RSA 25

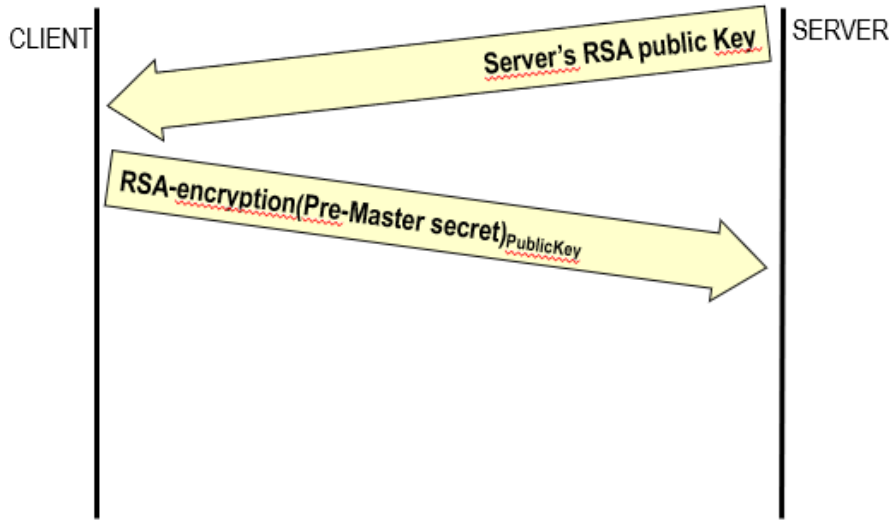
2 dicembre 2025

Indice

1 Sicurezza del Trasporto Chiavi RSA

Le slide presentano un approfondimento critico sull'utilizzo di RSA per lo scambio di chiavi (Key Transport), evidenziando come l'implementazione "scolastica" (Textbook RSA) sia vulnerabile a sofisticati attacchi crittografici se non adeguatamente protetta.

1.1 Il Protocollo di RSA Key Transport



La prima immagine illustra il classico meccanismo di *Key Establishment* utilizzato in protocolli storici come le prime versioni di SSL/TLS. Il flusso avviene come segue:

1. **Negoziazione:** Il Server invia al Client la propria **Chiave Pubblica RSA** (N, e).
2. **Generazione del Segreto:** Il Client genera un numero casuale, definito **Pre-Master Secret**. Questo valore sarà la base per generare le chiavi simmetriche di sessione.
3. **Cifratura:** Il Client cifra il Pre-Master Secret utilizzando la chiave pubblica del Server:
$$C = \text{ENC}_{\text{RSA}}(K_{\text{pre-master}})$$
4. **Trasmissione:** Il Client invia il crittogramma al Server, che lo decifrerà con la propria chiave privata.

Nota Didattica: Perché il Key Transport?

In questo schema, RSA non viene usato per cifrare tutto il traffico dati (sarebbe troppo lento), ma solo per trasportare in sicurezza una piccola chiave ("Pre-Master Secret"). Una volta che entrambe le parti possiedono questo segreto, possono derivare chiavi simmetriche (es. AES) molto più efficienti per cifrare la sessione vera e propria. Questo è il cuore della crittografia ibrida.

1.2 Vulnerabilità: Chosen Ciphertext Attack (CCA)

La slide successiva pone una domanda fondamentale: *"Is RSA robust against Chosen ciphertext attacks?"*. La risposta per l'RSA "puro" (senza padding, o Textbook RSA) è **NO**. L'attacco descritto sfrutta la proprietà di **Malleabilità** (o Omomorfismo moltiplicativo) di RSA.

1.2.1 Scenario dell'Attacco

- **Obiettivo:** L'attaccante intercetta un messaggio cifrato $C = M^e \pmod N$ (dove M è il segreto) e vuole recuperare M.
- **Potere dell'Attaccante (Oracolo):** Si assume che l'attaccante abbia accesso a un "Oracolo di Decifratura". Questo significa che può inviare al server qualsiasi testo cifrato (tranne l'originale C, altrimenti l'attacco sarebbe banale) e ottenere la decifratura o osservare il comportamento del server (es. messaggi di errore, tempi di risposta).

1.2.2 Esecuzione Matematica dell'Attacco

L'attaccante procede come segue per aggirare il divieto di inviare C:

1. **Mascheramento (Blinding):** L'attaccante sceglie un numero casuale r. Costruisce un *nuovo* testo cifrato X moltiplicando il crittogramma originale per la cifratura di r:
$$X = (C \cdot r^e) \pmod N \tag{1}$$
Per il server, X appare come un messaggio casuale, diverso da C.
2. **Interrogazione dell'Oracolo:** L'attaccante invia X all'oracolo di decifratura. Il server esegue onestamente la decifratura con la sua chiave privata d:
$$\begin{aligned} \text{Dec}(X) &= X^d \pmod N \\ &= (C \cdot r^e)^d \pmod N \\ &= (M^e \cdot r^e)^d \pmod N \\ &= (M \cdot r)^{ed} \pmod N \\ &= M \cdot r \pmod N \end{aligned}$$
Poiché $e \cdot d \equiv 1$, l'operazione rimuove gli esponenti. L'oracolo restituisce il valore $Y = M \cdot r$.
3. **Recupero del Messaggio (Unblinding):** L'attaccante riceve $Y = M \cdot r$. Poiché l'attaccante conosce r, può calcolare il suo inverso modulare r^{-1}. Per trovare il messaggio segreto M, basta moltiplicare il risultato dell'oracolo per l'inverso di r:
$$M = Y \cdot r^{-1} \pmod N \tag{2}$$

1.2.3 Conclusione

Questo attacco dimostra che l'RSA puro non garantisce la sicurezza CCA (Chosen Ciphertext Security). Per mitigare questo problema nei sistemi reali (come TLS), è obbligatorio utilizzare schemi di **Padding** robusti (come PKCS#1 v1.5 o meglio **OAEP**) che rompono la struttura algebrica che rende possibile la malleabilità.

2 Il Problema della Malleabilità in RSA

La slide affronta una debolezza intrinseca dell'RSA nella sua forma base (Textbook RSA), nota come **Malleabilità**. Questa proprietà è la causa diretta che rende possibili gli attacchi a testo cifrato scelto (CCA) analizzati nella sezione precedente.

