Public Key pdf 23

2 dicembre 2025

Indice

1 Introduzione alla Crittografia Asimmetrica

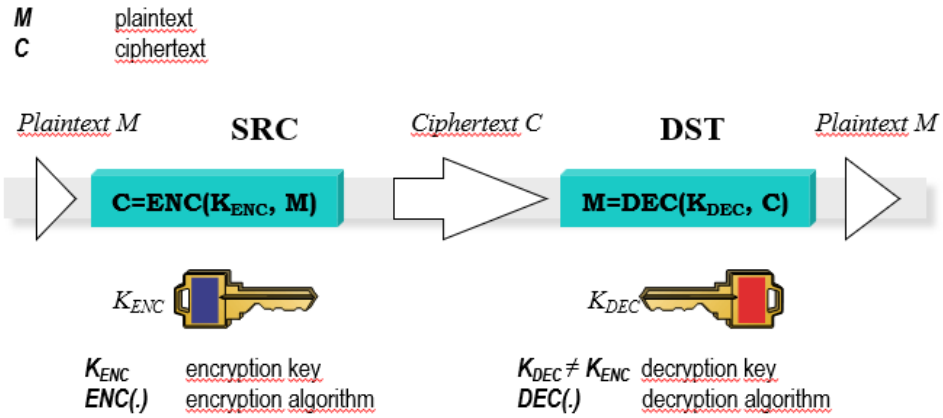
La slide illustra i concetti fondamentali della crittografia asimmetrica (nota anche come crittografia a chiave pubblica), evidenziando il flusso di trasmissione dei dati e le relazioni matematiche tra le chiavi coinvolte.

1.1 Componenti del Sistema

Il sistema crittografico viene descritto attraverso le seguenti variabili fondamentali:

- **M (Plaintext):** Il messaggio in chiaro originale, leggibile.
- **C (Ciphertext):** Il messaggio cifrato, reso incomprensibile per proteggerne il contenuto durante la trasmissione.
- **SRC (Source):** La sorgente o mittente del messaggio.
- **DST (Destination):** La destinazione o destinatario del messaggio.

1.2 Il Processo di Cifratura e Decifratura



A differenza della crittografia simmetrica, questo schema utilizza due chiavi distinte per le operazioni di cifratura e decifratura. Il processo, visualizzato nella figura da sinistra a destra, avviene come segue:

1. **Cifratura (Encryption):** La sorgente (SRC) utilizza un algoritmo di cifratura, denotato come $\mathrm{ENC}(\cdot)$, e una chiave di cifratura K_{ENC}. Applicando l'algoritmo al messaggio in chiaro M, si ottiene il testo cifrato C. L'operazione è matematicamente espressa come:
$$C = \mathrm{ENC}(K_{\mathrm{ENC}}, M) \tag{1}$$
2. **Decifratura (Decryption):** Una volta che il testo cifrato C raggiunge la destinazione (DST), il destinatario utilizza un algoritmo di decifratura, denotato come $\mathrm{DEC}(\cdot)$, e una chiave di decifratura K_{DEC}. Questa operazione inverte il processo, recuperando il messaggio originale M:
$$M = \mathrm{DEC}(K_{\mathrm{DEC}}, C) \tag{2}$$

1.3 Proprietà delle Chiavi

L'aspetto distintivo della crittografia asimmetrica risiede nella natura delle chiavi utilizzate, rappresentate graficamente con colori diversi (blu per la cifratura, rosso per la decifratura) per enfatizzarne la distinzione:

- **Asimmetria:** Le chiavi sono matematicamente diverse, ovvero:

  $$K_{\text{ENC}} \neq K_{\text{DEC}}$$

  Generalmente, K_{ENC} è la *chiave pubblica* (nota a tutti), mentre K_{DEC} è la *chiave privata* (segreta e posseduta solo dal destinatario).

- **Relazione Matematica:** Come indicato nella nota finale della slide, sebbene le chiavi di cifratura e decifratura siano intrinsecamente legate tra loro (poiché ciò che una cifra, l'altra deve decifrare), esiste una proprietà di sicurezza fondamentale:

  > Non è computazionalmente fattibile determinare K_{DEC} partendo dalla conoscenza di K_{ENC}.

  Questa impossibilità di derivazione (a meno che non si disponga di informazioni privilegiate o "trapdoor information") è ciò che garantisce la sicurezza del sistema, permettendo la distribuzione pubblica della chiave di cifratura senza compromettere la segretezza della chiave di decifratura.

2 Modello di Utilizzo Pratico: HTTPS/TLS

La slide "Practical usage model #1" illustra l'applicazione concreta dei principi crittografici nei protocolli di rete moderni, prendendo come riferimento lo standard **HTTPS/TLS**. Questo modello rappresenta un approccio *ibrido*, che combina la sicurezza della crittografia asimmetrica per lo scambio delle chiavi con l'efficienza della crittografia simmetrica per il trasferimento dei dati. Il ciclo di vita di una sessione sicura viene suddiviso in tre fasi distinte:

2.1 Prima Fase: Handshake (Negoziazione)

Questa è la fase di segnalazione iniziale ed è l'unico momento in cui viene utilizzata pesantemente la **crittografia asimmetrica**.

- **Obiettivo:** Stabilire un *Shared Secret* (segreto condiviso) tra client e server in un canale non sicuro.
- **Funzionamento:** Il protocollo utilizza coppie di chiavi pubblica/privata per autenticare le parti (solitamente il server tramite certificati digitali) e per negoziare in sicurezza il segreto condiviso.
- **Importanza:** Sebbene computazionalmente costosa, questa fase è necessaria solo all'inizio per garantire che solo le parti legittime conoscano il segreto.

2.2 Seconda Fase: Data Transfer (Trasferimento Dati)

Una volta stabilito il segreto condiviso, il sistema passa alla crittografia simmetrica, molto più veloce ed efficiente per grandi moli di dati.

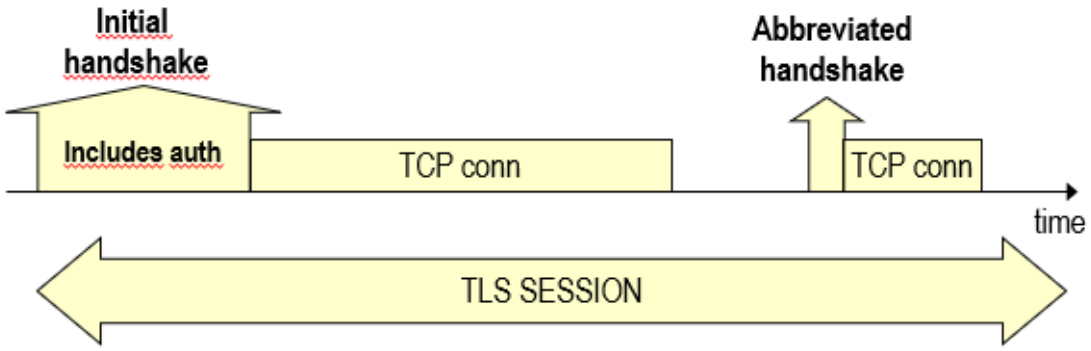
- **Derivazione delle Chiavi:** Il *Shared Secret* non viene usato direttamente per cifrare i dati. Viene invece utilizzato come input per una **KDF** (Key Derivation Function).
- **Chiavi Prodotte:** Dalla KDF si ottengono due tipi di chiavi simmetriche:
  1. *Symmetric Encryption Keys:* Per garantire la confidenzialità (cifratura del payload).
  2. *Message Authentication Keys:* Per garantire l'integrità e l'autenticità dei messaggi (evitando manomissioni).

2.3 Fasi Successive: Rekeying (Rinnovo delle Chiavi)

Per mantenere un elevato livello di sicurezza durante sessioni lunghe, le chiavi non rimangono statiche.

- **Refresh:** Il protocollo prevede il rinnovo periodico (*rekeying*) delle chiavi di cifratura e autenticazione.
- **Meccanismo:** Si continua a utilizzare il *Shared Secret* originale, ma la KDF viene alimentata con nuovi *nonces* (numeri casuali usati una sola volta). Questo garantisce che, anche se una chiave di sessione venisse compromessa, le comunicazioni precedenti o future resterebbero protette (principio simile alla *Perfect Forward Secrecy*).

2.4 Analisi Temporale della Sessione TLS



Il diagramma temporale mostra la distinzione tra una **Connessione TCP** e una **Sessione TLS**:

- **Initial Handshake:** Richiede l'autenticazione completa ed è più oneroso in termini di tempo. Apre la sessione TLS.
- **Abbreviated Handshake:** Se una sessione TLS è già attiva o memorizzata in cache, è possibile avviare nuove connessioni TCP utilizzando un handshake abbreviato (Session Resumption), riducendo la latenza poiché non è necessario ripetere la pesante crittografia asimmetrica iniziale.

3 Modello di Utilizzo Pratico #2: Crittografia Ibrida (Hybrid Encryption)

La slide presenta un secondo approccio fondamentale all'uso combinato di crittografia simmetrica e asimmetrica, progettato specificamente per la trasmissione efficiente di grandi moli di dati (*long messages*) o per comunicazioni asincrone (dove mittente e destinatario non sono necessariamente online contemporaneamente).

3.1 Il Processo di Cifratura Ibrida

Il meccanismo si articola in tre passaggi sequenziali, mirati a combinare la velocità degli algoritmi simmetrici con la sicurezza nella distribuzione delle chiavi degli algoritmi asimmetrici:

1. **Generazione della Chiave di Sessione (K):** Il mittente genera una chiave simmetrica K in modo casuale (random). Questa chiave è temporanea e verrà utilizzata esclusivamente per questo specifico messaggio o sessione.

2. **Cifratura del Messaggio (Data Encapsulation):** Utilizzando la chiave K appena generata e un cifrario simmetrico standard ed efficiente (nella slide viene citato l'**AES**, *Advanced Encryption Standard*), il mittente cifra il messaggio in chiaro m (spesso di grandi dimensioni).

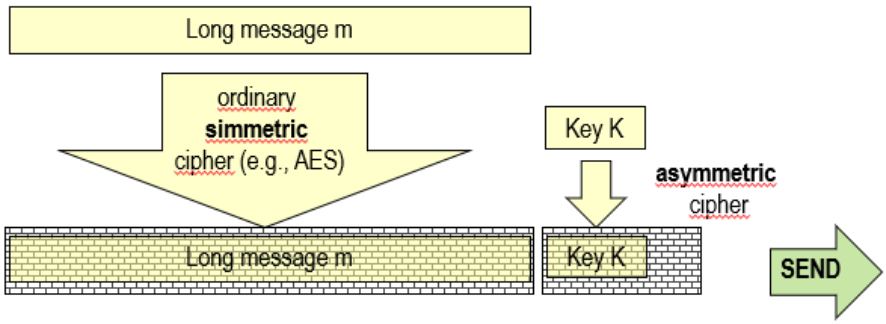
   $$C_{\text{dati}} = \text{ENC}_{\text{simmetrica}}(K, m)$$

   L'uso della crittografia simmetrica qui è cruciale per garantire prestazioni elevate nel trattamento del payload.

3. **Cifratura della Chiave (Key Encapsulation):** Per permettere al destinatario di leggere il messaggio, il mittente deve trasmettere anche la chiave K. Tuttavia, non può inviarla in chiaro. Pertanto, cifra la chiave K utilizzando un algoritmo asimmetrico e la chiave pubblica del destinatario.

   $$C_{\text{chiave}} = \text{ENC}_{\text{asimmetrica}}(\text{ChiavePubblica}_{\text{DST}}, K)$$

   Il pacchetto finale inviato ("SEND") comprende sia il messaggio cifrato che la chiave cifrata.



3.2 Applicazione Reale: 5G e Privacy

La slide evidenzia un'applicazione moderna e critica di questo schema: la protezione dell'identità nelle reti mobili di quinta generazione (**5G**).

- **ECIES (Elliptic Curve Integrated Encryption Scheme):** È lo standard crittografico ibrido citato. Utilizza le curve ellittiche per la parte asimmetrica (cifratura della chiave) per ridurre l'overhead computazionale e la dimensione delle chiavi rispetto a RSA.
- **5G SUCI (Subscription Concealed Identifier):** Nelle reti precedenti (4G/3G), l'identificativo dell'utente (IMSI) veniva spesso trasmesso in chiaro durante la prima connessione, rendendo possibile il tracciamento degli utenti (tramite *IMSI Catchers*). Nel 5G, grazie a questo schema ibrido, l'identificativo permanente (SUPI) viene cifrato utilizzando la chiave pubblica della rete domestica dell'operatore prima di essere trasmesso via etere. L'identificativo cifrato prende il nome di **SUCI**. Questo impedisce a chiunque intercetti il segnale radio di conoscere l'identità dell'abbonato, garantendo un livello superiore di privacy.

4 I Due Utilizzi della Crittografia a Chiave Pubblica

La slide "PubKey crypto: two ways to use it" esplora la versatilità degli algoritmi asimmetrici (come RSA). La premessa matematica fondamentale per questa dualità è la **proprietà commutativa** delle operazioni. Normalmente, sappiamo che decifrare un messaggio cifrato restituisce il testo originale:

$$\text{DEC}(\text{ENC}(M)) = M$$

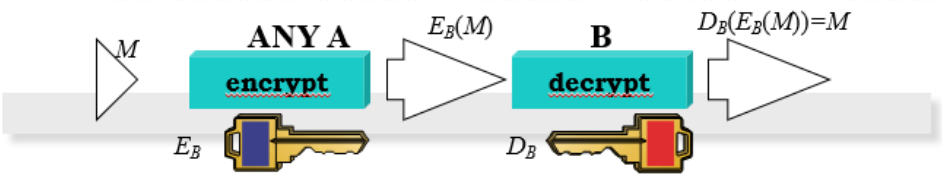
Per il funzionamento della firma digitale, assumiamo che l'algoritmo sia commutativo, ovvero che applicando le operazioni in ordine inverso il risultato non cambi:

$$\text{ENC}(\text{DEC}(M)) = M$$

Sfruttando questa proprietà, possiamo configurare il sistema in due modalità distinte:

4.1 1. Cifratura a Chiave Pubblica (Confidenzialità)

Questo è l'utilizzo tradizionale per garantire la segretezza del messaggio, illustrato nella parte superiore della slide.



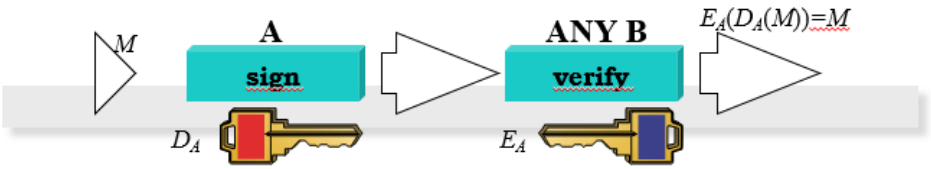
- **Scenario:** Chiunque ("ANY A") vuole inviare un messaggio segreto a B.
- **Processo:**
  1. Il mittente A utilizza la **Chiave Pubblica di B** (E_B, chiave blu) per cifrare il messaggio: $C = E_B(M)$.

2. Il messaggio viaggia cifrato.
3. Il destinatario B utilizza la propria **Chiave Privata** (D_B, chiave rossa) per decifrare: $M = D_B(C)$.

- **Garanzia:** *Confidenzialità.* Solo B, l'unico possessore della chiave privata D_B, può leggere il messaggio.

