



Timing Attack ad RSA

di

Ettore Mirto

matricola: 53/10265
e-mail: ettore.mirto@st.com

Tesina presentata all'Università di Salerno

Laurea in Scienze dell'Informazione

Esame di **“Sicurezza su Reti”**

Prof. Alfredo De Santis

Salerno, Marzo 2006



Index

Index	2
1. Timing Analysis.....	3
1.1 Introduzione	3
1.2 RSA.....	4
2. Timing Attack	6
2.1. L'idea	6
2.2. Dettagli dell'attacco	8
2.2.1. RSAREF 2.0	8
2.2.2. Analisi	12
3. Contromisure.....	18
4. Conclusioni	21
5. Riferimenti	23
6. Appendice A	25
7. Appendice B.....	26
8. Appendice C.....	27



1. Timing Analysis

1.1 Introduzione

Gli algoritmi crittografici, spesso, eseguono dei calcoli in tempi non costanti, a causa di ottimizzazioni sulle prestazioni. Nel caso in cui tali operazioni manipolano dei parametri segreti, queste variazioni sui tempi possono divulgare alcune informazioni, ed un'attenta analisi statistica conduce al loro totale recupero.

Lo schema di figura 1 rappresenta il principio che sta alla base del timing attack, nel quale l'implementazione che usa i dati segreti potrebbe essere un protocollo, una smart card, etc., e l'attaccante misura il tempo impiegato da tale implementazione per rispondere ad una sua richiesta.

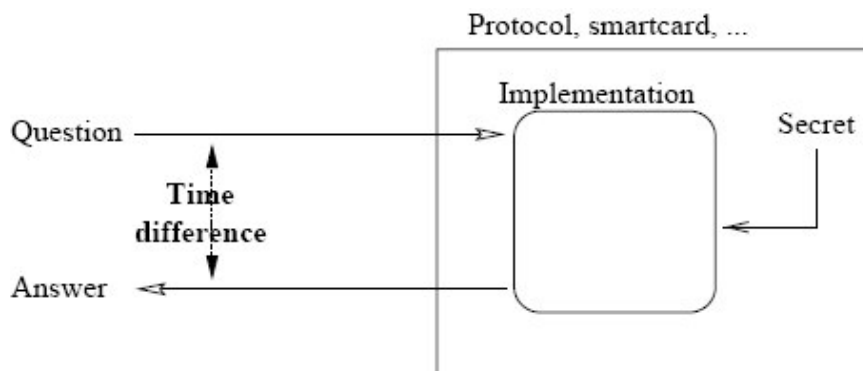


Fig.1 Il principio del timing attack

Nel Novembre del 1995, Paul Kocher [3] è stato il primo a discutere la tecnica del timing attack; infatti Kocher presentò il suo risultato preliminare, attirando l'attenzione dei crittografi di tutto il mondo, inclusi gli inventori del crittosistema RSA. I timing attack sono una forma di "**Side Channel Analysis**", o "**analisi del canale collaterale**" dove un attaccante prende informazioni dall'implementazione di un crittosistema, piuttosto che da debolezze presenti nelle proprietà matematiche del sistema. "**Side Channel Analysis**" espone informazioni sui tempi, consumo di alimentazione, emissioni elettromagnetiche per recuperare delle informazioni su un crittosistema [8].



Il timing attack analizza le variazioni dei tempi durante le operazioni crittografiche, poichè le ottimizzazioni sulle prestazioni agiscono sulle operazioni eseguite da un algoritmo crittografico e spesso sono eseguite con un differente ammontare di tempo, dipendente dall'input e dal valore dei parametri segreti. Se le operazioni riguardanti la chiave privata di RSA possono essere determinate in un tempo accuratamente ragionevole, in alcuni casi l'analisi statistica può essere applicata per recuperare la chiave segreta usata nei calcoli.

I prossimi paragrafi descriveranno l'algoritmo crittografico, nonché l'algoritmo di firme RSA [par.1.2] su cui si concentrerà la discussione, e la tecnica della timing analysis. In particolare sarà illustrata l'idea che sta alla base del timing attack [2.1], per poi essere approfondita, applicandola ad una specifica libreria RSAREF 2.0 [par.2.2.1], rilasciata dalla RSA Data Security per il calcolo della firma RSA. Un'analisi più dettagliata, basata su calcoli probabilistici [par.2.2.2], spiegherà come determinare i bit segreti, e la probabilità di successo ottenuta sui bit segreti che sono stati trovati. Il Capitolo 3 e 4 forniscono le conclusioni ed in particolare informazioni riguardo alcune tecniche di contromisura.

1.2 RSA

RSA è un algoritmo crittografico a chiave pubblica, oggi ampiamente usato, per trasferire informazioni elettroniche in modo sicuro. L'algoritmo RSA è stato inventato dal team di Rivest, Shamir e Adleman nel 1978.

RSA è un cifrario a chiave pubblica che permette di cifrare un messaggio attraverso un procedimento che sfrutta le proprietà dei numeri primi. RSA usa un esponente pubblico e per cifrare ed un esponente privato d per decifrare. Esso usa un modulo N che è il prodotto di due grandi numeri primi p e q , ovvero $N = p * q$. Gli esponenti e e d devono essere scelti in modo da soddisfare la condizione:

$$ed=1 \text{ mod}(p-1)(q-1)$$



Allora, la coppia di chiavi RSA ha come chiave pubblica la coppia (N, e) e come chiave privata la coppia (N, d) .

Ad esempio, se selezioniamo due numeri primi $p=11$ e $q=3$, allora $N = 11 * 3 = 33$. Ora calcoliamo $(p-1) * (q-1) = 10 * 2 = 20$ e scegliamo un valore e primo su 20, diciamo 3. Poi d è scelto in modo che $ed = 1 \pmod{20}$. Allora il valore di d è 7, poiché $3 * 7 = 21 = 1 \pmod{20}$ (**Appendice A**).

