The RSA Cryptosystem

Gianluca Dini
Dept. of Ingegneria dell'Informazione
University of Pisa
Email: gianluca.dini@.unipi.it
Version: 2021-03-29

The RSA Cryptosystem

BASICS

RSA in a nutshell

- Rivest-Shamir-Adleman, 1978
  - Rivest, R.; Shamir, A.; Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM 21 (2): 120–126, February 1978.
- The most widely used asymmetric crypto-system
  - Patented until 2000 in US
- Many applications
  - Encryption of small pieces (e.g., key transport)
  - Digital Signatures
- Underlying one-way function
  - integer factorization problem

RSA one-way function

- One-way function y = f(x)
  - y = f(x) is easy
  - x = f^{-1}(y) is hard
- RSA one-way function
  - Multiplication is easy
  - Factoring is hard

NON ESISTE UN ALGORITMO POLINOMIALE

Mathematical setting

- RSA encryption and decryption is done in the integer ring $\mathbb{Z}_n$
  - PT and CT are elements in $\mathbb{Z}_n = \{0, 1, \ldots, n - 1\}$
  - Modular computation plays a central role

Key Generation

G, E, D

1024 Bit

DISPARI, UNICO NUMERO PARI PRIMO È 2
CHE È TROPPO PICCOLO

1. Choose two large, distinct primes p, q
2. Compute modulus n = p × q
   NUMERI IN $\mathbb{Z}_n$ CHE SONO COPRIMI RISPETTO A n
3. Compute Euler's Phi function $\phi(n) = \underbrace{(p-1)}_{PARI} \times (q-1)$ DATO CHE n=p×q ALLORA
4. Randomly select the public (encryption) exponent e,
   1 < e < ϕ(n), s.t. (SUCH THAT) gcd(e, ϕ(n)) = 1 , e DEVE ESSERE PRIMO RELATIVAMENTE A ϕ(n)
   3, 17, 65, $2^{16}+1$
5. Compute the unique private (decryption) exponent
   d, 1 < d < ϕ, such that e·d ≡ 1 (mod ϕ) → SOLO d UNIVOCA
   $d = e^{-1} \bmod \phi$
6. Private key = (d, n), Public key = (e, n) GRAZIE A

$e \cdot d = 1 + t \cdot \phi$ ⇒ RISOLVIBILE CON 1 SOLA SOLUZIONE

- Φ(n) is the Euler's Phi Functions which denotes the number of integers in Z_m relatively prime to n.
- Condition gcd(e, φ(n)) = 1 ensures that the inverse of *e* mod φ(n)exists.
- Two parts of the algorithm are not trivial, namely step 1 and step 3 and 4.
- Step 4 is fundamental for the prrof of RSA consistency.

RSA Key Generation

- Comments
  - Primes p and q are 100÷200 decimal digits
    - Nowadays, p and q are 1024 bit
  - Condition gcd(e, Φ(n)) = 1 guarantees that d exists and is unique
  - At the end of key generation, p and q must be deleted
  - Two parts of the algorithm are nontrivial:
    - Step 1
    - Steps 4-5
      - Step 5 is crucial for RSA correctness

- Φ(n) is the Euler's Phi Functions which denotes the number of integers in Z_m relatively prime to n.
- Condition gcd(e, φ(n)) = 1 ensures that the inverse of *e* mod φ(n)exists.
- Two parts of the algorithm are not trivial, namely step 1 and step 3 and 4.
- Step 4 is fundamental for the prrof of RSA consistency.

RSA Encryption and Decryption Algorithm

- Encryption algorithm: to generate the ciphertext y from the plaintext x $\in$ [0, n – 1] → DEVE ESSERE CONSIDERATO UN NUMERO, NON UNA SEQUENZA DI BIT
  $\mathbb{Z}_n$
  - Obtain receiver's authentic public key (n, e)
  - Compute $\boxed{y = x^e \bmod n}$ ENCRIPTION $x^e = q \cdot n + R$ → RESIDUO CIPHERTEXT

- Decryption algorithm: to obtain the plaintext x from the ciphertext y $\in$ [0, n - 1]
  - Compute $x = y^d \bmod n$
    → PRIVATO!

Example with artificially small numbers

Key generation

IN REALTÀ NON SI POSSONO USARE PERCHÉ SONO TROPPO PICCOLI

- Let p = 47 e q = 71

  n = p × q = 3337

  ϕ= (p-1) × (q-1)= 46 × 70 = 3220
- Let e = 79

  ed = 1 mod ϕ

  79 × d = 1 mod 3220

  d = 1019

Encryption

APPLICO UNA SORTA DI ECB

Let m = 9666683 MESSAGE → SEQUENZA DI BIT INTERPRETATA COME UN NUMERO!

Divide m into blocks $m_i < n$

$m_1 = 966$; $m_2 = 668$; $m_3 = 3$

Compute → CIFRO SEPARATAMENTE!

