Crittosistemi asimmetrici

Sommario

- **Scambio di chiave**:
  - Diffie-Hellman (logaritmo discreto)
- **Cifratura asimmetrica**:
  - RSA (fattorizzazione di interi)
  - ElGamal (logaritmo discreto)
- **Firma digitale**:
  - RSA (fattorizzazione di interi)
  - ElGamal (logaritmo discreto)
  - DSA (logaritmo discreto)

Diffie-Hellman

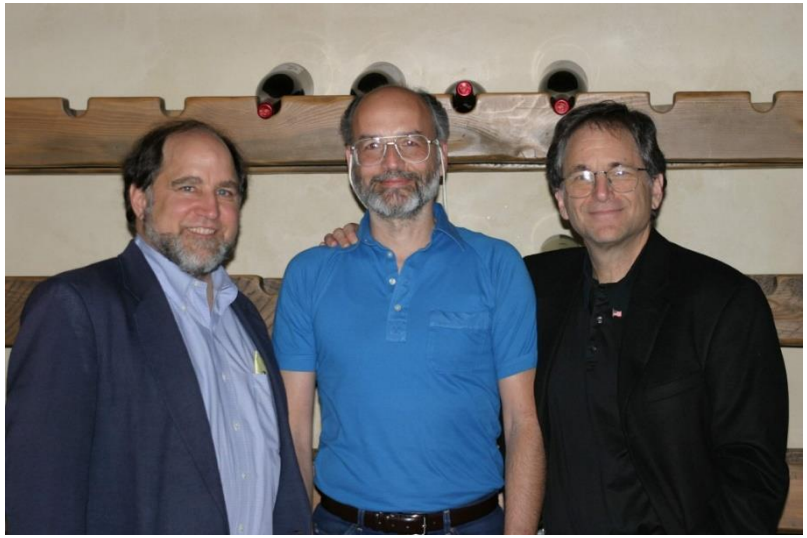
- Nel loro celebre lavoro del 1976 [1], Whitfield Diffie e Martin Hellman introdussero il paradigma della crittografia asimmetrica come soluzione per **scambio di chiavi**, **cifratura** e **autenticazione** (firma digitale).

[1] W. Diffie and M. Hellman, "New directions in cryptography," in *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, November 1976.

RSA

- In termini pratici, Diffie e Hellman introdussero una procedura per lo **scambio di chiavi** basata sul logaritmo discreto.

- Nel 1977 Ronald Rivest, Adi Shamir e Leonard Adleman proposero un metodo per la **cifratura asimmetrica** basato sulla fattorizzazione di numeri interi.



Prima di loro…

- **James Henry Ellis** nel 1970 concepì l'idea di una "cifratura non segreta", ovvero della crittografia a chiave pubblica
- **Clifford Cocks** nel 1973 ideò lo schema che noi oggi conosciamo come RSA
- **Malcolm John Williamson** nel 1974 ideò lo schema che noi oggi conosciamo come "scambio di chiavi Diffie–Hellman"
- Tutte queste informazioni furono considerate classificate, pertanto rimasero segrete
- Solo nel 1997 il governo Britannico le ha declassificate, rendendo pubblico il contributo di Ellis, Cocks e Williamson

Diffie-Hellman: procedura

1. Alice sceglie un numero primo p per cui sia difficile calcolare il logaritmo discreto (mod p) e una radice primitiva α (mod p).

