termodinamica**
  Limite di Landauer: per cambiare 1 bit almeno $k \times T \times \ln(2)$ (3×10^{-23} J a 3°K)
  Tutta l'energia emessa dal Sole in un anno = 1.2×10^{34} J
  → 4×10^{56} bit flip, come contare da 0 a 2^{188}
  Energia emessa dall'esplosione di una supernova = 2×10^{44} J
  → 7×10^{66} bit flip, come contare da 0 a 2^{222}

Resistenza alla fattorizzazione

- Il miglior attacco a RSA non è la forza bruta ma la ricerca dei fattori del modulo
- Stime non facilissime

  https://www.keylength.com/en/8/
  - 3000 bit robusti fino al 2026
- Stato attuale
  - Boudot et alii (2020) hanno fattorizzato RSA-250 (829 bit) usando 2700-anni-core

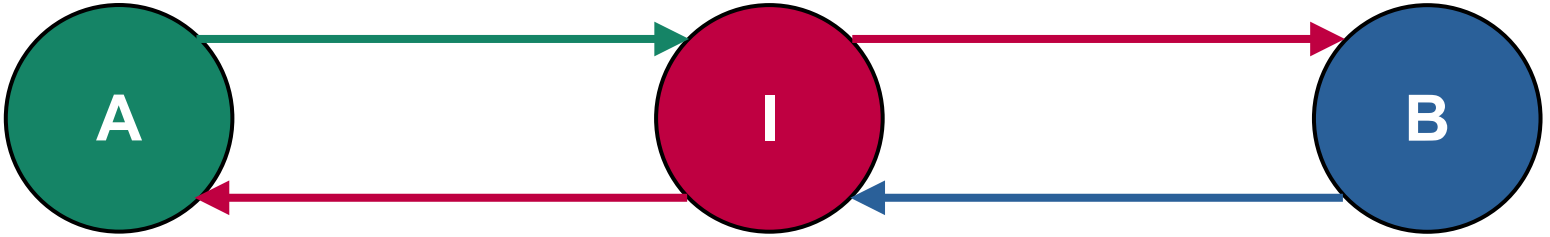
    https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html

Memorizzazione

- Chiave di decifrazione: requisiti contrastanti
  - perdita devastante → backup
  - segretezza fondamentale → non diffusione
- Accorgimenti di memorizzazione
  - Cifratura con passphrase
  - Hardware Security Module
  - Key escrow
  - Secret sharing
- Chiave di firma
  - se compromessa si sostituisce, nessuno ha bisogno di recuperarla in assenza del titolare → nessun backup!
  - non deve essere usata contro la volontà del titolare → cifratura con passphrase, HSM

Distribuzione

- Un attaccante passivo non apprende nulla dalla visione di una chiave pubblica o dall'intercettazione dei parametri di DH, un attaccante attivo può invece sostituire i valori inviati da una parte all'altra coi propri
- RSA: l'attaccante ha una propria coppia di chiavi $PRIV_I$ e PUB_i, e quando due utenti cercano uno la chiave pubblica dell'altro, ricevono invece PUB_i



  - quando il mittente cifra i messaggi, l'attaccante li può decifrare, e con la chiave pubblica del destinatario legittimo, per ri-cifrarli per non insospettirlo (magari alterati)
  - l'attaccante può firmare messaggi con $PRIV_I$ e il destinatario, verificandoli correttamente con $PRIV_I$ si convincerà che siano del mittente legittimo
- DH: l'attaccante stabilisce due chiavi separate con A e B e continua a fare da "passacarte" senza insospettirli
- Il problema quindi per i sistemi asimmetrici non è la riservatezza, ma **l'autenticità** dei dati pubblici ricevuti

Autenticazione delle chiavi pubbliche

- Serve un modo per associare con certezza una chiave pubblica al suo legittimo titolare (nonché unico possessore della corrispondente chiave privata)
- Modello *web of trust:*
  - L'autenticità di una chiave pubblica è testimoniata da altri utenti
  - L'utente che riceve una chiave da uno sconosciuto può decidere di accettarla per autentica se è firmata da qualcuno fidato
  - **Vantaggio**: nessuna entità "super partes" di cui doversi fidare
  - **Svantaggio**: pessima scalabilità
- Modello infrastrutturale:
  - esiste una terza parte fidata che documenta l'associazione

Secure Shell

- Necessità: amministrazione remota
- Predecessori: TELNET
  - Nessuna confidenzialità del canale
  - Nessuna autenticazione dell'host
  - Autenticazione passiva dell'utente

Secure Shell

- Il collegamento SSH tra client (ssh) e server (sshd) avviene attraverso questi passi essenziali
  - Negoziazione dei cifrari disponibili
  - Autenticazione dell'host remoto per mezzo della sua chiave pubblica
  - Inizializzazione di un canale di comunicazione cifrato
  - Negoziazione dei metodi disponibili per l'autenticazione dell'utente
  - Autenticazione dell'utente
- Ognuno dei passi elencati può essere portato a termine in modo configurabile, al fine di garantire il compromesso tra sicurezza e flessibilità più adatto al contesto.

Secure Shell – host authentication

- L'autenticazione dell'host remoto è importante per evitare di cadere nella trappola tesa da un eventuale uomo nel mezzo, che potrebbe così catturare la password dell'amministratore spacciandosi per l'host su cui egli vuole effettuare il login
  - Non è previsto un sistema centralizzato di attestazione dell'autenticità della chiave dell'host
    - solo supporto non ufficiale a X.509
  - Alla prima connessione l'amministratore deve utilizzare un metodo out-of-band per determinare la correttezza della chiave pubblica presentata dall'host
  - Alle connessioni successive la chiave pubblica memorizzata dal client dell'amministratore permette di effettuare un'autenticazione attiva

- Le chiavi pubbliche vengono memorizzate nel file **known_hosts** nella directory **.ssh** posta nella home dell'utente sul client.

Secure Shell – user authentication

- Ci sono due possibilità per l'autenticazione dell'utente sull'host remoto
  - Autenticazione passiva, tradizionale, con username e password – i dati sono trasmessi all'host autenticato su di un canale cifrato, quindi con buon livello di sicurezza
  - Autenticazione attiva, per mezzo di un protocollo challenge-response a chiave pubblica – presuppone che l'utente si doti della coppia di chiavi, e che installi correttamente sull'host remoto la chiave pubblica

Secure Shell – user authentication

- In entrambi i casi, l'identità dell'utente con cui viene tentato il login sull'host remoto può essere selezionata
  - in assenza di indicazioni specifiche verrà usato lo stesso nome utente con cui l'operatore sta lavorando sul client

  Es:
  - utente `marco` sul client esegue `ssh remoteserver`

    → si presenta come utente `marco` su `remoteserver` e si deve autenticare di conseguenza
  - utente `marco` sul client esegue `ssh root@remoteserver`

    → si presenta come utente `root` su `remoteserver` e si deve autenticare di conseguenza

Secure Shell – key generation

- Per poter effettuare l'autenticazione attiva un utente deve
  - generare una coppia di chiavi asimmetriche
    - es. `ssh-keygen -t rsa -b 2048`
      - chiave privata `.ssh/id_rsa`
      - chiave pubblica `.ssh/id_rsa.pub`
  - installare sull'host remoto la chiave pubblica.
    - tool di copia

      `ssh-copy-id [-i file] user@remote`
    - oppure
      - copia manuale su host remoto

      `scp .ssh/id_rsa.pub user@remote:`
      - append alla lista di utenti autorizzati (su *remote*)

      `cat id_rsa.pub >> .ssh/authorized_keys`

Secure Shell – avvertenze

Il ruolo autenticante della password viene sostituito dalla presenza della chiave privata dell'utente sul client – la segretezza della password è quindi sostituita dalla riservatezza del file che contiene la chiave privata

- Grande cura nell'impostazione dei permessi di file e directory (nota di tipo pratico: spesso il passwordless login non funziona semplicemente perché i permessi sulla directory *.ssh* dell'host remoto sono troppo larghi, e quindi il server sshd "non si fida" dell'integrità del suo contenuto)
- Possibilità di proteggere la chiave privata con una password

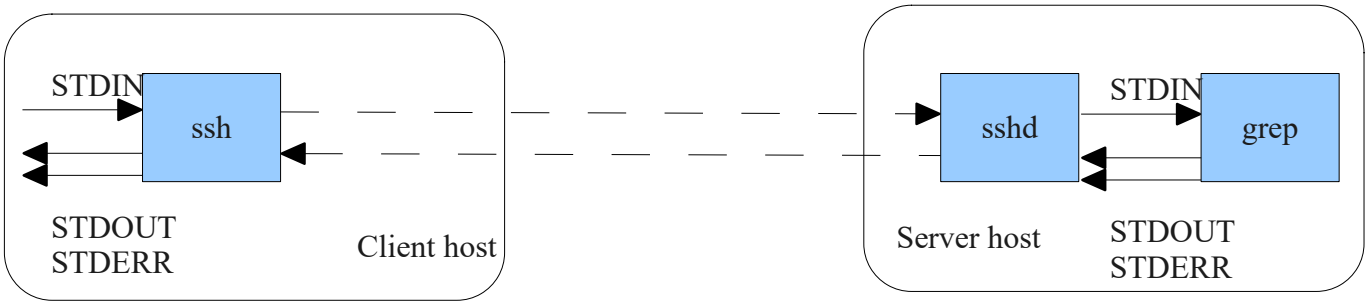
  - Vi priva della possibilità di passwordless login
  - Più sicuro comunque che utilizzare direttamente la password dell'account remoto, e più pratico se si amministrano molti host remoti

Secure Shell – esecuzione remota

- Lanciando `ssh utente@host` si ottiene un *terminale remoto* interattivo.
- Aggiungendo un ulteriore parametro, viene interpretato come **comando** da eseguire sull'host remoto al posto della shell interattiva; gli stream di I/O di tale comando vengono riportati attraverso il canale cifrato sul client.