4.2 2. Firma Digitale (Autenticazione e Integrità)

Questo è l'approccio "DUALE", illustrato nella parte inferiore della slide. Qui i ruoli delle chiavi si invertono per garantire l'autenticità del mittente.



- **Scenario:** Il mittente A vuole dimostrare a chiunque ("ANY B") che il messaggio proviene indiscutibilmente da lui.
- **Processo:**
  1. Il mittente A utilizza la propria **Chiave Privata** (D_A, chiave rossa) per trasformare il messaggio. In questo contesto, l'operazione di "decifratura" matematica agisce come una **Firma**:

$$S = \text{Sign}(M) = D_A(M)$$

  2. Il destinatario B (chiunque possieda la chiave pubblica di A) utilizza la **Chiave Pubblica di A** (E_A, chiave blu) per verificare la firma. Matematicamente, applica l'operazione di "cifratura" al testo firmato:

$$M' = \text{Verify}(S) = E_A(S)$$

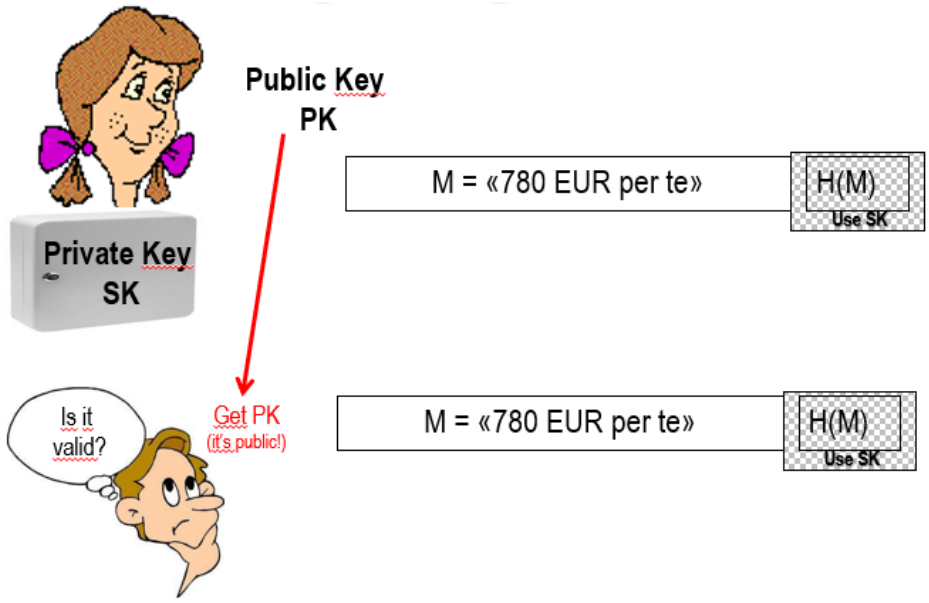
- **Garanzia:** *Autenticazione e Non Ripudio.* Se l'operazione $E_A(S)$ restituisce un messaggio M di senso compiuto (o corrispondente al messaggio inviato in chiaro), significa che S poteva essere stato generato *solo* da qualcuno in possesso di D_A. Poiché solo A possiede D_A, la firma è autentica.

  Nota Tecnica: Sebbene la slide mostri la firma dell'intero messaggio M per semplicità concettuale, nella pratica reale (per motivi di efficienza e sicurezza) si firma solitamente l'*hash* del messaggio ($H(M)$) e non il messaggio intero.

  Non usate la stessa chiave per l' RSA e per la Sign

5 Firma Digitale: Processo con Hashing

La slide illustra un esempio concreto di applicazione della firma digitale per una transazione economica (un messaggio contenente "780 EUR per te"). Rispetto allo schema teorico generale, qui viene introdotto un componente fondamentale per l'efficienza e la sicurezza: la funzione di **Hash** (H). Il processo viene descritto attraverso l'interazione tra due parti (Mittente e Destinatario) e l'uso delle chiavi asimmetriche.



5.1 Componenti Chiave

- **PK (Public Key):** La chiave pubblica del mittente, accessibile a chiunque.
- **SK (Secret Key):** La chiave privata (o segreta) del mittente, custodita gelosamente (rappresentata da una cassaforte).
- **M (Messaggio):** Il contenuto della transazione, ad esempio "780 EUR per te".
- **H(M):** L'impronta digitale (digest) del messaggio calcolata tramite una funzione di hash.

5.2 Il Flusso della Firma (Lato Mittente)

Nella parte superiore dell'immagine, il mittente prepara il pacchetto da inviare. Invece di cifrare l'intero messaggio M con la chiave privata (operazione computazionalmente lenta per messaggi lunghi), avviene quanto segue:

1. Viene calcolato l'hash del messaggio: $h = H(M)$.
2. Viene applicata la firma **solo sull'hash** utilizzando la chiave privata:

$$\text{Firma} = \text{ENC}_{SK}(H(M))$$

   Questo passaggio è rappresentato dal riquadro etichettato "H(M) Use SK".

3. Il mittente invia al destinatario il messaggio in chiaro M accompagnato dalla firma digitale.

5.3 Il Flusso di Verifica (Lato Destinatario)

Nella parte inferiore, il destinatario riceve il messaggio e la firma e si pone la domanda: "*Is it valid?*" (È valido?). Per rispondere, esegue la verifica:

1. **Recupero Chiave:** Il destinatario ottiene la chiave pubblica (PK) del mittente ("Get PK, it's public!").
2. **Verifica Matematica:** Il destinatario esegue due operazioni parallele:
   - Calcola autonomamente l'hash del messaggio in chiaro ricevuto: $H(M)_{locale}$.
   - Decifra la firma ricevuta usando la chiave pubblica PK per estrarre l'hash originale inviato dal mittente:

$$H(M)_{estratto} = \mathrm{DEC}_{PK}(\text{Firma})$$

3. **Confronto:** Se $H(M)_{locale} == H(M)_{estratto}$, la firma è valida.

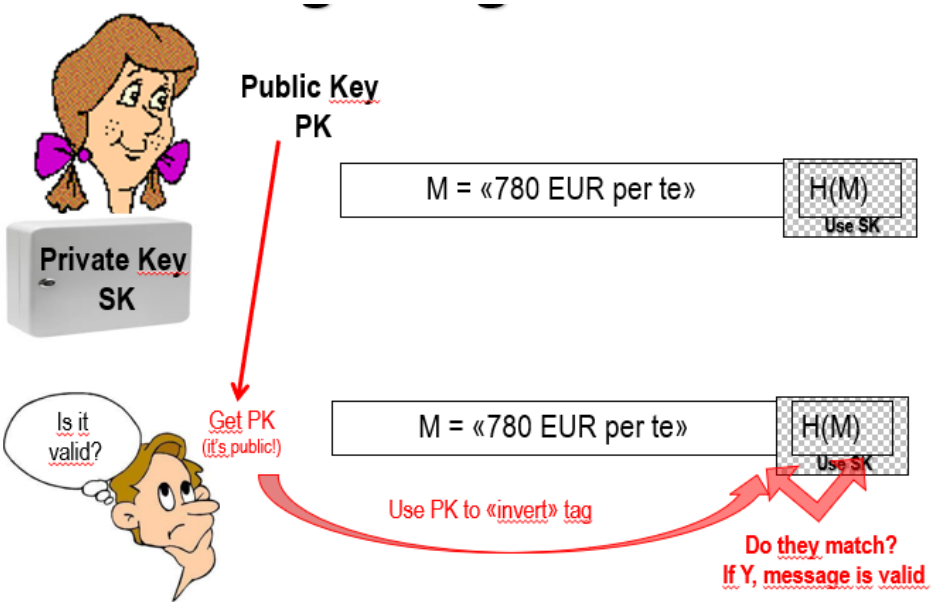
5.4 Garanzie di Sicurezza

Questo processo garantisce tre proprietà fondamentali:

- **Autenticità:** Solo il possessore di SK poteva generare una firma che si decifra correttamente con PK.
- **Integrità:** Se il messaggio "780 EUR" fosse stato modificato in "7800 EUR" durante il transito, l'hash calcolato dal destinatario non corrisponderebbe a quello contenuto nella firma, invalidando la transazione.
- **Non Ripudio:** Il mittente non può negare di aver inviato il messaggio, poiché la firma è matematicamente legata alla sua chiave privata.

6 Firma Digitale: La Fase di Verifica

Questa slide completa il concetto di firma digitale illustrando, tramite il percorso evidenziato in rosso, i passaggi critici che il destinatario deve compiere per validare l'autenticità e l'integrità del messaggio. Il processo risponde alla domanda fondamentale posta nel diagramma: *"Is it valid?"*.



Il protocollo di verifica si basa su tre azioni sequenziali:

6.1 1. Acquisizione della Chiave Pubblica

Il primo passo (indicato dalla freccia rossa verticale "Get PK") consiste nel reperire la chiave necessaria per la verifica.

- Poiché si tratta di crittografia asimmetrica, il destinatario non ha bisogno di aver scambiato segreti in precedenza.
- Deve semplicemente ottenere la **Chiave Pubblica (PK)** del mittente. Come nota la slide (*"it's public!"*), questa chiave è liberamente accessibile e non richiede canali segreti per la distribuzione.

6.2 2. Inversione del "Tag" (Firma)

La seconda azione (freccia curva inferiore: *"Use PK to invert tag"*) è il cuore matematico della verifica.

- Il destinatario applica la Chiave Pubblica (PK) al "Tag" crittografico (la firma) allegato al messaggio.
- Il termine "invertire" è usato perché l'operazione annulla la trasformazione fatta inizialmente dal mittente con la Chiave Privata (SK).
- Matematicamente, se il tag era $S = \text{ENC}_{SK}(H(M))$, il destinatario calcola:

$$\text{DEC}_{PK}(S) \rightarrow H(M)_{\text{originale}}$$

  Questo rivela l'impronta hash che il mittente ha calcolato al momento dell'invio.

6.3 3. Il Confronto (Matching)

L'ultimo passaggio è il test logico indicato dalla domanda *"Do they match?"*. Il destinatario possiede ora due valori da confrontare:

1. L'hash estratto dalla firma (che rappresenta "ciò che il mittente dice di aver inviato").
2. L'hash calcolato localmente sul messaggio M ricevuto in chiaro ("780 EUR per te").

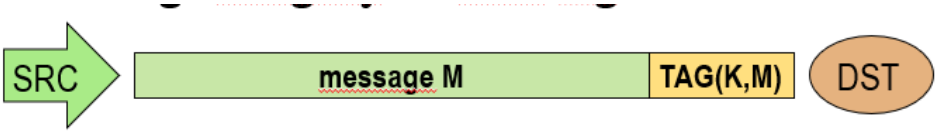
Esito della verifica:

If Y (Yes), message is valid: Se i due hash coincidono perfettamente, si ha la certezza matematica che:

- Il messaggio non è stato alterato (Integrità).
- Il messaggio è stato firmato indiscutibilmente dal proprietario della chiave privata associata a quella pubblica utilizzata (Autenticità).

7 Integrità del Messaggio e Autenticazione della Sorgente

La slide analizza la differenza cruciale tra garantire la semplice integrità di un messaggio e garantirne anche l'autenticazione inconfutabile della sorgente (non ripudio). Entrambi i meccanismi si basano sull'aggiunta di un'etichetta di controllo, denominata **Auth Tag**, al messaggio originale M.



Il modello generale, mostrato nella parte superiore, vede la sorgente (SRC) inviare:

$$\text{Pacchetto} = M + \text{TAG}(K, M)$$

Dove il TAG è una funzione crittografica che dipende sia dal contenuto del messaggio M che da una chiave segreta K.

7.1 Approccio Simmetrico: MAC (es. HMAC)

Nel caso della crittografia simmetrica, il tag viene chiamato **MAC** (Message Authentication Code), e l'esempio più comune è l'HMAC.

- **Gestione della Chiave:** La chiave K è una *chiave simmetrica condivisa* (Shared Secret). Sia il mittente (SRC) che il destinatario (DST) possiedono la stessa identica chiave.
- **Verifica:** Il destinatario deve conoscere la chiave del mittente (che è uguale alla propria) per ricalcolare il MAC e verificarne la correttezza.

- **Limite (Il problema dell'autenticazione):** Poiché la chiave è condivisa, se il destinatario verifica correttamente il MAC, sa che il messaggio non è stato alterato (Integrità). Tuttavia, non può provare a terze parti che il messaggio è stato scritto dal mittente. Poiché anche il destinatario possiede K, avrebbe potuto generare quel MAC da solo. Manca quindi il "Non Ripudio".

7.2 Approccio Asimmetrico: Firma Digitale

La Firma Digitale risolve il problema dell'identificazione univoca del mittente.

- **Gestione della Chiave:** La chiave K utilizzata per generare il tag è la **Chiave Privata** del mittente (SRC Private Key), che possiede solo lui.
- **Verifica:** Il destinatario (DST) non ha bisogno di segreti condivisi; gli basta conoscere la **Chiave Pubblica** (PubKey) del mittente per verificare la firma.

7.3 Conseguenza: Il Non Ripudio

La differenza fondamentale, evidenziata in blu nella slide, è la proprietà del **Non Ripudio** (o Autenticazione della Sorgente forte).

> Poiché la firma è generata con una chiave che possiede *esclusivamente* il mittente, quest'ultimo non può negare di aver inviato il messaggio. A differenza del MAC, dove la capacità di generare il tag è condivisa tra due o più parti, nella Firma Digitale la capacità di firma è unica.

8 Crittografia Asimmetrica: Algoritmi Base e Pionieri

Questa sezione introduce le origini della crittografia a chiave pubblica e i principi matematici fondamentali che ne garantiscono la sicurezza.

8.1 I Pionieri: La Nascita della Crittografia Moderna

Fino alla metà degli anni '70, la crittografia era esclusivamente simmetrica. La rivoluzione avvenne in due tappe fondamentali, guidate da matematici visionari definiti "The Pioneers":

- **1976 - L'Invenzione (Diffie & Hellman):** Whitfield Diffie e Martin E. Hellman pubblicarono il paper storico *"New Directions in Cryptography"*.
  - Introdussero per la prima volta il concetto teorico di crittografia asimmetrica.
  - Proposero il primo algoritmo asimmetrico in assoluto: il **Diffie-Hellman Key Agreement**. È importante notare che questo algoritmo serve specificamente per l'*accordo di chiave* (scambiare un segreto in modo sicuro) e non per la cifratura diretta di messaggi o per la firma.
- **1977 - La Maturità (Rivest, Shamir, Adleman):** L'anno successivo, R.L. Rivest, A. Shamir e L. Adleman realizzarono concretamente il concetto di *Public-Key Cryptosystem*.

– Inventarono l'algoritmo **RSA** (dalle loro iniziali).

– RSA fu il primo sistema a supportare sia la **cifratura a chiave pubblica** che la **firma digitale**, diventando lo standard de facto per decenni.

8.2 L'Ingrediente Base: Problemi Matematici "Difficili"

La sicurezza di questi algoritmi non si basa sull'oscurità del metodo, ma su problemi matematici definiti **computazionalmente difficili** (Hard Problems). Il principio cardine è l'asimmetria dello sforzo computazionale:

> Deve essere computazionalmente **FACILE** eseguire l'operazione in una direzione, ma computazionalmente **DIFFICILE** (impossibile in tempi umani) invertirla senza una scorciatoia (trapdoor).