2.1 Definizione di (Non) Malleabilità

In crittografia, la malleabilità è una proprietà indesiderata per un algoritmo di cifratura.

- **Definizione:** Un sistema è *malleabile* se, dato un testo cifrato c corrispondente a un messaggio m, un attaccante è in grado di generare un diverso testo cifrato c' che, una volta decifrato, produce un messaggio m' correlato matematicamente a m in modo prevedibile.
- **Il caso RSA:** RSA è perfettamente malleabile a causa della sua struttura moltiplicativa (omomorfismo).

$$\text{Se } c = m^e \pmod N \quad \text{e l'attaccante calcola } c' = c \cdot x^e \pmod N$$

$$\text{Allora } \text{DEC}(c') = m \cdot x \pmod N$$

L'attaccante ha modificato il contenuto del messaggio (moltiplicandolo per x) senza decifrarlo e senza conoscere la chiave privata.

Approfondimento: Malleabilità = Omomorfismo

In matematica, un "omomorfismo" è una trasformazione che preserva le operazioni. RSA preserva la moltiplicazione: la cifratura del prodotto è uguale al prodotto delle cifrature ($Enc(A \cdot B) = Enc(A) \cdot Enc(B)$). Questa proprietà è utile in alcuni contesti (es. calcolo sicuro nel cloud), ma è disastrosa per la cifratura standard perché permette a un attaccante di manipolare il testo in chiaro agendo solo su quello cifrato.

2.2 L'Importanza della Non-Malleabilità

Un sistema sicuro deve garantire la **Non-Malleabilità**. L'attaccante non dovrebbe essere in grado di produrre un c' valido. Se tenta di modificare c, il risultato della decifratura dovrebbe essere un errore o un messaggio pseudo-casuale privo di relazione con l'originale.

2.3 Il Ruolo del Padding RSA

Per mitigare questo problema, lo standard RSA prevede l'utilizzo del **Padding** (come PKCS#1 v1.5 o OAEP).

- **Funzione:** Il padding inserisce una struttura specifica e bit casuali nel messaggio prima dell'esponenziazione.
- **Effetto:** Se un attaccante prova a modificare il testo cifrato sfruttando la proprietà matematica, il risultato decifrato non rispetterà più la struttura rigida del padding.
- **Risultato:** Il sistema di decifratura rileverà che il padding è errato e scarterà il messaggio, bloccando l'attacco di malleabilità.

2.4 Nota Critica (L'Anticipazione)

La slide si chiude con una nota amara (la faccina triste): *"But we will see later on whether this is the case..."*. Sebbene il padding sia progettato per risolvere la malleabilità, implementarlo non è banale. Se il sistema rivela *perché* un messaggio è stato scartato (ad esempio distinguendo tra "errore di padding" e "errore generico"), si apre la strada a nuovi attacchi devastanti noti come **Padding Oracle Attacks** (es. Attacco di Bleichenbacher), che verranno trattati successivamente.

3 RSA Padding: Lo Standard PKCS #1 v1.5

La slide introduce lo standard industriale più noto per l'implementazione sicura di RSA: il **PKCS #1 v1.5**. Fino ad ora abbiamo discusso la malleabilità e i rischi dell'RSA puro; questo standard definisce una struttura rigida per il blocco dati prima che venga cifrato, mitigando molti di questi rischi.

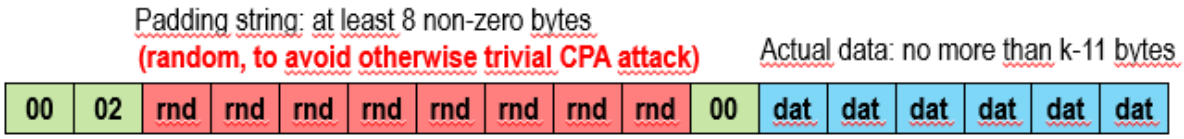
3.1 Che cos'è PKCS?

PKCS sta per *Public-Key Cryptography Standards*. È una famiglia di standard sviluppata inizialmente dai laboratori RSA (l'azienda fondata dagli inventori) per garantire l'interoperabilità tra sistemi diversi.

- **PKCS #1:** È il primo di questi standard e specifica le primitive per la crittografia e la firma RSA.
- **Riferimenti:** È stato formalizzato dall'IETF nelle RFC 2313 e, più recentemente, RFC 8017.

3.2 La Struttura del Blocco (Padding Scheme)

Lo standard impone che il messaggio M non venga elevato a potenza direttamente. Invece, viene incapsulato in una struttura più grande che occupa l'intera lunghezza del modulo.



k = size of the RSA modulus in bytes – formally: k = integer value which satisfies

$$2^{8(k-1)} \leq n < 2^{8k}$$

e.g. with RSA-1024, k = 128 (in most cases) → you can encrypt up to 117 bytes

Come mostrato nel diagramma colorato della slide, il blocco è composto da una sequenza specifica di byte:

1. **Header (Byte 00):** Il primo byte è sempre `00`. Questo garantisce che il numero intero risultante dal blocco sia sempre strettamente minore del modulo n (requisito matematico di RSA).

2. **Block Type (Byte 02):** Il secondo byte indica il tipo di operazione. Il valore `02` specifica che si tratta di una operazione di **Cifratura** con chiave pubblica.

3. **Padding String (Blocchi Rossi):** Questa è la parte fondamentale per la sicurezza.

   - Deve essere composta da almeno **8 byte**.
   - I byte devono essere **Casuali (Random)** e **Non-Zero**.
   - **Obiettivo:** L'uso di byte casuali introduce la *probabilistic encryption*. Cifrare due volte lo stesso messaggio produrrà due crittogrammi completamente diversi. Questo previene gli attacchi a testo in chiaro scelto (CPA - Chosen Plaintext Attacks) e impedisce a un attaccante di verificare se un certo crittogramma corrisponde a un messaggio ipotizzato.

4. **Separatore (Byte 00):** Un singolo byte a zero che serve a segnalare la fine della stringa di padding casuale e l'inizio dei dati reali. (Ecco perché il padding non può contenere zeri).

5. **Data (Blocchi Blu):** Il messaggio vero e proprio ("Actual data").