  Es: `ssh root@server "grep pattern"`

  - I dati forniti attraverso STDIN al processo ssh sul client vengono resi disponibili sullo STDIN del processo grep sul server
  - STDOUT e STDERR prodotti dal processo grep sul server "fuoriescono" dagli analoghi stream dal processo ssh sul client



SSH tunnelling “L”

Dalla man page:

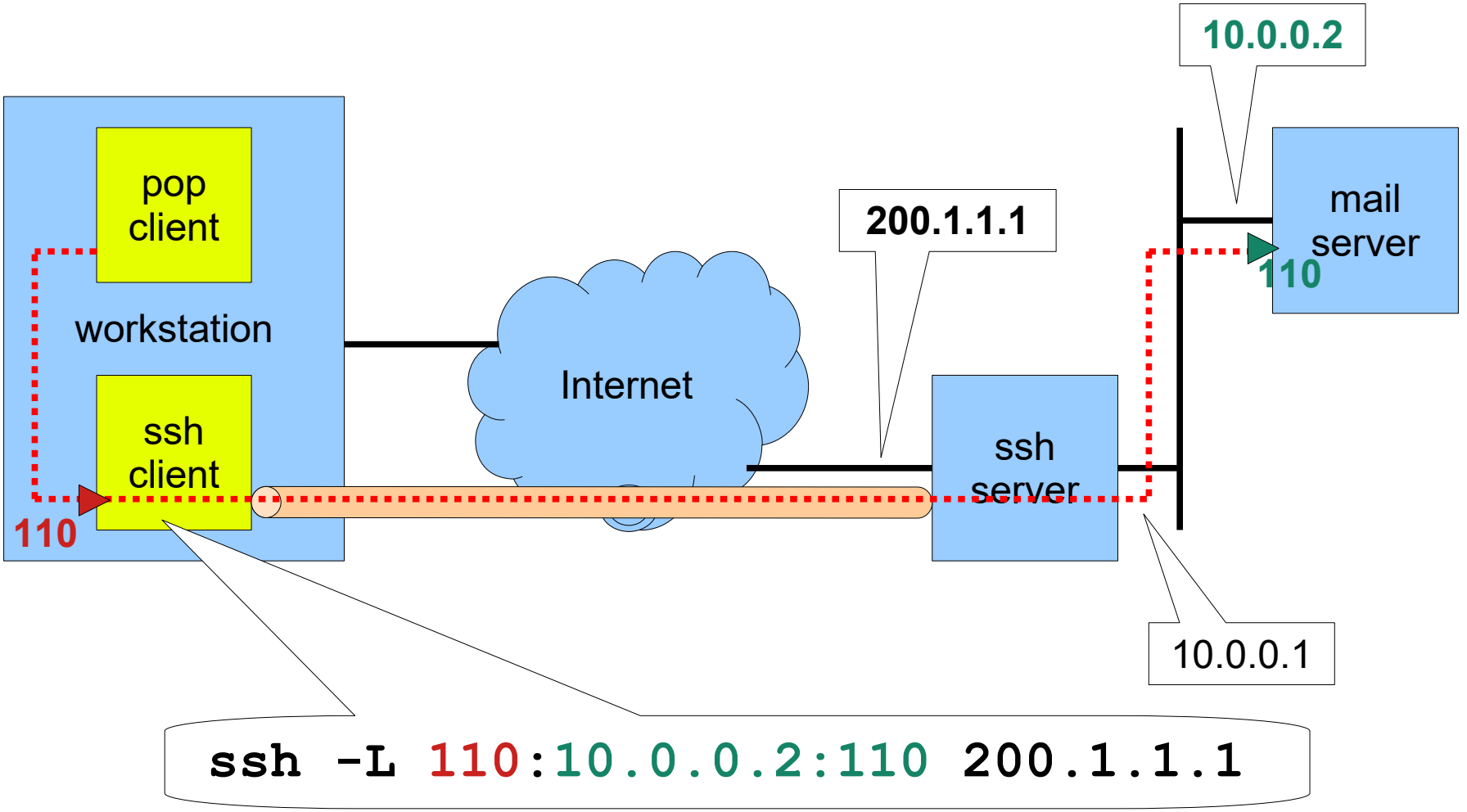
- **"poor man's VPN"**

`-L [bind_address:]port:host:hostport`

Specifica che la porta specificata sull'host locale (client) deve essere inoltrata all'host e alla porta host specificati sul lato remoto. Funziona allocando un socket per ascoltare la porta sul lato locale, facoltativamente associato al bind_address specificato. Ogni volta che viene effettuata una connessione a questa porta, la connessione viene inoltrata sul canale protetto e viene stabilita una connessione alla porta host hostport dalla macchina remota.

- **Prerequisiti: dare a un utente accesso SSH a un gateway “di frontiera” per l’organizzazione**
  - **Autenticazione forte**
  - **Possibilità di restringere le operazioni ammissibili**
- **Effetto: rendere raggiungibili host e servizi al di là del gateway**

SSH tunnelling “L” – esempio



SSH tunnelling “R”

Dalla man page:

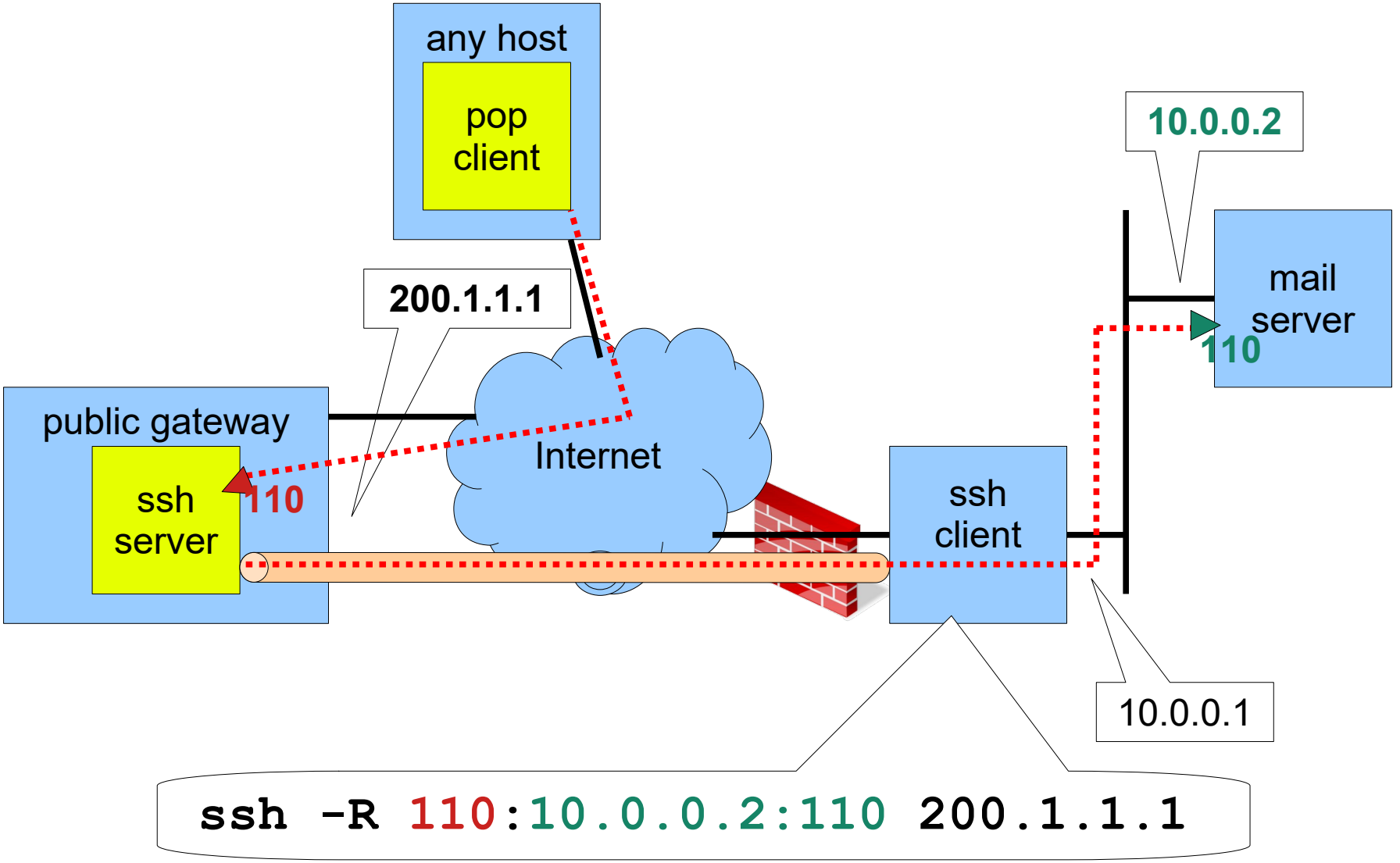
- **creare un accesso pubblico a una rete privata**

`-R port:host:hostport`

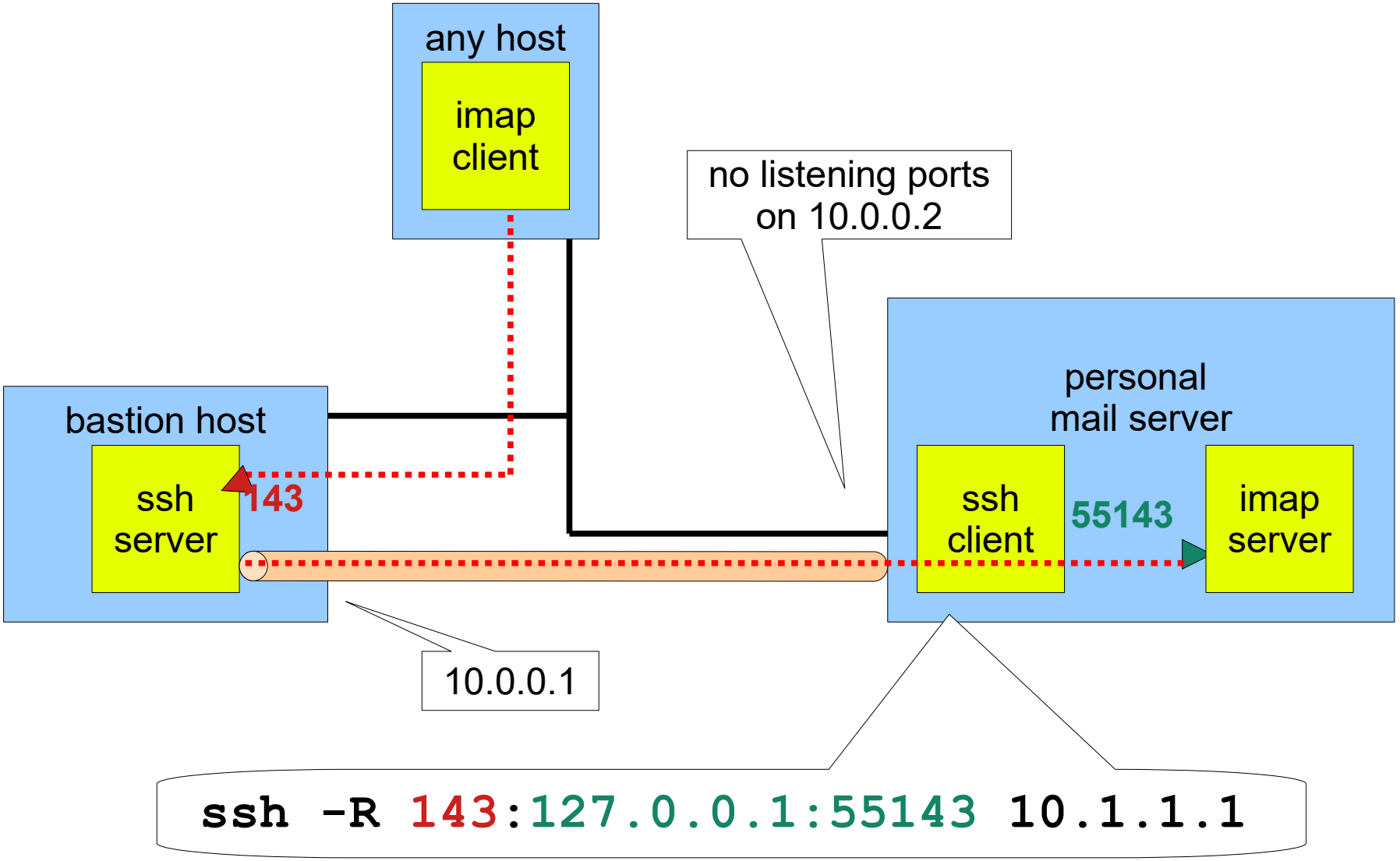
Specifica che la porta specificata sull'host remoto (server) deve essere inoltrata all'host e alla porta host specificati sul lato locale. Funziona allocando un socket per ascoltare la porta sul lato remoto e ogni volta che viene effettuata una connessione a questa porta, la connessione viene inoltrata sul canale protetto e viene stabilita una connessione alla porta host hostport dalla macchina locale.