2. Alice pubblica p ed α.

3. Alice sceglie a caso un esponente x segreto, con $1 \leq x \leq p - 2$, ed invia α^x (mod p) a Bob

4. Bob sceglie a caso un esponente y segreto, con $1 \leq y \leq p - 2$, ed invia α^y (mod p) ad Alice.

5. Alice calcola $(\alpha^y)^x = \alpha^{xy}$

6. Bob calcola $(\alpha^x)^y = \alpha^{xy}$

Diffie-Hellman: sicurezza

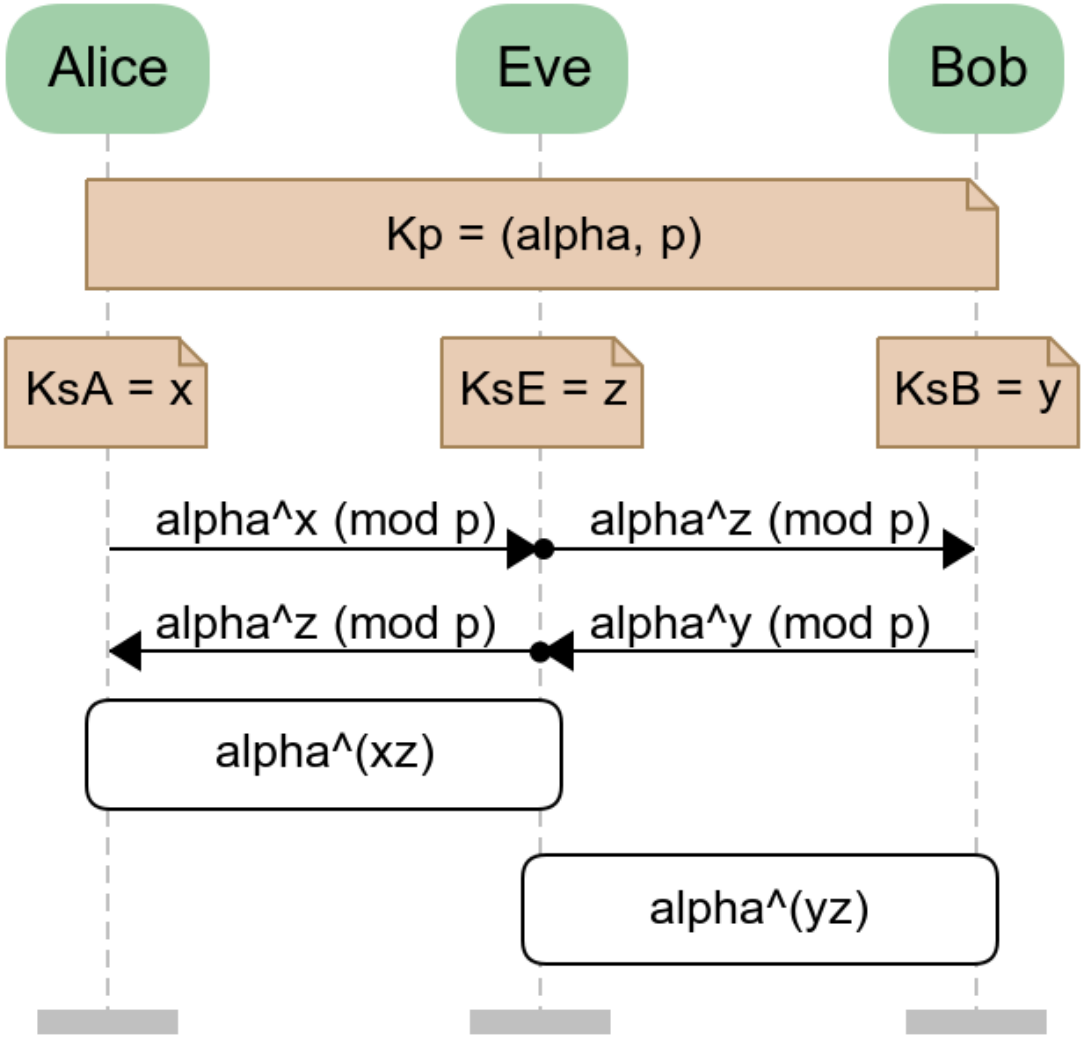
- Eve conosce α^x e α^y
- Se Eve è in grado di calcolare logaritmi discreti può violare il sistema, in quanto può calcolare x oppure y e ricavare α^{xy}.
- Ma Eve potrebbe non avere necessariamente bisogno di calcolare x o y per recuperare il segreto.

Problema computazionale di Diffie-Hellman: Sia p un primo e sia α una radice primitiva (mod p). Dati α^x (mod p) e α^y (mod p), trovare α^{xy} (mod p).

- Non è noto se questo problema sia più facile del calcolo del logaritmo discreto.
- Certamente non è più difficile.

Attacco man-in-the-middle (MITM)

Diffie-Hellman: attacco MITM



RSA


HERE'S TO ADI, RON AND LEN.
FOR GIVING US RSA PUBLIC-KEY CRYPTOGRAPHY.
IN 1977, THESE THREE INVENTED THE MODERN DIGITAL SECURITY FRAMEWORK.
Credit: Ron Rivest

RSA: generazione delle chiavi

- RSA si basa sulla difficoltà di fattorizzare gli interi in fattori primi.
- Bob sceglie due primi p e q grandi e distinti e li moltiplica per formare il numero semiprimo $n = pq$ (detto **modulo**)
- Bob sceglie un intero e (detto **esponente di cifratura**) tale che $\text{MCD}(e, (p-1)(q-1)) = 1$
- Conoscendo p e q, Bob può calcolare $\varphi(n) = (p-1)(q-1)$ e può quindi calcolare d (detto **esponente di decifratura**) tale per cui $de \equiv 1 \pmod{(p-1)(q-1)}$
- La coppia di chiavi di Bob è
  - chiave pubblica: $\{n, e\}$
  - chiave privata: $\{p, q, d\}$

RSA: cifratura e decifratura

- Alice scrive il messaggio come un numero m.
- Se m è più grande di n, Alice spezza il messaggio in blocchi, ognuno rappresentato da un numero $< n$.
- Per il momento supponiamo che $m < n$.
- Alice cifra m calcolando: $c \equiv m^e \pmod{n}$ e invia c a Bob.
- Bob decifra c calcolando $m \equiv c^d \pmod{n}$.