Concetto Fondamentale: Probabilistic Encryption

Senza il padding casuale (blocchi rossi), RSA sarebbe deterministico: cifrare "CIAO" produrrebbe sempre lo stesso output. Un attaccante potrebbe cifrare "CIAO" e confrontare il risultato. Con il padding, l'output cambia ogni volta, rendendo impossibile questo confronto diretto.

3.3 Vincoli Dimensionali

L'introduzione di questa struttura comporta un overhead (spazio occupato dal padding) che riduce lo spazio disponibile per il messaggio vero e proprio. Definiamo k come la lunghezza del modulo RSA in byte (es. per RSA-1024, $k = 128$ byte). Il blocco totale deve essere lungo k byte. Calcoliamo lo spazio occupato dalla struttura fissa:

- 1 byte iniziale (`00`)
- 1 byte tipo (`02`)
- Almeno 8 byte di padding
- 1 byte separatore (`00`)

Totale overhead minimo: $1+1+8+1 = \mathbf{11\ byte}$. Di conseguenza, la lunghezza massima del messaggio cifrabile in un singolo blocco è:

$$\text{Max Data} = k - 11 \text{ bytes} \tag{3}$$

Esempio riportato nella slide: Con una chiave RSA a 1024 bit ($k = 128$ byte), possiamo cifrare al massimo:

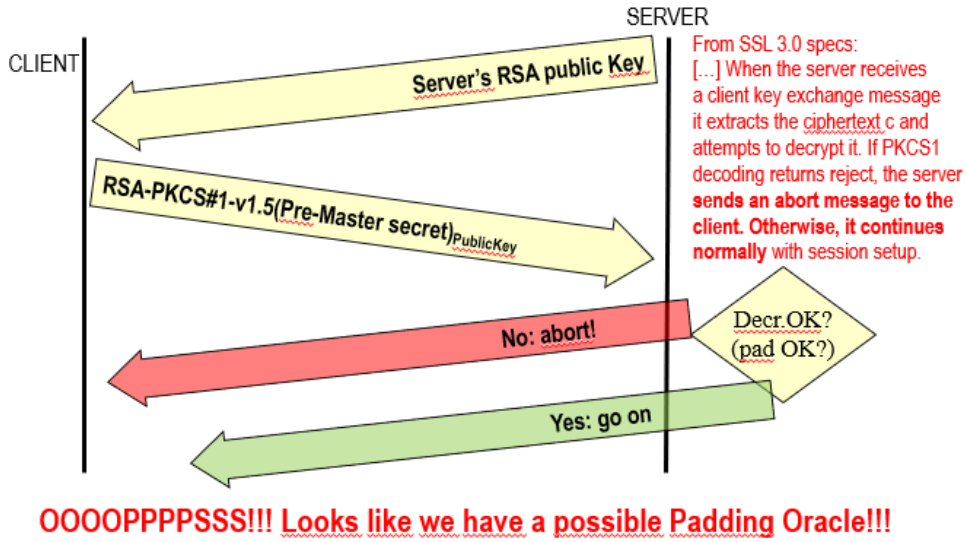
$$128 - 11 = 117 \text{ bytes}$$

Se il messaggio è più lungo, si deve ricorrere alla crittografia ibrida (cifrare il messaggio con AES e cifrare solo la chiave AES con RSA).

4 Vulnerabilità in SSL v3.0: La Nascita del Padding Oracle

La slide "RSA key transport (until SSL v3.0 included)" analizza il flusso di handshake utilizzato storicamente nei protocolli sicuri fino a SSL 3.0, evidenziando un errore di progettazione nella gestione degli errori di decifratura RSA.

4.1 Il Protocollo di Handshake



Il diagramma mostra il classico scambio di chiavi RSA:

1. Il **Server** invia la sua Chiave Pubblica RSA al Client.
2. Il **Client** genera il *Pre-Master secret* (il seme della chiave di sessione), lo formatta secondo lo standard **PKCS #1 v1.5** (visto nella sezione precedente), lo cifra con la chiave pubblica del server e lo invia.

4.2 Il Comportamento del Server (L'Oracolo)

La parte critica è descritta nel blocco di testo rosso a destra, estratto dalle specifiche ufficiali di SSL 3.0. Quando il server riceve il testo cifrato, tenta di decifrarlo e verifica la correttezza del padding PKCS #1. Qui si crea una biforcazione comportamentale (il rombo giallo "Decr. OK?"):

- **Caso A (Padding Errato):** Se la decifratura produce un blocco che non inizia con `00 02`, o non ha il separatore `00` nella posizione corretta, il server dichiara l'errore e invia immediatamente un messaggio di **Abort (Alert)** al client.
- **Caso B (Padding Corretto):** Se la struttura del padding è valida, il server prosegue con la negoziazione della sessione ("continues normally").

4.3 Perché questo è un problema?

La scritta in basso "OOOOPPPSSS!!! Looks like we have a possible Padding Oracle!!!" identifica la vulnerabilità. Il server si comporta involontariamente come un **Oracolo di Padding**. Un "Oracolo", in crittografia, è un sistema che risponde a domande specifiche. In questo caso, l'attaccante può inviare migliaia di messaggi cifrati casuali (o manipolati appositamente) al server e osservare la sua reazione:

- Se riceve un messaggio di "Abort", sa che il testo decifrato **non** rispettava lo standard PKCS #1.
- Se **non** riceve l'errore (o nota che la connessione resta aperta un istante in più), sa che il testo decifrato **rispettava** lo standard PKCS #1 (cioè iniziava con `00 02` ...).

Questa semplice informazione binaria (Sì/No), combinata con la malleabilità di RSA, è sufficiente per permettere a un attaccante di decifrare completamente il messaggio originale senza conoscere la chiave privata (il famoso *Attacco di Bleichenbacher* del 1998).

5 L'Attacco di Bleichenbacher (1998)

La slide "Bleichenbacher's Oracle" introduce uno degli attacchi più celebri e longevi nella storia della crittografia, presentato dal ricercatore Daniel Bleichenbacher nel 1998. Questo attacco ha dimostrato come la vulnerabilità teorica del "Padding Oracle" in SSL v3.0 potesse essere sfruttata praticamente per decifrare chiavi RSA.

5.1 Tipologia di Attacco: Adaptive Chosen Ciphertext

L'attacco è classificato come **Adaptive Chosen Ciphertext Attack (CCA2)**.