- **Prerequisiti: avere a disposizione un gateway (con almeno un lato) esterno alla rete privata (pubblicamente) raggiungibile**
- **Effetto: rendere raggiungibile dalla rete esterna (non necessariamente pubblica)**
  - **un servizio locale non esposto sulla rete privata**
  - **un servizio della rete privata (inaccessibile dall’esterno)**

SSH tunnelling “R” – esempio “buca firewall”



SSH tunnelling “R” – esempio “filtro locale”



SSH tunnelling “D”

Dalla man page:

- **attivazione di un proxy SOCKS**

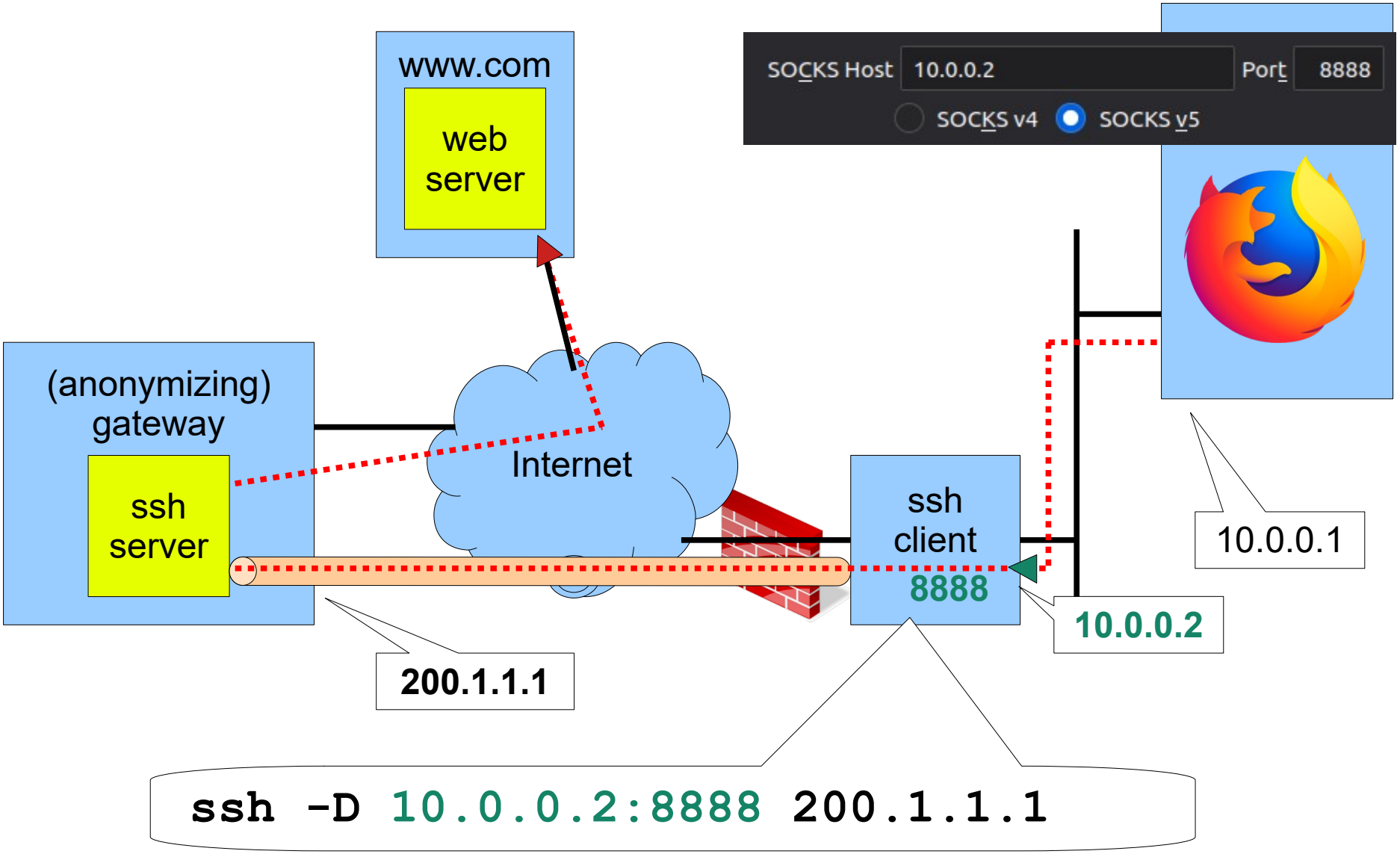
`-D [bind_address:]port`

Specifica un port forwarding "dinamico" a livello di applicazione locale.

Funziona allocando un socket per ascoltare la porta sul lato client, facoltativamente associata al bind_address specificato. Ogni volta che viene stabilita una connessione a questa porta, la connessione viene inoltrata sul canale protetto e là viene utilizzato il protocollo dell'applicazione per determinare dove connettersi dalla macchina remota. Attualmente i protocolli SOCKS4 e SOCKS5 sono supportati e ssh agirà come server SOCKS.

- **Prerequisiti: avere a disposizione un gateway “sensato” da cui far apparentemente originare**
- **Effetto: simil-ToR (senza l’elusione del triplo salto), o simil-”L” ma rendendo accessibili porte arbitrarie, un po’ come una VPN**

SSH tunnelling “D”



SSH tunnelling “J”

Dalla man page:

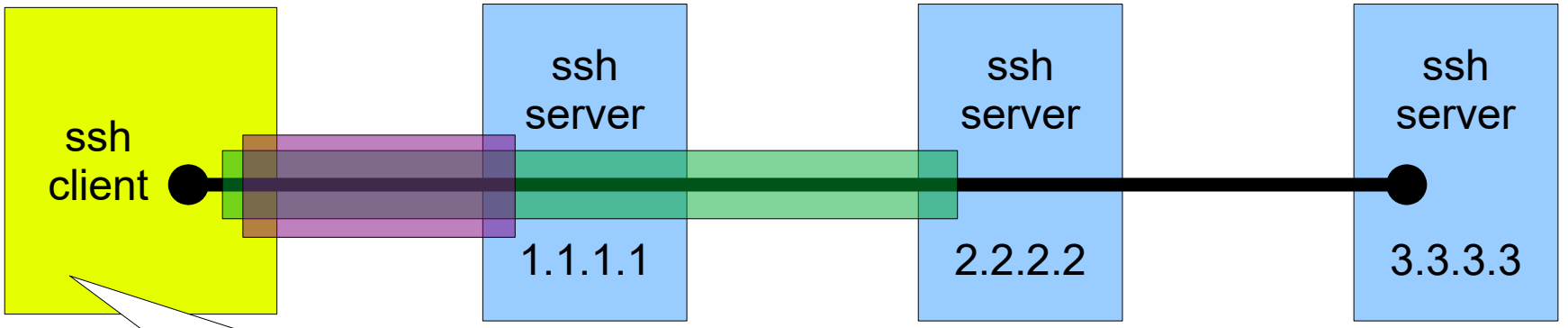
- **creare un accesso pubblico a una rete privata**

`-J jumphost`

Connette via SSH all'host di destinazione effettuando prima una connessione SSH al jumphost passato come parametro. È possibile specificare molteplici salti separati da virgole.

