Azzolini Riccardo 2021-04-22

RSA

1 Aspetti computazionali

I calcoli richiesti dall'algoritmo RSA sono relativamente costosi, dunque nell'implementazione bisogna prestare attenzione agli aspetti computazionali, cercando di ottimizzare le prestazioni. Ciò è importante soprattutto per la cifratura e la decifratura, che vengono eseguite molto più spesso rispetto alla generazione delle chiavi, ma è comunque bene ottimizzare anche quest'ultima.

1.1 Cifratura e decifratura

Il costo della cifratura ($M^e \bmod n$) e della decifratura ($C^d \bmod n$) è dato:

- dall'elevamento a potenza di un numero grande (M o C) con un esponente grande (e o d);
- dal calcolo del modulo tra due numeri grandi (M^e o C^d e n).

Adesso verranno presentate le principali ottimizzazioni che si possono applicare per ridurre il costo di ciascuna di queste due operazioni.

1.1.1 Elevamento a potenza

Calcolare una potenza in modo "tradizionale" richiede un numero di moltiplicazioni lineare nel valore dell'esponente. Ad esempio, per calcolare x^{16} servono 15 moltiplicazioni:

$$x^{16} = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x$$

Il numero di moltiplicazioni necessarie può essere ridotto scomponendo il calcolo in una sequenza di elevamenti al quadrato:

$$x \cdot x = x^2, \quad x^2 \cdot x^2 = x^4, \quad x^4 \cdot x^4 = x^8, \quad x^8 \cdot x^8 = x^{16}$$

così servono solo 4 moltiplicazioni.

L'esempio appena mostrato è particolarmente semplice perché l'esponente è una potenza di 2, ma esistono degli algoritmi, chiamati *square-multiply*, *binary exponentiation* o *left-to-right*, che eseguono una scomposizione del genere anche per esponenti che non lo sono. Come suggerisce il nome "binary exponentiation", essi calcolano una potenza x^k

operando sulla rappresentazione binaria dell'esponente k, leggendo i bit dal più significativo al meno significativo, cioè da sinistra a destra (da cui il nome "left-to-right"), e procedendo a ogni passo in questo modo:

- per il primo bit 1 incontrato a partire da sinistra si prende semplicemente come risultato del passo il valore x;
- a ogni bit 0 successivo si calcola il quadrato del risultato precedente;
- a ogni bit 1 successivo si effettua un'operazione "square-multiply": si calcola il quadrato del risultato precedente e poi lo si moltiplica per x.

Ad esempio, si supponga di voler calcolare x^{37}. Siccome la rappresentazione binaria dell'esponente 37 è 100101, il calcolo avviene come mostrato nella seguente tabella:

Bit	Risultato del passo
1	x
0	$(x)^2 = x^2$
0	$(x^2)^2 = x^4$
1	$(x^4)^2 \cdot x = x^8 \cdot x = x^9$
0	$(x^9)^2 = x^{18}$
1	$(x^{18})^2 \cdot x = x^{36} \cdot x = x^{37}$

Per ogni bit dell'esponente (eccetto il primo, per cui non si fa nulla) bisogna eseguire una moltiplicazione (l'elevamento al quadrato) se il bit vale 0 o due moltiplicazioni (il quadrato e la moltiplicazione per x) se il bit vale 1. Di conseguenza, il numero di moltiplicazioni è *lineare nel numero di bit* dell'esponente, ovvero *logaritmico nel valore* dell'esponente (mentre il calcolo con il metodo $x \cdot x \cdots x$ richiedeva un numero di moltiplicazioni *lineare nel valore* dell'esponente, cioè *esponenziale nel numero di bit*). Inoltre, tra diversi esponenti aventi lo stesso numeri di bit, quelli con tanti bit a 0 e pochi bit a 1 richiedono meno moltiplicazioni. Perciò, nella generazione delle chiavi RSA, invece di selezionare un e casuale si fissa spesso $e = 65537$, che ha la rappresentazione binaria 10000000000000001, composta da 17 bit di cui 15 sono a 0, quindi l'elevamento a questo esponente richiede 17 moltiplicazioni (nessuna per l'uno più significativo, 15 per gli zeri e 2 per l'uno meno significativo).

Scegliendo l'esponente $e = 65537$ (o un altro con caratteristiche simili) si ottimizzano le operazioni che usano appunto e (la cifratura per la segretezza e la verifica delle firme), ma una volta fissato e non è possibile scegliere anche un d ottimale, perché il valore di d deve per forza essere calcolato come inverso moltiplicativo modulo $\phi(n)$ di e, $d = e^{-1} \bmod \phi(n)$, dunque le operazioni che usano d (la decifratura e la generazione delle firme) potrebbero risultare meno efficienti. Allora, come già anticipato, se si vogliono invece ottimizzare le operazioni con d (a scapito di quelle con e) si fissa d e si calcola $e = d^{-1} \bmod \phi(n)$.