Le due famiglie principali di algoritmi si basano su due diversi problemi matematici:

8.2.1 1. Diffie-Hellman: Il Logaritmo Discreto

La sicurezza del protocollo Diffie-Hellman si fonda sul *Discrete Logarithm Problem* (DLP) all'interno di campi finiti primi. Dati un grande numero primo p, un generatore g e un intero segreto x:

- **Direzione Facile (Esponenziazione Modulare):** Calcolare y conoscendo g, x, p è molto rapido, anche per numeri grandi:

$$y \equiv g^x \pmod{p} \quad \rightarrow \text{EASY} \tag{3}$$

- **Direzione Difficile (Logaritmo Discreto):** Conoscendo il risultato y, il generatore g e il modulo p, risalire all'esponente originale x è estremamente arduo:

$$x = \text{DLog}_g(y) \pmod{p} \quad \rightarrow \text{HARD!!} \tag{4}$$

8.2.2 2. RSA: La Fattorizzazione di Interi

La sicurezza del sistema RSA si basa sulla difficoltà di fattorizzare il prodotto di due grandi numeri primi.

- **Direzione Facile (Moltiplicazione):** Dati due numeri primi molto grandi p e q, calcolarne il prodotto N è banale:

$$N = p \cdot q \quad \rightarrow \text{EASY} \tag{5}$$

- **Direzione Difficile (Fattorizzazione):** Dato solo il numero N (che è pubblico), trovare i due fattori primi originali p e q è un problema intrattabile per N sufficientemente grandi (es. 2048 o 4096 bit):

$$\text{Trovare } p, q \text{ dato } N \quad \rightarrow \text{HARD} \tag{6}$$

SECURITY FOCUS: Attacco a Forza Bruta (Brute Force)

In crittografia, quando un problema è matematicamente "Hard" (come il DLOG o la Fattorizzazione), significa che non esiste un algoritmo analitico efficiente per risolverlo. L'unico modo per un attaccante di trovare la chiave segreta (es. x nel DLOG o p, q in RSA) è provare tutte le possibili combinazioni. Questo si chiama **Attacco a Forza Bruta**.
Se le chiavi sono sufficientemente lunghe (es. 2048 bit o più), il numero di tentativi necessari è astronomico (maggiore del numero di atomi nell'universo), rendendo l'attacco impossibile anche con i supercomputer più potenti esistenti oggi.

9 Perché l'Esponenziazione Modulare è "Facile"

La slide illustra l'efficienza computazionale dell'esponenziazione modulare, ovvero il calcolo di $g^x \pmod p$. Se dovessimo calcolare g^{1437} moltiplicando g per se stesso 1436 volte, il processo sarebbe troppo lento per i numeri giganteschi usati in crittografia (che hanno centinaia di cifre).

Why modular exponentiation is easy

- **Goal: compute g^{1437} mod p**
- **Express x in bits**
  - $1437_{10}=10110011101_2$
- **list bits from lsb to msb**
- **First line: initialize**
  - Column Square = g
  - Column Multiply = 1 (if b=0) or g (if b=1)
- **Start from 2° lsb to msb**
  - For every bit
    - →Square;
    - →If 1: multiply to result
- **Complexity:**
  - O(nbit) squares
  - O(nbit/2) multiplications

$b: lsb->msb$	*Square*	*Multiply*
1	g	$\times g \to g$
0	g^2	.
1	$(g^2)^2 = g^4$	$\times g^4 \to g^5$
1	$(g^4)^2 = g^8$	$\times g^8 \to g^{13}$
1	$(g^8)^2 = g^{16}$	$\times g^{16} \to g^{29}$
0	$(g^{16})^2 = g^{32}$	.
0	$(g^{32})^2 = g^{64}$	.
1	$(g^{64})^2 = g^{128}$	$\times g^{128} \to g^{157}$
1	$(g^{128})^2 = g^{256}$	$\times g^{256} \to g^{413}$
0	$(g^{256})^2 = g^{512}$	.
1	$(g^{512})^2 = g^{1024}$	$\times g^{1024} \to g^{1437}$

No algorithm such as this for the opposite problem!! That's why it is hard!

La soluzione adottata è l'algoritmo **Square-and-Multiply**, che riduce drasticamente il numero di operazioni sfruttando la rappresentazione binaria dell'esponente.

9.1 L'Algoritmo Passo dopo Passo

L'esempio pratico mostrato è il calcolo di $g^{1437} \pmod p$.

1. **Conversione Binaria:** L'esponente $x = 1437$ viene convertito in base 2:

$$1437_{10} = 10110011101_2$$

2. **Scansione dei Bit (LSB verso MSB):** L'algoritmo analizza i bit partendo dal meno significativo (LSB, destra) verso il più significativo (MSB, sinistra). La tabella

nella slide mostra questa progressione dal basso verso l'alto (o riga per riga a seconda dell'implementazione, qui i bit sono elencati verticalmente).

3. **Operazioni (Square & Multiply):** Ad ogni passo dell'iterazione corrispondono due possibili azioni sulla variabile accumulatore:

   - **Square (Quadrato):** In ogni singolo passaggio, indipendentemente dal bit, si eleva al quadrato la base corrente ($g \to g^2 \to g^4 \to g^8 \dots$). Questo costruisce le potenze di 2.
   - **Multiply (Moltiplicazione):** Questa operazione viene eseguita *solo se* il bit corrente è equal a **1**. Il risultato parziale viene moltiplicato per la potenza di g corrente.

9.2 Analisi dell'Esempio

Guardando la colonna di destra della slide, vediamo come viene costruito il risultato finale g^{1437}:

$$1437 = 1 + 4 + 8 + 16 + 128 + 256 + 1024$$

L'algoritmo seleziona e moltiplica tra loro solo le potenze di g corrispondenti ai bit a '1':

$$g^{1437} = g^1 \cdot g^4 \cdot g^8 \cdot g^{16} \cdot g^{128} \cdot g^{256} \cdot g^{1024}$$

9.3 Complessità Computazionale

Questo è il punto chiave che definisce il problema come "EASY" (trattabile):

- Il numero di operazioni di "Squaring" è pari esattamente al numero di bit dell'esponente: $O(n_{\text{bit}})$.
- Il numero di operazioni di "Multiply" è, nel caso peggiore, pari al numero di bit, e in media $O(n_{\text{bit}}/2)$ (assumendo il 50% di zeri e uni).

Per un numero a 2048 bit, invece di fare 2^{2048} moltiplicazioni (impossibile), ne facciamo solo circa 3000. È una differenza abissale.

9.4 Conclusione: L'Asimmetria

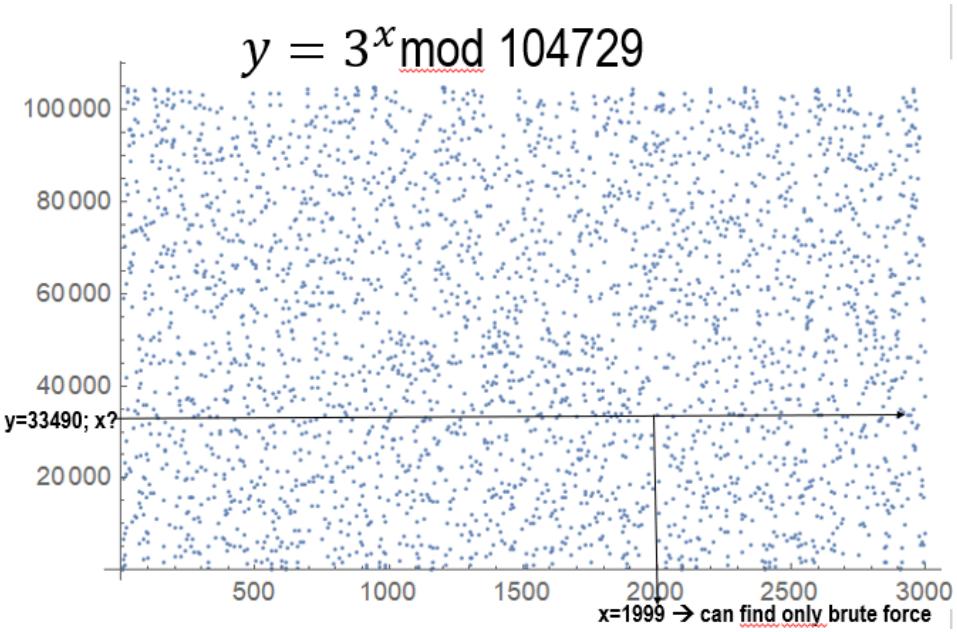
La nota in rosso in fondo alla slide ribadisce il concetto fondamentale della crittografia a chiave pubblica:

> **"No algorithm such as this for the opposite problem!!"**

Mentre calcolare $g^x \to y$ è velocissimo grazie a questo trucco binario, il problema inverso (dato y, trovare x, ovvero il Logaritmo Discreto) non ha scorciatoie simili. Per risolvere il problema inverso bisogna procedere quasi per tentativi, rendendolo computazionalmente "HARD".

10 Visualizzazione della Difficoltà del DLOG

La slide "And why DLOG is hard" fornisce una rappresentazione grafica fondamentale per comprendere la robustezza del problema del Logaritmo Discreto.



Il grafico mostra la distribuzione dei risultati della funzione di esponenziazione modulare:

$$y = 3^x \pmod{104729}$$

dove:

- L'asse orizzontale (x) rappresenta l'esponente (l'input).
- L'asse verticale (y) rappresenta il risultato modulare (l'output).

10.1 Il Caos Deterministico

A differenza dell'esponenziazione classica ($y = 3^x$) che produce una curva esponenziale liscia, continua e monotonamente crescente, l'introduzione dell'operatore modulo spezza ogni continuità. Come si evince dalla "nuvola" di punti blu nel grafico:

- **Assenza di Pattern Visibile:** I risultati appaiono distribuiti in modo pseudo-casuale (random). Non c'è una linea o una curva prevedibile che colleghi i punti.
- **Mancanza di Correlazione Locale:** Valori vicini di x producono valori di y completamente distanti e slegati tra loro. Sapere quanto vale $3^{1998} \pmod p$ non dà alcuna informazione utile per stimare quanto varrà $3^{1999} \pmod p$.

10.2 Il Problema Inverso (Trovare x)

La slide pone una sfida specifica:

> Dato $y = 33490$, trova x.

Se avessimo una funzione continua, potremmo usare metodi analitici (come il metodo di Newton o l'interpolazione) per trovare x. Tuttavia, a causa della dispersione caotica dei punti mostrata nel grafico, non esistono strumenti matematici "geometrici" per risalire all'origine. L'unica strategia certa rimane la **Forza Bruta** (o algoritmi leggermente più ottimizzati come *Baby-step Giant-step*, che restano comunque esponenziali rispetto al numero di bit):

1. Calcolo $3^1 \pmod p \rightarrow$ non è 33490.
2. Calcolo $3^2 \pmod p \rightarrow$ non è 33490.
3. ...
4. Calcolo $3^{1999} \pmod p \rightarrow$ **Trovato!** È 33490.

10.3 Conclusione sulla Sicurezza

Nell'esempio della slide, il modulo è piccolo (104.729), quindi un computer può trovare $x = 1999$ istantaneamente. Tuttavia, nei sistemi crittografici reali (come Diffie-Hellman o RSA), i numeri utilizzati sono nell'ordine di 2^{2048}. Cercare un ago in un pagliaio di quelle dimensioni, senza una mappa (pattern), richiede un tempo superiore all'età dell'universo. Ecco perché il problema è considerato "HARD".

11 Diffie-Hellman Key Agreement

La slide introduce il protocollo **Diffie-Hellman (DH)**, una pietra miliare nella storia della sicurezza informatica. Pubblicato nel 1976, questo protocollo rappresenta la vera nascita della crittografia asimmetrica, risolvendo uno dei problemi più antichi della comunicazione segreta: la distribuzione delle chiavi.

11.1 Contesto Storico e Innovazione

Il 1976 è considerato l'anno di svolta ("breakthrough") per la crittografia moderna.

- **L'Invenzione:** Diffie e Hellman introdussero il concetto di asimmetria, dimostrando che non era necessario che due parti si fossero incontrate in precedenza per comunicare in modo sicuro.
- **Il Limite Iniziale:** La slide specifica un dettaglio storico importante: sebbene DH abbia introdotto la crittografia asimmetrica, *non* risolse immediatamente il problema completo del "Public Key Cryptosystem" (ovvero cifratura diretta e firma digitale). Per quello fu necessario attendere l'anno successivo (1977) con l'avvento di **RSA**.

11.2 La Natura del Protocollo: Key Agreement

È fondamentale comprendere cosa *è* e cosa *non è* Diffie-Hellman.

- **Non è un cifrario:** DH non serve per cifrare un messaggio (come "Ciao").
- **È un protocollo di accordo chiavi:** Lo scopo unico del protocollo è permettere a due parti di generare e condividere un segreto comune (una chiave simmetrica).

11.3 Il Paradosso del Canale Insicuro

La parte centrale della slide, evidenziata in rosso, descrive la "magia" del protocollo:

> Come stabilire in modo sicuro un segreto condiviso (Shared Secret) scambiando **solamente valori pubblici**?

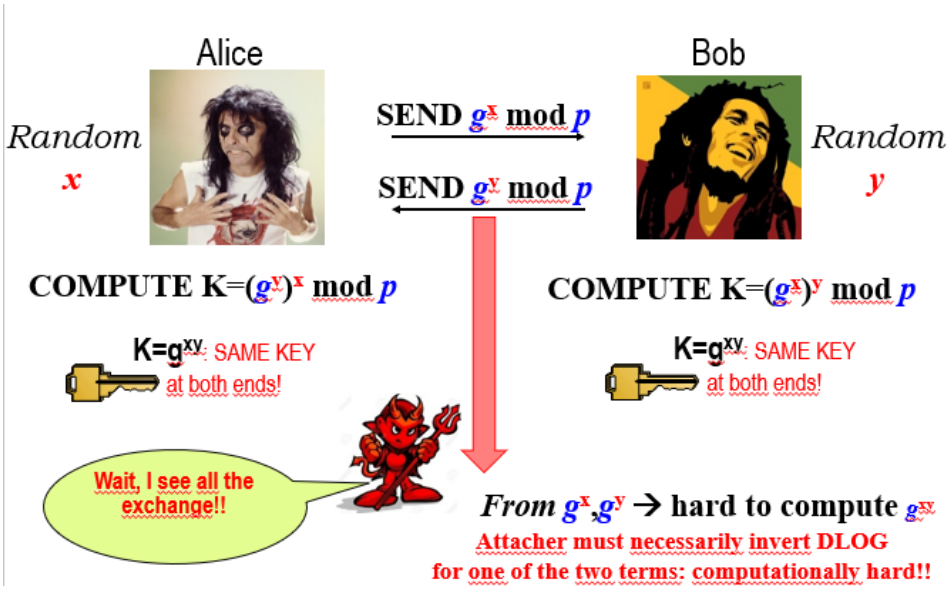
Prima del 1976, per concordare una chiave segreta, era necessario un canale sicuro (es. incontrarsi di persona o usare un corriere fidato). Con Diffie-Hellman, è possibile stabilire questa chiave segreta comunicando attraverso un canale completamente insicuro e sorvegliato (come Internet), senza che un osservatore esterno (eavesdropper) possa dedurre la chiave finale.

11.4 Fondamento Matematico

Come anticipato nelle sezioni precedenti, la sicurezza di questo scambio miracoloso si basa sul problema matematico del Logaritmo Discreto (**DLOG**). La garanzia è che, pur vedendo passare i valori pubblici scambiati, un attaccante non possa risalire agli esponenti segreti scelti dalle parti a causa della difficoltà computazionale del DLOG.