- **Prerequisiti: avere a disposizione un gateway raggiungibile dal client e dal quale si possa raggiungere la destinazione finale**
- **Effetto: come “L” ma specifico per SSH, e multi-salto**

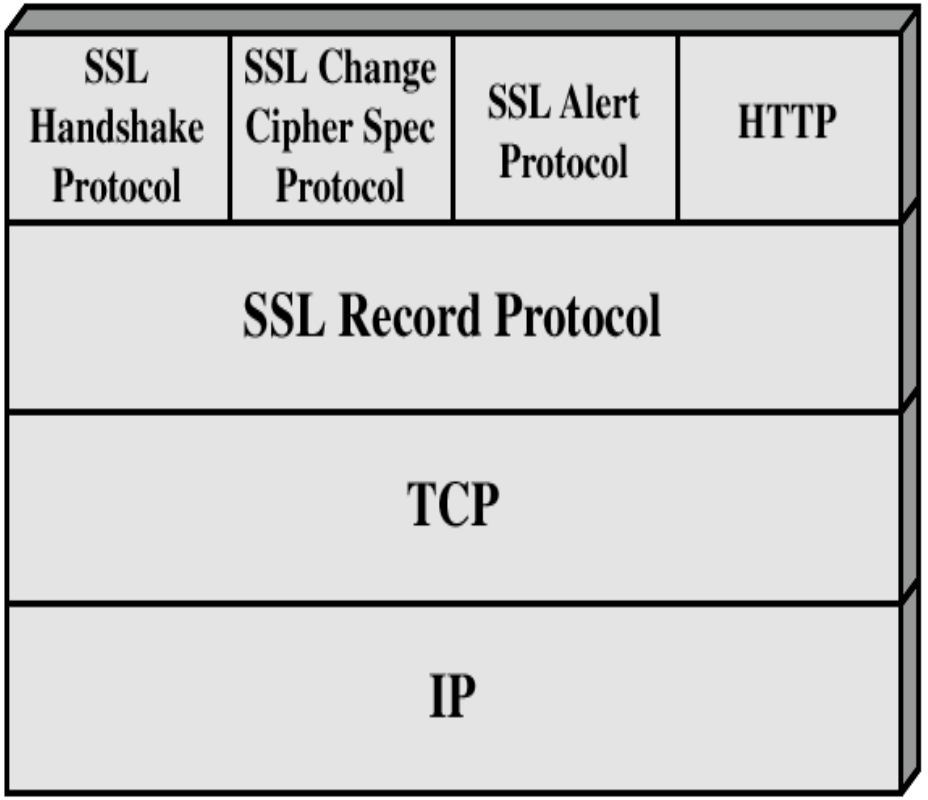
SSH tunnelling “J” – esempio




```
ssh  -J  u1@1.1.1.1,u2@2.2.2.2  u3@3.3.3.3
```

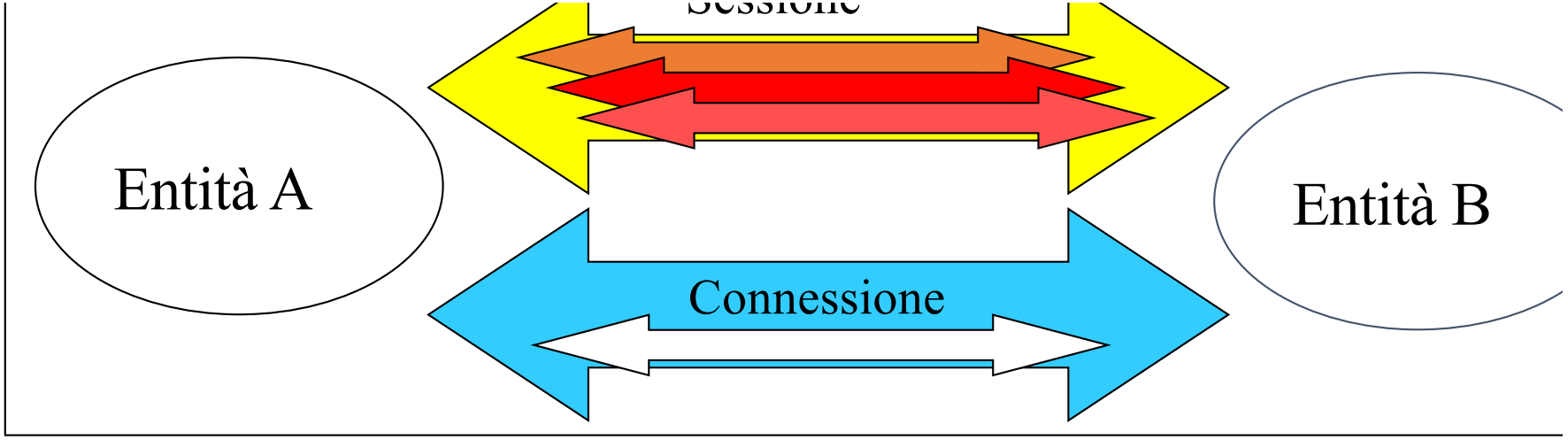
SSL/TLS

- SSL è stato progettato come uno strato di protocolli indipendente
- Si colloca logicamente fra strato di trasporto e applicazioni
  - Grazie a questa architettura non richiede una modifica dei protocolli di rete
- L'implementazione si presenta come una serie di funzioni di libreria (SSLeay, OpenSSL, ...)
  - Per rendere una applicazione capace di usare SSL è sufficiente inserire nel codice le chiamate a tali funzioni di libreria



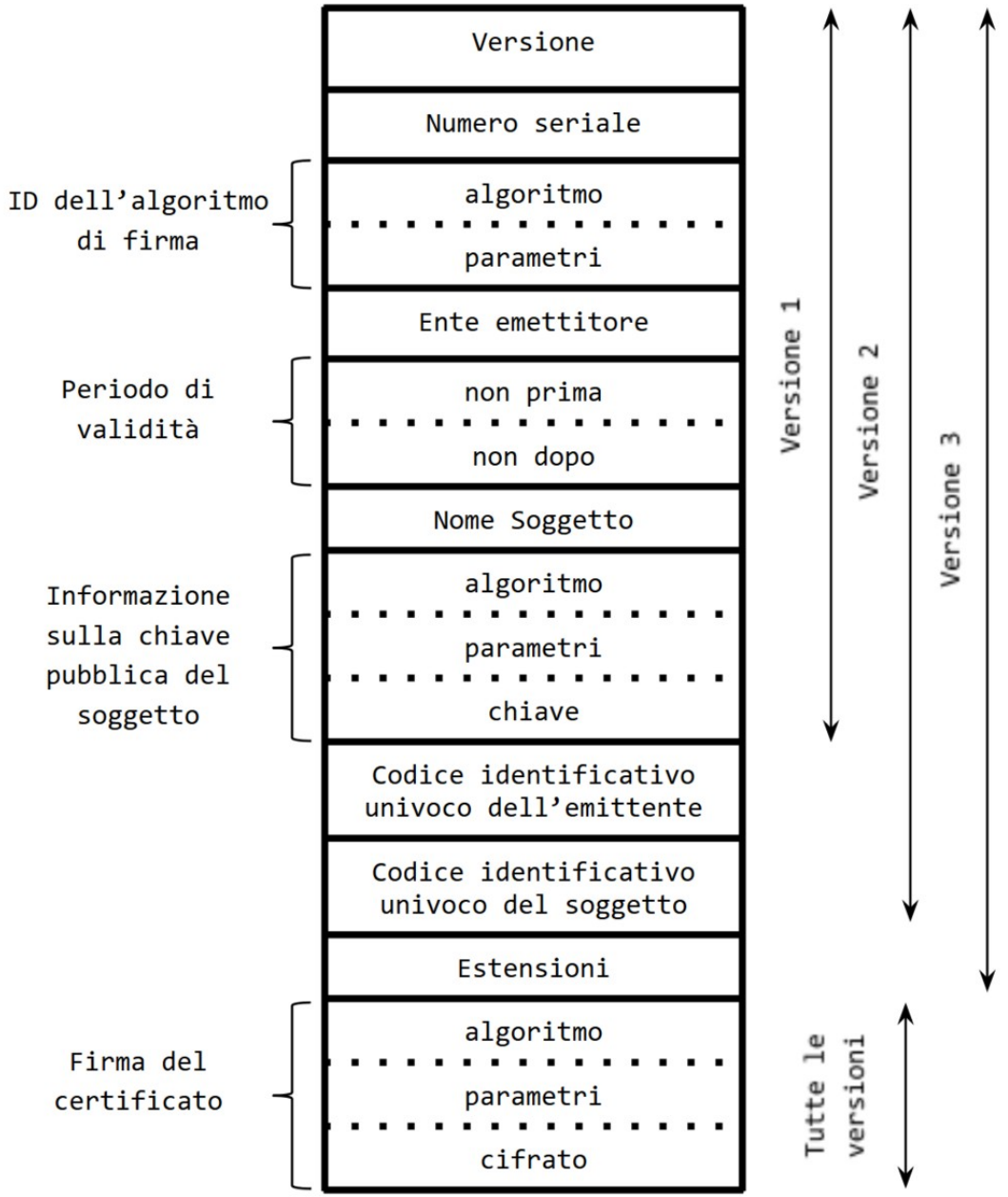
Sessione e connessione

- Due entità che colloquiano utilizzando SSL devono avere aperto una sessione
- Una singola sessione può includere numerose connessioni sicure contemporanee
- Due entità possono avere attive numerose sessioni contemporanee



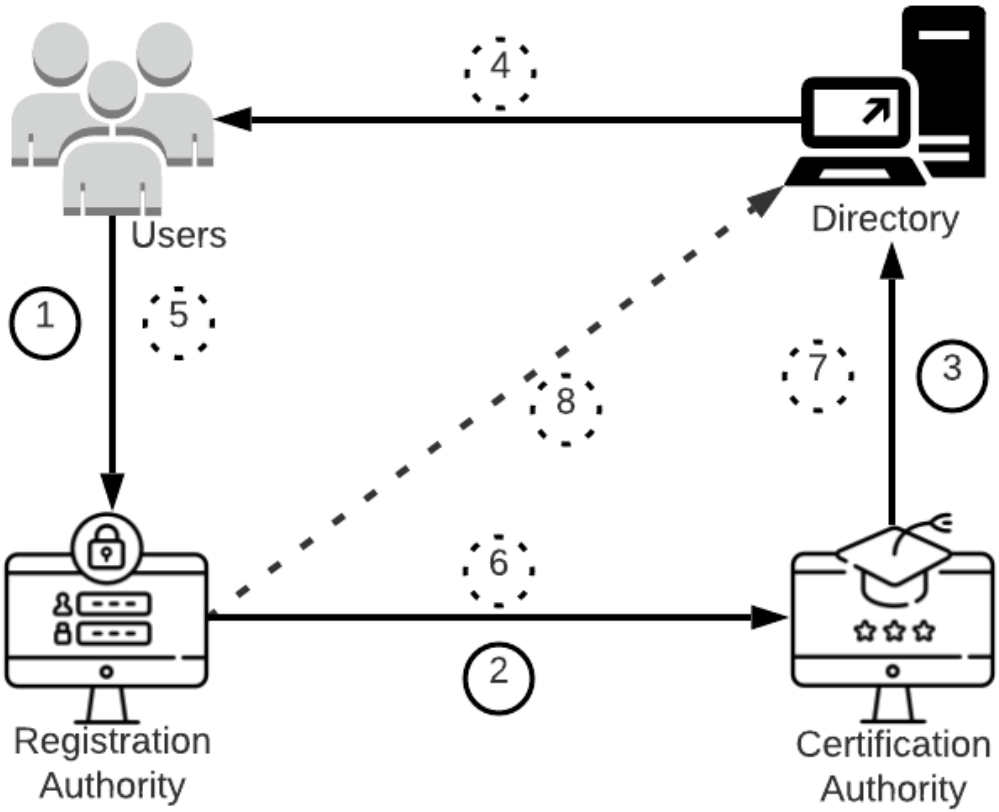
Certificati

- Esiste uno standard per rappresentare l'associazione chiave-titolare attestata da una terza parte: il *certificato di chiave pubblica* (standard ITU-T X.509v3)