Così prendiamo la chiave pubblica $(N = 33, e = 3)$ e la corrispondente chiave privata $d = 7$. Scartiamo il fattore originale p, q .

Per cifrare il messaggio M , calcoliamo $C = M^e \pmod{N}$, dove C sarà il messaggio cifrato. Per decifrare il testo cifrato C , usiamo la formula $M = C^d \pmod{N}$, che conterrà il messaggio originale M .

Continuando con il nostro esempio, supponiamo di voler spedire un messaggio $M = 19$. La codifica genera un messaggio codificato $C = M^e \pmod{N} = 19^3 \pmod{33} = 28$. Il mittente spedirà il messaggio cifrato $C = 28$. Per decifrare il messaggio C , il destinatario usa la chiave privata d e calcola $M = C^d \pmod{N} = 28^7 \pmod{33} = 19$, che corrisponde al messaggio originale.

Un altro uso dell'algorithmo RSA è la generazione di una firma digitale il cui scopo è quello di verificare la sorgente del messaggio. Durante la firma si usa la chiave privata d e si esegue l'operazione matematica $F = M^d \pmod{N}$ per creare la firma. Il mittente invia al destinatario messaggio e firma, così il ricevente, per verificare la firma, ovvero che quel messaggio è stato spedito effettivamente dal mittente, usa la chiave pubblica e del mittente ed esegue l'operazione $M^e \pmod{N}$, la cui firma va confrontata con quella ricevuta per verificare la validità del messaggio.



2. Timing Attack

2.1. L'idea

Un'operazione fondamentale usata nel crittosistema RSA è l'esponentiazione modulare. Essa è usata sia per cifrare sia per decifrare oppure per firmare blocchi di messaggio. Quando RSA è stata introdotta, gli inventori hanno suggerito un algoritmo basato su operazioni ripetitive d'elevazione al quadrato e di moltiplicazione (fig 2).

INPUT: $M, N, d = (d_{n-1}d_{n-2} \dots d_1d_0)_2$

OUTPUT: $S = M^d \bmod N$

1 $S \leftarrow 1$

2 **for** $j = n - 1 \dots 0$ **do**

3 $S \leftarrow S^2 \bmod N$

4 **if** $d_j = 1$ **then**

5 $S \leftarrow S \cdot M \bmod N$

6 **return** S

Figura 2: algoritmo left-to-right

La figura 2 descrive l'algoritmo left-to-right per il calcolo dell'esponentiazione modulare. In input abbiamo il messaggio M , il modulo RSA, ovvero N , e l'esponente privato d , espresso in forma binaria (pedice 2). L'esponente privato d è rappresentato usando al più n bit, dove n è la lunghezza dei bit del modulo RSA, ovvero N , che abbiamo usato nel paragrafo precedente, mentre l'output S rappresenta la firma digitale del messaggio M . Inoltre dobbiamo considerare, che questo algoritmo è usato ipotizzando che il messaggio M è minore del modulo N .



Kocher ha fatto alcune importanti osservazioni sull'algoritmo rappresentato in figura 2, infatti la condizione espressa alla linea 4 causa l'esecuzione del percorso di quest'algoritmo in base al valore dell'esponente. In ogni ciclo di iterazione, se il bit rilevante di d è 1, allora sono eseguiti entrambi il modulo del quadrato e della moltiplicazione (linea 3 e 5 rispettivamente); se il bit rilevante è 0 è eseguito solo il modulo del quadrato (linea 3). In questo modo, l'ammontare di calcolo richiesto, e quindi il tempo di esecuzione, per completare gli n cicli iterativi è influenzato dal valore dell'esponente.

Se un attaccante osserva e confronta il tempo di esecuzione di diversi cicli iterativi nell'algoritmo left-to-right, è in grado di dedurre il valore dei corrispondenti bit dell'esponente. Questa tecnica, quando è applicata contro un'operazione di firme RSA, rileverà i bit della chiave segreta del firmatario.

Il timing attack di Kocher descrive come un attaccante può usare il tempo totale di esecuzione dell'algoritmo per dedurre i bit dell'esponente privato. Questa informazione sul tempo può essere facilmente osservata da un attaccante passivo.

Supponiamo che un utente malintenzionato, Oscar, spedisce una serie di richieste di firme ad un PC che implementa l'algoritmo RSA, usando l'algoritmo ripetuto left-to-right.

Oscar memorizza i tempi T_1, T_2, \dots, T_k che il PC impiega al ritorno di una firma su ognuno dei messaggi conosciuti $M_1, M_2, \dots, M_k \in Z_N$. L'attacco, poi, procede in modo che Oscar recupera i bit di d , uno alla volta.

Poiché $d < N$ ed n è la lunghezza dei bit di N , la rappresentazione binaria di d può contenere degli zero precedenti, ma per semplificare la nostra discussione assumeremo che $d_{n-1} = 1$.

Passando attraverso il pseudocodice della figura 2, Oscar conosce che all'inizio del 2° ciclo iterativo $S = M \bmod N$ e poi, dopo il passo del quadrato, $S = M^2 \bmod N$. Se $d_{n-2} = 1$, il PC calcola il prodotto $M * M^2 \bmod N$, altrimenti non lo esegue.



Usando la conoscenza delle caratteristiche fisiche del PC destinatario, Oscar simula su un PC identico (ad esempio con stesso processore, stessa RAM cache, etc.) il tempo \hat{t}_i preso per calcolare $M_i^2 * M_i \bmod N$ per ognuno dei messaggi conosciuti. Il valore di M_i influenza l'ammontare di tempo richiesto per eseguire questo calcolo.

Kocher nota che, quando $d_{n-2} = 1$, i 2 insiemi $\{\hat{t}_i\}$ e $\{T_i\}$ sono correlati. Per esempio, se \hat{t}_i è molto più grande di quello atteso, allora anche T_i deve essere più grande di quello atteso. Se $d_{n-2} = 0$, allora i 2 insiemi si comportano come variabili casuali indipendenti. Misurando la correlazione Oscar può decidere il valore di d_{n-2} . A questo punto Oscar conosce il valore di S all'inizio del 3° ciclo iterativo.