- **Chosen Ciphertext:** L'attaccante sceglie dei testi cifrati da inviare al server.
- **Adaptive:** L'attaccante non sceglie tutti i testi all'inizio, ma costruisce ogni nuova richiesta ("query") basandosi sulla risposta ottenuta da quella precedente.
- **Obiettivo:** Colpire l'implementazione di RSA con padding **PKCS #1 v1.5**.

5.2 Il Funzionamento dell'Oracolo

L'attacco sfrutta il comportamento del server descritto nella sezione precedente. Il server agisce come un oracolo che risponde (involontariamente) a una domanda binaria:

> *"Il messaggio decifrato inizia con i byte corretti '00 02'?"*

Inviando circa un milione di messaggi cifrati manipolati (sfruttando la malleabilità di RSA), l'attaccante può restringere progressivamente l'intervallo matematico in cui si trova il messaggio originale m, fino a determinarlo completamente. Nel contesto di SSL/TLS, questo significa recuperare il *Pre-Master Secret* e quindi decifrare l'intera sessione HTTPS.

5.3 Evoluzione dei Protocolli e Contromisure

La storia di questo attacco ha influenzato l'evoluzione degli standard di sicurezza per vent'anni:

1. **Target (SSL v3.0):** Era vulnerabile perché inviava esplicitamente un messaggio di errore distinto in caso di padding errato.
2. **Mitigazione (TLS 1.0 - 1.2):** Non potendo cambiare l'algoritmo RSA (troppo diffuso), gli standard successivi hanno introdotto contromisure complesse. La regola imposta era: *"Se il padding è errato, non dare errore subito. Genera un numero casuale e procedi come se fosse il Pre-Master Secret corretto, fallendo solo alla fine del protocollo (sul controllo MAC)"*. Questo rendeva indistinguibile l'errore di padding da un errore di chiave, accecando l'oracolo.
3. **Soluzione Definitiva (TLS 1.3):** L'unica vera soluzione è stata **rimuovere** il problema alla radice. Il protocollo TLS 1.3 ha eliminato completamente il supporto a *RSA Key Transport*. Oggi RSA si usa solo per le firme digitali, mentre lo scambio delle chiavi avviene via Diffie-Hellman (EC-DHE).

5.4 La Persistenza del Problema ("Zombie Attack")

La slide nota con enfasi: **"But still around... at least until 2017/18!!"**. Nonostante le correzioni teoriche in TLS 1.0+, implementare correttamente la contromisura (gestire gli errori in "tempo costante" senza lasciar trapelare nulla) è difficilissimo. Nel 2017 è stato scoperto l'attacco **ROBOT** (Return Of Bleichenbacher's Oracle Threat), che ha dimostrato come molti server moderni fossero ancora vulnerabili a varianti di questo attacco vecchio di 20 anni, a causa di imperfezioni nell'implementazione del codice di gestione degli errori.

Analisi: Side Channel Attack

L'attacco di Bleichenbacher non viola la matematica di RSA, ma sfrutta un "canale laterale": il messaggio di errore. In crittografia, qualsiasi informazione che il sistema lascia trapelare sul suo stato interno (errori, tempo di esecuzione, consumo di energia) può essere usata per rompere la chiave, anche se l'algoritmo matematico è perfetto.

6 Dettagli Tecnici dell'Attacco di Bleichenbacher

La slide "Bleichenbacher's Oracle" scompone i passaggi matematici che permettono a un attaccante di decifrare un messaggio RSA intercettato senza conoscere la chiave privata, sfruttando un server SSL v3.0 come oracolo. L'attacco si configura come un processo iterativo e adattivo.

6.1 1. L'Obiettivo dell'Attaccante

L'attaccante ha intercettato un testo cifrato C e conosce la chiave pubblica del server (n, e).

$$C = M^e \pmod{n}$$

Il suo scopo è recuperare il messaggio in chiaro originale M (che in SSL è il *Pre-Master Secret*).

6.2 2. Sfruttamento della Malleabilità

L'attaccante non può decifrare C direttamente. Tuttavia, grazie alla proprietà di **malleabilità** di RSA, può manipolare il testo cifrato per creare varianti correlate.

1. L'attaccante sceglie un numero intero casuale r.
2. Costruisce un nuovo testo cifrato C' moltiplicando il crittogramma originale per la cifratura di r:
$$C' = C \cdot r^e \pmod{n} \tag{4}$$
3. Matematicamente, questo nuovo testo cifrato corrisponde alla cifratura del messaggio originale moltiplicato per r:
$$C' = (M^e \cdot r^e) \pmod{n} = (M \cdot r)^e \pmod{n}$$

In sostanza, inviando C', l'attaccante sta chiedendo al server di decifrare il valore $(M \cdot r)$.

6.3 3. Interrogazione dell'Oracolo (Il Server SSL v3.0)

L'attaccante invia C' al server. Il server decifra il messaggio ottenendo $X = (M \cdot r) \pmod{n}$ e verifica se il padding è corretto. Secondo lo standard PKCS #1 v1.5, un messaggio valido deve iniziare con i byte `00 02`. L'oracolo fornisce una risposta binaria (Sì/No):

- **Risposta Negativa (Errore/Abort):** Il server dice "Errore di padding". L'attaccante impara che $(M \cdot r) \pmod{n}$ **non** inizia con `00 02`.
- **Risposta Positiva (Continua):** Il server non dà errore (o impiega più tempo). L'attaccante impara che $(M \cdot r) \pmod{n}$ **inizia effettivamente** con `00 02`.

6.4 4. La Fuga di Informazioni (Information Leak)

Quando l'attaccante trova un r che produce una risposta positiva, ha ottenuto un'informazione preziosa. Sapere che $(M \cdot r) \pmod{n}$ inizia con `00 02` significa sapere che il risultato numerico cade in un intervallo specifico:

$$2 \cdot B \leq (M \cdot r \pmod{n}) < 3 \cdot B$$

(dove B è una potenza di 2 che dipende dalla lunghezza della chiave). Questa disuguaglianza riduce drasticamente lo spazio delle possibili soluzioni per M.