12 Il Protocollo Diffie-Hellman: Scambio e Calcolo

La slide illustra la sequenza di operazioni matematiche eseguite da Alice e Bob per concordare una chiave comune, evidenziando la sicurezza del protocollo contro un attaccante passivo (rappresentato dal diavolo in basso). Assumiamo preliminarmente che esistano due parametri pubblici noti a tutti (incluso l'attaccante): un grande numero primo p e un generatore g.



12.1 Fase 1: Generazione dei Segreti (Private Values)

Inizialmente, ciascuna delle due parti genera un numero casuale segreto che non verrà mai trasmesso:

- **Alice:** Sceglie un intero casuale segreto x.

- **Bob:** Sceglie un intero casuale segreto y.

12.2 Fase 2: Scambio dei Valori Pubblici

Le parti calcolano i rispettivi valori pubblici utilizzando l'esponenziazione modulare e se li scambiano attraverso il canale insicuro (frecce nere "SEND"):

- Alice calcola $A = g^x \pmod{p}$ e lo invia a Bob.
- Bob calcola $B = g^y \pmod{p}$ e lo invia ad Alice.

Nota: L'attaccante, osservando il canale, intercetta e conosce sia g^x che g^y.

12.3 Fase 3: Derivazione della Chiave Comune

Una volta ricevuti i valori pubblici della controparte, Alice e Bob eseguono l'ultimo calcolo matematico per ottenere la chiave di sessione K:

- **Alice:** Prende il valore ricevuto da Bob (g^y) e lo eleva al suo segreto (x):

$$K_{\text{Alice}} = (g^y)^x \pmod{p}$$

- **Bob:** Prende il valore ricevuto da Alice (g^x) e lo eleva al suo segreto (y):

$$K_{\text{Bob}} = (g^x)^y \pmod{p}$$

12.4 La Magia Matematica (Correttezza)

Per le proprietà delle potenze, l'ordine degli esponenti non cambia il risultato finale:

$$(g^y)^x = g^{yx} = g^{xy} = (g^x)^y$$

Di conseguenza, Alice e Bob hanno calcolato esattamente lo stesso numero $K = g^{xy} \pmod{p}$, che diventa la loro chiave simmetrica ("SAME KEY at both ends!").

12.5 Analisi di Sicurezza (L'Attaccante)

La parte inferiore della slide spiega perché il protocollo è sicuro nonostante l'attaccante veda tutto lo scambio ("Wait, I see all the exchange!!"). L'attaccante possiede le seguenti informazioni: g, p, g^x, g^y. Il suo obiettivo è calcolare la chiave $K = g^{xy}$.

- **L'Impossibilità Computazionale:** Non esiste un'operazione matematica diretta per combinare g^x e g^y e ottenere g^{xy} senza conoscere almeno uno degli esponenti (x o y). (Nota: moltiplicandoli si otterrebbe g^{x+y}, che non è la chiave).
- **Il Muro del DLOG:** Per trovare la chiave, l'attaccante è costretto a risalire a x partendo da g^x (o a y da g^y). Come visto nelle slide precedenti, questa operazione corrisponde al calcolo del **Logaritmo Discreto (DLOG)**, che è un problema computazionalmente difficile ("Hard!!").

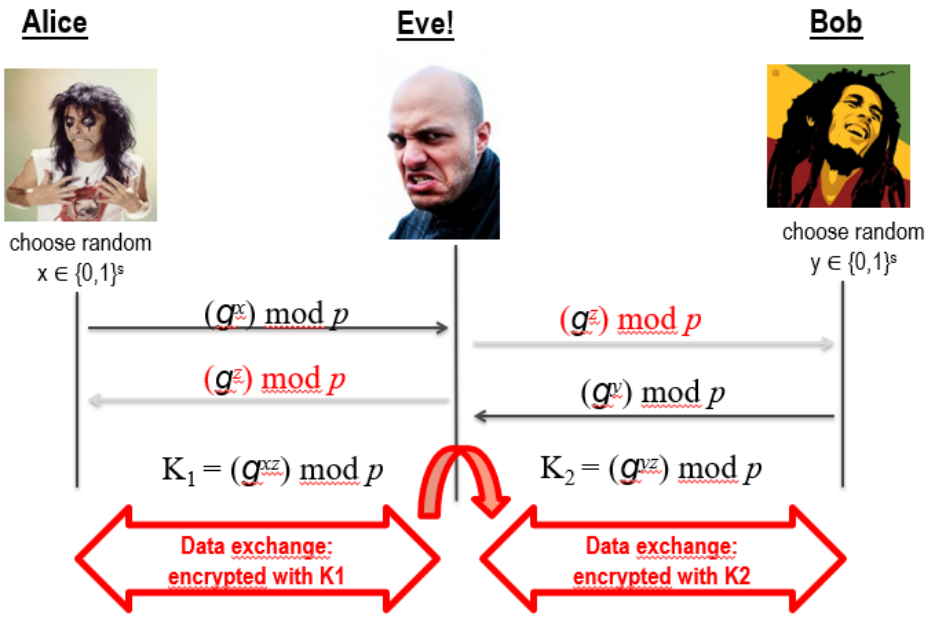
13 Vulnerabilità di Diffie-Hellman: L'Attacco Man-in-the-Middle

La slide intitolata "Vanilla DH is... anonymous!" evidenzia il tallone d'Achille del protocollo Diffie-Hellman puro (o "Vanilla"). Sebbene il protocollo garantisca la confidenzialità contro un ascoltatore passivo, non offrendo alcun meccanismo di *autenticazione*, è totalmente vulnerabile contro un attaccante attivo che si posiziona nel mezzo della comunicazione. Questo scenario è noto come attacco **Man-in-the-Middle (MitM)**.

SECURITY FOCUS: Man-in-the-Middle (MitM): Attivo vs Passivo

È cruciale distinguere tra due tipi di attaccanti:

- **Attaccante Passivo (Eavesdropper):** Si limita ad ascoltare e copiare i dati che passano sulla rete. Diffie-Hellman protegge da questo attacco.
- **Attaccante Attivo (Man-in-the-Middle):** Intercetta i dati, li blocca e ne invia di falsi. Agisce come un "postino infedele" che apre la busta di Alice, ne scrive una falsa per Bob, e viceversa. Diffie-Hellman non protegge da questo attacco perché Alice e Bob non hanno modo di verificare chi ha generato i valori g^x e g^y.



13.1 Lo Scenario dell'Attacco

In questa situazione, abbiamo tre attori:

- **Alice e Bob:** Le vittime, che intendono stabilire un canale sicuro tra loro.
- **Eve (The Attacker):** Un attaccante attivo capace di intercettare, bloccare e modificare i pacchetti di rete tra Alice e Bob.

13.2 Esecuzione dell'Attacco

L'attacco sfrutta il fatto che, nello scambio DH base, le chiavi pubbliche (g^x, g^y) sono anonime: non c'è nulla che garantisca che g^x provenga davvero da Alice o g^y da Bob. L'attacco procede come segue:

1. **Intercettazione:** Alice invia il suo valore pubblico g^x destinato a Bob. Eve lo intercetta e impedisce che arrivi a Bob.
2. **Sostituzione verso Bob:** Eve genera un proprio segreto casuale z, calcola g^z e lo invia a Bob fingendosi Alice. Bob accetta il valore credendo provenga da Alice.
3. **Risposta di Bob:** Bob invia il suo valore g^y destinato ad Alice. Eve lo intercetta.
4. **Sostituzione verso Alice:** Eve invia il valore g^z (basato sul suo segreto) ad Alice, fingendosi Bob. Alice accetta il valore credendo provenga da Bob.

13.3 Il Risultato: Due Canali Separati

Come mostrato nella parte inferiore della slide, la conseguenza è che non si crea un unico tunnel sicuro tra Alice e Bob, ma due tunnel separati controllati da Eve:

- **Lato Alice (K_1):** Alice calcola la chiave K_1 combinando il suo segreto x con il valore di Eve g^z.
$$K_1 = (g^z)^x = g^{xz} \pmod{p}$$
Alice crede di condividere questa chiave con Bob, ma in realtà la condivide con Eve.
- **Lato Bob (K_2):** Bob calcola la chiave K_2 combinando il suo segreto y con il valore di Eve g^z.
$$K_2 = (g^z)^y = g^{yz} \pmod{p}$$
Bob crede di condividere questa chiave con Alice, ma la condivide con Eve.

13.4 Le Conseguenze Operative

Eve possiede il segreto z e ha intercettato g^x e g^y. Pertanto, è in grado di calcolare sia K_1 che K_2.

> Quando Alice invia un messaggio cifrato con K_1, Eve lo decifra, lo legge (o lo modifica), lo ricifra con K_2 e lo inoltra a Bob.

Alice e Bob comunicano pensando di essere al sicuro, mentre Eve ha il controllo totale (lettura e scrittura) sul traffico. Questo dimostra perché Diffie-Hellman deve essere sempre abbinato a un meccanismo di autenticazione (come le firme digitali RSA) per essere sicuro.

14 Approfondimento: Il Protocollo BTNS

In risposta alle limitazioni di scalabilità delle infrastrutture a chiave pubblica (PKI), è stato proposto il concetto di *Opportunistic Encryption* (Crittografia Opportunistica), concretizzato nel protocollo BTNS.

BTNS: Better-Than-Nothing Security

Definizione Il protocollo **BTNS** (Better-Than-Nothing Security) è un'estensione per IPsec che permette di stabilire tunnel crittografati tra due nodi senza richiedere alcuna configurazione preliminare di autenticazione (niente certificati, niente chiavi pre-condivise). **Funzionamento Tecnico** Il protocollo utilizza uno scambio di chiavi **Diffie-Hellman Anonimo** (proprio come il "Vanilla DH" visto in precedenza).

- Alice e Bob negoziano una chiave di sessione K.
- Non viene verificata l'identità delle controparti (Autenticazione = Null).
- Il traffico viene cifrato con K.

Modello di Sicurezza

- **Contro Attaccanti Passivi (Eavesdroppers):** SICURO. Poiché il traffico è cifrato, chi si limita ad ascoltare la rete (sniffing) non può leggere i dati.
- **Contro Attaccanti Attivi (Man-in-the-Middle):** VULNERABILE. Poiché non c'è autenticazione, un attaccante attivo (Eve) può inserirsi nel mezzo, scambiando chiavi diverse con Alice e Bob senza essere rilevato.

Perché "Better Than Nothing"? La filosofia alla base è pragmatica: la maggior parte del traffico Internet è in chiaro (plaintext).

> *"È meglio avere una comunicazione cifrata ma non autenticata (che protegge dall'ascolto di massa), piuttosto che una comunicazione completamente in chiaro."*

Il BTNS alza l'asticella per l'attaccante: mentre intercettare il traffico in chiaro è banale e può essere fatto su larga scala (mass surveillance), eseguire un attacco MitM richiede risorse attive e mirate per ogni singola connessione.

15 Soluzione Pratica al MitM: Verifica Manuale (Trust on First Use)

La slide "How to fix this? A practical approach" propone una contromisura pragmatica alla vulnerabilità del protocollo Diffie-Hellman "Vanilla" contro gli attacchi Man-in-the-Middle (MitM). Questo approccio è tipico delle applicazioni di messaggistica istantanea sicura (emerse intorno al 2013, come citato nel fumetto) che offrono **End-to-End Encryption (E2EE)**.

15.1 Il Concetto di "Auth Code" o Fingerprint

Il problema del MitM è che Alice e Bob non sanno se stanno parlando direttamente tra loro o tramite Eve. Per risolvere questo problema senza un'autorità centrale, si delega

la verifica agli utenti finali tramite un confronto manuale dei segreti derivati. Il processo avviene in tre fasi:

1. **Scambio DH Standard:** Alice e Bob eseguono il protocollo Diffie-Hellman come di consueto, scambiandosi g^x e g^y attraverso il server dell'applicazione. Calcolano le rispettive chiavi:
$$K = g^{xy} \pmod{p}$$
Nota: Il fornitore del servizio ("messaging application") promette che nemmeno i propri server possono intercettare la chat, poiché non conoscono i segreti x e y.

2. **Visualizzazione della Chiave (Fingerprint):** L'applicazione trasforma la chiave derivata K (o più comunemente un hash crittografico della stessa) in un formato leggibile o scansionabile dall'uomo. Nella slide, questo è rappresentato dai **Codici QR**. In termini tecnici, questo è spesso chiamato *Safety Number* o *Short Authentication String (SAS).*

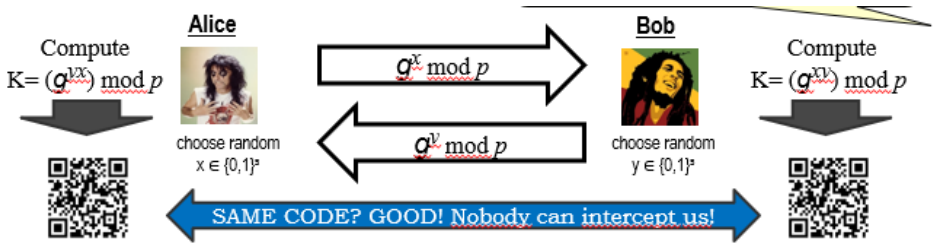
3. **Verifica Fuori Banda (Out-of-Band Verification):** Alice e Bob devono confrontare questi codici utilizzando un canale diverso da quello che stanno cercando di proteggere (ad esempio incontrandosi di persona e scansionando i QR code, oppure leggendosi i numeri al telefono).

15.2 Perché questo blocca il Man-in-the-Middle?

Il meccanismo di sicurezza si basa sull'unicità della chiave condivisa. Tornando allo scenario dell'attacco:

- Se **Eve** fosse nel mezzo, avrebbe stabilito una chiave K_1 con Alice e una chiave diversa K_2 con Bob.
- Di conseguenza, il QR Code generato da Alice (basato su K_1) sarebbe matematicamente diverso da quello generato da Bob (basato su K_2).

La verifica visiva ("SAME CODE? GOOD!") è la prova crittografica che $K_{\text{Alice}} = K_{\text{Bob}}$. Se i codici coincidono, esiste un'unica chiave condivisa e, per le proprietà del Diffie-Hellman, questo implica l'assenza di intermediari.

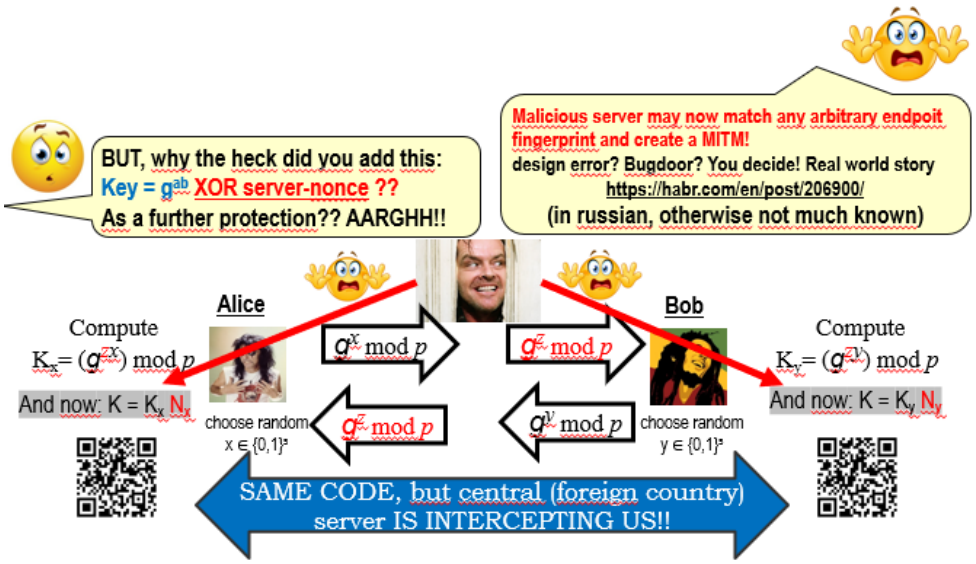


16 Case Study: Bug o Backdoor? Un "Errore" Sottile

Le slide presentano un caso reale (discusso nella community crittografica, spesso associato alle prime implementazioni di protocolli proprietari come quello discusso su Habr) che illustra il concetto di **"Bugdoor"** (una crasi tra Bug e Backdoor). Si tratta di una situazione in cui un errore di design o implementazione è indistinguibile da una backdoor inserita intenzionalmente per permettere l'intercettazione (Plausible Deniability).