Di Andrei Jizdan - opera mia, https://it.wikipedia.org/w/index.php?curid=7795838

Public Key Infrastructure



Initialization

1. Certificate Signing Request (CSR)
2. Approved CSR
3. Certificate Publication

Operation

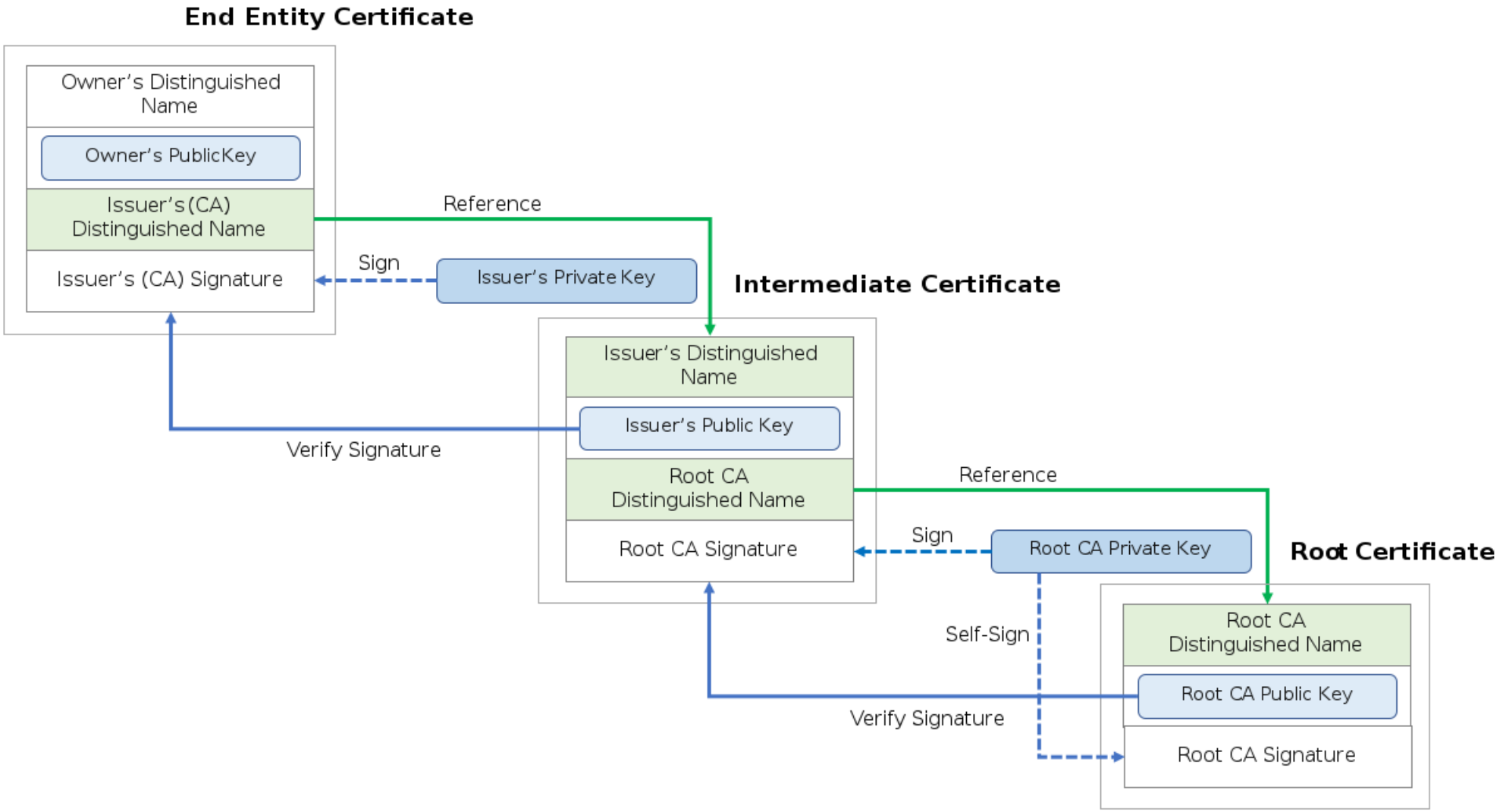
4. Certificate and status Information distribution
5. Revocation/Removal Request (RR)
6. Approved RR
7. Off-line status information publication
8. On-line status information publication

Verifica dei certificati

- **Ricevo un messaggio firmato**
  - Mi procuro il certificato del mittente
  - con la chiave verifico la firma → se ok, messaggio integro e autentico
- **E il certificato è integro e autentico?**
  - è firmato dalla CA
  - mi procuro il certificato della CA
  - con la chiave verifico la firma → se ok, certificato integro e autentico

quando finisce questo iter?