RSA: razionale

- Per il Teorema di Eulero, se MCD(a, n) = 1, allora $a^{\varphi(n)} \equiv 1 \pmod{n}$.
- Nel caso in esame, $\varphi(n) = \varphi(pq) = (p-1)(q-1)$.
- Dal momento che p e q sono grandi, probabilmente m non contiene nessuno di essi come fattori, e MCD (m, n) = 1.
- Poiché $de \equiv 1 \pmod{\varphi(n)} \rightarrow de = 1 + k\varphi(n)$, con k intero.
- Ne segue che:
  $c^d \equiv (m^e)^d \equiv m^{1+k\varphi(n)} \equiv m \cdot \left(m^{\varphi(n)}\right)^k \equiv m \cdot (1)^k \equiv m \pmod{n}$
- È molto probabile che Bob possa recuperare il messaggio anche se MCD(m, n) $\neq$ 1.

RSA: esempio

$p = 885320963, \qquad q = 238855417$
$\rightarrow n = pq = 211463707796206571$

$e = 9007$

Alice deve inviare *cat*.
Convenzione: $a = 01$
$b = 02$
…
$z = 26$

$\rightarrow cat = 030120 = 30120$

$c \equiv m^e \equiv 30120^{9007} \equiv 113535859035722866 \pmod{n}$

RSA: esempio (cont.)

Mediante l'algoritmo Euclideo esteso, Bob calcola:

$d = 116402471153538991$

Infine:

$c^d \equiv 113535859035722866^{116402471153538991} \equiv 30120 \pmod{n}$

coincidente con il messaggio originale.

RSA: sicurezza

- Eve conosce *n* ed *e* e può intercettare *c*.
- Eve non conosce *p*, *q* e *d*.
- *d* va mantenuto segreto perché la fattorizzazione di *n* è possibile se si conosce *d*.
- Eve conosce $c = m^e$; non può fare la radice *e*-sima?
  - se non si lavora in aritmetica modulare questo è banale, ma…
  - nel contesto in esame, questo è oltremodo complesso (se *n* è grande).

RSA: sicurezza

- Bob sceglie *p* e *q* a caso, e indipendentemente l'uno dall'altro.
- I valori di *p* e *q* sono molto grandi: almeno 100 cifre.
- È preferibile sceglierli con lunghezze lievemente diverse tra loro.
- Alcuni valori di *p* e *q* devono essere evitati perché facilitano la fattorizzazione.

RSA: sicurezza

- Trovare $\varphi(n)$ o l'esponente di decifratura è difficile quanto fattorizzare n.
- Se fattorizzare è difficile allora non dovrebbero esistere metodi veloci ed ingegnosi per trovare d.
- Sia $n = pq$ il prodotto di due primi distinti. Se n e $\varphi(n)$ sono noti, allora p e q possono essere calcolati rapidamente. Infatti:

$$p, q = \frac{n - \varphi(n) + 1 \pm \sqrt{(n - \varphi(n) + 1)^2 - 4n}}{2}$$

- come radici del polinomio:

$$X^2 - (n - \varphi(n) + 1)X + n = X^2 - (pq - (p-1)(q-1) + 1)X + pq$$
$$= X^2 - (p+q)X + pq = (X - p)(X - q)$$

- Se d ed e sono noti, allora n può essere probabilmente fattorizzato (metodo dell'*esponente universale*).

RSA: velocità

- Cifratura e decifratura richiedono il calcolo di potenze in aritmetica modulare, come m^e (mod n).
- Questo calcolo può essere svolto rapidamente e senza troppa memoria, ad esempio mediante quadrature successive.
- Se invece si tentasse di calcolare prima m^e e poi ridurlo (mod n) probabilmente si incapperebbe in un overflow di memoria.
- Le operazioni richieste da RSA richiedono un tempo pari a una potenza di log(n).
- Sono tempi accettabili se la mole di dati da cifrare (o firmare) è contenuta.

Attacchi a RSA

Teorema:
Sia t il numero delle cifre di $n = pq$. Se si conoscono le prime $t/4$, o le ultime $t/4$, cifre di p, allora si può fattorizzare n in modo efficiente.

Teorema:
Sia (n, e) una chiave pubblica RSA, sia t il numero delle cifre di n e sia d l'esponente di decifratura. Se si hanno almeno le ultime $t/4$ cifre di d, allora si può trovare d in modo efficiente in un tempo che è lineare in $e \cdot \log_2 e$.

- Se e è piccolo, allora è piuttosto veloce trovare d quando si conosce una parte consistente di esso. Se e è grande (Es.: circa n) il teorema non dà un risultato più favorevole di una ricerca di d caso per caso.