6.5 5. Attacco Adattivo e Risultato Finale

L'attacco è definito **Adaptive** perché ogni nuovo valore di r viene scelto in base ai risultati delle iterazioni precedenti, restringendo progressivamente l'intervallo possibile per M. **Efficienza:** Come indicato nella slide, per una chiave RSA a 1024 bit, sono necessari circa un milione di tentativi (2^{20} queries). Sebbene sembri un numero alto, per un computer automatizzato che invia richieste di rete, questo attacco può essere completato in tempi ragionevoli (minuti o ore), portando alla **decifrazione completa** del messaggio M.

7 Toy Example: Capire la Logica di Bleichenbacher

La slide "Toy Bleichenbacher example" propone un modello semplificato per comprendere l'intuizione dietro l'attacco, rimuovendo la complessità del padding PKCS #1 reale. L'idea è dimostrare come un oracolo che fornisce anche solo un'informazione binaria (Sì/No) su una proprietà del messaggio decifrato possa essere sfruttato per rivelare l'intero messaggio in pochissimi passaggi.

7.1 L'Oracolo Semplificato

Immaginiamo uno scenario ideale ("Toy Oracle") con queste caratteristiche:

- **Lunghezza del Messaggio:** Il messaggio è lungo $\log_2 n$ bit (ovvero, la lunghezza in bit del modulo RSA).
- **Comportamento dell'Oracolo:** Invece di controllare il padding complesso `00 02...`, questo oracolo controlla solo il **primo bit** (il bit più significativo, MSB).
  - Risponde **SÌ** (1) se il primo bit è 1.
  - Risponde **NO** (0) se il primo bit è 0.

7.2 La Strategia di Attacco: Ricerca Binaria

Con un tale oracolo, decifrare il messaggio diventa banale come indovinare un numero tra 0 e 100 facendosi dire solo "più alto" o "più basso". Sfruttando la malleabilità di RSA, l'attaccante può "spostare" i bit del messaggio a sinistra (moltiplicando per 2, ovvero inviando $C' = C \cdot 2^e \pmod n$) e interrogare l'oracolo su ogni singolo bit. Il processo è equivalente a una **Binary Search** (ricerca dicotomica):

1. Chiedo all'oracolo il valore del 1° bit.
2. Moltiplico per 2 (shift a sinistra) e chiedo il valore del 2° bit.
3. Ripeto per tutti i bit.

7.3 Efficienza Devastante

La slide evidenzia in basso la potenza di questo approccio:

> **Con un numero di tentativi pari all'ordine di $\log_2 n$, puoi decifrare l'INTERO messaggio!**

Se la chiave RSA è di 1024 bit, $\log_2 n = 1024$. Significa che bastano **soli 1024 tentativi** per decifrare completamente il segreto. In termini computazionali, è istantaneo.

7.4 Differenza con l'Attacco Reale

Il testo rosso al centro della slide fa il ponte con la realtà:

- Nell'attacco reale di Bleichenbacher (contro SSL v3.0), l'oracolo non ci dice il valore di un singolo bit.

- Ci dice se i primi **2 byte** (16 bit) sono uguali a `00 02`.
- Il concetto logico rimane lo stesso (restringere l'intervallo delle possibilità), ma la matematica è molto più elaborata ("much more elaborate math") perché non stiamo recuperando un bit alla volta in modo pulito, ma stiamo restringendo intervalli numerici multipli.
- Ecco perché l'attacco reale richiede circa un milione (2^{20}) di tentativi invece di mille (2^{10}), ma rimane comunque fattibile e pericoloso.

Analisi: Ricerca Binaria

Immagina di dover indovinare un numero segreto. Se l'oracolo ti dicesse solo "Sì/No" ma tu potessi raddoppiare il tuo numero segreto a piacimento, potresti spostare le cifre finché l'oracolo non ti rivela ogni singolo bit. Questo trasforma un problema esponenzialmente difficile (indovinare la chiave) in un problema linearmente facile (indovinare bit per bit).

8 Esempio Numerico: Toy Bleichenbacher Attack

La slide "Example (see math file)" traduce la teoria dell'attacco adattivo in un esempio pratico di calcolo. Qui vediamo come l'attaccante, sfruttando la malleabilità di RSA e un oracolo semplificato, possa ricostruire il messaggio segreto M bit dopo bit.

8.1 1. Setup dei Parametri RSA

L'esempio imposta un ambiente RSA con numeri molto piccoli (8 bit) per rendere i calcoli leggibili:

- **Generazione Chiavi:**
  - Primi: $p = 13, q = 19$.
  - Modulo: $n = p \cdot q = 247$. (Questo numero sta in 8 bit, poiché $2^8 = 256$).
  - Totiente: $\Phi(n) = (13 - 1)(19 - 1) = 12 \cdot 18 = 216$.
  - Chiave Pubblica: $e = 29$.
  - Chiave Privata: $d = e^{-1} \pmod{216} = 149$.

8.2 2. Lo Scenario dell'Attacco

- **Il Bersaglio:** L'attaccante intercetta il testo cifrato $C = 90$.
- **L'Obiettivo:** Trovare il messaggio M che ha generato C, senza conoscere $d = 149$.

8.3 3. Generazione delle Query (Chosen Ciphertext)

L'attaccante utilizza la formula mostrata nella slide (sintassi stile Mathematica):

```
Mod[c * PowerMod[2^Range[1, 8], e, n], n]
```

Analizziamo cosa significa matematicamente questa operazione. L'attaccante invia al server una serie di 8 testi cifrati manipolati $(C_1, C_2, \ldots, C_8)$, calcolati come:

$$C_i = C \cdot (2^i)^e \pmod{n} \quad \text{per } i = 1 \ldots 8 \tag{5}$$

Sfruttando la proprietà omomorfica (malleabilità) di RSA, decifrare questi testi equivale a:

$$M_i = \text{DEC}(C_i) = \text{DEC}(C \cdot (2^i)^e) = M \cdot 2^i \pmod{n}$$

In pratica, l'attaccante sta chiedendo al server di decifrare il messaggio originale moltiplicato per 2, poi per 4, poi per 8, ecc. In binario, moltiplicare per 2 equivale a uno **shift a sinistra** (spostare i bit verso la posizione più significativa).