16.1 La Modifica al Protocollo ("Make it more secure")

Nel tentativo dichiarato di "rendere più sicuro" il protocollo o aggiungere entropia, gli sviluppatori hanno introdotto una variante alla verifica del codice di sicurezza (Fingerprint) vista nella sezione precedente.



- **Procedura Standard:** Alice e Bob confrontano un'impronta derivata direttamente dalla chiave DH (K). Se $K_A = K_B$, allora non c'è MitM.
- **La Modifica Introdotta:** Il server partecipa alla generazione dell'impronta inviando un **Nonce** (un numero casuale) sia ad Alice che a Bob. L'impronta visiva (QR Code) viene calcolata come:

$$\text{Fingerprint} = K_{\text{DH}} \oplus \text{Nonce}_{\text{Server}} \tag{7}$$

  Dove $\oplus$ indica l'operazione XOR.

L'idea venduta agli utenti è: "Aggiungiamo ulteriore casualità dal server per proteggervi meglio".

16.2 L'Attacco: Come il Server crea un MITM Invisibile

La seconda slide mostra perché questa aggiunta è catastrofica. Se il server è malevolo (o compromesso), può annullare l'efficacia della verifica manuale dei QR Code. Ecco come avviene l'attacco:

1. **Posizionamento MitM:** Il server si interpone tra Alice e Bob (come Eve).
   - Stabilisce una chiave K_x con Alice.
   - Stabilisce una chiave diversa K_y con Bob.

   In un protocollo normale, i QR code sarebbero diversi ($K_x \neq K_y$) e gli utenti se ne accorgerebbero.

2. **Manipolazione dei Nonce:** Il server controlla i Nonce (N_x inviato ad Alice e N_y inviato a Bob). Poiché conosce entrambe le chiavi false (K_x e K_y), può calcolare matematicamente i Nonce necessari per far *sembrare* uguali i QR code. Il server calcola N_y tale che:
$$K_x \oplus N_x = K_y \oplus N_y \tag{8}$$

3. **Risultato:**
   - Alice vede il QR Code generato da $K_x \oplus N_x$.
   - Bob vede il QR Code generato da $K_y \oplus N_y$.
   - **I codici sono identici ("SAME CODE")**.

16.3 Conclusione: La "Deniable Bugdoor"

Gli utenti verificano i codici, vedono che coincidono e si fidano della connessione ("Nobody can intercept us!"). In realtà, il server sta decifrando, leggendo e ricifrando tutto il traffico. Questo è un classico esempio di **Bugdoor**:

- Se scoperto, lo sviluppatore può dire: "Ops, è stato un errore di implementazione, non avevamo pensato a questa conseguenza matematica" (*Bug*).
- Se non scoperto, permette al gestore del server (o a un'agenzia governativa che controlla il server) di intercettare le conversazioni segrete (*Backdoor*).

17 Limitazioni Funzionali di Diffie-Hellman

Nonostante l'importanza rivoluzionaria del protocollo Diffie-Hellman (DH) per l'accordo delle chiavi su canali insicuri, la slide evidenzia come questo algoritmo, da solo, non costituisca un sistema crittografico a chiave pubblica completo. Esso soffre di due limitazioni funzionali principali che ne restringono il campo d'applicazione al solo scambio di segreti.

17.1 1. Mancanza di un Sistema di Cifratura a Chiave Pubblica

Il protocollo DH permette a due parti di calcolare un numero comune, ma non fornisce un meccanismo diretto per cifrare un messaggio specifico M.

- **Il problema:** In un vero sistema a chiave pubblica (come inteso oggi), Alice dovrebbe poter prendere un messaggio M, usare la chiave pubblica di Bob (PK_B) e creare un crittogramma C che solo Bob può leggere.
- **La limitazione DH:** Con DH puro, questo non è possibile. Non esiste una funzione $C = \text{Encrypt}(PK_{\text{receiver}}, \text{data})$. Il protocollo serve solo a generare la chiave ("key agreement"), non a trasportare dati cifrati ("data transfer"). Per inviare dati, bisogna successivamente usare un algoritmo simmetrico (come AES) con la chiave derivata da DH.

17.2 2. Assenza della Firma Digitale

La seconda grave lacuna è l'impossibilità di garantire l'autenticità e il non ripudio tramite firma.

- **Il problema:** La firma digitale richiede che un utente usi la propria chiave segreta (SK) per trasformare un dato, in modo che chiunque possieda la corrispondente chiave pubblica possa verificare l'identità del firmatario.
- **La limitazione DH:** La matematica di Diffie-Hellman è simmetrica nel risultato ($g^{xy} = g^{yx}$) e cooperativa. Non esiste un modo per Alice di "firmare" un documento usando il suo esponente segreto x in modo tale che il mondo intero possa verificarlo usando il suo valore pubblico g^x.

17.3 L'Evoluzione: RSA (1977)

La soluzione storica a queste mancanze. Nel 1977, l'algoritmo **RSA** "viene in soccorso" (*to the rescue*). RSA è stato il primo algoritmo a risolvere contemporaneamente tutti e tre i problemi:

1. Permette lo scambio di chiavi (Key Encapsulation).
2. Permette la cifratura diretta con chiave pubblica (Public Key Encryption).
3. Permette la firma digitale (Digital Signature).

18 L'Algoritmo RSA

La slide presenta l'**RSA**, l'algoritmo di crittografia asimmetrica più famoso e diffuso al mondo. Acronimo dei cognomi dei suoi inventori (**R**ivest, **S**hamir, **A**dleman), è stato pubblicato nel 1977, un anno dopo il paper teorico di Diffie-Hellman.

18.1 Storia e Diffusione

- **Origine:** Creato presso il MIT nel 1977.
- **Brevetto:** È stato coperto da brevetto negli Stati Uniti fino all'anno 2000. La scadenza del brevetto ha contribuito ulteriormente alla sua adozione universale.
- **Stato Attuale:** È lo standard "de facto" per la crittografia a chiave pubblica, definito nella slide come "Ultra-well known".

18.2 Il Problema Matematico (Hard Problem)

A differenza di Diffie-Hellman che si basava sul Logaritmo Discreto, la sicurezza di RSA si fonda su un problema matematico diverso: la **Fattorizzazione di Interi** (Integer Factoring). Il principio è l'asimmetria della moltiplicazione tra numeri primi:

- **Facile (Creazione):** Dati due numeri primi molto grandi p e q, è banale calcolare il loro prodotto:

$$N = p \times q$$

- **Difficile (Attacco):** Dato solamente il numero N (che è pubblico), è computazionalmente intrattabile risalire ai due fattori originali p e q (che sono segreti).

La grandezza di N determina la forza del sistema (oggi si usano N a 2048 o 4096 bit).

18.3 Funzionamento: Cifratura e Decifratura

Il cuore operativo di RSA è, ancora una volta, l'**esponenziazione modulare**. Le chiavi sono costituite da esponenti matematici. Definiamo:

- N: Il modulo (prodotto di p e q, pubblico).
- PK: La chiave pubblica (esponente pubblico, spesso indicato come e nei testi accademici).
- SK: La chiave segreta (esponente privato, spesso indicato come d).

Le operazioni sono l'una l'inversa dell'altra:

1. **Cifratura (Encryption):** Il mittente usa la chiave pubblica del destinatario $\{N, PK\}$ per cifrare il messaggio M:

$$\text{ciphertext} = M^{PK} \pmod{N} \tag{9}$$

2. **Decifratura (Decryption):** Il destinatario usa la propria chiave segreta $\{N, SK\}$ per recuperare il messaggio originale:

$$\text{message} = (\text{ciphertext})^{SK} \pmod{N} \tag{10}$$

18.4 Versatilità

La nota finale della slide sottolinea la superiorità funzionale di RSA rispetto a Diffie-Hellman. RSA supporta nativamente entrambe le funzioni principali della crittografia asimmetrica:

- **Confidenzialità:** Cifratura con la chiave pubblica del destinatario.
- **Autenticazione/Non Ripudio:** Firma digitale cifrando (firmando) con la propria chiave privata.

19 Il Principio Matematico di RSA: Periodicità e Teorema di Eulero

La slide "The principle behind RSA" illustra il motore matematico che permette alla crittografia a chiave pubblica di funzionare: la natura ciclica dell'esponenziazione modulare.

19.1 La Periodicità della Funzione Modulare

La funzione fondamentale utilizzata in RSA è $f(x) = m^x \pmod N$. Osservando il comportamento di questa funzione con numeri piccoli (ad esempio modulo 10), notiamo che i risultati non crescono all'infinito, ma si ripetono secondo uno schema ciclico (periodico). Gli esempi nella slide mostrano le potenze modulo 10:

- **Base 3:** $3^1 = 3, 3^2 = 9, 3^3 = 27(\equiv 7), 3^4 = 81(\equiv 1)$. La sequenza è $\{3, 9, 7, 1\}$ e poi si ripete.
- **Base 7:** La sequenza è $\{7, 9, 3, 1\}$ e poi si ripete.
- **Base 9:** La sequenza è $\{9, 1\}$ e poi si ripete.

Tutte queste sequenze terminano con il valore **1** prima di ricominciare. Questo "1" è cruciale perché rappresenta l'elemento neutro della moltiplicazione (il punto di reset del ciclo).

19.2 Il Teorema di Eulero (Euler's Totient Theorem)

Per utilizzare questa proprietà in crittografia, dobbiamo sapere esattamente *quando* (dopo quanti passaggi) la sequenza torna a 1. Non possiamo andare per tentativi con numeri di 2048 bit. Qui entra in gioco il **Teorema di Eulero** (o Teorema di Eulero-Fermat), che fornisce una regola esplicita per calcolare questo periodo.

19.2.1 La Funzione Totiente di Eulero $\Phi(N)$

Il teorema introduce la funzione $\Phi(N)$, che conta quanti numeri interi positivi minori di N sono coprimi con N (cioè non hanno divisori comuni con N eccetto 1). Il teorema afferma che il periodo massimo della funzione modulare è determinato proprio da $\Phi(N)$.

19.2.2 Enunciato Formale

La formula fondamentale, evidenziata in rosso nella slide, afferma che:

$$\text{Se GCD}(m, N) = 1 \implies m^{\Phi(N)} \equiv 1 \pmod N \tag{11}$$

Cosa significa in pratica? Significa che se prendiamo un numero m, lo eleviamo alla potenza $\Phi(N)$ e calcoliamo il resto della divisione per N, otteniamo sempre 1. Di conseguenza, se andiamo avanti di un passo ulteriore (moltiplicando per m), otteniamo:

$$m^{\Phi(N)+1} \equiv m \pmod N \tag{12}$$

Questa identità è la base della decifratura RSA: ci permette di trasformare il messaggio cifrato di nuovo nel messaggio originale m "completando il giro" dell'esponenziazione modulare.

20 Calcolo della Funzione Totiente di Eulero $\Phi(N)$

Dopo aver stabilito che $\Phi(N)$ determina il periodo dopo il quale le potenze modulari ritornano a 1 (ossia $m^{\Phi(N)} \equiv 1 \pmod N$), questa slide fornisce le regole operative per calcolare questo valore per diverse tipologie di numeri. La comprensione di queste regole è essenziale per capire la "botola" (trapdoor) di RSA.

20.1 1. Caso Base: Numeri Primi

Se p è un numero primo, il calcolo è immediato. Poiché un numero primo non è divisibile per nessuno dei numeri che lo precedono, tutti gli interi da 1 a $p-1$ sono coprimi con p.

$$\Phi(p) = p - 1 \tag{13}$$

Esempio della slide: Per $p = 7$, $\Phi(7) = 6$. Infatti, $3^6 \equiv 729 \equiv 1 \pmod 7$.

20.2 2. Il Caso RSA: Prodotto di due Primi

Questo è il caso evidenziato nel rettangolo rosso, poiché è la struttura utilizzata nell'algoritmo RSA ($N = p \cdot q$). La funzione Totiente è **moltiplicativa** per numeri coprimi. Dato che p e q sono primi distinti, sono coprimi tra loro, quindi:

$$\Phi(N) = \Phi(p \cdot q) = \Phi(p) \cdot \Phi(q) \tag{14}$$

Sostituendo con la regola del caso base ($p-1$ e $q-1$), otteniamo la formula fondamentale di RSA:

$$\Phi(N) = (p-1)(q-1) \tag{15}$$

Esempio della slide: Dato $N = 10$ (dove $p = 2, q = 5$):

$$\Phi(10) = (2-1)(5-1) = 1 \cdot 4 = 4$$

Verifica: $3^4 = 81$. Dividendo 81 per 10, il resto è 1. Il teorema funziona.

20.3 3. Casi Avanzati (Potenze e Caso Generale)

La slide mostra anche le regole per numeri più complessi, utili per generalizzare il concetto.

- **Potenza di un Primo (p^k):** Non tutti i numeri minori di p^k sono coprimi (i multipli di p vanno esclusi). La formula è:

  $$\Phi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$$

  Esempio: $N = 25 = 5^2$. $\Phi(25) = 5^1(4) = 20$.

- **Caso Generale (Fattorizzazione):** Per un numero N qualsiasi, si scompone in fattori primi $N = p_1^{k_1} \dots p_z^{k_z}$ e si applica la proprietà moltiplicativa ai singoli blocchi.

20.4 Implicazione per la Sicurezza (La Trapdoor)

La formula del "Caso RSA" rivela il segreto della sicurezza dell'algoritmo:

- **Per chi conosce la fattorizzazione (p, q):** Calcolare $\Phi(N) = (p-1)(q-1)$ è un'operazione banale e velocissima.

- **Per chi conosce solo N:** Per calcolare $\Phi(N)$, bisognerebbe prima trovare p e q (fattorizzare N). Ma come visto in precedenza, la fattorizzazione di grandi numeri è un problema **HARD**.

Senza conoscere $\Phi(N)$, non è possibile calcolare la chiave privata di decifratura.

21 Conseguenze della Periodicità: Aritmetica degli Esponenti

La slide "Consequence of periodicity" illustra come la natura ciclica dell'esponenziazione modulare influenzi le regole di calcolo, in particolare quando si opera sugli esponenti.

21.1 Il Periodo $\Phi(N)$

Come stabilito dal Teorema di Eulero, la funzione $f(x) = m^x \pmod N$ è periodica. La lunghezza di questo ciclo (il periodo) è data esattamente da $\Phi(N)$. Nell'esempio della slide, viene scelto $N = 11$ (un numero primo).

- Il periodo è $\Phi(11) = 11 - 1 = 10$.
- Osservando le potenze di 2 modulo 11: $\{2, 4, 8, 5, 10, 9, 7, 3, 6, 1\}$.
- La sequenza ha 10 elementi distinti e poi ricomincia da 2.