Fissare il valore di e in modo non casuale significa che molte chiavi pubbliche hanno lo stesso valore di e, e si distinguono solo per i valori di n. Ciò non è un problema di

sicurezza, perché se cambia n cambia anche $\phi(n)$, e uno stesso e ha inversi moltiplicativi diversi per moduli $\phi(n)$ diversi, quindi anche per chiavi pubbliche con lo stesso e il parametro privato d assume valori completamente diversi.

Un'ulteriore ottimizzazione dell'elevamento a potenza, realizzata da molte librerie crittografiche, può essere ottenuta sfruttando il "teorema cinese del resto" (Chinese Remainder Theorem). Essa non verrà presentata nel dettaglio, ma in sostanza consiste nel precomputare dei particolari parametri privati (derivati da p, q e d) che sostituiscono la chiave privata "tradizionale" e forniscono un modo alternativo di calcolare il valore $C^d \bmod n$ (cioè eseguire la decifratura o la generazione di una firma) nel quale si opera su numeri più piccoli, ottenendo così una maggiore efficienza.

1.1.2 Calcolo del modulo

Il calcolo del modulo di numeri molto grandi può essere evitato sfruttando la seguente proprietà dell'aritmetica modulare per scomporre il calcolo in calcoli su numeri più piccoli:

$$ab \bmod n = (a \bmod n)(b \bmod n) \bmod n$$

Ad esempio, invece di calcolare

$$88^7 \bmod 187 = 40867559636992 \bmod 187 = 11$$

si può calcolare

$$\begin{aligned} 88^7 \bmod 187 &= 88^{4+2+1} \bmod 187 = 88^4 \cdot 88^2 \cdot 88^1 \bmod 187 \\ &= (88^4 \bmod 187)(88^2 \bmod 187)(88^1 \bmod 187) \bmod 187 \\ &= (59969536 \bmod 187)(7744 \bmod 187)(88 \bmod 187) \bmod 187 \\ &= 132 \cdot 77 \cdot 88 \bmod 187 = 894432 \bmod 187 = 11 \end{aligned}$$

dove il numero più grande su cui si opera è $88^4 = 59969536$, decisamente più piccolo di $88^7 = 40867559636992$.

Nel fare questa scomposizione del calcolo bisogna trovare un equilibrio tra la dimensione dei numeri su cui si opera e le moltiplicazioni da effettuare: se si esagerasse con la scomposizione da un lato si riporterebbe il calcolo a valori piccoli, ma dall'altro si introdurrebbe un numero eccessivo di moltiplicazioni. Un modo efficace di fare la scomposizione, usato nell'esempio precedente, è scrivere l'esponente come somma di potenze di due, il che può essere fatto tramite la sua rappresentazione binaria, e poi applicare le proprietà delle potenze per trasformare la somma all'esponente in un prodotto di potenze: 7 in binario è 111, ovvero $2^2 + 2^1 + 2^0 = 4 + 2 + 1$, quindi $88^7 = 88^{4+2+1} = 88^4 \cdot 88^2 \cdot 88^1$.

È anche possibile combinare le ottimizzazioni per l'elevamento a potenza e per il calcolo del modulo, eseguendo direttamente in modulo n i calcoli dell'algoritmo square-multiply. Ad esempio, per $88^7 \bmod 187$, ricordando che i bit dell'esponente 7 sono 111:[1]

Bit	Risultato del passo
1	88
1	$88^2 \cdot 88 \bmod 187 = 681472 \bmod 187 = 44$
1	$44^2 \cdot 88 \bmod 187 = 170368 \bmod 187 = 11$

1.2 Generazione delle chiavi

Nella generazione delle chiavi, le operazioni più interessanti dal punti di vista computazionale sono la selezione dei due numeri primi p e q, la selezione di un e o d coprimo con $\phi(n)$ e il calcolo del suo inverso moltiplicativo modulo $\phi(n)$. Come valore del parametro e o d si sceglie solitamente un numero primo, rendendo così più facile la verifica che esso sia coprimo con $\phi(n)$, e l'inverso moltiplicativo viene calcolato in modo efficiente usando, come già detto, l'algoritmo esteso di Euclide. Invece, la selezione dei numeri primi è più difficile.