Per prendere d_{n-3} Oscar ricostruisce l'insieme $\{\hat{t}_i\}$, simulando il tempo preso dal PC per calcolare $S * M \bmod N$, dove il valore di S è conosciuto, e compare con l'insieme $\{T_i\}$. Oscar continua allo stesso modo per recuperare i rimanenti bit di d .

2.2. Dettagli dell'attacco

Analizziamo il metodo d'attacco di Kocher e spieghiamo come può essere applicato contro un'esponenziazione modulare nella libreria crittografica RSAREF.

2.2.1. RSAREF 2.0

L' RSAREF 2.0 è stata rilasciata dalla RSA Laboratories nel 1994. Questa libreria è stata un'implementazione di riferimento per diversi schemi comuni crittografici. Nella libreria RSAREF 2.0 sono inclusi gli algoritmi per l'accordo di chiavi Diffie-Hellman e la firma RSA. In entrambi i casi, l'esponenziazione modulare è trattata attraverso la funzione NN-ModExp. Il pseudo-codice della funzione NN-modExp è dato dalla figura 3.



INPUT: $M, N, d = (d_{n-1}d_{n-2} \dots d_1d_0)_2$

OUTPUT: $S = M^d \bmod N$

1 $m_1 \leftarrow M \bmod N$

2 $m_2 \leftarrow m_1 * M \bmod N$

3 $m_3 \leftarrow m_2 * M \bmod N$

4 $S \leftarrow 1$

5 **for** $j = n - 1 \dots 0$ **by 2 do**

6 $S \leftarrow S^2 \bmod N$

7 $S \leftarrow S^2 \bmod N$

8 **if** $(d_j d_{j-1})_2 \neq 0$ **then**

9 $S \leftarrow S * m_{(d_j d_{j-1})_2} \bmod N$

10 **return** S

Figura 3: Metodo left-to-right per il calcolo dell'esponentiazione modulare nel quale è stata usata una finestra di 2 bit.

Un'idea sbagliata molto comune sul timing attack è che esso determina solo se è eseguita o no una condizione moltiplicativa. Se fosse stato così, allora l'attacco non avrebbe successo contro l'algoritmo di figura 3. Conoscendo che una moltiplicazione è eseguita in un particolare ciclo iterativo, eliminerebbe soltanto uno dei quattro possibili valori per la coppia rilevante dei bit dell'esponente.

Per determinare il valore di una coppia dei bit dell'esponente è necessario conoscere quali operandi sono usati nella condizione moltiplicativa. Il timing attack è capace di esplodere la variazione dei tempi nelle moltiplicazioni e nel quadrato.



Supponiamo che Alice e Oscar comunicano con un protocollo di firme usando i propri PC. Quando Oscar spedisce un messaggio ad Alice, Alice usa le routine di RSAREF e la sua coppia di chiavi private (N,d) per firmare. Alice allora spedisce la sua firma a Oscar. Oscar, intanto, memorizza il tempo T_i che Alice impiega per rispondere, dopo che egli ha spedito ad Alice il messaggio M_i .

Ci sono diversi fattori che contribuiscono al valore di T_i , infatti ritornando alla figura 3, il tempo richiesto per eseguire le istruzioni da 1 a 4, danno una contribuzione che denotiamo con c_i . Inoltre nel ciclo della figura 3, per un particolare valore di j , il tempo richiesto per eseguire le linee 6, 7 e 9 anch'esse contribuiscono a T_i . Denotiamo queste ultime condizioni con $r_{i,j}, s_{i,j}, t_{i,j}$ rispettivamente.

Notiamo che $r_{i,j}$ e $s_{i,j}$ sono valori strettamente positivi, ma $t_{i,j}$ potrebbe valere zero. Altri fattori, come errori di misurazione e distanza di trasmissione, contribuiscono anche a T_i e possono essere trattati come sorgenti di errore. Denotiamo queste contribuzioni con e_i . A questo punto, possiamo scrivere:

$$T_i = e_i + c_i + (r_{i,n-1} + s_{i,n-1} + t_{i,n-1}) + (r_{i,n-3} + s_{i,n-3} + t_{i,n-3}) + \dots + (r_{i,1} + s_{i,1} + t_{i,1}) = e_i + c_i + \sum_j (r_{i,j} + s_{i,j} + t_{i,j})$$

I bit dell'esponente segreto di Alice influenza il valore di quasi tutti i componenti in questa somma. Per un particolare valore di j , gli operandi usati nelle due operazioni quadratiche sono completamente determinate dal valore dei bit dell'esponente $d_{n-1}, d_{n-2}, \dots, d_{j+2}, d_{j+1}$. Gli operandi usati nei passi della moltiplicazione sono affetti da questi stessi bit così come i bit d_j, d_{j-1} . Così, gli esponenti $r_{i,j}, s_{i,j}, t_{i,j}$ sono tutti influenzati dai bit dell'esponente. Il valore di c_i è influenzato soltanto dal valore di M_i .