Scelta degli esponenti *e* e *d*

- Esponenti di cifratura e decifratura bassi sono attraenti perché accelerano i tempi di elaborazione.
- Ci sono però alcuni pericoli, che devono essere evitati.
- Queste “trappole” possono essere evitate usando esponenti grandi.
- Scelta abbastanza comune: $e = 2^{16} + 1 = 65537$.
- Questo numero è primo e quindi la condizione MCD(e, $(p-1)(q-1)$) = 1 è molto probabilmente verificata.
- Poiché è più grande di 1 di una potenza di 2, l’elevamento a potenza per questo numero può essere eseguito rapidamente:

$$m^{65537} = \left\{\left\{\left[(m^2)^2\right]^2\right\}^{\cdot^{\cdot}}\right\}^2 \cdot m$$

dove l’elevamento al quadrato è fatto 16 volte.

Attacchi con esponenti bassi

- L'esponente di decifratura d dovrebbe essere sufficientemente grande in modo che sia impossibile trovarlo con la sola forza bruta.
- Considerare gli attacchi a forza bruta non è sufficiente.

Teorema.
Siano p e q due primi con $q < p < 2q$. Sia $n = pq$, e siano $1 \leq d, e \leq \varphi(n)$ tali che $de \equiv 1 \pmod{(p-1)(q-1)}$. Se $d < 1/3n^{1/4}$, allora d può essere calcolato rapidamente (in un tempo polinomiale in $\log(n)$).

- Rilassando le ipotesi del teorema (cioè imponendo condizioni meno stringenti), Eva può usare il metodo per calcolare d in molti casi.
- Per questo motivo, si consiglia di scegliere d sempre molto grande.

Testo in chiaro corto

- Un uso comune di RSA è per trasmettere chiavi da usare con cifrari simmetrici come DES o AES.
- Essendo la chiave DES lunga 56 bit ed essendo $2^{56} - 1 \approx 7.2 \cdot 10^{16}$, si può pensare di scrivere la chiave come un numero $m < 10^{17}$.
- m viene cifrato con RSA ottenendo $c \equiv m^e \pmod{n}$.
- Anche se m è "piccolo", c è probabilmente un numero della stessa grandezza di n (per esempio di circa 200 cifre).
- Tuttavia, la dimensione ridotta di m rende efficiente un attacco specifico da parte di Eva.

Testo in chiaro corto (2)

- Eva genera due liste:
  1. $cx^{-e} \pmod{n}$ per ogni x con $1 \leq x \leq 10^9$.
  2. $y^e \pmod{n}$ per ogni y con $1 \leq y \leq 10^9$.
- Poi cerca una corrispondenza tra un elemento della prima lista e un elemento della seconda lista.
- Se ne trova una, allora $cx^{-e} \equiv y^e \pmod{n}$ per qualche x e y.
- Pertanto: $c \equiv (xy)^e \pmod{n}$.
- E quindi: $m = xy$.
- Se m è il prodotto di due interi $< 10^9$ l'attacco ha successo.

Testo in chiaro corto (3)

- Questo attacco è molto più efficiente della prova di tutte le 10^{17} possibilità per *m*.
- L'attacco è molto simile al meet-in-the-middle che si usa per attaccare l'algoritmo DES.
- Si previene aggiungendo a caso qualche cifra all'inizio o alla fine di *m* in modo da formare un testo in chiaro molto più lungo.
- Quando Bob decifra, rimuove queste cifre casuali e riottiene *m*.

Optimal Asymmetric Encryption Padding (OAEP)

Mihir Bellare

Philip Rogaway

- n è il modulo RSA di k bit (quindi $< 2^k$)
- Si fissano due interi positivi k_0 e k_1 con $k_0 + k_1 < k$.
- m può essere formato da $k - k_0 - k_1$ bit.
- Valori tipici: $k = 1024$, $k_0 = k_1 = 128$, $k - k_0 - k_1 = 768$.
- Sia G una funzione che prende in input stringhe di k_0 bit e restituisce in output stringhe di $k - k_0$ bit.
- Sia H una funzione che prende in input stringhe di $k - k_0$ bit e restituisce in output stringhe di k_0 bit.

OAEP (2)

- *G* e *H* sono costruite mediante funzioni *hash*.
- Quando Alice deve cifrare *m*:
  1. Lo allunga a $k - k_0$ bit aggiungendo k_1 bit uguali a zero: $m0^{k_1}$
  2. Sceglie a caso una stringa *r* di k_0 bit e calcola:
     $$x_1 = m0^{k_1} \oplus G(r), \qquad x_2 = r \oplus H(x_1)$$
  3. Se $x_1 || x_2 < n$, Alice forma il testo cifrato:
     $$E(m) = (x_1 || x_2)^e \pmod{n}$$
  4. Altrimenti Alice fa un nuovo tentativo cambiando *r*.

