03 - Public Key RSA Algorithm

Notazioni

- $\mathbb{Z} = \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$ denota l'isieme dei numeri interi
- $\mathbb{Z}_n = 0, 1, 2, 3, .., n-1$ denota l'isniemi degli interi di modulo n
- $GCD(m, n)$ denota il massimo comunedivisore di m e n
- $\mathbb{Z}_n^*$ denota i numeri primi relativi a n
- $\varphi(n) = |\mathbb{Z}_n^*|$ denota il **Toziente di Eulero**

Alcuni fatti
Se il $GCD(m, n) = 1$ n e m sono numeri primi relativi o *coprimi*, allora:

$$\varphi(nm) = \varphi(n)\varphi(m)$$

Se p e q sono numeri primi allora

- $\varphi(p) = (p-1)$
- $\varphi(pq) = (p-1)(q-1)$

Per definire RSA è necessario definire prima le seguenti operazioni:

- come generare le chiavi
- come criptare: $C(m)$
- come decriptare: $D(m)$

Generare chiavi RSA

Si scelgano due numeri primi molto grandi p e q
Si calcoli $n = p \times q$
Si calcoli $\varphi(n) = (p-1)(n-1)$
Si scelga $1 < e < \varphi(n)$ tale che $GCD(e, \varphi(n)) = 1$ (e e $\varphi(n)$ sono coprimi)
Si calcoli d come l'inverso moltiplicativo di e: $d \times e \mod \varphi(n) = 1$
Chiave pubblica (e, n)
Chiave privata (d, n)

Criptazione

$$C(m) = m^e \mod n$$

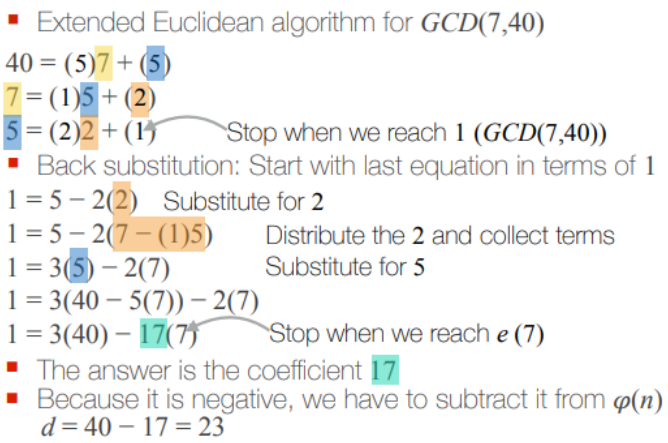
Decriptazione

$$D(c) = c^d \mod n$$

Esempio
Siano $p = 5, q = 11$ (non realistici!!)
Quindi $n = 5 \times 11 = 55$, $\varphi(n) = (5-1)(11-1) = 40$
Sia $e = 7$ (si verifichi che $GCD(e, \varphi(n)) = GCD(7, 40) = 1$)
Si calcoli d come l'inverso moltiplicativo di e:
- $d \times e \mod \varphi(n) = 1$
- $d \times 7 \mod 40 = 1$
d può essere calcolato tramite la formula estesa dell'algortimo di Euclide

- L'algoritmo di Euclide calcola $GCD(e, \varphi(n))$
- La formula Euclidea estesa esprime il $GCD(e, \varphi(n))$ come combinazione lineare tra e e $\varphi(n)$

Formula estesa dell'algoritmo di Euclide



Verifica:
con $d = 23$, $e = 7$
$23 \times 7 \mod 40 = 1 (23 \times 7 = 161 = 40 \times 4 + 1)$
Quindi, le chiavi pubbliche e private sono:
$K[\text{priv}] = (23, 40)$
$K[\text{pub}] = (7, 40)$

Acuni problemi
Come si codificano messaggi plaintext in un intero m tale che $0 < m < n$ (necessario dividere il messaggio in blocchi più piccoli)
Come si può garantire che la criptazione e decriptazione siano uno l'inverso dell'altro ($D(C(m)) = m$)
Quanto è effettivamente sicura l'RSA?
Quanto è efficiente l'RSA?
Come svolgere i vari passaggi dell'algoritmo?