21.2 L'Esempio di Moltiplicazione

La slide propone il calcolo di $9 \cdot 7 \pmod{11}$ utilizzando due approcci diversi per dimostrare la regola degli esponenti.

1. **Approccio Classico:** Si esegue la moltiplicazione diretta e poi il modulo.
$$9 \cdot 7 = 63 \equiv 8 \pmod{11}$$
2. **Approccio Alternativo (Esponenziale):** Possiamo riscrivere i numeri 9 e 7 come potenze del generatore 2 (guardando la sequenza sopra):
$$9 \equiv 2^6 \pmod{11} \quad \text{e} \quad 7 \equiv 2^7 \pmod{11}$$
Sostituendo nel calcolo:
$$9 \cdot 7 \equiv 2^6 \cdot 2^7 \equiv 2^{6+7} \equiv 2^{13} \pmod{11}$$

21.3 La Regola Fondamentale degli Esponenti

Qui sorge il problema cruciale: come riduciamo $2^{13} \pmod{11}$? Non dobbiamo calcolare 2^{13} (che è 8192) e dividerlo per 11. Possiamo semplificare l'esponente. Poiché il ciclo si ripete ogni $\Phi(N) = 10$ passi, possiamo sottrarre multipli di 10 dall'esponente senza cambiare il risultato.

$$2^{13} = 2^{10+3} = 2^{10} \cdot 2^3$$

Sappiamo che $2^{10} \equiv 1 \pmod{11}$ (fine del ciclo). Quindi:

$$1 \cdot 2^3 = 8 \pmod{11}$$

REGOLA AUREA: Quando si lavora con gli esponenti, l'aritmetica non segue il modulo N, ma il modulo del periodo $\Phi(N)$.

$$\text{Esponente finale} = \text{Esponente iniziale} \pmod{\Phi(N)}$$

Nell'esempio: $13 \pmod{10} = 3$.

21.4 Conseguenza Finale (La base di RSA)

L'ultima riga della slide formalizza il concetto che rende possibile la decifratura RSA:

$$m^x \equiv m \pmod{N} \quad \text{SE} \quad x \equiv 1 \pmod{\Phi(N)}$$

Ciò significa che se riusciamo a trovare un esponente x che, diviso per $\Phi(N)$, dà resto 1, allora elevare m alla potenza x equivale a non fare nulla (restituisce m). In RSA, questo esponente x sarà il prodotto delle chiavi $e \cdot d$.

22 Costruzione del Sistema RSA (Key Generation)

La slide "RSA construction" descrive la procedura step-by-step per la generazione delle chiavi (Setup). La sicurezza dell'intero sistema dipende dalla corretta esecuzione di questi passaggi e dal mantenimento della segretezza di alcuni parametri intermedi. La procedura si articola in 5 fasi sequenziali:

22.1 1. Generazione dei Numeri Primi

Il primo passo consiste nel generare due numeri primi, denominati p e q.

- **Requisito:** Devono essere numeri primi **grandi** (LARGE primes). Nella pratica moderna, questi numeri hanno centinaia di cifre decimali (per ottenere chiavi a 2048 o 4096 bit).
- **Visibilità:** **SECRET!** Questi numeri non devono mai essere rivelati a nessuno né salvati in chiaro una volta finito il calcolo.

22.2 2. Calcolo del Modulo RSA

Si calcola il modulo N moltiplicando i due primi:

$$N = p \times q$$

- **Visibilità:** **PUBLIC!** Il valore N è parte integrante della chiave pubblica e viene distribuito a tutti. La sua lunghezza in bit determina la "dimensione della chiave" (es. RSA-2048 significa che N è lungo 2048 bit).

22.3 3. Calcolo della Funzione Totiente $\Phi(N)$

Si calcola la funzione di Eulero sfruttando la formula per il prodotto di due primi (vista nella slide precedente):

$$\Phi(N) = (p-1)(q-1)$$

- **Visibilità:** **SECRET!** Questo è il passaggio più critico. Anche se N è pubblico, $\Phi(N)$ deve rimanere segreto. Se un attaccante scoprisse $\Phi(N)$, il sistema crollerebbe immediatamente.

22.4 4. Generazione della Chiave Pubblica (e)

Si sceglie un numero intero e che soddisfi due condizioni:

1. Deve essere compreso tra 1 e $\Phi(N)$.
2. Deve essere **coprimo** con $\Phi(N)$, ovvero $\text{MCD}(e, \Phi(N)) = 1$.

Questa condizione di coprimalità è necessaria matematicamente affinché e sia invertibile (cioè affinché esista la chiave privata). *Nota:* e è l'esponente pubblico usato per la cifratura.

22.5 5. Generazione della Chiave Privata (d)

Si calcola l'esponente privato d come l'**inverso modulare** di e rispetto a $\Phi(N)$. L'equazione fondamentale da risolvere è:

$$e \times d \equiv 1 \pmod{\Phi(N)}$$

In termini matematici, stiamo cercando un numero d tale che, moltiplicato per e e diviso per $\Phi(N)$, dia resto 1. Questo calcolo si esegue tipicamente con l'*Algoritmo di Euclide Esteso*.

22.6 Riepilogo delle Chiavi e Assunzioni di Sicurezza

Al termine della procedura, le chiavi sono così composte:

- **Chiave Pubblica (Public Key):** La coppia (N, e). Viene usata per cifrare e verificare firme.
- **Chiave Privata (Private Key):** Il numero d (associato a N). Viene usato per decifrare e firmare.

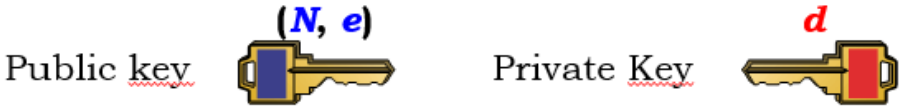
La Catena della Sicurezza: La slide riassume la logica di sicurezza nel box grigio in basso:

1. Dato N (pubblico), è computazionalmente difficile trovare i fattori p e q (Problema della Fattorizzazione).
2. Se non si conoscono p e q, è difficile calcolare $\Phi(N) = (p-1)(q-1)$.
3. Senza conoscere $\Phi(N)$, è impossibile calcolare d partendo da e. L'equazione $e \times d \equiv 1 \pmod{\Phi(N)}$ non può essere risolta dall'attaccante perché gli manca il modulo $\Phi(N)$.

23 RSA: Operazioni di Cifratura, Decifratura e Verifica

La slide riassume le due operazioni fondamentali del protocollo RSA e fornisce la dimostrazione matematica della loro correttezza (ovvero, dimostra che la decifratura inverte esattamente la cifratura).

23.1 Le Chiavi in Azione



Vengono confermati i ruoli delle chiavi generate in precedenza:

- **Public Key** (N, e)**:** Rappresentata dalla chiave blu. È accessibile a chiunque e viene utilizzata per l'operazione di **Cifratura** (chiudere il lucchetto).
- **Private Key** d**:** Rappresentata dalla chiave rossa. È posseduta solo dal destinatario e serve per l'operazione di **Decifratura** (aprire il lucchetto).

23.2 Il Processo Matematico

23.2.1 1. Cifratura (Encryption)

Il mittente vuole inviare un messaggio M (interpretato come un numero). Utilizza la chiave pubblica (N, e) per calcolare il crittogramma C:

$$C = M^e \pmod{N} \tag{16}$$

23.2.2 2. Decifratura (Decryption)

Il destinatario riceve C. Per recuperare il messaggio, applica la propria chiave privata d:

$$M' = C^d \pmod{N} \tag{17}$$

23.3 Dimostrazione di Correttezza

La parte inferiore della slide mostra la catena di uguaglianze che prova che M' è effettivamente uguale al messaggio originale M. Sostituendo l'equazione di cifratura in quella di decifratura:

$$\begin{aligned} M_{\text{recuperato}} &= C^d \pmod{N} \\ &= (M^e)^d \pmod{N} && \text{(Sostituzione di } C\text{)} \\ &= M^{e \cdot d} \pmod{N} && \text{(Proprietà delle potenze)} \end{aligned}$$

Qui entra in gioco la proprietà fondamentale costruita durante la generazione delle chiavi. Sappiamo che d è stato scelto come inverso di e modulo $\Phi(N)$, il che significa che:

$$e \cdot d \equiv 1 \pmod{\Phi(N)}$$

Grazie al teorema di Eulero (visto nelle slide precedenti), operare con un esponente pari a 1 (mod $\Phi(N)$) equivale ad avere esponente 1 nel calcolo reale. Pertanto la catena si conclude:

$$\begin{aligned} M^{e \cdot d} \pmod{N} &= M^1 \pmod{N} \\ &= M \end{aligned}$$

Conclusione: Il sistema funziona perfettamente perché il prodotto delle chiavi $e \cdot d$, agendo nell'esponente, "cancella" la trasformazione, riportando il valore al messaggio M originale.

24 RSA come Funzione Trapdoor

La slide "Why RSA works? Trapdoor function!" spiega il meccanismo logico che rende RSA sicuro per un attaccante ma facilmente utilizzabile per il legittimo proprietario. Tutto ruota attorno alla disponibilità di un'informazione segreta: la funzione Totiente $\Phi(N)$. Il problema della decifratura viene presentato come una sfida matematica:

> Dati il modulo N, l'esponente pubblico e e un messaggio cifrato $C = m^e$, trovare la chiave di decifratura x tale che:
>
> $$(m^e)^x \equiv m \pmod{N}$$

In questo contesto, x rappresenta la nostra chiave privata (precedentemente indicata come d). La difficoltà di risolvere questa equazione cambia drasticamente a seconda delle informazioni in possesso:

24.1 1. Caso "Normale": Il Problema Difficile (HARD)

Per un attaccante che conosce solo la chiave pubblica (N, e), trovare x è un problema di **complessità esponenziale**.

- L'attaccante non conosce $\Phi(N)$.
- Senza $\Phi(N)$, non esiste una scorciatoia matematica per correlare e a x.
- L'unica opzione è procedere per tentativi (Brute Force) su tutti i possibili valori di x. Dato che x è un numero dell'ordine di grandezza di N (migliaia di bit), questo approccio richiederebbe milioni di anni.

24.2 2. Caso con Trapdoor: Il Problema Facile (EASY)

Per il legittimo destinatario, che possiede la "Trapdoor information" (ovvero conosce $\Phi(N)$ perché conosce i fattori primi p e q), il problema diventa di **complessità polinomiale** (estremamente veloce). Conoscendo $\Phi(N)$, la ricerca di x si riduce alla risoluzione di una semplice equazione lineare modulare:

$$e \cdot x \equiv 1 \pmod{\Phi(N)} \tag{18}$$

Questo significa che x è semplicemente l'**inverso moltiplicativo modulare** di e. La slide evidenzia nel box giallo la soluzione diretta:

$$x = e^{-1} \pmod{\Phi(N)} \tag{19}$$

24.3 L'Algoritmo di Risoluzione

Per calcolare questo valore concretamente, non si va per tentativi. Esiste un algoritmo matematico efficiente ed antico, citato in fondo alla slide: l'**Algoritmo di Euclide Esteso** (Extended Euclidean Algorithm). Questo algoritmo permette di trovare x in frazioni di secondo, purché (e solo se) si conosca il modulo $\Phi(N)$. **Sintesi:** La sicurezza di RSA risiede nel fatto che calcolare $\Phi(N)$ partendo da N (fattorizzazione) è "impossibile", mentre calcolare l'inverso modulare conoscendo $\Phi(N)$ è banale.

25 Promemoria: Algoritmo di Euclide Esteso

Per completare la generazione delle chiavi RSA, è necessario risolvere l'equazione $e \cdot d \equiv 1 \pmod{\Phi(N)}$. Questo corrisponde al calcolo dell'inverso modulare. Lo strumento matematico per effettuare questa operazione in modo efficiente è l'**Algoritmo di Euclide Esteso**. La slide mostra un esempio pratico di esecuzione dell'algoritmo con numeri piccoli.

25.1 L'Obiettivo

Dati due numeri:

- Il modulo (che nel contesto RSA sarebbe $\Phi(N)$): **51**
- L'esponente pubblico (che nel contesto RSA sarebbe e): **11**

Sappiamo che $\text{MCD}(51, 11) = 1$ (sono coprimi), quindi l'inverso esiste. L'obiettivo è trovare due coefficienti interi a e b che soddisfino l'identità di Bézout:

$$51a + 11b = 1 \tag{20}$$

In questa equazione, il valore b rappresenterà il nostro inverso modulare (la chiave privata).

➔GCD[51,11]=1 coprime

➔Find a,b s.t. 51 a + 11 b = 1

a	b	val	rem	
1	0	51		
0	1	11	4	Subtract 4x
1	-4	7	1	Subtract 1x
-1	5	4	1	Subtract 1x
2	-9	3	1	Subtract 1x
-3	14	1	=	

51x(-3)+11x(14)=1

25.2 Esecuzione Tabellare Passo-Passo

La tabella mostra l'evoluzione dell'algoritmo, che combina la divisione euclidea con la tracciatura dei coefficienti. Le colonne rappresentano:

- **a:** Il coefficiente del numero 51.
- **b:** Il coefficiente del numero 11.
- **val:** Il resto corrente (si parte dai numeri originali 51 e 11).

Il processo consiste nel sottrarre multipli della riga inferiore da quella superiore per ridurre il valore "val":

1. **Inizializzazione:** Si parte con le identità ovvie:

$$1 \cdot 51 + 0 \cdot 11 = 51 \quad \text{(Riga 1)}$$

$$0 \cdot 51 + 1 \cdot 11 = 11 \quad \text{(Riga 2)}$$

2. **Passo 1 (51 diviso 11):** $51 = 4 \times 11 + 7$. Sottraiamo 4 volte la Riga 2 dalla Riga 1.
   - $a : 1 - 4(0) = 1$
   - $b : 0 - 4(1) = -4$
   - $val : 51 - 4(11) = 7$
3. **Passo 2 (11 diviso 7):** $11 = 1 \times 7 + 4$. Sottraiamo 1 volta la nuova riga dalla precedente.
   - $a : 0 - 1(1) = -1$
   - $b : 1 - 1(-4) = 5$
   - $val : 11 - 1(7) = 4$
4. **Passi successivi:** Si continua iterativamente fino a ottenere un valore "val" pari a **1** (il MCD). L'ultima riga della tabella mostra il risultato finale:

$$a = -3, \quad b = 14, \quad \text{val} = 1$$

25.3 Interpretazione del Risultato

L'ultima riga ci dice che:

$$51(-3) + 11(14) = 1 \tag{21}$$

Verifica numerica: $-153 + 154 = 1$. Corretto.

Cosa significa per RSA? Applicando il modulo 51 a tutta l'equazione, il termine $51(-3)$ diventa 0. Resta:

$$11(14) \equiv 1 \pmod{51}$$

Questo significa che **14** è l'inverso moltiplicativo di 11 modulo 51. Se 51 fosse il nostro $\Phi(N)$ e 11 la nostra chiave pubblica e, allora **14 sarebbe la chiave privata** d.

26 Esempi Pratici di RSA: Calcolo e Ciclo Completo

Le slide conclusive presentano due esempi numerici ("Toy Examples") che utilizzano numeri piccoli per rendere trasparenti i calcoli matematici sottostanti all'algoritmo RSA.

26.1 Dettaglio del Calcolo dell'Inverso (Chiave Privata)

La prima slide ("How to compute RSA inverse") si concentra sul passaggio più ostico per gli studenti: l'uso dell'Algoritmo di Euclide Esteso per trovare la chiave privata d. **Dati di input:**

- Chiave pubblica $e = 13$.
- Modulo RSA $N = 77$ (dato da 11×7).
- Funzione Totiente $\Phi(77) = (11 - 1)(7 - 1) = 60$.