OAEP (3)

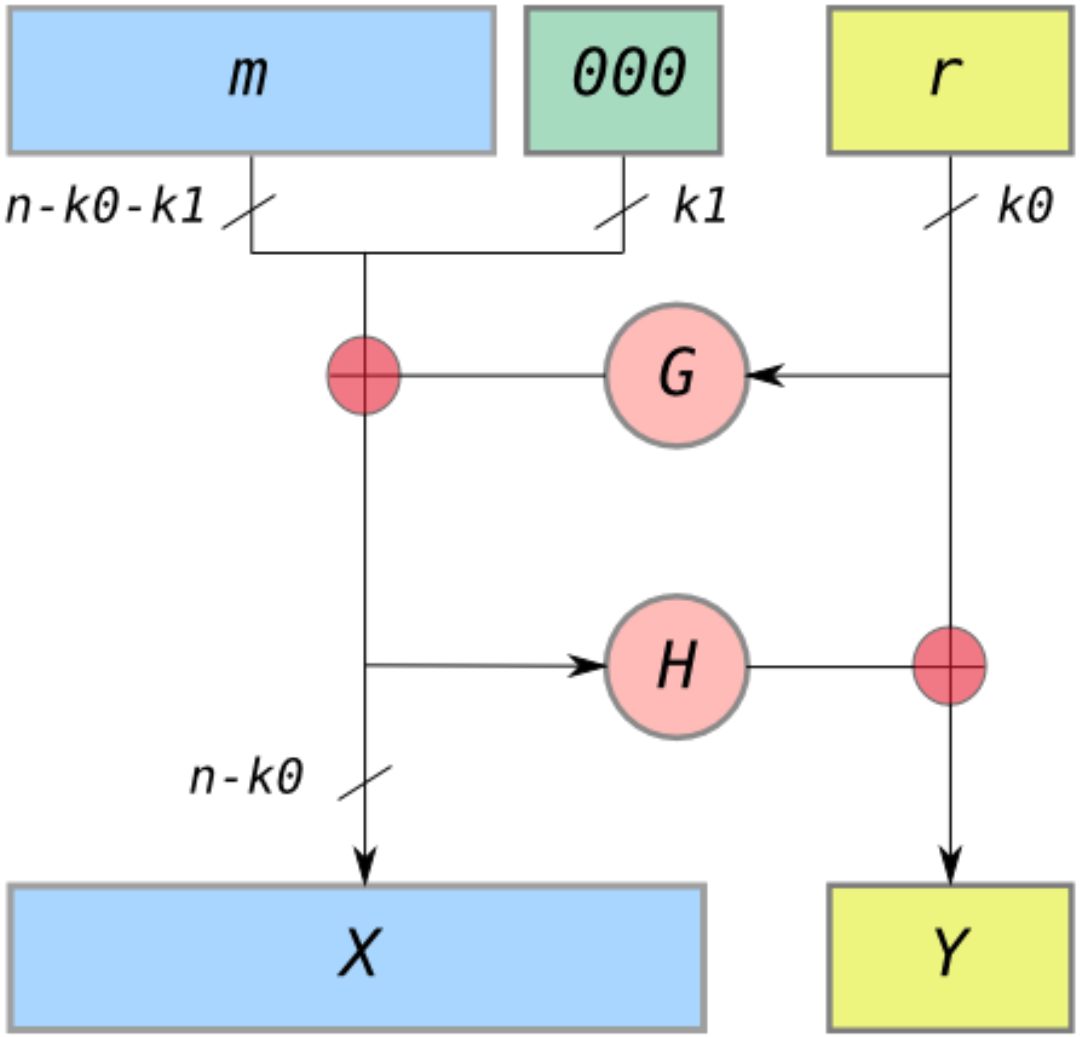
- In decifratura: $c^d \pmod n = y_1 || y_2 = x_1 || x_2$, con y_1 formato da $k - k_0$ bit e y_2 formato da k_0 bit.

- Bob calcola:

$$y_1 \oplus G(H(y_1) \oplus y_2) = y_1 \oplus G\big(H(x_1) \oplus r \oplus H(x_1)\big)$$
$$= y_1 \oplus G(r) = x_1 \oplus G(r) = m0^{k_1}$$

- Bob rimuove i k_1 bit nulli finali e ottiene m.

OAEP



Vantaggi OAEP (cifratura plaintext-aware)

1. Controllo sull'integrità (se non ci sono gli zeri finali il testo cifrato non corrisponde a una cifratura valida)

2. Il riempimento con x_2 dipende dal messaggio m e dal parametro casuale r
   $\rightarrow$ più difficili gli attacchi di testo cifrato scelto.

Attacchi basati sul tempo di esecuzione

- Attacco proposto da **Paul Kocher** nel 1995.
- Si supponga che Eva sia in grado di osservare (a distanza) Bob mentre decifra diversi testi cifrati y.
- Eva misura il tempo utilizzato per ogni y.
- La misurazione del tempo di decifratura è possibile, ad esempio, quando il terminale che decifra manda un ACK al mittente. È sufficiente la misura dei tempi di risposta.
- La conoscenza di ogni y e del tempo necessario per la decifratura permette ad Eva di trovare d.
- Bisogna conoscere l'hardware usato per calcolare y^d.