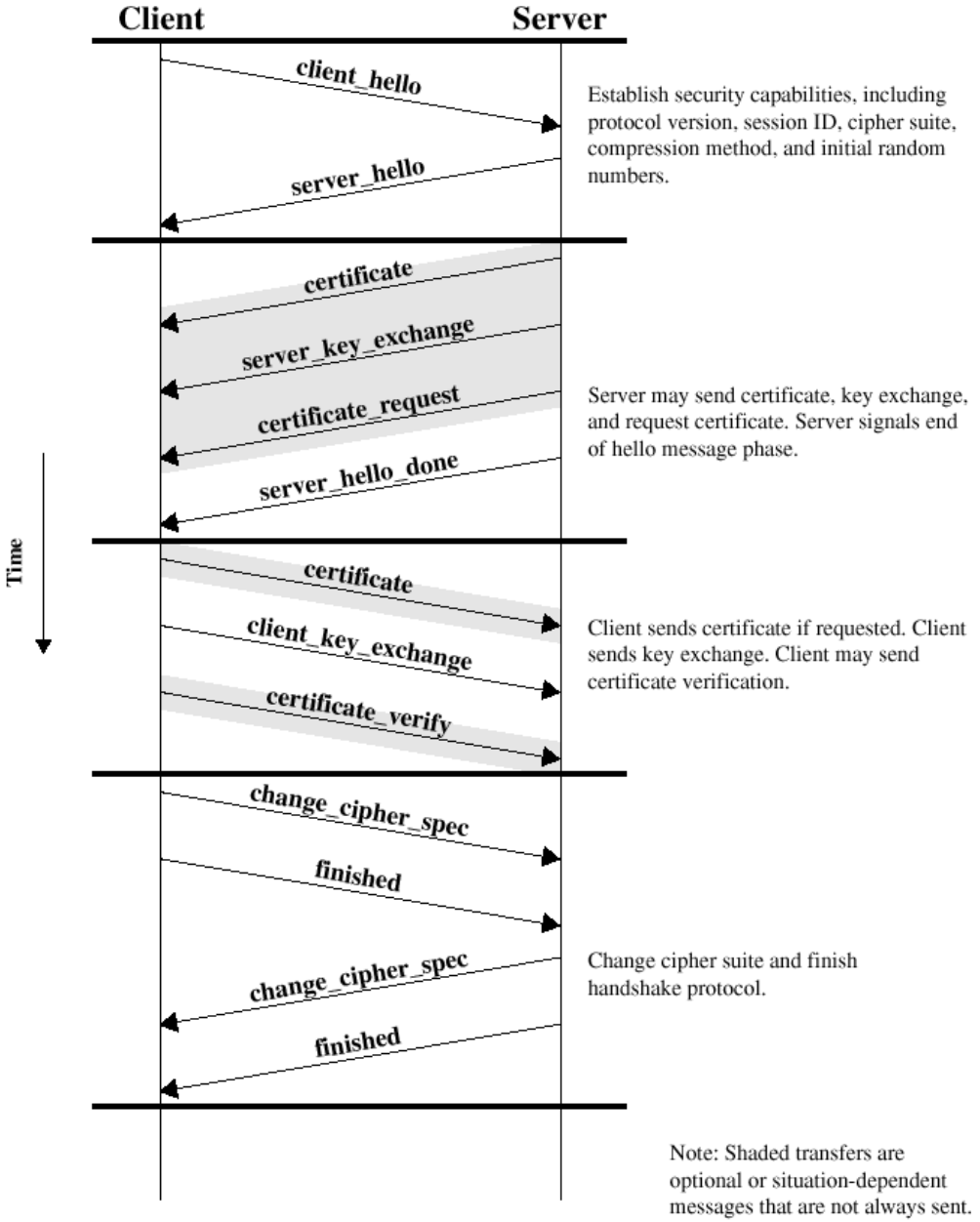
Certificate chain e root CA



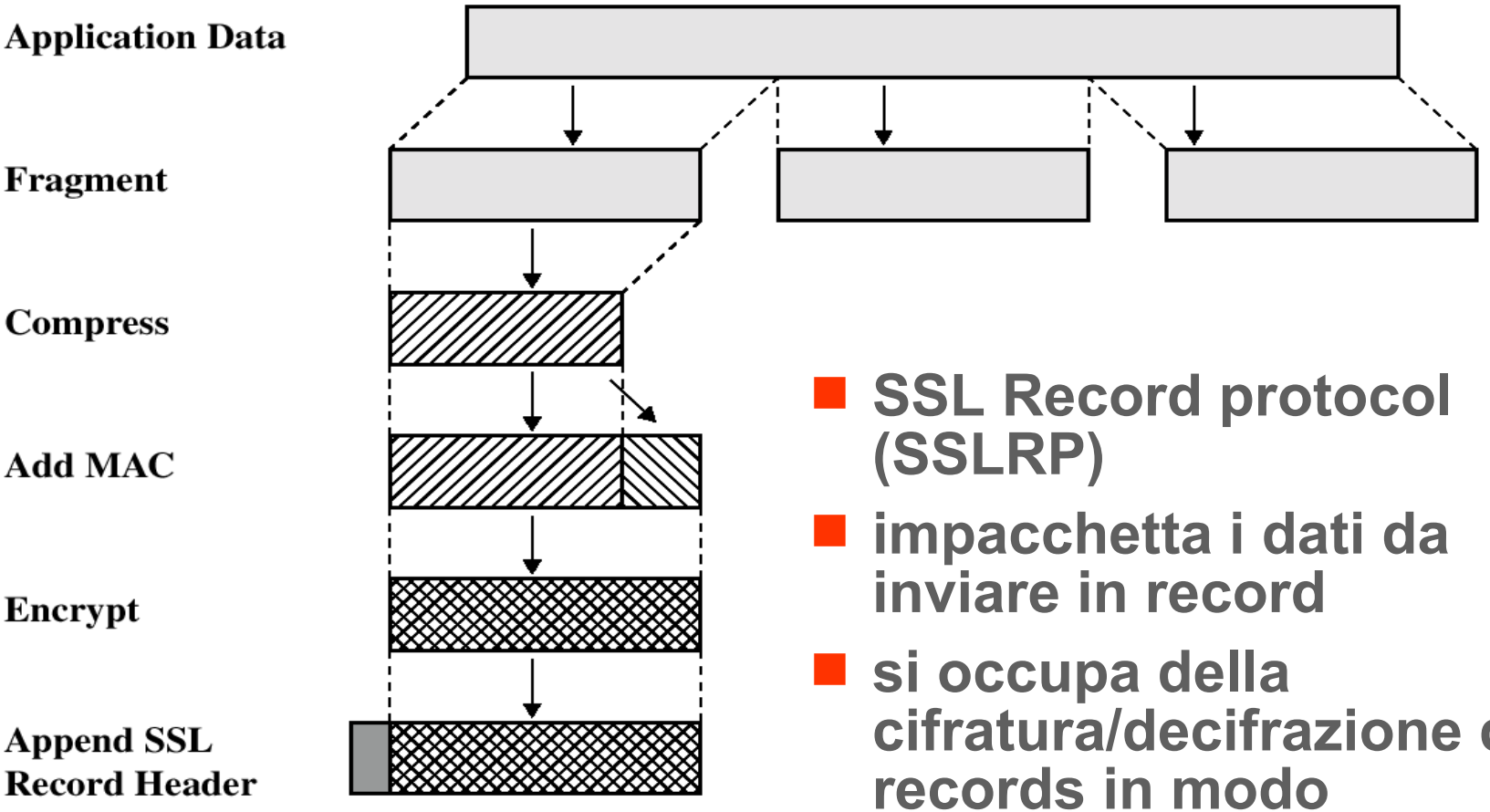
- By Yuhkih - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=94177019

Architettura di SSL – Handshake Protocol

- La parte più complessa di SSL.
- Consente al server ed al client di autenticarsi reciprocamente
  - challenge response basato su crittografia asimmetrica e certificati X.509
  - nelle applicazioni web è comune che solo il server provi la sua autenticità al client
- Negozia gli algoritmi e le chiavi per la cifratura ed i controlli di integrità
- Interviene prima che qualsiasi dato sia trasmesso
- Progettato per limitare il carico di elaborazione e di rete
  - Implementa un meccanismo di chaching delle sessioni per limitare il numero di aperture e quindi il carico di elaborazione
  - Tenta di ridurre al minimo l'attività sulla rete (scambio di messaggi)
  - E' possibile negoziare un numero di sessione: se la comunicazione si interrompe temporaneamente, vengono poi recuperati i parametri della sessione senza rinegoziarli



Architettura di SSL – record protocol

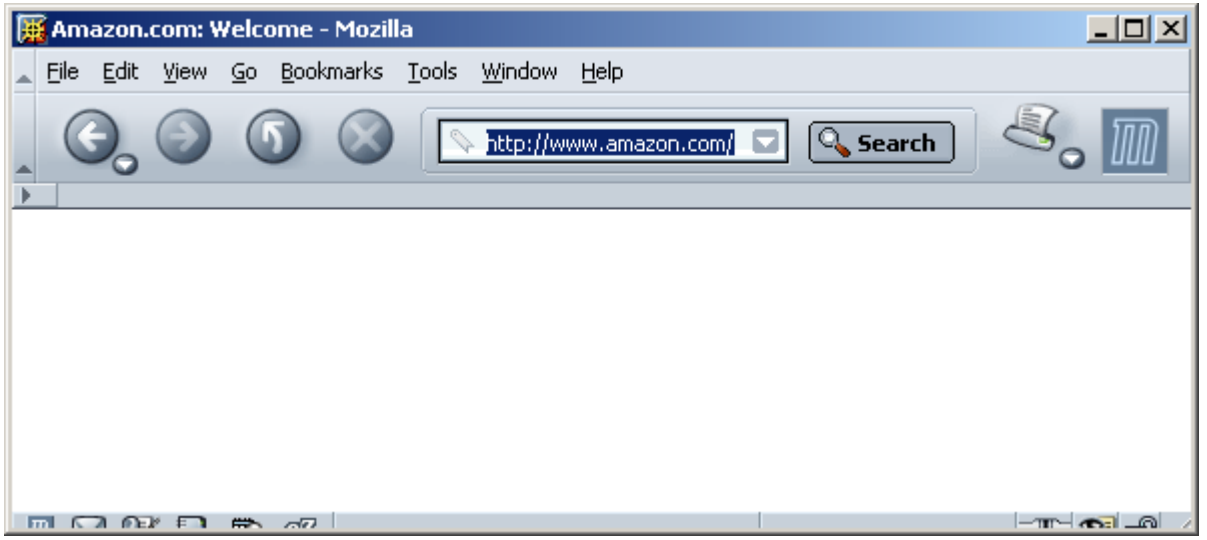


- SSL Record protocol (SSLRP)
- impacchetta i dati da inviare in record
- si occupa della cifratura/decifrazione dei records in modo conforme a quanto negoziato

SSL → TLS

- Transport Layer Security è l'evoluzione di SSL
  - Lo stesso formato di record
  - Definito in RFC 5246 (v1.2) e 8446 (v1.3)
- Simile a SSLv3, ma differenziato in:
  - numero della versione
  - codice di autenticazione del messaggio
  - funzione pseudocasuale
  - codici di avviso
  - suite di cifratura
  - tipi di certificato client
  - certificate_verify e messaggio finito
  - calcoli crittografici
  - padding

HTTPS

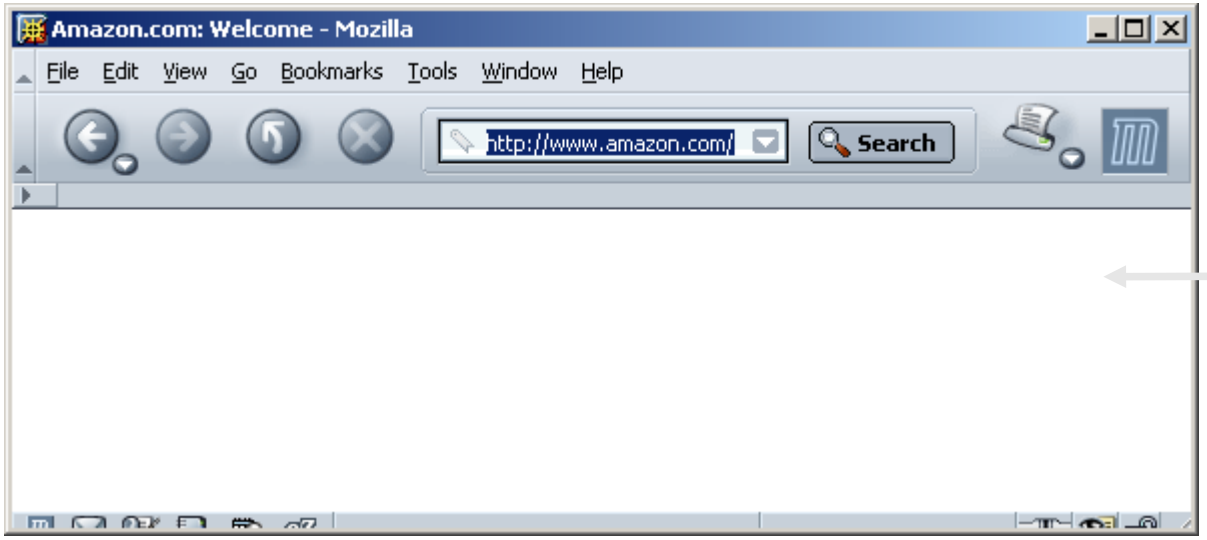


"Provami che hai la chiave privata di amazon"

amazon.com.

Web server @ crackeraddress

HTTPS

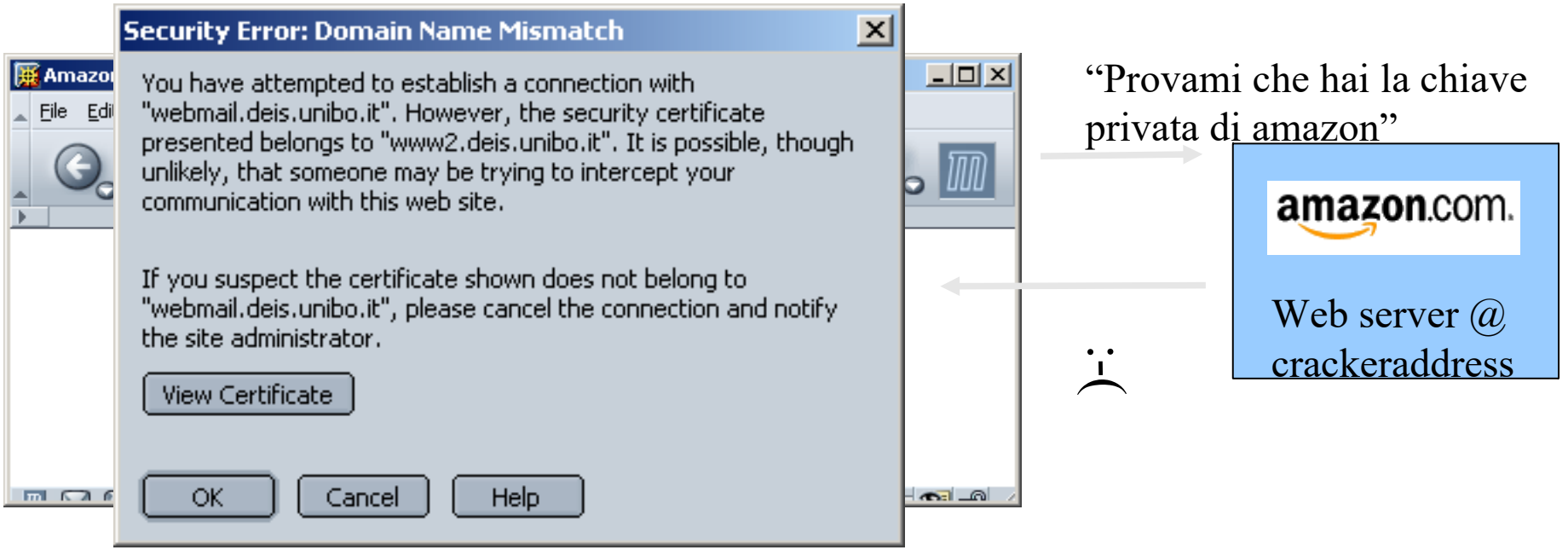


"Provami che hai la chiave privata di amazon"