8.4 4. La Risposta dell'Oracolo

Il server riceve i testi cifrati $\{20, 224, 187, \ldots\}$ e, agendo come "Toy Oracle", per ciascuno risponde dicendo se il messaggio decifrato ha il primo bit (MSB) a 0 o a 1.

$$\text{Server returns: } \{0, 1, 1, 1, 0, 1, 1, 1\}$$

8.5 5. Ricostruzione del Messaggio

Questa sequenza di risposte permette di ricostruire il messaggio. Poiché a ogni passaggio stiamo "spingendo" i bit di M verso sinistra (moltiplicando per 2), il bit che trabocca o che diventa il più significativo viene rivelato dall'oracolo. La sequenza binaria restituita $\{0, 1, 1, 1, 0, 1, 1, 1\}$ corrisponde, letta nel giusto ordine, alla rappresentazione binaria del messaggio M. Facciamo una verifica (Reverse Engineering dell'esempio):

- La sequenza binaria 01110111_2 corrisponde al numero decimale:

$$64 + 32 + 16 + 4 + 2 + 1 = 119$$

- Verifichiamo se $M = 119$ è corretto cifrandolo con la chiave pubblica:

$$119^{29} \pmod{247} = 90$$

- Esattamente il $C = 90$ che l'attaccante aveva visto all'inizio!

Conclusione: Con sole 8 richieste al server (tante quanti sono i bit del modulo), l'attaccante ha decifrato completamente il messaggio segreto.

9 Analisi dell'Esempio: La Riduzione degli Intervalli

La slide analizza passo dopo passo come l'attaccante elabora le risposte ricevute dall'oracolo nel "Toy Example" precedente. Il processo è una forma di **Ricerca Binaria** applicata su intervalli modulari.

9.1 La Logica della Restrizione

Inizialmente, il messaggio M potrebbe essere un qualsiasi numero compreso tra 0 e $N-1$ (dove $N = 247$). Ad ogni passaggio, l'attaccante moltiplica il messaggio per potenze di 2 ($2M, 4M, 8M \dots$) e chiede all'oracolo il valore del primo bit (MSB).

1. **Step 1 ($2M$ inizia con 0):** L'oracolo ci dice che il primo bit di ($2M \pmod N$) è 0. Questo significa che il risultato dell'operazione modulare è nella "metà inferiore" dei valori possibili (minore di 128 su 8 bit). Matematicamente, questo esclude molti valori di M. I candidati rimasti validi cadono in due intervalli disgiunti:

   $$M \in (0, 63) \cup (124, 187)$$

   Interpretazione: Se M fosse stato, ad esempio, 100, $2M = 200$, che ha il primo bit a 1 (poiché > 127). Quindi 100 viene scartato.

2. **Step 2 ($4M$ inizia con 1):** L'attaccante ora sa che M è negli intervalli sopra. Chiede dell'MSB di $4M$. L'oracolo risponde "1". L'attaccante interseca i nuovi vincoli con gli intervalli precedenti. Lo spazio si restringe ulteriormente:

   $$M \in (32, 61) \cup (156, 185)$$

3. **Step 3 ($8M$ inizia con 1):** Si continua il processo. Ogni risposta dimezza (circa) lo spazio di ricerca rimanente.

   $$M \in (47, 61) \cup (171, 185)$$

9.2 Conclusione della Ricerca

La slide nota che all'ottavo tentativo (8° attempt), ovvero interrogando su $256M$ (che equivale a completare il ciclo dei bit del modulo), l'ultimo "pareggio" (tie) viene risolto. Gli intervalli si restringono fino a contenere un singolo numero intero: il messaggio $M = 119$.

9.3 Complessità Lineare

Il punto di forza di questo attacco è l'efficienza:

> **Binary Search: linear with #bit!!**

Non stiamo provando tutte le chiavi (complessità esponenziale). Stiamo risolvendo bit per bit. Se la chiave è lunga k bit, servono circa k tentativi. Questo rende l'attacco fattibilissimo anche per chiavi enormi.

9.4 L'Insegnamento Generale ("Aftermath")

Il testo rosso finale ("Aftermath") generalizza il concetto, enunciando un principio fondamentale della crittanalisi:

- **Partial Information Leak:** Non è necessario che l'oracolo ci riveli tutto il messaggio.

- **From One to All:** "Finché si riesce a conoscere un singolo bit (che sia l'MSB, il bit di parità, o il bit di validità del padding PKCS#1), è possibile rivelare **tutto** il testo cifrato".

L'attacco di Bleichenbacher non è altro che un'applicazione complessa di questo principio: l'oracolo del padding ci dà un "bit di informazione" (valido/non valido), e l'attaccante lo usa per decifrare tutto.

10 Corollario: L'Attacco alla Parità (Parity Oracle)

La slide conclusiva, "Corollary: attacking parity", estende il ragionamento visto con Bleichenbacher per dimostrare quanto sia fragile la crittografia RSA non autenticata (o malleabile). L'idea centrale è che non serve un errore complesso di padding per violare il sistema: basta conoscere la **parità** del messaggio decifrato.

10.1 Lo Scenario Ipotetico: Il Negozio

Viene presentato un esempio pratico di un "Side Channel" (canale laterale) a livello applicativo:

- Immaginiamo un negozio online ("Shop") che riceve ordini cifrati con RSA.
- Per qualche logica di business (o un bug), il server **rifiuta** le richieste se il numero di oggetti ordinati è **pari**.
- L'attaccante invia un testo cifrato C, il server lo decifra in M, controlla se M è pari o dispari e risponde con un errore (se pari) o un OK (se dispari).

Questo comportamento trasforma il server in un **Parity Oracle**.

10.2 Equivalenza con la Lettura dei Bit

Matematicamente, sapere se un numero M è pari o dispari equivale a conoscere il suo **LSB (Least Significant Bit)**, ovvero il bit meno significativo (l'ultimo a destra):

- Se M è pari $\rightarrow$ LSB $= 0$.
- Se M è dispari $\rightarrow$ LSB $= 1$.

10.3 Dall'LSB all'Intero Messaggio

La slide ribadisce con forza il concetto chiave ("Same situation as before"):

"As long as just 1 bit gets revealed, ALL plaintext is at stake!"