Esempio

- Possibile algoritmo per il calcolo di $y^d \pmod n$.

Let $d = b_1 b_2 \ldots b_w$ *be written in binary (for example, when* $x = 1011$, *we have* $b_1 = 1, b_2 = 0, b_3 = 1, b_4 = 1$) *Let* y *and* n *be integers Perform the followng procedure:*

1. *Start with* $k = 1$ *and* $s_1 = 1$.
2. *If* $b_k = 1$, *let* $r_k \equiv s_k y \pmod n$. *If* $b_k = 0$, *let* $r_k = s_k$.
3. *Let* $s_{k+1} \equiv r_k^2 \pmod n$.
4. *If* $k = w$, *stop. If* $k < w$, *add 1 to* k *and go to (2)*

Then $r_w \equiv y^d \pmod n$.

- Si vede che la moltiplicazione $s_k y$ viene effettuata solo quando $b_k = 1$.
- Conoscendo l'hardware, si può allora avere immediatamente un'idea del numero di bit 1 presenti in *d*, ma questo non è sufficiente per conoscere *d*.

Esempio (2)

- Il tempo richiesto da una moltiplicazione (come pure da altre operazioni coinvolte nel calcolo) può presentare una variabilità molto grande.
- Necessità di un'analisi statistica.
- Eva osserva n testi cifrati y_1, …, y_n e determina i tempi t_i necessari per calcolare ogni $y_i^d \pmod n$.
- Eva stima:
  - valore medio: $\mu = \frac{t_1 + \cdots + t_n}{n}$
  - varianza: $\mathrm{Var}(\{t_i\}) = \frac{(t_1 - \mu)^2 + \cdots + (t_n - \mu)^2}{n}$

Esempio (3)

- Supponiamo che per ogni y_i Eva sappia stimare il tempo t_i' necessario per effettuare la moltiplicazione $s_k y_i$, pur non sapendo se essa viene eseguita o meno.
- Di conseguenza, Eva può anche stimare il tempo $t_i'' = t_i - t_i'$ necessario per tutte le altre operazioni.
- Eva stima:
  - valore medio: $\mu'' = \frac{t_1'' + \cdots + t_n''}{n}$
  - varianza: $\mathrm{Var}(\{t_i''\}) = \frac{(t_1'' - \mu'')^2 + \cdots + (t_n'' - \mu'')^2}{n}$

Esempio (4)

- Se la moltiplicazione viene eseguita, è ragionevole supporre che t_i' e t_i'' siano tra loro **statisticamente indipendenti**. Essendo $t_i = t_i' + t_i''$, si ha allora:

$$\mathrm{Var}(\{t_i\}) \approx \mathrm{Var}(\{t_i'\}) + \mathrm{Var}(\{t_i''\}) > \mathrm{Var}(\{t_i''\})$$

- Se la moltiplicazione non viene eseguita, t_i' è il tempo necessario per un'operazione che non ha legami con il calcolo, e quindi è ragionevole supporre che t_i e t_i' siano tra loro **statisticamente indipendenti**. Essendo $t_i'' = t_i - t_i'$, si ha allora:

$$\mathrm{Var}(\{t_i''\}) \approx \mathrm{Var}(\{t_i\}) + \mathrm{Var}(\{-t_i'\}) > \mathrm{Var}(\{t_i\})$$

poiché la varianza è sempre positiva.

Esempio (5)

- In sintesi:

$$\mathrm{Var}(\{t_i\}) > \mathrm{Var}(\{t_i''\}) \quad \rightarrow \quad b_k = 1$$
$$\mathrm{Var}(\{t_i\}) < \mathrm{Var}(\{t_i''\}) \quad \rightarrow \quad b_k = 0$$

- Eva dunque deve calcolare $\mathrm{Var}(\{t_i\})$ e $\mathrm{Var}(\{t_i''\})$ e procedere al confronto.
- La procedura è ricorsiva: Eva cerca di ricostruire i bit b_k uno alla volta.
- L'attacco presuppone che la decifratura non abbia una durata fissa.

Side channel attacks

- Attacchi come i precedenti, basati su «informazioni laterali» (tempo di esecuzione, consumo di potenza, emissioni elettromagnetiche…), si definiscono **side channel attacks**.
- Essi presuppongono la conoscenza del software e/o hardware su cui si esegue la decifratura.
- Sono fattibili quando software e/o hardware variano il loro funzionamento in funzione della chiave segreta.
- Devono essere prevenuti con opportune scelte implementative (implementazioni a **tempo/potenza costante**, indipendenti dalla chiave segreta).