amazon.com

Web server @ crackeraddress

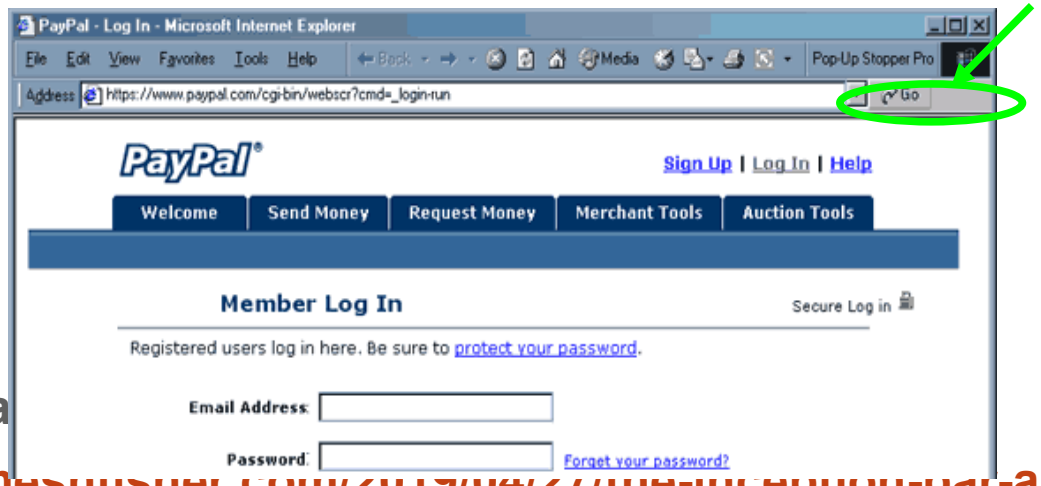
HTTPS



- Come fa il browser a verificare la prova fornita dal web server?
  - Certificate store
  - Trusted CAs