Esistono algoritmi noti (come l'attacco al bit di parità di RSA) che permettono di decifrare l'intero messaggio sfruttando questo singolo bit di informazione. La logica è simile alla ricerca binaria vista prima, ma lavorando all'inverso: 1. L'attaccante usa la malleabilità di RSA per manipolare il testo cifrato (es. moltiplicando per l'inverso di 2 modulare, che equivale a uno shift a destra o dimezzamento). 2. Invia il nuovo testo

cifrato all'oracolo. 3. In base alla nuova parità, deduce il bit successivo. 4. Ripetendo il processo k volte (dove k sono i bit del modulo), recupera tutto il messaggio.

Conclusione del Corso: Questo esempio finale serve a cementare l'idea che la sicurezza semantica di RSA dipende dal non rivelare *alcuna* informazione sul testo in chiaro. Qualsiasi "perdita" (leak), anche di un solo bit (parità, padding corretto, MSB), porta alla compromissione totale.

11 La Persistenza di Bleichenbacher: Esempi nel Mondo Reale

La slide "Recent examples of Bleichenbacher's Oracle in the wild" dimostra come la vulnerabilità scoperta da Daniel Bleichenbacher nel 1998 non sia solo un reperto storico, ma una minaccia ricorrente che ha afflitto Internet anche decenni dopo. Nonostante le correzioni teoriche negli standard (TLS 1.0+), la complessità di implementare correttamente la gestione degli errori RSA ha portato a numerose "resurrezioni" dell'attacco. Vengono citati tre casi emblematici recenti:

11.1 1. L'Attacco DROWN (2016)

DROWN (*Decrypting RSA with Obsolete and Weakened eNcryption*) è un esempio critico di come il supporto a protocolli obsoleti possa compromettere quelli moderni.

- **Il Meccanismo:** L'attacco sfrutta i server che supportano ancora l'antico protocollo **SSLv2** (risalente agli anni '90) per compatibilità retroattiva ("legacy").
- **Il Paradosso:** Anche se un client si connette in modo sicuro via TLS 1.2, se il server condivide la stessa chiave privata RSA con una configurazione SSLv2 attiva, l'attaccante può usare l'oracolo di Bleichenbacher presente in SSLv2 per decifrare le connessioni TLS moderne.
- **L'Impatto:** La slide sottolinea con incredulità ("SSLv2? Are you kidding?") la vastità del problema: nel 2016, circa un terzo dei siti web mondiali (inclusi giganti del web) era vulnerabile perché non aveva disabilitato il vecchio SSLv2.

11.2 2. L'Attacco ROBOT (2018)

L'attacco ROBOT (*Return Of Bleichenbacher's Oracle Threat*) ha colpito nel 2018, esattamente 20 anni dopo la scoperta originale.

- **La Causa:** Non si trattava di un difetto nel protocollo TLS, ma di **"poor implementations"** (implementazioni scadenti) da parte dei venditori di apparati di rete (es. F5, Citrix, Cisco).
- **Il Problema:** Gli standard richiedevano di gestire gli errori di padding in modo che non trapelassero informazioni (tempi di risposta identici, nessun messaggio d'errore distinto). Tuttavia, molti sviluppatori hanno fallito in questo compito delicato, reintroducendo involontariamente l'oracolo laterale.

11.3 3. Vulnerabilità in IPsec IKE (2018)

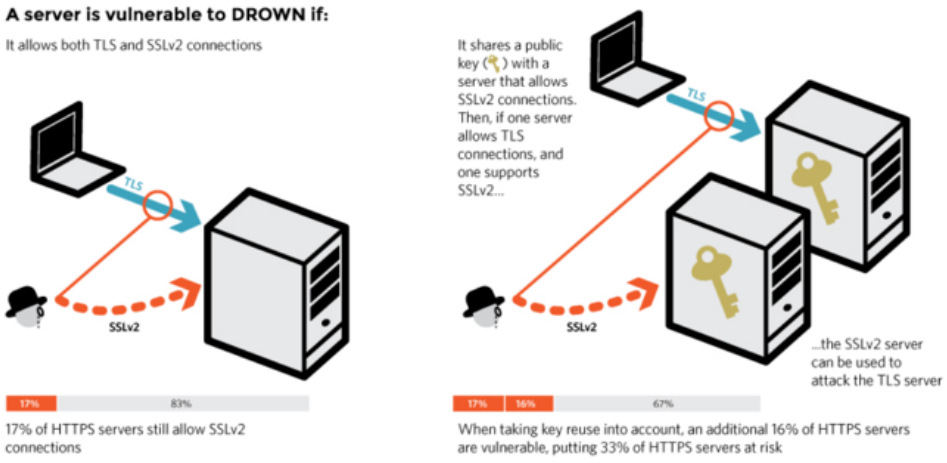
Sempre nel 2018, si è scoperto che l'attacco non riguardava solo il web (HTTPS/TLS), ma anche le VPN basate su IPsec.

- **Contesto:** Il protocollo IKE (Internet Key Exchange) serve a scambiare le chiavi per le VPN.
- **L'Attacco:** Combinando un attacco di **Downgrade** (forzando l'uso della vecchia versione IKEv1) con la logica di Bleichenbacher, è stato possibile recuperare le chiavi di autenticazione.
- **Vendor Colpiti:** La slide elenca importanti produttori di hardware di rete come Cisco, Huawei, Clavister e ZyXEL, citando le specifiche CVE (Common Vulnerabilities and Exposures) associate, a dimostrazione della diffusione trasversale del problema.

Conclusione: Questi esempi insegnano che in crittografia la teoria corretta non basta; l'implementazione pratica "senza leak" è estremamente difficile da mantenere nel tempo.

12 L'Attacco DROWN (2016)

La slide illustra i dettagli operativi e l'impatto dell'attacco **DROWN** (*Decrypting RSA with Obsolete and Weakened eNcryption*), scoperto nel 2016. Questo attacco rappresenta un classico esempio di **Cross-Protocol Attack**: una vulnerabilità in un protocollo obsoleto (SSLv2) viene sfruttata per compromettere connessioni effettuate su protocolli moderni e sicuri (TLS).



DROWN dimostra che la sicurezza di una connessione non dipende solo dal protocollo che si sta usando in quel momento, ma anche dalla configurazione del server e dalla gestione delle chiavi.