Consideriamo il primo ciclo iterativo di NN-ModExp. Usando un PC identico ad Alice, Oscar può simulare e temporizzare i quattro possibili insiemi di calcoli che Alice esegue nel primo ciclo iterativo quando firma il messaggio M . Effettivamente, Oscar genera quattro candidati per il valore di



$c_i + r_{i,n-1} + s_{i,n-1} + t_{i,n-1}$. Per costruire ogni candidato, Oscar può semplicemente firmare il messaggio M_i quattro volte usando gli esponenti 0, 1, 2, 3 equivalenti a 00, 01, 10 e 11, espressi in forma binaria. Denotiamo con $\hat{T}_{i,n-1,0}, \hat{T}_{i,n-1,1}, \hat{T}_{i,n-1,2}, \hat{T}_{i,n-1,3}$ il tempo richiesto per queste quattro firme, dove i primi due indici indicano il relativo messaggio e il ciclo iterativo, mentre l'ultimo indice rappresenta un'ipotesi per i bit $d_{n-1}d_{n-2}$. Oscar può, quindi, costruire la seguente tabella:

00	01	10	11
$T_1 - \hat{T}_{1,n-1,0}$	$T_1 - \hat{T}_{1,n-1,1}$	$T_1 - \hat{T}_{1,n-1,2}$	$T_1 - \hat{T}_{1,n-1,3}$
$T_2 - \hat{T}_{2,n-1,0}$	$T_2 - \hat{T}_{2,n-1,1}$	$T_2 - \hat{T}_{2,n-1,2}$	$T_2 - \hat{T}_{2,n-1,3}$
$T_3 - \hat{T}_{3,n-1,0}$	$T_3 - \hat{T}_{3,n-1,1}$	$T_3 - \hat{T}_{3,n-1,2}$	$T_3 - \hat{T}_{3,n-1,3}$
\vdots	\vdots	\vdots	\vdots

In una delle quattro colonne, Oscar ha simulato le operazioni che saranno le stesse di Alice, eseguite alla fine del primo ciclo iterativo. In una di queste colonne, il valore candidato di Oscar sarà più vicino a $c_i + r_{i,n-1} + s_{i,n-1} + t_{i,n-1}$ degli altri tre candidati. Dall'analisi nella seguente sezione, vediamo che, con un'alta probabilità, questa produrrà che la varianza¹ della colonna corretta sia più piccola delle altre. Confrontando le quattro varianze ottenute, Oscar può determinare il valore di $d_{n-1}d_{n-2}$.

La prossima coppia di bit dell'esponente, $d_{n-3}d_{n-4}$, possono essere dedotti calcolando i tempi dei quattro possibili insiemi di calcoli che Alice ha eseguito prima della fine del secondo ciclo iterativo. Per ogni messaggio, Oscar può misurare il tempo preso per firmare M_i usando i quattro esponenti

¹ Questa statistica è usualmente denotata con S^2 . Se Y_1, Y_2, \dots, Y_k è un insieme di osservazioni e \bar{Y} è la loro media aritmetica, allora $S^2 = \frac{1}{k-1} \sum_{i=1}^k (Y_i - \bar{Y})^2$



$d_{n-1}d_{n-2}00$, $d_{n-1}d_{n-2}01$, $d_{n-1}d_{n-2}10$ e $d_{n-1}d_{n-2}11$. Denotiamo il tempo richiesto per queste quattro firme con $\hat{T}_{i,n-3,0}, \hat{T}_{i,n-3,1}, \hat{T}_{i,n-3,2}, \hat{T}_{i,n-3,3}$.

Oscar allora può ricostruire la sua tabella in cui le righe hanno la forma:

$T_i - \hat{T}_{i,n-3,0}$	$T_i - \hat{T}_{i,n-3,1}$	$T_i - \hat{T}_{i,n-3,2}$	$T_i - \hat{T}_{i,n-3,3}$
---------------------------	---------------------------	---------------------------	---------------------------

Inoltre, Oscar calcola la varianza di ogni colonna per determinare i valori attuali dei bit. Il valore delle altre coppie di bit può essere deciso a turno, usando una tabella simile.

2.2.2. Analisi

Sia j_0 un particolare valore di j nell'algoritmo left-to-right di figura 3, e sia $g \in \{0,1,2,3\}$. Oscar procede con il timing attack riempiendo le colonne della tabella con valori della forma $T_i - \hat{T}_{i,j_0,g}$, dove g è l'ipotesi per il valore dei bit $d_{j_0}d_{j_0-1}$ dell'esponente. Assumendo che Oscar ha determinato correttamente il valore dei bit $d_{n-1}d_{n-2} \dots d_{j_0+2}d_{j_0+1}$, abbiamo:

$$\hat{T}_{i,j_0,g} = c_i + \sum_{j>j_0} (r_{i,j} + s_{i,j} + t_{i,j}) + (r_{i,j_0} + s_{i,j_0} + \hat{t}_{i,j_0,g})$$

dove $\hat{t}_{i,j_0,g}$ è un valore candidato per t_{i,j_0} . Se $g = 0$ allora $\hat{t}_{i,j_0,g} = 0$, altrimenti $\hat{t}_{i,j_0,g} > 0$. Quindi, noi abbiamo:

$$\begin{aligned} T_i - \hat{T}_{i,j_0,g} &= e_i + c_i + \sum_j (r_{i,j} + s_{i,j} + t_{i,j}) - c_i - \sum_{j>j_0} (r_{i,j} + s_{i,j} + t_{i,j}) - (r_{i,j_0} + s_{i,j_0} + \hat{t}_{i,j_0,g}) = \\ &= e_i + \sum_{j<j_0} (r_{i,j} + s_{i,j} + t_{i,j}) + (t_{i,j_0} - \hat{t}_{i,j_0,g}) \end{aligned}$$

Allora si può verificare che il valore $\hat{t}_{i,j_0,g}$ è una misura corretta del tempo presa da Alice per calcolare la moltiplicazione alla linea 9 della figura 3 quando $j = j_0$, oppure non lo è. Se è corretto, allora $\hat{t}_{i,j_0,g}$ è uguale a t_{i,j_0} , e allora:



$$T_i - \hat{T}_{i,j_0,g} = e_i + \sum_{j < j_0} (r_{i,j} + s_{i,j} + t_{i,j})$$

Se non è corretto, allora $\hat{t}_{i,j_0,g}$ solitamente non è uguale a t_{i,j_0} , così non ci saranno cancellazioni. Oscar può usare la statistica per determinare se questa cancellazione avviene o no e quindi verificare l'ipotesi su g .