Occultamento della barra degli indirizzi

Il vecchio: qualche riga di codice js o activeX

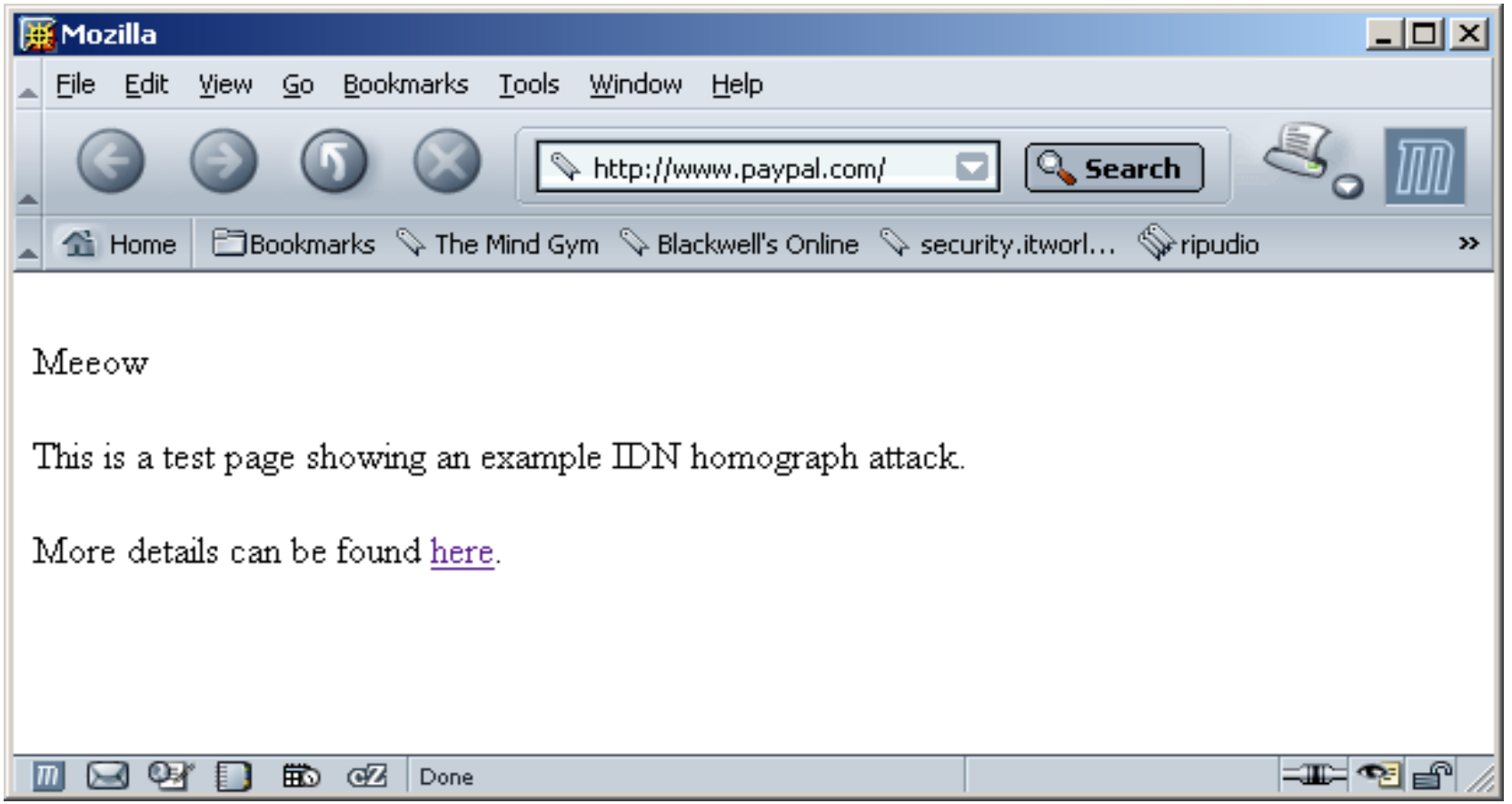


Il nuovo: a

https://jamesfmsher.com/2019/04/27/the-inception-bar-a-new-phishing-method/

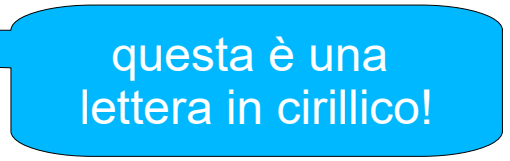
Occultamento dell'URL

RFC3490/1/2: International Domain Names
e attacci omografici

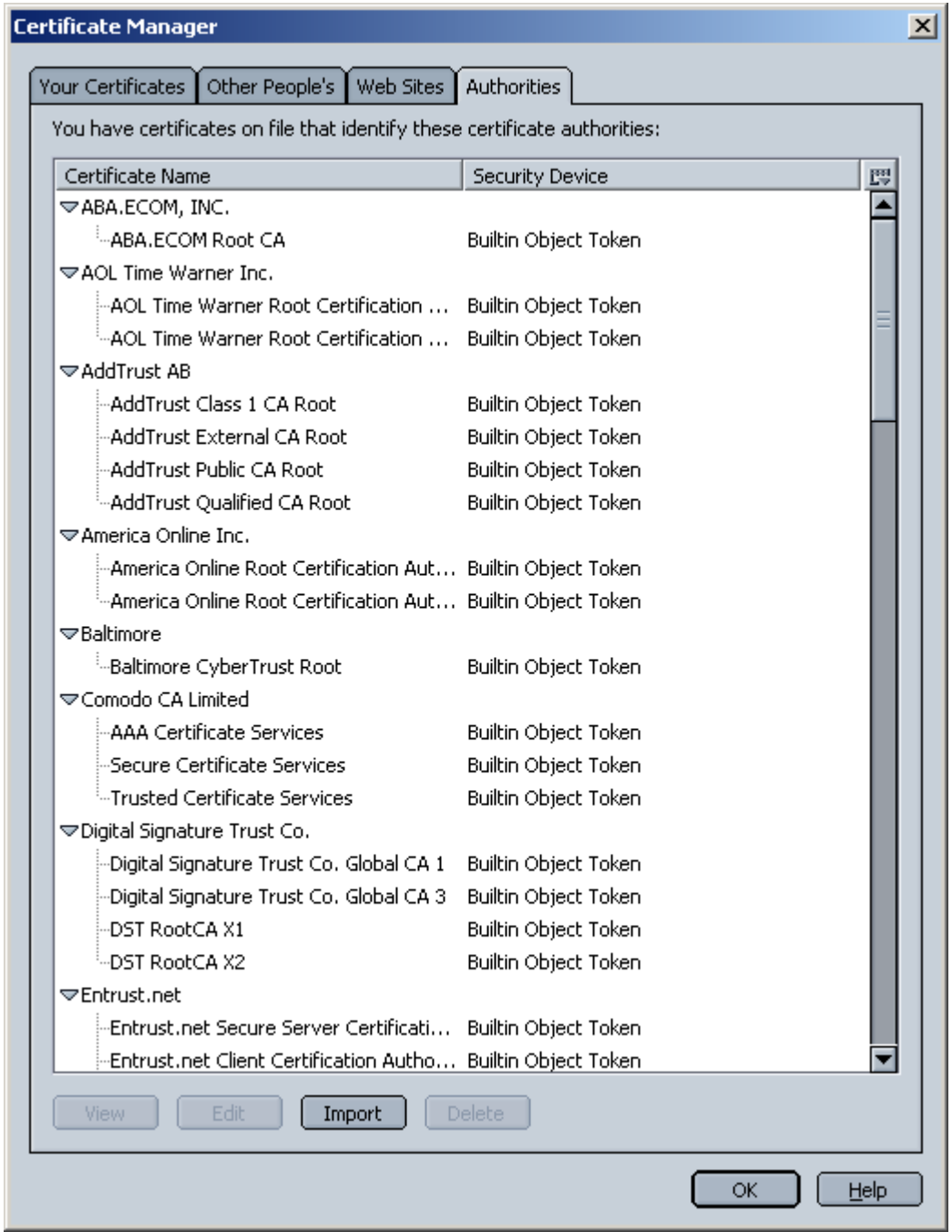


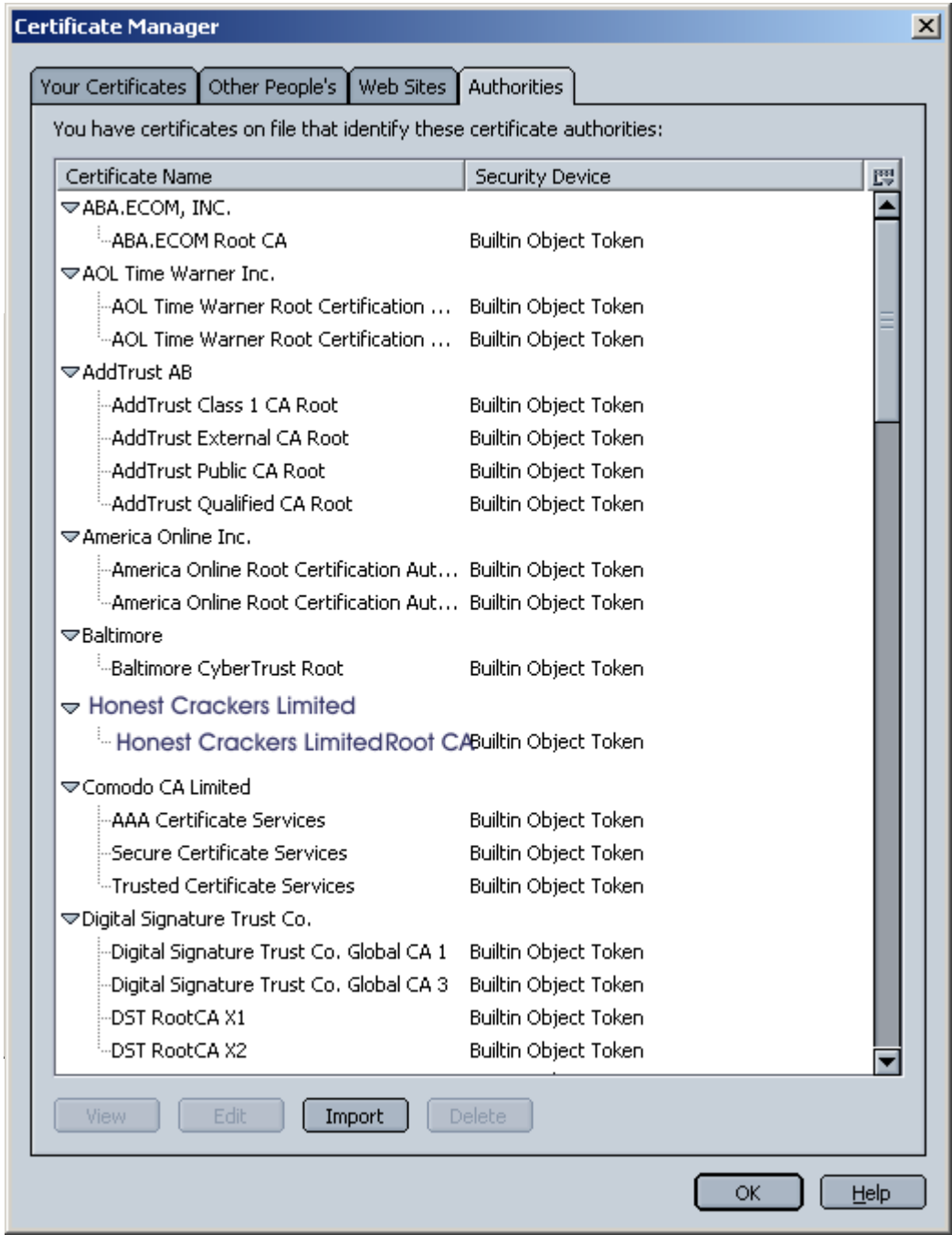
Attacchi omografici

- Problema strutturale
  - IDN consente nomi di dominio in alfabeti diversi dal latino di base
  - Il sistema DNS supporta solo il latino di base
- Soluzione: punycode
  - conversione dei caratteri internazionali in sequenze di caratteri base
    - es: 點看 → xn--c1yn36f
  - ma ci sono caratteri identici (omografi) a quelli latini
    - es: paypal.com → xn--pypal-4ve.com
- Non si può impedire a un attaccante
  - di registrare il dominio xn--pypal-4ve.com
    - nessuna necessità di DNS hijacking
  - di ottenere un legittimo certificato X.509 a tale nome
    - corretto funzionamento di HTTPS
- Soluzione più comune lato browser: visualizzare il punycode invece del font internazionale che potrebbe trarre in inganno



Iniezione di CA nel certificate store





Fake certificates

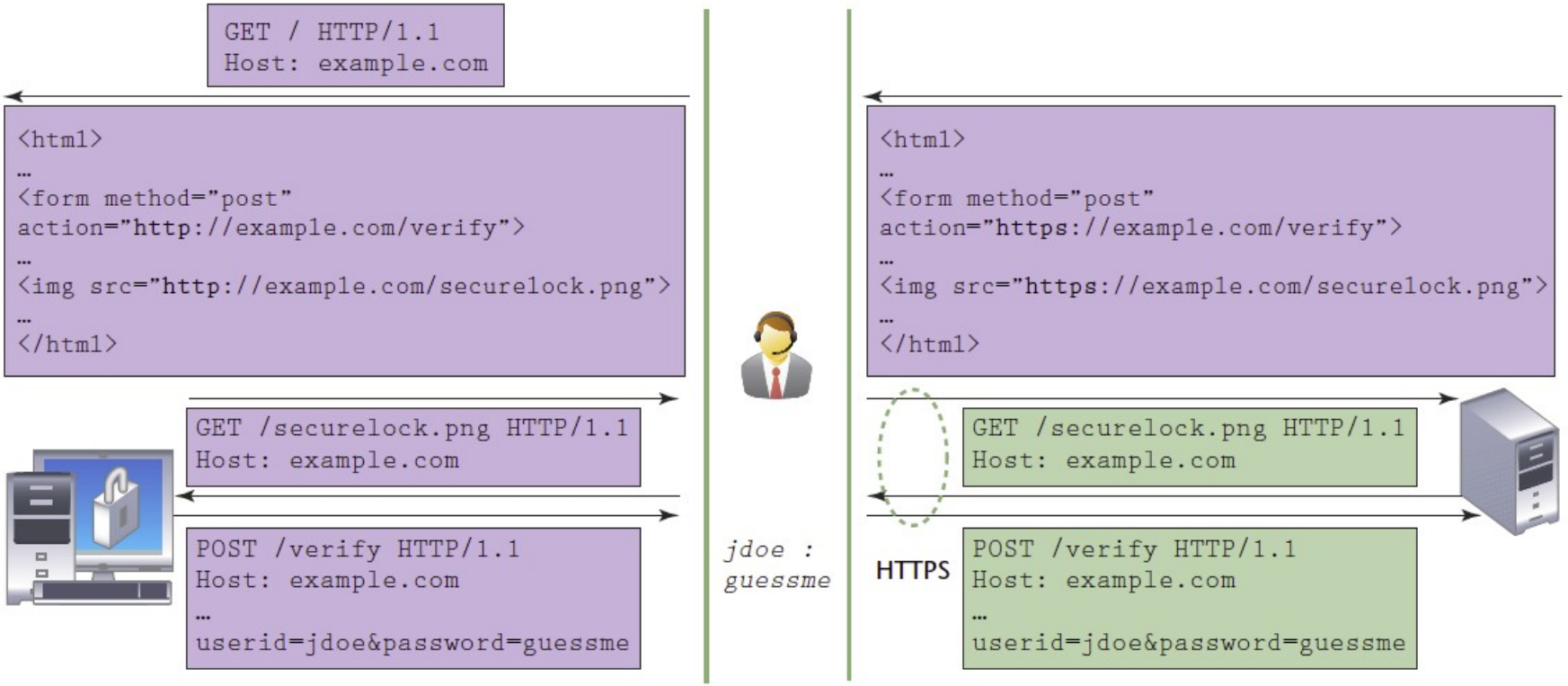
- Il modo più "semplice" di impersonare un sito è falsificare il certificato

  https://www.thesslstore.com/blog/what-is-a-rogue-certificate/

  - raro ma pericolosissimo: compromissione di una CA
  - se non emesso da un'autorità riconosciuta servono vari sotterfugi (vedi seguito)
- *HTTP Public Key Pinning (HPKP - RFC7469)*
  - limita le chiavi associabili a un dominio
    - root, intermediate, o end-entity
  - dichiarato nell'header HTTP
    - `Public-Key-Pins`
    - `Public-Key-Pins-Report-Only`
  - deprecato dal team di Google Chrome
- *Certificate Transparency (CT – RFC 6962)*
  - un framework aperto per scrutinare il processo di rilascio dei certificati; dal sito ufficiale di Google:
    - Make it impossible (or at least very difficult) for a CA to issue a SSL certificate for a domain without the certificate being visible to the owner of that domain.
    - Provide an open auditing and monitoring system that lets any domain owner or CA determine whether certificates have been mistakenly or maliciously issued.
    - Protect users (as much as possible) from being duped by certificates that were mistakenly or maliciously issued.

Stripping

- pagine HTTP che inviano dati sensibili via HTTPS possono essere modificate da un MITM



Mitigazione

- HSTS (HTTP Strict Transport Security)
  - policy implementata dai server per forzare i browser a interagire via HTTPS
  - RFC 6797
- Campo nell'header della risposta
  - inefficace sulla prima richiesta!
- Attacchi simili per i quali non c'è un equivalente di HSTS: STARTTLS command injection
  - connessioni che partono insicure e chiedono l'upgrade
  - MITM interferisce o accoda comandi in ordine errato

HTTP security - riferimenti

- Iniziative per evitare connessioni in chiaro
  - https://tools.ietf.org/html/rfc6797
  - https://www.eff.org/it/https-everywhere
  - https://tools.ietf.org/html/rfc8555
  - https://letsencrypt.org/
- HSTS / pinning / CT
  - https://tools.ietf.org/html/rfc7469
  - https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning
  - https://tools.ietf.org/html/rfc6962
  - https://www.certificate-transparency.org/
- address bar
  - https://en.wikipedia.org/wiki/IDN_homograph_attack
  - https://www.netsparker.com/blog/web-security/web-browser-address-bar-spoofing/
- TLS – dalle basi alle vulnerabilità
  - https://www.acunetix.com/blog/articles/tls-security-what-is-tls-ssl-part-1/
    - primo di 6 articoli
  - https://tools.ietf.org/html/rfc7457