Correttezza, sicurezza e efficienza dell'RSA

Correttezza

Si deve dimostrare che:

$$\forall m : D(C(m)) = m$$

Risultato classico dalla teoria dei numeri
Teorema di Eulero:
Se $GCD(m, n) = 1$ allora $m^{\varphi(n)} \mod n = 1$

Proprietà dell'aritmetica modualare:
- Se $x \mod n = 1$, allora per ogni intero y, si ha $x^y \mod n = 1$
- Se $x \mod n = 0$, allora per ogni intero y, si ha $x^y \mod n = 0$
- $(m^x \mod n)^y = (m^x)^y \mod n$

Sia m un intero che codifica il messaggio originale tale che $0 < m < n$
Per definizione si ha che:
$D(C(m)) = D(m^c \mod n)$
$= (m^e \mod n)^d \mod n$
$= (m^e)^d \mod n$
$= m^{ed} \mod n$

Per definizione si sa che $ed \mod \varphi(n) = 1$
Quindi deve esistere un intero positivo k tale che $ed = k\,\varphi(n) + 1$
Sostituendo si ottiene:
$D(C(m)) = m^{ed} \mod n = m^{k\,\varphi(n)+1} \mod n$
$= m m^{k\,\varphi(n)+1} \mod n$
$= m * 1 = m$

Sicurezza dell'RSA

La sicurezza dell'RSA potrebbe essere cmpromessa tramite:

- Bruteforce:
  - Attacco: prova tutte le possibili chiavi
  - Difesa: usare una chiave abbastanza grande
- Attacco matematico:
  - Fattorizzare n nei suoi fattori primi p, q, si computi $\varphi(n)$ e in fine si computi $d = e^{-1}(\mod \varphi(n))$
  - Si calcoli $\varphi(n)$ senza fattorizzare n, e quindi calcolare $d = e^{-1}(\mod \varphi(n))$
    Entrambi gli approcci soni caratterizzati dalla difficoltà di fattorizzare n

Il problema della fattorizzazione

Nessun teorema o limite minimo
Solo evidenze empiriche sulla sua difficoltà
Nessuna garanzia sul fatto che se se è sicura oggi è sicura anche domani.

Number of decimal digits	Number of bits	Date achieved	MIPS-years	Algorithm
100	332	April 1991	7	Quadratic Sieve
110	365	April 1992	75	Quadratic Sieve
120	398	June 1993	830	Quadratic Sieve
129	428	April 1994	5000	Quadratic Sieve
130	431	April 1996	1000	Generalized number field sieve
140	465	February 1999	2000	Generalized number field sieve
155	512	August 1999	8000	Generalized number field sieve
160	530	April 2003	-	Lattice sieve
174	576	December 2003	-	Lattice sieve
200	663	May 2005	37500	Lattice sieve (18 months using 80 Opteron processors)

- 1GHz Pentium is about a 250-MIPS machine

Efficienza dell'RSA

Come si calcola $(x^2 \mod n)$ efficientemente
x^{32}: $x \to x^2 \to x^4 \to x^8 \to x^{16} \to x^{32}$
5 moltiplicazioni totali perchè $5 = \log_2(32)$

Cosa succede se z non è una potenza di 2:
Da x^y si possono ottenere x^{2y} e x^{2y+1} con al massimo due moltiplicazioni in più:

- $x^{2y} = (x^y)^2 = x^y * x^y$
- $x^{2y+1} = (x^y)^2 = x^{2y} * x = x^y * x^y * x$
  Come si decompone z come combinazione linearre di x^{2y} e x^{2y+1}
- Si supponga che si debba calcolare $1284^{54} \mod 3233$
- Si converta l'esponente 54 in binario 110110_2
- Ora si deve calcolare $1284^{110110_2} \mod 3233$
- Per semplicità si ignori *mod* e si consideri l'esponente un bit alla volta
  $1284^{1_2} = 1284$
  $1284^{11_2} = 1284^2 \cdot 1284$
  $1284^{110_2} = (1284^2 \cdot 1284)^2$
  $1284^{1101_2} = ((1284^2 \cdot 1284)^2)^2 \cdot 1284$
  $\dots$
  1284^{110110_2}