La sottrazione del termine $\hat{t}_{i,j_0,g}$ influenza la varianza di una colonna dei dati. Per vedere ciò, trattiamo le misure dei tempi come occorrenze di variabili casuali. La variabile casuale T descrive quanto tempo è necessario per firmare un messaggio in Z_N , usando d , l'esponente privato di Alice. La variabile casuale $\hat{T}_{j_0,g}$ descrive quanto tempo è necessario per esponenziare un messaggio usando gli $n - j_0$ bit più significativi di d , nel quale sono state aggiunte le ipotesi di due bit (dipendenti dal valore di g).

La variabile casuale r ed s descrive quanto tempo serve per eseguire il quadrato di un elemento di Z_N . La variabile casuale t descrive quanto tempo è necessario per eseguire la moltiplicazione di due elementi di Z_N (notiamo che t è strettamente positivo). Infine, la variabile casuale e descrive gli effetti di errore.

Assumendo che il tempo per il calcolo del quadrato e della moltiplicazione nei cicli successivi sono indipendenti da ognuno e dall'errore, la varianza della variabile casuale $T - \hat{T}_{j_0,g}$ quando l'ipotesi di g è corretta, vale:

$$\begin{aligned} \text{Var}(T - \hat{T}_{j_0,g}) &= \text{Var}\left(e + \sum_{j < j_0} (r + s) + \sum_{\substack{j < j_0 \\ d_j d_{j-1} \neq 00}} t\right) \\ &= \text{Var}(e) + \frac{j_0 - 2}{2} \text{Var}(r) + \frac{j_0 - 2}{2} \text{Var}(s) + l * \text{Var}(t) \end{aligned}$$

La variabile l è un intero che è determinato dal numero di coppie di bit prese da d che non sono uguali a 00 (per un valore casuale di d , l è approssimativamente uguale a $\frac{3(j_0 - 2)}{4}$). Ricordiamo che la variabili casuali r ed s descrivono il tempo necessario per fare un'operazione di calcolo del quadrato.



Così, r ed s sono identicamente distribuite e la varianza di $T - \hat{T}_{j_0, g}$ può in futuro essere semplificata a:

$$Var(T - \hat{T}_{j_0, g}) = Var(e) + (j_0 - 2)Var(s) + l * Var(t).$$

Quando il valore di g ipotizzato è scorretto, allora ci sono due possibilità per la varianza di $T - \hat{T}_{j_0, g}$, dipendente dal valore di g . Ricordiamo che, nel caso g sia scorretto, si ha::

$$T_i - \hat{T}_{i, j_0, g} = e_i + \sum_{j < j_0} (r_{i, j} + s_{i, j} + t_{i, j}) + (t_{i, j_0} - \hat{t}_{i, j_0, g}).$$

Innanzitutto, supponiamo che entrambi t_{i, j_0} e $\hat{t}_{i, j_0, g}$ sono diversi da zero. Allora, il valore $t_{i, j_0} - \hat{t}_{i, j_0, g}$ è la differenza di due occorrenze (solitamente non uguali) della variabile casuale t . Poiché la varianza della variabile casuale $t - t$ è data da $Var(t) + Var(-t) = 2 * Var(t)$, abbiamo nelle colonne della relativa tabella:

$$Var(T - \hat{T}_{j_0, g}) = Var(e) + (j_0 - 2)Var(s) + (l + 2)Var(t)$$

Poi, supponiamo che una tra t_{i, j_0} o $\hat{t}_{i, j_0, g}$ è zero, allora, per ogni colonna di dati con questa proprietà, si ha:

$$Var(T - \hat{T}_{j_0, g}) = Var(e) + (j_0 - 2)Var(s) + (l + 1)Var(t).$$

In questo modo, la colonna dei dati basata su una corretta ipotesi ha una varianza che è $Var(t)$ oppure $2 * Var(t)$ più bassa delle altre colonne di dati. La varianza campione, S^2 , è una buona stima della varianza effettiva e noi potremmo presentare una stima euristica della probabilità che questa statistica distinguerà la corretta colonna.

Per sviluppare la nostra stima, prima introduciamo alcune note e consideriamo due argomenti che sono stati stabiliti in molti testi introduttivi della probabilità [10]. Scriviamo $X \sim N(\mu, \sigma^2)$ per indicare che la variabile casuale X è normalmente distribuita con la media μ e la varianza σ^2 . La media di una variabile casuale X è anche denotata con $E(X)$. Se Y è una variabile casuale espressa come $Y = aX + b$, dove a e b sono costanti, e $X \sim N(\mu, \sigma^2)$, allora



$Y \sim N(a\mu + b, a^2\sigma^2)$. Se $X \sim N(\mu_x, \sigma_x^2)$ e $Y \sim N(\mu_y, \sigma_y^2)$, dove X e Y sono indipendenti, allora $X + Y \sim N(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$.

La colonna dei dati nella tabella di Oscar, che corrisponde ad una corretta ipotesi, ha una varianza attesa di $Var(e) + (j_0 - 2)Var(s) + l * Var(t)$. C'è una seconda colonna nella tabella di Oscar che ha una varianza attesa di $Var(e) + (j_0 - 2)Var(s) + (l + 1)Var(t)$. Queste due varianze differiscono per il valore di $Var(t)$. Supponiamo che esiste una terza colonna di dati con varianza attesa $Var(e) + (j_0 - 2)Var(s) + (l + 2)Var(t)$. In questo caso la varianza differisce dalla prima colonna per un valore di $2 * Var(t)$.

La probabilità di successo del test statistico di Oscar, che consiste nel calcolare S^2 , è più basso quando egli la calcola alla prima e seconda colonna, opposto a quando la applica alla prima e terza colonna. Quindi, noi deriviamo una stima di questa probabilità di successo nel caso peggiore. Una stima della probabilità in altri casi, può essere ricavata in modo simile.