Attacchi basati su fattorizzazione

- RSA si può attaccare fattorizzando *n*, almeno in principio
- Fattorizzare un numero e testarne la primalità non sono la stessa cosa
- È molto più facile testare se un numero è composto che non fattorizzarlo
- Esistono molti grandi interi che non sono mai stati fattorizzati, anche se si sa che sono composti

Accorgimenti

- Bisogna garantire che $p - 1$ abbia almeno un fattore molto grande.
- Supponiamo di volere che p abbia circa 100 cifre.
- Si sceglie un primo $p_0 \approx 10^{40}$ (grande).
- Si cercano gli interi della forma: $kp_0 + 1$, con k che varia tra alcuni interi attorno a 10^{60}.
- Si controlla la primalità.
- In media si ottiene un valore di p in meno di 100 passi.
- Si ripete la procedura per q.
- In questo modo $n = pq$ sarà difficile da fattorizzare usando metodi noti.

Test di primalità

- Si supponga di avere un intero *n* di 200 cifre e di dover stabilire se è o meno un numero primo.
  - Metodo a forza bruta: si divide *n* per tutti i numeri primi che gli sono minori.
  - Metodo a forza bruta migliorato: si considerano solo i numeri primi minori o uguali alla radice quadrata di *n*.
- Ci sono circa $4 \cdot 10^{97}$ numeri primi minori di 10^{100} e testarli tutti in un tempo accettabile è infattibile.
- **Esempio**. Capacità di elaborazione di 10^9 numeri primi al secondo: tempo stimato: 10^{81} anni!

Test di primalità di Fermat

- Sia $n > 1$ un intero. Sia a un intero casuale tale che $1 < a < n - 1$. Se $a^{n-1} \neq 1 \pmod{n}$, allora n è composto. Se invece $a^{n-1} \equiv 1 \pmod{n}$, allora n è probabilmente primo.
- **Esempio**: $n = 35$, $a = 2$, $2^{34} \equiv 9 \pmod{35} \rightarrow 35$ è composto.
- Il test di Fermat è molto accurato per n grande.
- Può essere eseguito rapidamente poiché l'elevamento a potenza modulare è veloce.
- Se gli elevamenti a potenza sono eseguiti opportunamente, il test di Fermat può essere combinato con il Principio Fondamentale per ottenere un risultato più forte.

Altri test di primalità

- I test di Miller-Rabin e di Solovay-Strassen possono essere eseguiti rapidamente.
- Questi test non danno una dimostrazione rigorosa del fatto che un numero sia primo.
- Questi metodi sono quasi tutti probabilistici: non garantiscono il successo, anche se la probabilità di successo è normalmente molto alta.
- Esistono test che danno una dimostrazione rigorosa della primalità ma sono in generale molto più lenti.
- Un algoritmo deterministico con tempo polinomiale è stato introdotto da Agrawal, Kayal e Saxena nel 2002, ma non è stato ancora migliorato al punto da poter competere con gli algoritmi probabilistici.

Record di fattorizzazione

- Nell'ultima metà del ventesimo secolo si sono fatti enormi progressi nella fattorizzazione, in parte per lo sviluppo dei computer e in parte per il miglioramento degli algoritmi.

Anno	Numero di cifre
1964	20
1974	45
1984	71
1994	129
1999	155
2003	174
2005	200
2009	232

RSA challenge

- Nel 1991, la RSA Laboratories (http://www.rsa.com/rsalabs/) pubblicò 54 semiprimi con numero di cifre decimali compreso tra 100 e 617.
- La sfida, che consisteva nel trovare la fattorizzazione di tali numeri, è stata dichiarata conclusa nel 2007.
- Ad alcuni di questi semiprimi fu associato un premio in denaro da destinare a chi ne avesse trovato per primo la fattorizzazione.
- La cifra nel nome dei primi numeri RSA generati, da RSA-100 a RSA-500, indica il numero delle cifre decimali; successivamente, a partire da RSA-576, quello indicato è il numero di cifre binarie.
- Il numero RSA-617 rappresenta un'eccezione, in quanto creato prima del cambiamento nel sistema di numerazione.

RSA challenge (2)

- Il primo dei numeri RSA fu fattorizzato in pochi giorni, ma per la maggior parte degli altri numeri il problema è ancora aperto e per molti di loro ci si aspetta che rimanga aperto ancora a lungo.
- Fino a giugno 2010, sono stati fattorizzati 15 dei 54 numeri RSA, ossia tutti i 12 più piccoli (da RSA-100 a RSA-180), oltre che RSA-640, RSA-768 (232 cifre decimali, nel 2009) e RSA-200.
- RSA-768 ha richiesto la raccolta di oltre 64 miliardi di relazioni e la soluzione di una matrice 192.796.550×192.795.550.