Siccome il parametro $n = pq$ è pubblico, p e q devono essere scelti da un insieme sufficientemente esteso di valori, altrimenti sarebbe facile provare in modo esaustivo tutte le combinazioni dei possibili valori per risalire a p e q da n. Non esistono però algoritmi per generare un numero primo arbitrariamente grandi. Allora, si sceglie un grande numero dispari (al fine di evitare il caso banale in cui un numero non è primo in quanto divisibile per 2), si applica un *test di primalità* per determinare se esso è primo, e se non lo è si ripete la selezione finché non si trova un numero primo.

Per determinare in modo efficiente se un numero grande è primo si usano dei test di primalità *probabilistici* (come ad esempio il test di Miller-Rabin): essi non danno la conferma che un numero sia primo, ma indicano se ha una certa probabilità di essere primo, e possono essere ripetuti più volte sullo stesso numero per far avvicinare a 1 questa probabilità.

2 Sicurezza

La sicurezza di RSA è misurata dalla lunghezza in bit del modulo n, che perciò viene considerata la lunghezza della chiave (nonostante la chiave contenga anche il parametro e o d). Come al solito, i tipi di attacchi da considerare sono la forza bruta e la crittoanalisi.

[1] Qui nei passi square-multiply sono state effettuati insieme l'elevamento al quadrato e la moltiplicazione per 88, applicando il modulo solo alla fine, ma con numeri più grandi potrebbe essere conveniente applicare il modulo anche al risultato intermedio del quadrato per ridurre ulteriormente la dimensione dei numeri su cui si opera.

- Un attacco a forza bruta consisterebbe nel provare tutti i possibili valori dell'esponente d, che sono numeri grandi, ovvero ci sono tanti valori possibili, dunque la possibilità della forza bruta può sostanzialmente essere esclusa.
- Un attacco di crittoanalisi cerca di ricavare d conoscendo la chiave pubblica $\{e, n\}$, il che può essere fatto in vari modi:
  - fattorizzando n nei due fattori primi p e q si possono calcolare $\phi(n) = (p-1)(q-1)$ e $d = e^{-1} \bmod \phi(n)$, ma la fattorizzazione di un numero grande è costosa;
  - si può calcolare direttamente $\phi(n)$ senza determinare p e q, ma ciò ha un costo sostanzialmente uguale alla fattorizzazione di n;
  - si può cercare di determinare in qualche modo d senza calcolare $\phi(n)$, per mezzo di particolari attacchi matematici, ma anche questi hanno un costo uguale alla fattorizzazione di n.

  Di conseguenza, il costo degli attacchi di crittoanalisi è appunto dato dalla difficoltà della fattorizzazione di un numero n grande.

Negli anni sono state fatte varie sfide (challenge) di decodifica di cifrature RSA. La prima di queste sfide, indetta nel 1977, chiedeva di decifrare (senza conoscere la chiave privata) un messaggio cifrato con un n a 428 bit; essa fu risolta solo nel 1994, quasi 20 anni dopo, da un gruppo che riuscì a fattorizzare n in 9 mesi usando 1600 computer. Successivamente sono state proposte sfide con chiavi sempre più grandi, e nel 2020 si è arrivati a fattorizzare un n di 829 bit. Per questo motivo, le chiavi di 1024 bit che si potevano usare fino a un po' di anni fa non sono più considerate sufficienti: si consigliano almeno 2048 bit.

Per massimizzare la sicurezza si potrebbe pensare di usare chiavi anche molto più grandi di 2048 bit, ma ciò renderebbe troppo costose le operazioni di cifratura e decifratura: raddoppiare la lunghezza della chiave (cioè del modulo n) aumenta di un fattore 4 il costo delle operazioni con la chiave pubblica e di un fattore 8 il costo delle operazioni con la chiave privata (o il contrario se nella generazione delle chiavi è stato ottimizzato l'esponente d invece di e). In sostanza, la scelta della lunghezza della chiave è un compromesso tra sicurezza ed efficienza, ed esistono diverse raccomandazioni ufficiali (NIST, NSA, RFC 3766, ECRYPT, ecc.) su come scegliere in base al livello di sicurezza desiderato (misurato ad esempio come "security strength", il numero di bit di una chiave per un algoritmo tradizionale, simmetrico, che darebbe un livello di sicurezza equivalente a una chiave RSA di una certa lunghezza).

Infine, dagli algoritmi di fattorizzazione usati per risolvere le varie sfide sono stati ricavati degli accorgimenti su come scegliere p e q in modo da generare valori di n resistenti alla fattorizzazione. Ad esempio:

- p e q devono differire per poche cifre;

- sia $p-1$ che $q-1$ devono contenere un grosso fattore primo;
- MCD$(p-1, q-1)$ deve essere piccolo.