12.1 Meccanismo dell'Attacco

DROWN sfrutta una variante dell'attacco di Bleichenbacher (Padding Oracle) contro l'implementazione di RSA nel vecchio protocollo SSLv2. Un attaccante può catturare una

sessione TLS cifrata (registrando il traffico passivamente) e successivamente utilizzare il server SSLv2 come oracolo per decifrare il *Pre-Master Secret* della sessione TLS catturata. La slide evidenzia due scenari di vulnerabilità:

12.1.1 Scenario 1: Vulnerabilità Diretta (Stesso Server)

Come mostrato nella parte sinistra dell'immagine:

- Un server è configurato per accettare connessioni sia moderne (**TLS**) che obsolete (**SSLv2**).
- Un client legittimo si connette in modo sicuro via TLS.
- L'attaccante, avendo registrato il traffico TLS, invia pacchetti malformati appositamente costruiti alla porta che gestisce SSLv2 sullo stesso server.
- Sfruttando le risposte del server SSLv2 (l'oracolo), l'attaccante decifra la chiave di sessione TLS.
- *Statistica:* Nel 2016, il 17% dei server HTTPS permetteva ancora connessioni SSLv2, esponendosi direttamente.

12.1.2 Scenario 2: Riutilizzo delle Chiavi (Key Reuse)

La parte destra dell'immagine mostra un rischio ancora più sottile e diffuso: il riutilizzo delle credenziali crittografiche.

- Spesso le organizzazioni acquistano un certificato RSA e lo installano su più server (es. il server Web principale e il server di Posta).
- Supponiamo che il Server A (Web) supporti solo TLS (sicuro), ma condivida la stessa coppia di chiavi (chiave pubblica/privata) con il Server B (Posta) che supporta ancora SSLv2.
- L'attaccante registra il traffico verso il Server A (sicuro).
- Successivamente, attacca il Server B (insicuro) usando la suite SSLv2. Poiché la chiave privata RSA è la stessa, l'oracolo su B permette di decifrare il traffico di A.
- *Statistica:* Considerando il riutilizzo delle chiavi, un ulteriore 16% di server risultava vulnerabile.

12.2 Impatto Globale

Sommando i due scenari, la slide conclude con un dato allarmante:

> **Il 33% di tutti i server HTTPS mondiali era a rischio.**

Questo ha costretto gli amministratori di sistema di tutto il mondo a disabilitare definitivamente il supporto a SSLv2 e a rivedere le politiche di gestione delle chiavi.

13 Contromisure e Conclusioni (Take-home Messages)

Le slide finali discutono come l'industria ha cercato di difendersi dall'attacco di Bleichenbacher nel corso degli anni e quali lezioni fondamentali possiamo trarre da questa storia ventennale.

13.1 Le Contromisure Tecniche

La slide "Countermeasures" elenca tre approcci principali per mitigare la vulnerabilità del Padding Oracle su RSA PKCS#1 v1.5:

13.1.1 1. Cambiare lo Schema di Padding (La soluzione teorica)

La soluzione crittograficamente più corretta sarebbe abbandonare il vecchio standard PKCS#1 v1.5 in favore di schemi più moderni e sicuri, come **RSA-OAEP** (Optimal Asymmetric Encryption Padding).

- **Pro:** OAEP è dimostrabilmente sicuro contro gli attacchi a testo cifrato scelto (IND-CCA2).
- **Contro:** Richiede di cambiare il formato dei dati scambiati. Per protocolli diffusi globalmente come TLS, questo rappresentava un cambiamento troppo radicale ("Way too major change!!"), rompendo la compatibilità con vecchi client e server. Pertanto, questa strada è stata spesso declinata per motivi di retro-compatibilità.

13.1.2 2. Implementazione "Attenta" (Il Workaround)

Poiché non si poteva cambiare il protocollo, si è cercato di rendere l'implementazione resistente all'attacco. L'idea è eliminare l'oracolo mascherando l'errore:

> *"Se il padding è errato, non dare errore. Genera una chiave casuale e procedi come se nulla fosse."*

In questo modo, l'attaccante non riceve un messaggio di "Abort" immediato e non dovrebbe poter distinguere tra un padding valido e uno invalido. **Il problema:** Questa strategia è incredibilmente difficile da implementare correttamente ("Not nearly easy!!").

- Anche se non si inviano messaggi di errore, le differenze nei tempi di esecuzione (Timing Attacks) possono tradire il server.
- Ad esempio, generare 48 byte casuali richiede un tempo diverso rispetto a estrarli dal pacchetto decifrato.
- Come dimostrato dal paper citato (2003), molte implementazioni hanno fallito, portando a vulnerabilità latenti (come ROBOT).

13.1.3 3. Abbandonare RSA Key Transport (La Soluzione Finale)

L'unica soluzione definitiva è rimuovere la funzionalità vulnerabile. **TLS 1.3** ha rimosso completamente il supporto per il trasporto chiavi tramite RSA. Oggi si usa RSA solo per le firme, mentre lo scambio chiavi avviene via Diffie-Hellman (DHE/ECDHE), che offre anche la Forward Secrecy.

Sintesi: Legacy e Sicurezza

Il caso DROWN/ROBOT ci insegna che in sicurezza "compatibilità retroattiva" (supportare vecchi standard per non escludere vecchi client) è spesso nemica della sicurezza. Finché un server supporta un protocollo bacato (anche se non lo usa di default), la sua chiave privata è a rischio.

13.2 Lezioni Apprese (Take-home messages)

La slide conclusiva sintetizza ciò che l'incidente Bleichenbacher insegna agli ingegneri della sicurezza:

1. **L'Implementazione è Trappola:** È molto facile commettere errori sottili nell'implementazione di primitive crittografiche, ed è difficilissimo correggerli ("patch") in modo incrementale senza rifare tutto da capo.
2. **La Fallacia della "Military Grade Encryption":** La vignetta finale ironizza su un malinteso comune.
   - **Ricercatore:** "Il tuo server è vulnerabile all'attacco di Bleichenbacher (un difetto logico nel protocollo)".
   - **Amministratore:** "Nessun problema, usiamo crittografia di grado militare (algoritmi forti come AES-256 o RSA-4096)".

   Morale: Avere algoritmi forti è inutile se il protocollo che li utilizza perde informazioni (Side Channels). Un algoritmo robusto (RSA) usato male (PKCS#1 v1.5 con gestione errori verbosa) rende il sistema totalmente insicuro.