L'Obiettivo: Dobbiamo risolvere l'equazione lineare:

$$\Phi(N) \cdot a + e \cdot b = 1 \quad \Rightarrow \quad 60a + 13b = 1$$

Dove b sarà il nostro candidato per la chiave privata. **Esecuzione:** L'algoritmo di Euclide Esteso restituisce i coefficienti $\{5, -23\}$. Infatti:

$$60(5) + 13(-23) = 300 - 299 = 1$$

Gestione del Risultato Negativo: Il valore trovato per la chiave privata è -23. In aritmetica modulare, la chiave deve essere un intero positivo. Per convertire un numero negativo nel suo equivalente positivo modulo 60, basta aggiungere il modulo:

$$d = -23 \pmod{60} = -23 + 60 = 37$$

Quindi, la chiave privata finale è **37**.

26.2 Riepilogo del Ciclo di Vita RSA (Toy Example)

La seconda slide ("RSA recap") mostra un sistema completo, dalla generazione delle chiavi fino alla firma digitale, utilizzando un nuovo set di parametri.

26.2.1 1. Generazione delle Chiavi

- **Scelta dei Primi (Segreto):** $p = 11, q = 17$.
- **Modulo (Pubblico):** $N = 11 \times 17 = 187$.
- **Totiente (Segreto):** $\Phi(N) = 10 \times 16 = 160$.
- **Chiave Pubblica:** Si sceglie $e = 7$ (è primo rispetto a 160).
- **Chiave Privata:** Si calcola $d = 7^{-1} \pmod{160}$. Il risultato è 23.
  Verifica: $23 \times 7 = 161$. Diviso per 160 dà resto 1. Corretto.

26.2.2 2. Cifratura (Confidenzialità)

Per inviare un messaggio M cifrato:

$$C = M^7 \pmod{187}$$

Per decifrare, il destinatario usa $d = 23$:

$$M = C^{23} \pmod{187}$$

La matematica garantisce che $(M^7)^{23} = M^{161} \equiv M^1 \pmod{187}$.

26.2.3 3. Firma Digitale (Autenticazione)

Per firmare un messaggio M (o meglio, il suo hash $H(M)$):

- **Firma:** Il mittente usa la propria chiave **Privata** (23).

$$\text{TAG} = H(M)^{23} \pmod{187}$$

- **Verifica:** Il destinatario usa la chiave **Pubblica** del mittente (7).

$$\text{Check} = \text{TAG}^7 \pmod{187}$$

Se il risultato del calcolo TAG^7 corrisponde all'hash del messaggio M in chiaro, la firma è valida.

27 Il Problema dell'Identità: Man-in-the-Middle e la necessità di PKI

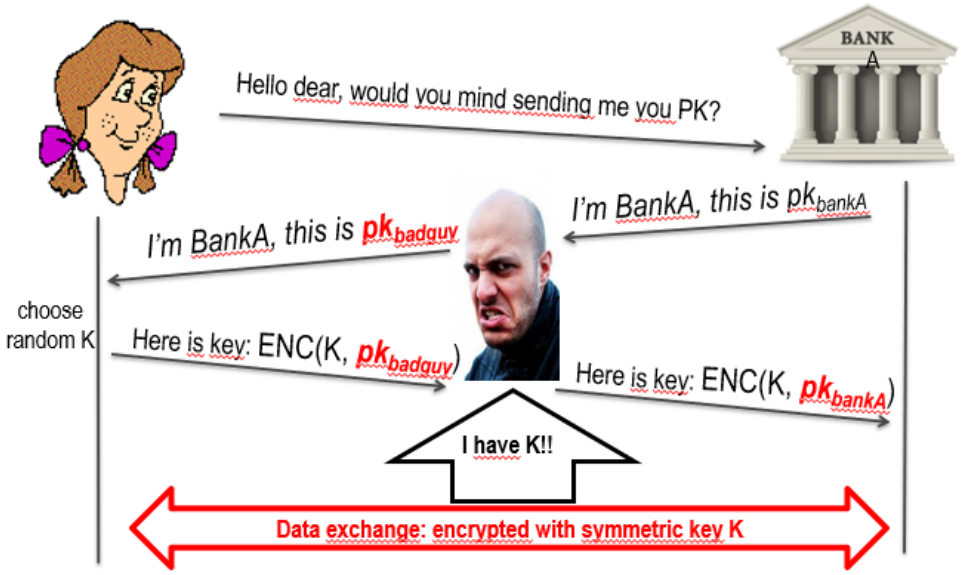
Le slide caricate illustrano come, nonostante la robustezza matematica degli algoritmi asimmetrici (come RSA e le Firme Digitali), i protocolli rimangano vulnerabili se non esiste un modo sicuro per legare una Chiave Pubblica alla reale Identità del suo proprietario.

27.1 Revisione: RSA Key Transport (Scenario Ideale)

Nel funzionamento corretto (mostrato nella slide "RSA Key Transport, review"), un utente (Flavia) vuole comunicare in modo sicuro con una Banca.

1. Flavia richiede la chiave pubblica della Banca.
2. La Banca invia la sua chiave pubblica PK_{BankA}.
3. Flavia genera una chiave di sessione simmetrica casuale K, la cifra con PK_{BankA} e la invia.
4. Risultato: Flavia e la Banca condividono il segreto K e possono comunicare cifrati.

27.2 L'Attacco: MITM su RSA Key Transport



La slide "MITM attack" mostra come un attaccante attivo (BadGuy) possa rompere questo schema senza decifrare RSA, ma semplicemente intercettando lo scambio iniziale.

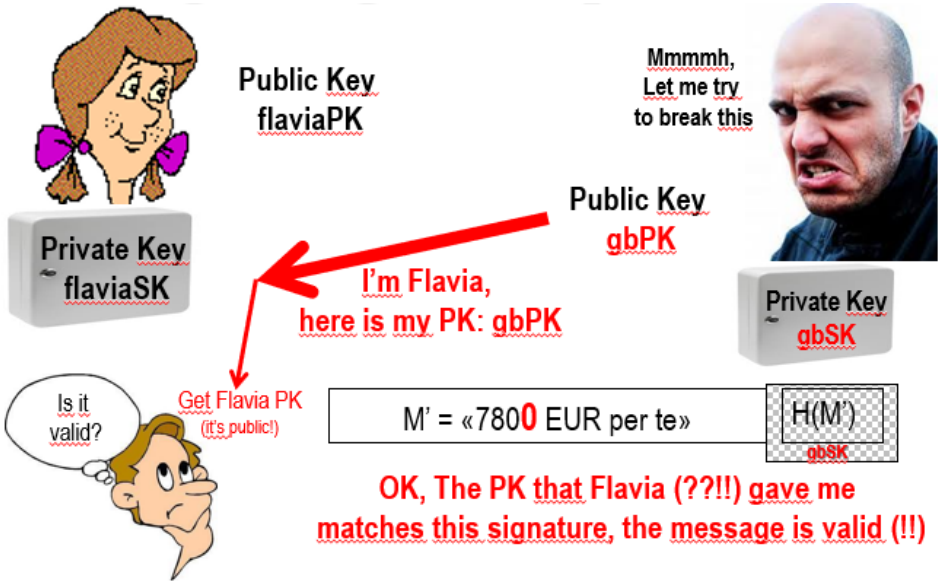
- **L'Inganno:** Quando Flavia chiede la chiave della Banca, l'attaccante intercetta la richiesta e risponde inviando la *propria* chiave pubblica PK_{badguy}, fingendo di essere la Banca.
- **L'Errore della Vittima:** Flavia non ha modo di distinguere PK_{BankA} da PK_{badguy} (sono entrambi solo numeri). Cifra quindi la chiave di sessione K usando la chiave dell'attaccante.
- **La Compromissione:** L'attaccante riceve il pacchetto, lo decifra con la sua chiave privata, ottiene K, e poi (per non farsi scoprire) ricifra K con la vera chiave della Banca e lo inoltra.
- **Conseguenza:** La Banca e Flavia comunicano credendo di essere al sicuro, ma l'attaccante possiede la chiave K e può leggere tutto.

SECURITY FOCUS: Attacco di Spoofing e Identity Binding

Qui la matematica di RSA non è stata violata (l'attaccante non ha scoperto la chiave privata della Banca). L'attaccante ha invece violato il **Binding** (il legame) tra Identità e Chiave. Questo tipo di attacco si chiama **Spoofing** (furto di identità o falsificazione). Senza un'autorità che certifichi "Questa chiave appartiene a BankA", Flavia non ha modo di sapere se la chiave ricevuta sia legittima o malevola.

27.3 Il Problema della Firma Digitale (Identity Binding)

Lo stesso problema affligge la Firma Digitale (slide "digital signature: problem").



- **Scenario:** L'attaccante ha una coppia di chiavi valida $(gbPK, gbSK)$. Invia un messaggio firmato correttamente con $gbSK$ ma dice "Ciao, sono Flavia, ecco la mia chiave: $gbPK$".
- **Verifica Matematica vs Verifica dell'Identità:** La vittima verifica la firma con $gbPK$: la matematica torna ("Is it valid? YES"). La firma è valida rispetto alla chiave fornita.
- **Il Bug Logico:** La firma prova che il messaggio è stato scritto dal proprietario di quella chiave, ma **non prova che il proprietario sia Flavia**.

27.4 La Radice del Problema: Legare Chiave e Identità

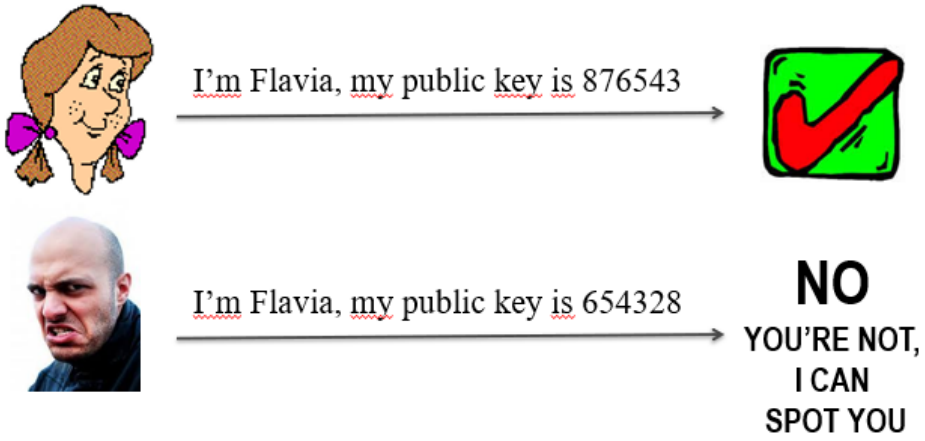
La slide "Three different scenarios, SAME problem" sintetizza il concetto fondamentale:

> Una chiave pubblica è solo una stringa di bit. Non ha il nome del proprietario scritto sopra.

Sia Flavia che l'Attaccante possono presentare le loro chiavi dicendo "Io sono Flavia". Senza un meccanismo esterno di verifica, non possiamo distinguere la verità.

- Abbiamo bisogno di un modo per creare un **legame crittografico** (cryptographic binding) tra una chiave pubblica e un'identità reale.

27.5 La Soluzione: Certificati Digitali e PKI



Per risolvere il problema dell'associazione (Binding) e prevenire il MITM sia su RSA che su Diffie-Hellman (anch'esso vulnerabile come ricordato nella slide apposita), viene introdotto il concetto di **Digital Certificates** e **Public Key Infrastructure (PKI)**. Un Certificato Digitale non è altro che un documento che attesta:

"La chiave pubblica X appartiene all'identità Y"

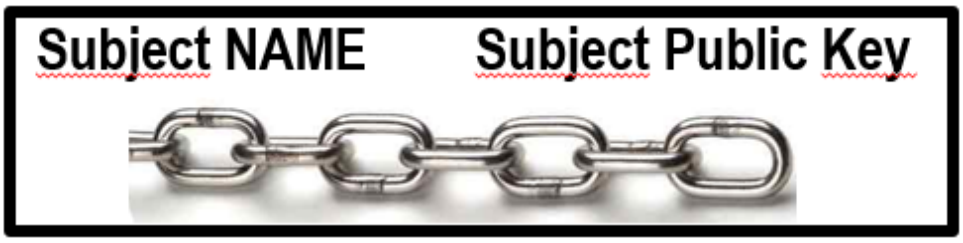
Questo documento è a sua volta firmato digitalmente da una terza parte fidata (Certification Authority), creando una catena di fiducia che sostituisce la fiducia "cieca" nella chiave ricevuta.

28 La Soluzione: I Certificati Digitali

Le slide introducono i **Certificati Digitali** come la risposta definitiva al problema della distribuzione sicura delle chiavi pubbliche e alla prevenzione degli attacchi Man-in-the-Middle.

28.1 Il Ruolo Principale: Il Binding

La funzione fondamentale di un certificato digitale è sintetizzata nella slide con una metafora visiva molto chiara: una catena d'acciaio. L'obiettivo è creare un legame indissolubile, tecnicamente chiamato **Binding**, tra due entità:



1. **Una Chiave Pubblica:** Di per sé anonima e costituita solo da numeri.
2. **Un Soggetto (Subject):** L'identità reale proprietaria di quella chiave. Può trattarsi di:

- Una persona fisica (es. "Mario Rossi").
- Un'azienda o organizzazione (es. "Google Inc.").
- Un'entità legale o un dispositivo (es. un server web).

28.2 La Natura del Legame: Cryptographic Binding

Non basta scrivere il nome accanto alla chiave (chiunque potrebbe farlo falsificando il documento). Il legame deve essere garantito matematicamente. La slide definisce questo concetto come **Cryptographic Binding**. Per realizzare questo legame sicuro, entra in gioco un terzo attore fondamentale:

- **Trusted Third Party (TTP):** Una terza parte fidata, nota come **Certification Authority (CA)** (Autorità di Certificazione).
- **Il Meccanismo di Garanzia:** La CA verifica l'identità del soggetto e la sua chiave pubblica, quindi "firma" l'associazione tra i due utilizzando la propria **Firma Digitale**.

In sintesi: Un certificato digitale è un file che dice "Questa chiave pubblica appartiene a questo Soggetto", e questa affermazione è firmata digitalmente da una CA di cui ci fidiamo. Se qualcuno (un attaccante) provasse a modificare la chiave o il nome nel certificato, la firma della CA non corrisponderebbe più (verifica dell'integrità), rendendo la manomissione evidente.

29 Ciclo di Vita del Certificato Digitale

Le slide illustrano il processo completo di gestione di un certificato digitale, suddiviso in due macro-fasi: l'emissione (Issuing) e la verifica (Verifying). Particolare attenzione viene posta sulla distinzione tra la validità formale del certificato e l'autenticità dell'interlocutore, introducendo il concetto di "Certificate Replay".