Supponiamo che r, s, t siano normalmente distribuite. Denotiamo con $N(\mu_s, \sigma_s^2)$ la distribuzione di r ed s , e denotiamo con $N(\mu_t, \sigma_t^2)$ la distribuzione di t . Entrambi:

$$\sum_{j < j_0} (r + s) \quad \text{e} \quad \sum_{\substack{j < j_0 \\ d_j d_{j-1} \neq 00}} t$$

sono normalmente distribuite e i dati nelle colonne corrette e incorrette della tabella sono distribuite in accordo alle seguenti somme:

$$\sum_{j < j_0} (r + s) + \sum_{\substack{j < j_0 \\ d_j d_{j-1} \neq 00}} t \quad \text{e} \quad \sum_{j < j_0} (r + s) + \sum_{\substack{j < j_0 \\ d_j d_{j-1} \neq 00}} t + t$$

Entrambe le somme di queste variabili casuali sono normalmente distribuite. Denotiamo la distribuzione del primo con $N(\mu_0, \sigma_0^2)$ e notiamo che $\sigma_0^2 = (j_0 - 2)\sigma_s^2 + l\sigma_t^2$.



Supponiamo che abbiamo un totale di k misure di tempo accurate. Siano X_1, X_2, \dots, X_k e Y_1, Y_2, \dots, Y_k le variazioni normali standard corrette ed errate rispettivamente. Se gli effetti di errore sono insignificanti, possiamo modellare i dati in due colonne come:

$\sigma_0 X_1 + \mu_0$	$(\sigma_0 X_1 + \mu_0) + (\sigma_t Y_1 + \mu_t)$
$\sigma_0 X_2 + \mu_0$	$(\sigma_0 X_2 + \mu_0) + (\sigma_t Y_2 + \mu_t)$
\vdots	\vdots
$\sigma_0 X_k + \mu_0$	$(\sigma_0 X_k + \mu_0) + (\sigma_t Y_k + \mu_t)$

Per semplificare la nostra notazione, poniamo $V_i = \sigma_0 X_i + \mu_0$ e $W_i = (\sigma_0 X_i + \mu_0) + (\sigma_t Y_i + \mu_t)$. Vogliamo stimare quanto segue:

$$\Pr(S_w^2 > S_v^2) = \Pr\left(\frac{1}{k-1} \sum_{i=1}^k (W_i - \bar{W})^2 > \frac{1}{k-1} \sum_{i=1}^k (V_i - \bar{V})^2\right) = \Pr\left(\sum_{i=1}^k (W_i - \bar{W})^2 > \sum_{i=1}^k (V_i - \bar{V})^2\right)$$

Le variabili V_i e W_i sono normalmente distribuite con le media di μ_0 e $\mu_0 + \mu_t$ rispettivamente. Così, se k è grande, allora $\bar{V} \approx \mu_0$ e $\bar{W} \approx \mu_0 + \mu_t$. Usando questa approssimazione ci dà:

$$\begin{aligned} \Pr(S_w^2 > S_v^2) &\approx \Pr\left(\sum_{i=1}^k (\sigma_0 X_i + \sigma_t Y_i)^2 > \sum_{i=1}^k (\sigma_0 X_i)^2\right) = \\ &= \Pr\left(\sum_{i=1}^k (\sigma_0^2 X_i^2 + 2\sigma_0 \sigma_t X_i Y_i + \sigma_t^2 Y_i^2) > \sum_{i=1}^k \sigma_0^2 X_i^2\right) = \\ &= \Pr\left(2\sigma_0 \sum_{i=1}^k X_i Y_i + \sigma_t \sum_{i=1}^k Y_i^2 > 0\right) \end{aligned}$$

Ora, consideriamo alcune nozioni del calcolo delle probabilità. L'identità $Var(X) = E(X^2) - E(X)^2$ visualizza che $E(X_i^2) = 1$ e $E(Y_i^2) = 1$, poiché X_i e Y_i sono normalmente distribuite. A questo punto, $E\left(\sum_{i=1}^k Y_i^2\right) = \sum_{i=1}^k E(Y_i^2) = k$ e noi useremo questo valore per approssimare $\sum_{i=1}^k Y_i^2$. Poiché X_i e Y_i sono indipendenti, $E(X_i, Y_i) = E(X_i)E(Y_i) = 0$.



Ancora, $Var(X_i, Y_i) = E(X_i^2 Y_i^2) - (E(X_i Y_i))^2 = E(X_i^2 Y_i^2) = E(X_i^2) E(Y_i^2) = 1$.

Applicando il teorema del limite centrale, allora $\sum_{i=1}^k X_i Y_i$ segue, approssimativamente, la distribuzione di $N(0, k)$. Se Z è una variazione standard normale, la probabilità di successo di Oscar, nel caso peggiore, è approssimativamente:

$$\Pr(S_W^2 > S_V^2) \approx \Pr(2\sigma_0(\sqrt{k}Z) + \sigma_t k > 0) = \Pr(Z > -\frac{\sigma_t}{\sigma_0} \frac{\sqrt{k}}{2}) = \Phi(\frac{\sigma_t}{\sigma_0} \frac{\sqrt{k}}{2})$$

dove $\Phi(z)$ è l'area situata sotto la curva normale standard da $-\infty$ a z . Riapplicando i passi della nostra approssimazione, possiamo stimare la probabilità di successo di Oscar in un caso alternativo, quale:

$$\Phi(\frac{\sigma_t}{\sigma_0} \sqrt{\frac{k}{2}})$$

Notiamo che, come ci aspettavamo, la probabilità è più grande del primo caso.