Si può calcolare x^y facendo solo $2[\log_2(y)]$ moltiplicazioni.
Proprietà dell'aritmetica modulare:
$(a \times b) \mod n = [(a \mod n) \times (b \mod n)] \mod n$
Quindi ogni risultato intermedio è riducibile dal modulo n
Esempio *completo*:
$1284^{1_2} = (1284) \mod 3233$
$1284^{112} = (1284^2 \cdot 1284) \mod 3233$
$1284^{110_2} = (1284^2 \cdot 1284)^2 \mod 3233$
$1284^{1101_2} = (((1284^2 \cdot 1284)^2)^2 \cdot 1284) \mod 3233$

Generazione di numeri primi grandi

Probabilità di scegliere casualmente un numero primo:

$$Pr(n\text{preso casualmente sia primo} \sim \frac{1}{\log(n)}$$

Se n ha 10 cifre, allora $Pr(n \text{ primo}) \sim \frac{1}{23}$, con 100 cifre sarebbe $\sim \frac{1}{230}$
Queste probabilità sono numeri troppo bassi per generare numeri casuali che possano essere usati come primi.
Bisogna creare una funzione di test **p_test(n)**, che possa essere usata per escludere numeri non primi e generare nuovi numeri cauali finchè non se ne trovi uno primo.

Come implementare p_test(n) affinchè ritorni True se n è primo e false altrimenti:

- metodo semplice: verificare che tutti i numeri da $2 \text{ a } n-1$ non siano divisibili per n
- Si possono controllare solo tutti i numeri fino a $\sqrt{n}$
- Complessità $O(\sqrt{n})$ ovvero $O(2^{\frac{m}{2}})$ dove $m = \log(n)$ è la dimensione dei bit in input

Esistono due algoritmi polinomiali con complessità $O(\log n)^4)$ e $O((\log n)^6)$ per testare se un numero è primo.
In ogni caso il costo computazionale è troppo grande per cui ci si affida a **test probabilistici**

Il teorema di Fermat
se n è primo, allora per ogni intero a, $0 < a < n$

$$a^{n-1} \mod n = 1$$

Qual è la probabilità che il teorema di fermat **regga** anche quando n **non è primo**?

- Sia n un **numero intero molto grande** (più di 100 cifre)
- per ogni numero positivo casuale a minore di n
  - $Pr(n \text{ non primo}) \text{ e } (a^{n-1} \mod n = 1) \simeq 10^{-13}$

```
def p_test(n):
      a = rand() mod n
      x = a^(n-1) mod n
      if x == 1:
            return “true”
      else:
            return “false”
```

Se ritona falso allora n non è primo
Se ritorna true allora n potrebbe non essere primo con una probabilità uguale 10^{-13}
La probabilità è molto bassa ma non accettabili
Idea: ripetere il test k volte con diversi valori di a ogni volta

```
def p_test(n, k):
      repeat k times:
            a = rand() mod n
            x = a^(n-1) mod n
            if x != 1:
                  return “false”
      return “true”
```

La probabilità che n non sia primo si riduce a $(10^{-13})^k$
In media si eseguono test $\frac{\log(n)}{2}$ volte
Es: per un numero casuale di 200-bit si testano $\log(2^{200})/2 = 70$ volte

Altri schemi per chiavi pubbliche
Benchè sia relativamente facile calcolare esponeziali moldulo un numero primo, è difficile calcolare **logaritmi discreti**
Il logaritmo discreto di **g** base **b** è l'intero **k** che solve l'equazione $b^k = g$ dove b e g sono elementi di un gruppo finito.
Esempi di schemi:

- Diffie-Hellman

- El Gamal