29.1 A) Emissione del Certificato (Issuing)

Questa fase avviene *offline* o comunque una tantum, prima che inizi qualsiasi comunicazione sicura. Coinvolge tre passaggi critici per garantire la sicurezza alla radice:

1. **Generazione delle Chiavi:** L'entità richiedente (es. la Banca) genera autonomamente la propria coppia di chiavi (Pubblica PK e Privata SK).
2. **Custodia della Chiave Segreta:** La chiave privata (SK) viene memorizzata localmente e **non deve mai essere trasmessa** o rivelata a nessuno, nemmeno alla CA.
3. **Certificazione:** La Banca invia alla Certification Authority (CA) le proprie credenziali (Nome e Chiave Pubblica) attraverso un **canale sicuro** (fisico o fortemente autenticato).
4. **Firma:** La CA verifica l'identità della Banca e crea il certificato firmando digitalmente l'associazione tra il nome e la chiave pubblica:

$$\text{CERT} = (\text{BankName}, \text{BankPK})_{\text{CA\_sign}}$$

29.2 B) Verifica della Validità (Lato Client)

Quando un client (es. Flavia) si connette alla Banca, riceve il certificato. Il processo di verifica formale segue due controlli logici:

1. **Trust della CA:** Il client verifica se la CA che ha emesso il certificato è presente nella propria lista di autorità fidate (*Trusted Root Store*).
2. **Integrità della Firma:** Il client verifica matematicamente la firma della CA sul certificato. Se la verifica passa, il client ha la certezza che il certificato non è stato manomesso e contiene una chiave pubblica valida.

29.3 Il Problema di Sicurezza: Certificate Replay

A questo punto sorge una domanda critica (scritta in rosso nelle slide): *"Il fatto che il certificato sia valido significa che sto parlando davvero con la Banca?"* La risposta è **NO (Not Yet)**. Come mostra la slide con l'attaccante:

- Il certificato è un file pubblico. Chiunque può scaricarlo e inviarlo a terzi.
- Un attaccante (Man-in-the-Middle o Replay Attacker) può intercettare il certificato valido della Banca e inviarlo al client presentandosi come "La Banca".
- I controlli 2 e 3 (Trust e Firma) avranno esito positivo perché il certificato in sé è autentico, anche se chi lo sta inviando è un impostore.

29.4 La Soluzione: Proof of Possession (PoP)

Per essere sicuri di parlare con il legittimo proprietario del certificato, non basta verificare il certificato, bisogna verificare il possesso della chiave privata associata (SK). Questo passaggio (Step 4) è chiamato **Proof of Possession**.

Il client deve sfidare l'interlocutore chiedendogli di compiere un'operazione che solo chi possiede SK può completare, utilizzando un dato "fresco" (*Nonce* o *Challenge*) per evitare la replica di vecchie risposte:

1. **Challenge con Firma:** "Ti invio un numero casuale, firmalo con la tua SK". Il client verificherà la firma con la PK del certificato.
2. **Challenge con Decifratura:** "Ti invio un numero casuale cifrato con la tua PK, dimmi che numero è". Solo chi ha SK può decifrarlo.

Solo superando questo ultimo passaggio si ha l'autenticazione completa dell'entità.

30 Protocolli di Autenticazione e Catene di Fiducia

Le slide analizzano come passare dalla teoria dei certificati alla pratica, rispondendo a due domande: "Come verifico concretamente che il server possieda la chiave privata?" e "Cosa succede se non conosco l'autorità che ha emesso il certificato?".

30.1 1. Approccio Pratico: Autenticazione Implicita (TLS/RSA)

La prima slide ("Practical TLS approach") illustra come avveniva storicamente lo scambio di chiavi in protocolli come SSL/TLS (nella modalità *RSA Key Transport*). Qui l'autenticazione non è un passaggio separato, ma è **implicita** nello scambio della chiave di sessione.

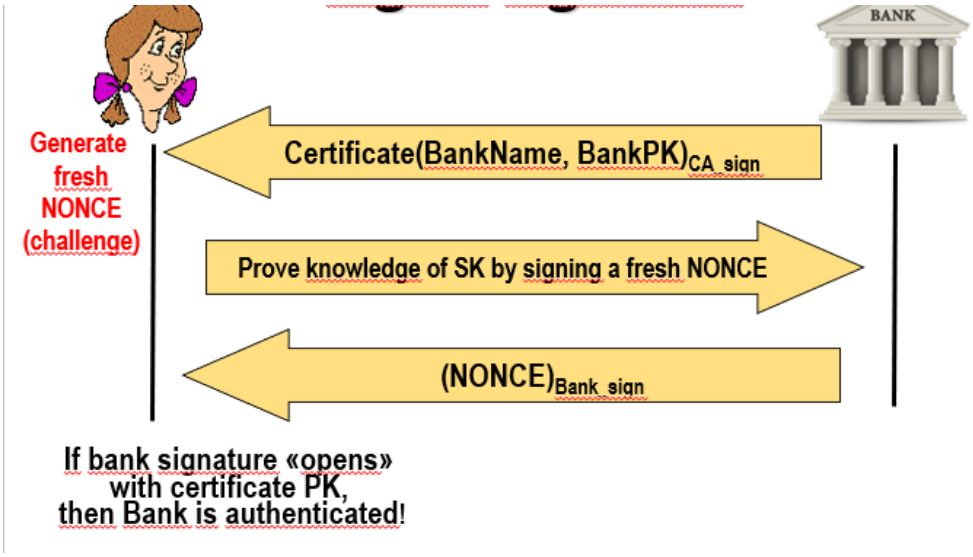
- **Il Flusso:**
  1. La Banca invia il suo certificato contenente la Chiave Pubblica (PK_{Bank}).
  2. Il Client (Flavia) verifica il certificato e, se valido, genera una chiave simmetrica fresca K (Session Key).
  3. Il Client cifra K usando la chiave pubblica della Banca: $C = \text{ENC}(K, PK_{\text{Bank}})$.
  4. Il Client invia C alla Banca.
- **Perché è sicuro?** Solo la vera Banca possiede la Chiave Privata (SK_{Bank}) necessaria per decifrare C e ottenere K.
- **Verifica:** Se c'è un impostore (Rogue/MITM), esso non potrà decifrare la chiave K. Di conseguenza, quando il Client inizierà a inviare dati cifrati con K (freccia rossa "Data exchange"), l'impostore vedrà solo spazzatura e la connessione cadrà. Il possesso della SK è provato dalla capacità di comunicare.

30.2 2. Autenticazione Esplicita: Challenge-Response

Le due slide successive ("Proof of knowledge of SK") mostrano metodi per verificare l'identità *prima* o indipendentemente dallo scambio di dati, utilizzando un meccanismo di sfida (Challenge-Response) basato su un **NONCE** (Number used ONCE). L'uso di un NONCE "fresco" (generato al momento) è fondamentale per prevenire attacchi di *Replay* (dove un attaccante riutilizza vecchie risposte valide).

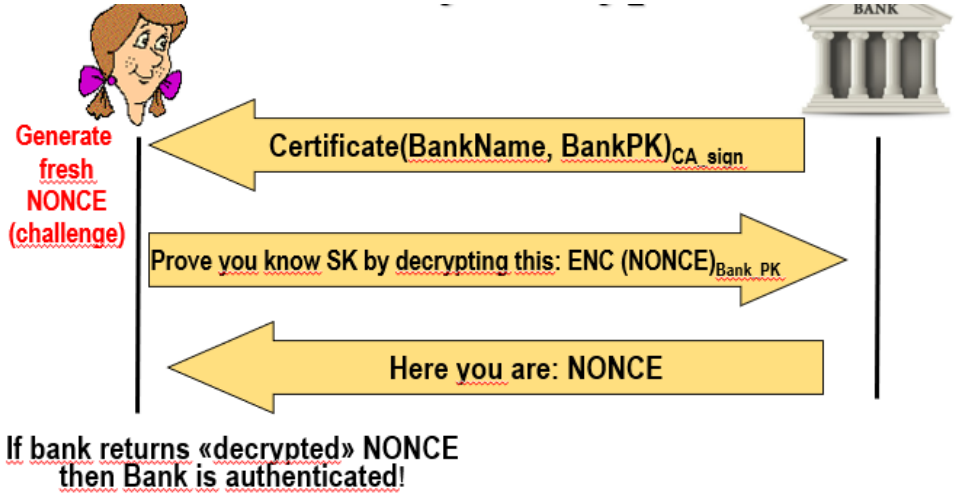
SECURITY FOCUS: Replay Attack e la Soluzione Nonce

Un **Replay Attack** avviene quando un attaccante registra una comunicazione valida (es. un comando di login cifrato inviato ieri) e la "riproduce" (invia nuovamente) al server oggi. Anche se l'attaccante non può decifrare il messaggio, se il server lo accetta come valido, l'attaccante ottiene l'accesso.
Il **NONCE** (Number Used Once) risolve il problema garantendo la "freschezza" (Liveness). Il server dice: "Per autenticarti ora, cifra questo numero casuale che ho appena generato: 84920". Se l'attaccante riproduce il messaggio di ieri (che conteneva la risposta al numero 12345), il server lo scarta.



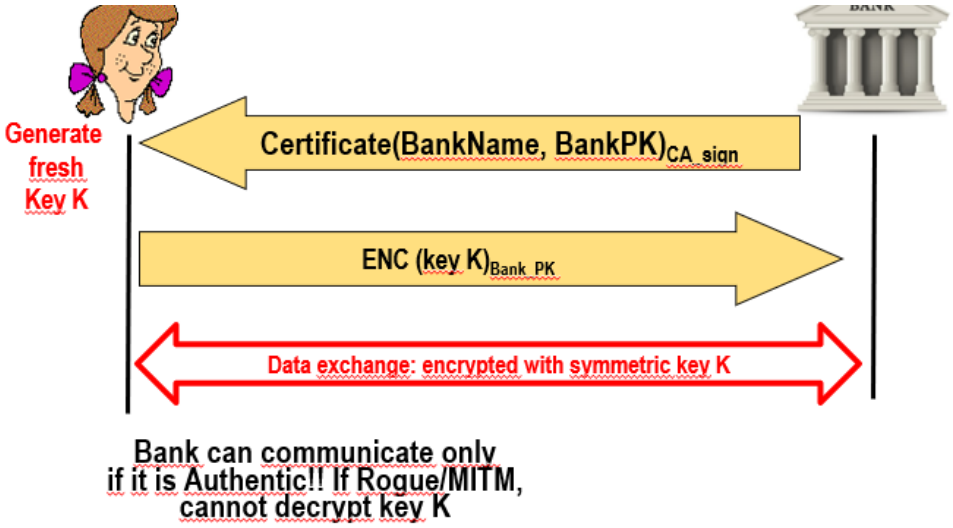
30.2.1 Metodo A: Via Firma Digitale

1. Il Client genera un NONCE e lo invia alla Banca ("Challenge").
2. La Banca firma il NONCE con la sua Chiave Privata (SK): Sign($NONCE, SK$).
3. Il Client verifica la firma con la PK del certificato. Se la firma è valida, la Banca è autenticata.



30.2.2 Metodo B: Via Decifratura (Public Key Encryption)

1. Il Client genera un NONCE, lo cifra con la Chiave Pubblica della Banca e lo invia.
2. La Banca deve decifrarlo usando la sua Chiave Privata (SK) e restituire il NONCE in chiaro.
3. Se il numero restituito corrisponde a quello generato, la Banca è autenticata.



30.3 3. Catene di Certificati (Chain of Trust)

L'ultima slide ("Certificate chains") affronta il problema della scalabilità della fiducia. Un client (browser/OS) non può memorizzare le chiavi pubbliche di tutte le possibili CA del mondo. Solitamente ha una lista preinstallata di poche "Root CA" super-fidate (es. Universetrust).

Lo Scenario:

- Il Client si connette alla Banca.
- La Banca presenta un certificato firmato da "LUNATRUST".
- Il Client controlla la sua lista e vede che **non conosce** LUNATRUST ("Not in my list of trusted").

La Soluzione Gerarchica: Il Client non rifiuta subito la connessione, ma chiede: "Chi garantisce per Lunatrust?".

1. Lunatrust presenta il proprio certificato, che è firmato da **UNIVERSETRUST**.

2. Il Client controlla la lista e trova UNIVERSETRUST ("OK, this is in my list").

3. **Validazione a catena:** Poiché il Client si fida di Universetrust, e Universetrust si fida di Lunatrust, allora il Client (per proprietà transitiva) si fida della Banca garantita da Lunatrust.

Questa struttura permette di delegare la fiducia e creare un'infrastruttura globale (PKI).

31 Public Key Infrastructure (PKI) e Standard X.509

Le slide introducono il concetto di **PKI** (Infrastruttura a Chiave Pubblica), il framework che governa l'intero ciclo di vita dei certificati digitali e delle chiavi pubbliche. Non si tratta di un singolo software, ma di un ecosistema complesso.

31.1 Definizione e Architettura

Come definito nella slide, una PKI è l'insieme di **protocolli, politiche (policies) e meccanismi tecnici** necessari per supportare lo scambio sicuro di chiavi pubbliche. Affinché una PKI funzioni correttamente, l'architettura deve soddisfare quattro requisiti fondamentali:

1. **Formato Standard:** Tutti i partecipanti devono parlare la stessa lingua. I certificati devono seguire una struttura dati universale (vedasi X.509).

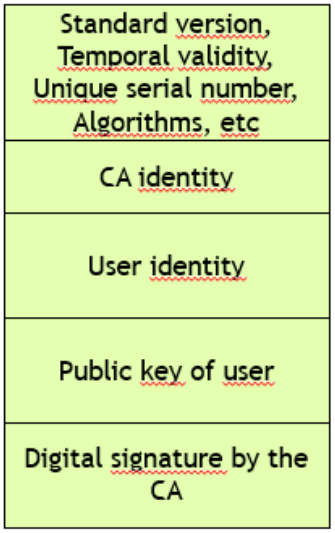
2. **Relazioni di Fiducia:** Devono essere definite le relazioni tra le Certification Authorities (CA) e gli utenti finali (es. gerarchie, catene di fiducia viste nella sezione precedente).

3. **Directory Services:** Deve esistere un sistema per rendere pubblici e reperibili i certificati (repository pubblici).

4. **Gestione del Ciclo di Vita (Emissione e Revoca):** Questo è un punto critico evidenziato con enfasi nella slide (*Revoking !!*). Una PKI non deve solo emettere certificati, ma deve avere procedure robuste per **revocarli** prima della loro scadenza naturale (ad esempio, se una chiave privata viene rubata o compromessa). Senza revoca, un certificato rubato rimarrebbe valido fino alla scadenza, creando un enorme rischio di sicurezza.

31.2 Il Formato Standard: X.509

Per garantire l'interoperabilità globale (ad esempio, permettere a un browser americano di validare un certificato emesso da una banca italiana), è stato adottato lo standard **X.509**. La slide "High Level" mostra la struttura interna di un certificato X.509, che può essere visto come un documento digitale diviso in sezioni:



- **Metadati (Header):** Contiene informazioni amministrative.
  - *Version:* La versione dello standard.
  - *Serial Number:* Un numero univoco assegnato dalla CA per identificare quello specifico certificato (fondamentale per le liste di revoca).
  - *Temporal Validity:* Le date di inizio e fine validità (*Not Before / Not After*). Fuori da questo intervallo, il certificato scade.
  - *Algorithms:* L'indicazione degli algoritmi usati per la firma (es. SHA-256 con RSA).
- **Identità (Soggetti):**
  - *Issuer (CA Identity):* Chi ha emesso il certificato.
  - *Subject (User Identity):* A chi appartiene il certificato (es. www.google.com).
- **Payload (Chiave Pubblica):** La parte più importante: la **Chiave Pubblica dell'utente**. Questa è l'informazione che il certificato intende proteggere e associare all'identità.
- **Sigillo di Garanzia (Firma):** In calce al documento si trova la **Firma Digitale della CA**. La CA prende tutti i campi precedenti (Metadati + Identità + Chiave Pubblica), ne calcola l'hash e lo cifra con la propria Chiave Privata. Questo rende il certificato inalterabile.