Ricordiamo che $\sigma_0^2 = (j_0 - 2)\sigma_s^2 + l\sigma_t^2$ e ipotizzando che $l = \frac{3(j_0 - 2)}{4}$, abbiamo che $\sigma_0^2 = \frac{(j_0 - 2)}{2}(2\sigma_s^2 + \frac{3}{4}\sigma_t^2)$. Adesso:

$$\frac{\sigma_t}{\sigma_0} = \sqrt{\frac{\sigma_t^2}{\frac{(j_0 - 2)}{2}(2\sigma_s^2 + \frac{3}{4}\sigma_t^2)}} = \sqrt{\frac{2}{(j_0 - 2)(2(\frac{\sigma_s}{\sigma_t})^2 + \frac{3}{4})}}$$

Questa è la probabilità di successo, e in ognuno dei due casi, dipende dai valori di $\sigma_s, \sigma_t, j_0, k$. In questo modo, Oscar procede con il timing attack, facendo variare j_0 da $n-1$ a 1. Vengono recuperati più bit dell'esponente segreto, j_0 decrementa, e così la probabilità di successo incrementa. Ancora, considerando molte misure sui tempi, k incrementa, così la probabilità di successo incrementa anch'essa.



3. Contromisure

Prima di descrivere come difendersi dal timing attack, consideriamo prima due altri approcci comuni verso lo sviluppo di contromisure.

Il primo e il metodo più ovvio, è assicurarsi che tutte le operazioni girino in un ammontare di tempo costante. Sfortunatamente, è difficile perseguire quest'obiettivo. Le ottimizzazioni dei compilatori e della memoria possono introdurre variazioni di tempo non attesi, i quali sfuggono al controllo delle implementazioni. Trattenere il risultato di un'operazione fino a che uno specifico ammontare di tempo terminerà, può sembrare un approccio promettente, ma la lunghezza del ritardo aggiunto può essere determinato, attraverso il consumo di alimentazione del sistema o dell'uso della CPU. Inoltre, usando questo metodo degraderemo l'efficienza del sistema, poiché tutte le operazioni si comporterebbero come se stessero processando degli input nel caso peggiore.

Allora, si potrebbe eseguire la moltiplicazione in ogni ciclo iterativo dell'algoritmo left-to-right come in figura 4, il quale non rende costante l'esecuzione del tempo dell'algoritmo. La variabilità nell'operazione della moltiplicazione e del quadrato rimarranno ancora e questo può essere scoperto.

INPUT: $M, N, d = (d_{n-1}d_{n-2} \dots d_1d_0)_2$

OUTPUT: $S = M^d \bmod N$

1 $S \leftarrow 1$

2 **for** $j = n - 1 \dots 0$ **do**

3 $S \leftarrow S^2 \bmod N$

4 $T \leftarrow S \cdot M \bmod N$

5 **if** $d_j = 1$ **then**

6 $S \leftarrow T$

7 **return** S



Figura 4: Questa modifica dell'algoritmo square and multiply è ancora vulnerabile al timing attack

Come menzionato in precedenza, il timing attack può determinare quali operandi sono usati in ogni passo dell'algoritmo e al percorso dell'esecuzione.

Se l'operazione di moltiplicazione e d'elevazione al quadrato sono eseguite in un tempo costante, allora il tempo per un'esponentiazione modulare sarebbe correlata soltanto dall'ampiezza di Hamming dell'esponente [Appendice B]. Per esponenti casuali, l'ampiezza di Hamming non rileva, in media, molte informazioni circa il suo valore. La moltiplicazione di Montgomery [Appendice C] è eseguita in un tempo quasi costante, ma c'è una piccola sorgente di variabilità risultante da una sottrazione condizionale. RSA con la moltiplicazione di Montgomery è vulnerabile al timing attack [11].

Il secondo metodo è aggiungere rumore al tempo di esecuzione delle varie operazioni. L'effetto desiderato è quello di incrementare il numero richiesto di misure dei tempi in modo tale che l'attacco diventi impraticabile. Il precedente metodo d'attacco e la successiva analisi ha ipotizzato che gli effetti di rumore fossero insignificanti, ma questo non potrebbe essere il caso. Inserendo dei ritardi casuali si produce una sorgente di rumore, ma questo ridurrà l'efficienza se la media del ritardo è abbastanza larga.

Per sconfiggere il timing attack, gli sviluppatori preventiveranno un attaccante dall'apprendere gli input per un'operazione vulnerabile. Nel caso di RSA, se Oscar non conoscesse il valore della base usata nell'esponentiazione modulare, allora la corrispondente informazione di timing non può essere usata. La struttura algebrica di Z_N permette che i messaggi siano blindati [12] prima di essere firmati. Piuttosto che firmare il messaggio $M \in Z_N^*$, Alice può scegliere un $r \in Z_N^*$ casuale e firmare il messaggio $M' = r^e * M \bmod N$. Denotiamo la firma risultante con S' . Alice, adesso, calcola $r^{-1}S' = r^{-1}r^{ed}M^d = r^{-1}rM^d = M^d \bmod N$ per ottenere la sua firma sul messaggio M.



La convenienza di questa tecnica dipende interamente dai dettagli del crittosistema, ma molti sistemi a chiave pubblica hanno una struttura algebrica a richiesta.



4. Conclusioni

La nostra analisi del timing attack, applicata all'esponentiazione modulare in RSAREF, è complicata dal fatto che il metodo d'esponentiazione processa gli esponenti usando una finestra di 2 bit. La nostra discussione sarebbe maggiormente semplificata se avesse considerato un metodo con una finestra da 1 bit. Nel documento [3], Kocher semplifica la sua analisi assumendo che si conosce ogni secondo bit dell'esponente.

Kocher presenta i risultati tratti da diversi esperimenti che sostengono la sua descrizione teorica del timing attack. Sfortunatamente, in quella pubblicazione, Kocher rileva pochi dettagli pratici di come ha eseguito i suoi esperimenti; questo rende il lavoro di riproduzione del suo esperimento come qualcosa di difficile per il lettore.

Altri autori sono stati più imprecisi con i dettagli dei loro esperimenti. Per esempio, esiste una discussione dettagliata nel documento [13] che descrive come precisare le informazioni sui timing che possono essere misurate su un PC.