RSA 129

- Non faceva propriamente parte della sfida RSA.
- È stato fattorizzato nell'aprile 1994 da un team diretto da D. Atkins, M. Gradd, A. K. Lenstra e P. Lyland, usando approssimativamente 1600 computer con la collaborazione di circa 600 volontari connessi tramite Internet.
- Un premio di $100 è stato assegnato dalla RSA Security per la sua fattorizzazione, il quale è stato donato alla Free Software Foundation.
- La fattorizzazione è stata calcolata usando l'algoritmo del crivello quadratico.

RSA 129 (2)

- La fattorizzazione di RSA-129 è la seguente:
- RSA-129 =
  114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541

  = 3490529510847650949147849619903898133417764638493387843990820577

  × 327691329932667095499619881908344614131776429679929425397982885333

- La sfida per la fattorizzazione includeva un messaggio da decriptare con RSA-129. Una volta decriptato usando la fattorizzazione il messaggio trovato fu “the magic words are squeamish ossifrage” (le parole magiche sono gipeto ipersensibile).

Crittosistema di ElGamal

- Sistema proposto nel 1985 da **Taher ElGamal**
- La sua sicurezza si basa sulla difficoltà di calcolare logaritmi discreti.
- Insieme dei possibili testi in chiaro = interi (mod p).
- Insieme dei possibili testi cifrati = coppie di interi (r, t) (mod p).
- Ne segue che il testo in chiaro risulta espanso nel testo cifrato.
- In RSA invece i due insiemi coincidono (interi mod n).

ElGamal – generazione delle chiavi

- Bob sceglie un primo p grande e una radice primitiva $\alpha \pmod{p}$.
- Bob sceglie un intero a e calcola $\beta \equiv \alpha^a \pmod{p}$.
- Chiave **pubblica** di Bob: (p, α, β)
- Chiave **privata** di Bob: a

ElGamal – cifratura

- Alice vuole inviare un messaggio m a Bob.
- Si assume $0 \leq m < p$, altrimenti si spezza il messaggio in blocchi tali che ciascun blocco corrisponda ad un numero $< p$ e si cifra un blocco alla volta.
- Alice ottiene la chiave pubblica di Bob (p, α, β).
- Alice Sceglie a caso un intero k segreto e calcola:
  - $r \equiv \alpha^k \pmod{p}$
  - $t \equiv \beta^k m \pmod{p}$
- Il testo cifrato è la coppia (r, t).

ElGamal – decifratura

- Bob decifra calcolando:

$$
\begin{aligned}
& tr^{-a} \equiv \beta^k m \left(\alpha^k\right)^{-a} \equiv \\
& \equiv (\alpha^a)^k m \alpha^{-ak} \equiv \\
& \equiv \alpha^{ak} m \alpha^{-ak} \equiv \\
& \equiv \ m \pmod{p}
\end{aligned}
$$

ElGamal – basi della sicurezza

- Se Eva determina *a* viola il sistema.
- Se Eva determina *k* viola il sistema ($m = \beta^{-k} t$).
- $m = 0$ deve essere evitato perché risulta in $t = 0$.
- Per ogni messaggio, è importante usare un *k* diverso.
- Supponiamo che Alice codifichi due messaggi m_1 e m_2 usando lo stesso *k*:
  - $m_1 \rightarrow (r, t_1)$
  - $m_2 \rightarrow (r, t_2)$
- Se Eva trova m_1 può determinare m_2:

$$t_1/m_1 \equiv \beta^k \equiv t_2/m_2 \pmod{p} \quad \rightarrow \quad m_2 \equiv t_2\, m_1/t_1 \pmod{p}.$$

RSA – problema di decisione

- Eva afferma di possedere il testo in chiaro *m* corrispondente ad un testo cifrato *c* con RSA.
- Si può verificare la sua affermazione?
- Sì, è facile, dal momento che *n* ed *e* sono pubblici
- Basta calcolare $m^e \pmod{n}$ e controllare se coincide con *c* o meno

ElGamal – problema di decisione

- Eva afferma di possedere il testo in chiaro m corrispondente ad un testo cifrato (r, t) con ElGamal.
- Si può verificare la sua affermazione?
- Il problema ha difficile soluzione in quanto non si conosce il valore di k usato per la cifratura.
- Questa verifica è difficile quanto il

Problema di decisione di Diffie-Hellman. Sia p un primo e sia α una radice primitiva (mod p). Dati $\alpha^x \pmod p$, $\alpha^y \pmod p$ e $c \neq 0 \pmod p$, decidere se $c \equiv \alpha^{xy} \pmod p$.

ElGamal – differenze con RSA

- Il problema decisionale di RSA è di facile soluzione, mentre quello di ElGamal no.
- Questo perché la cifratura con RSA è deterministica, mentre quella con ElGamal è statistica per via di *k*.
- Il carattere casuale si recupera in RSA con OAEP o protocolli simili.
- Inoltre, in RSA l'insieme dei testi in chiaro e quello dei testi cifrati coincidono, proprietà che facilita l'ottenimento di uno schema di firma a partire dallo schema di cifratura asimmetrica.