Si dovrebbe notare che il timing attack di Kocher, come presentato in [3], non applica direttamente le operazioni in RSAREF 2.0 alla firma RSA. Come molte implementazioni di RSA, RSAREF 2.0 usa il Chinese Remainder Theorem (CRT) per calcolare le firme. Una conseguenza di questo metodo è che gli input ai due componenti di esponentiazione modulare sono effettivamente blindati, così il timing attack non può essere applicato. Se un avversario ha l'abilità di scegliere quali messaggi devono essere firmati allora il timing attack può essere applicato alle implementazioni CRT come visualizzato in [14].

I timing attack sono più minacciosi per apparecchi crittografici dedicati (come ad esempio le smartcard) e ci sono stati tre differenti attacchi su OpenSSL basati su applicazioni di decifrazione RSA. E' stato dimostrato che è possibile recuperare le chiavi private RSA usate in questi sistemi.



La tecnica del timing attack illustra che gli attaccanti non necessariamente giocano un ruolo fondamentale, ma essi sfruttano ed attaccano sempre i punti deboli del sistema.



5. Riferimenti

1. D.Brumley and D. Boneh, "Remote timing attacks are practical", at crypto.stanford.edu/~dabo/papers/ssl-timing.pdf
2. J.F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, "A practical implementation of the timing attack", at www.cs.jhu.edu/~fabian/courses/CS600.624/Timing-full.pdf
3. P.Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", at www.cryptography.com/resources/whitepapers/TimingAttacks.pdf
4. S.Levy, "The open secret", *Wired*, issue 7.04, April 1999, at <http://www.wired.com/wired/archive/7.04/crypto.html>
5. OpenSSL Project. <http://www.openssl.org>
6. R. L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems". *Comm. ACM* 21 (1978) no. 2, 120-126.
7. RSA Laboratories. <http://www.rsasecurity.com/rsalabs/node.asp?id=2098>
8. B.Schneier, "Risks of relying on cryptography", Inside Risks 112, *Communications of the ACM*, vol. 42, no. 10, October 1999, at www.schneier.com/essay-021.html
9. US-CERT Vulnerability Note VU#997481, at <http://www.kb.cert.org/vuls/id/997481>



10. G.Blom. Probability and Statistics: Theory and Applications. Springer-Verlag, 1989.
11. J.F. Dhem, F. Koeune, P. A. Leroux, P. Mestrée, J.J. Quisquater, and J.L.Willems. A Practical Implementation of the Timing Attack. Technical Report CG-1998/1, Université catholique de Louvain, 1998. Available from <http://www.dice.ucl.ac.be/crypto>.
12. D. Chaum. Blind Signatures for Untraceable Payments. In R. Rivest and A. Sherman and D. Chaum, editor, *Advances in Cryptology - Proceedings of CRYPTO 82*, volume 0, pages 199–203. Plenum Press, 1983.
13. A. Hevia and M. Kiwi. Strength of Two Data Encryption Standard Implementation Under Timing Attacks. *ACM Transactions on Information and System Security*, 2(4):416–437, November 1999.
14. W. Schindler. A Timing Attack against RSA with the Chinese Remainder Theorem. In C. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of LNCS, pages 109–124. Springer-Verlag, August 2000.



6. Appendice A

Teorema di Eulero: se m è un intero positivo ed a è un intero con $MCD(a, m) = 1$,

allora $a^{\phi(m)} \equiv 1 \pmod{m}$.

Con $MCD(a, m)$ si intende il Massimo Comune Divisore di a ed m e $\phi(m)$ è la funzione phi di Eulero, definita come il numero di interi positivi minore di m che sono relativamente primi a m . Due numeri sono relativamente primi se il loro MCD è 1. E la cosa molto più importante è che se è possibile fattorizzare m , possiamo calcolare velocemente $\phi(m)$.



7. Appendice B

Distanza di Hamming: si chiama **distanza di Hamming** e s'indica come $d_H(s1,s2)$ la differenza di componenti tra due vettori booleani.

Esempio:

Dati

$$s1 = (0001101001)$$

e

$$s2 = (1011001011)$$

la loro distanza di Hamming $d_H(s1,s2)$ è 4.

Infatti le componenti diverse sono la prima, terza, quinta e nona; per calcolare velocemente tale distanza si può fare l'XOR dei due vettori, componente a componente, e contare il numero di componenti 1 ottenute.



8. Appendice C

Moltiplicazione di Montgomery.

L'algoritmo di Montgomery permette di fare il calcolo veloce della moltiplicazione fra numeri rappresentati con un numero elevato di bit. In particolare va a sfruttare alcune proprietà dell'algebra modulare per velocizzare al massimo questa operazione. Il problema in questione si riconduce al calcolo della seguente quantità:

$$ab \bmod n \quad (8.1)$$

Questa operazione, essendo a, b, n numeri molto grandi (dell'ordine di 1024 bit) è piuttosto dispendiosa e, dato che viene eseguita più volte negli algoritmi di crittografia a chiave pubblica, si cerca di ricondurla in una forma più snella.

Ovvero l'algoritmo di Montgomery permette il calcolo della seguente quantità:

$$\text{MonPro}(a, b) = abr^{-1} \bmod n \quad (8.2)$$

E' necessario scegliere un numero r maggiore di n in modo tale che i due numeri siano primi fra loro. Di conseguenza, il massimo comune divisore deve essere uguale ad 1 ($\text{gcd}(n, r) = 1$). La scelta di questo parametro può essere fatta in modo tale che si semplifichino le operazioni che si dovranno compiere. Di conseguenza, si sceglie come r il numero potenza di 2 ($r = 2^k$) tale che valga la seguente disuguaglianza $2^{k-1} \leq n < 2^k$.

In questo modo, le divisioni per r diventano semplici shift, mentre i moduli diventano delle semplici maschere. Infatti, per quanto riguarda il modulo si ha:

$$f \bmod r = f \wedge (r-1) \quad (8.3)$$

(Es. $27 \bmod 8 = 3$ espresso in binario $27 = 11011$ $8 = 1000$ per cui si vede che $27 \& 7 = 11011 \& 00111 = 3$ $27 \bmod 8 = 27 \& 7 = 3$).