$c_1 = 966^{79}$ mod 3337 = 2276

$c_2 = 668^{79}$ mod 3337 = 2423

$c_3 = 3^{79}$ mod 3337 = 158

$c = c_1c_2c_3$ = 2276 2423 158

Decryption

$m_1 = 2276^{1019}$ mod 3337 = 966

$m_2 = 2423^{1019}$ mod 3337 = 668

$m_3 = 158^{1019}$ mod 3337 = 3

m = 966 668 3

The RSA Cryptosystem

PROOF OF RSA

RSA consistency: proof

- We need to prove that decryption is the inverse operation of encryption, $D_{privK}(E_{pubK}(x)) = x$ RELAZIONE TRA CHIAVE PUBBLICA E PRIVATA
- Step 1
  - $d \cdot e = 1 \bmod \Phi(n)$
  - By definition of mod operator $\boxed{d \cdot e = 1 + t \cdot \Phi(n)}$ for some integer t COMPORTA CHE
  - Insert this expression in the decryption: $y^d \equiv x^{ed} \equiv x^{1+t \cdot \Phi(n)} \equiv x \cdot x^{t \cdot \Phi(n)} \equiv x \cdot (x^{\Phi(n)})^t \bmod n$
- Step 2: prove that $x \equiv x \cdot (x^{\Phi(n)})^t \bmod n$
  - Recall
    - Euler's Theorem: if gcd(x, n) = 1 then $1 \equiv x^{\Phi(n)} \bmod n$
    - Minor generalization $1 \equiv 1^t \equiv (x^{\Phi(n)})^t \bmod n$

$x^{\Phi} = 1 + t \cdot n$ TEOREMA DI EULERO

<u>6.3 LIBRO</u>

$\Phi(n)$ is the number of integers in $\mathbb{Z}_n$ relatively prime with respect to n. Of course $\Phi(p) = p - 1$. See Section 6.3 of Paar's book.

RSA consistency: proof

UNIVERSITÀ DI PISA

$x \in [0, n-1]$

- Step 2
  - case 1: gcd(x, n) = 1
    - Euler's theorem holds
    - $x \cdot (x^{\Phi(n)})^t \equiv x \cdot 1 \equiv x \bmod n$ Q.E.D.

  PRIMI

  $n = p \cdot q$, $x < n$

  - case 2: gcd(x, n) $\neq$ 1
    - Since p and q are primes (and $x < n$) then either $x = r \cdot p$ or $x = s \cdot q$ with $r < p$ and $s < q$
    - Assume $x = r \cdot p$ then gcd(x, q) = 1
    - Euler's Theorem holds in this form $1 \equiv (x^{\Phi(n)})^t \bmod q$
      - Proof: $(x^{\Phi(n)})^t \equiv (x^{(p-1)(q-1)})^t \equiv ((x^{\Phi(q)})^t)^{p-1} \equiv 1^{(p-1)} \equiv 1 \bmod q$
    - $(x^{\Phi(n)})^t = 1 + u \cdot q$, for some integer u
    - $x \cdot (x^{\Phi(n)})^t = x + x \cdot u \cdot q = x + (r \cdot p) \cdot u \cdot q = x + r \cdot u \cdot (q \cdot p) = x + r \cdot u \cdot n$
    - $x \cdot (x^{\Phi(n)})^t \equiv x \bmod n$ Q.E.D.

DIMOSTRAZIONE NON RICHIESTA.

$q - 1$, SE q È PRIMO E CONSIDERO $\mathbb{Z}_q$ = TUTTI I NUMERI PRIMI RISPETTO A q IN $\mathbb{Z}_q$.

$\phi(q) = q - 1$

RSA encryption and decryption

- Comments
  - RSA proof is based on Euler's theorem
  - The proof becomes simpler by using the Chinese Remainder Theorem

The RSA Cryptosystem

PERFORMANCE

RSA

- RSA algorithms for key generation, encryption and decryption are “easy” ~ ALGORITMO POLINOMIALE!
- They involve the following operations
  - Discrete exponentiation
  - Generation of large primes
  - Solving diophantine equations

$e \cdot d = 1 \bmod (\Phi) \iff e \cdot d = 1 + t\Phi$

Computation of e and d (refined)

- Select $e \in (1, \varphi(n))$
- Apply EEA with input parameters n and e and obtain the relationship

  RISOLTA IN MODO EFFICIENTE DA EXTENDED EULER'S ALGORITHM

  – $\gcd(\Phi(n), e) = s \cdot \varphi(n) + t \cdot e$ (Diophantine equation)
    - If $\underbrace{\gcd(e, \varphi(n)) = 1}$ then

      COPRIMI

      – Parameter e is a valid public key
      – Unknown $t = e^{-1} \bmod \Phi(n)$, i.e., $t = d \bmod \Phi(n)$
    - If $\gcd(e, \Phi(n)) \neq 1$ then
      – Select another value for e and repeat the process

  – Efficiency
    - Number of steps is close to the number of digit of the input parameter

See Extended Euclid Algorithm in Section 6.3 of Paar's book.

Finding large primes

UNIVERSITÀ DI PISA

- Algorithm

```
repeat
     p ← RNG(x);     // secure random generator
until isPrime(p);    // primality test
```

QUANTE VOLTE DEVO FARE QUESTO LOOP?

QUANTO COSTA CONTROLLARE SE UN NUMERO È PRIMO?

- Comment
  - RNG must be secure, i.e., unpredictable
- Problems
  - How many random numbers we must test before we have a prime?
  - How fast can we check whether a random integer is prime?
  - It turns out that both steps are reasonably fast

.

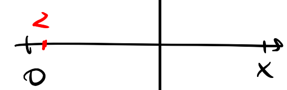
PRIMALITY TESTS

- Determining whether a number is prime or composite is **a much simpler problem** than factoring it.
- Typically a true primality test is more computationally expensive than a probabilistic one.
- A probabilistic primality test returns “composite” with certainty and “prime” with “high probability”. Therefore, before applying a true test to a candidate *n* the candidate should pass through a probabilistic test.

How common are primes?

UNIVERSITÀ DI PISA

- Let Pi(x) be the number of primes less than x
  $\lim_{x \to +\infty} Pi(x) = \frac{x}{\ln x}$
- Prime Numbers Theorem
  - For a very large x, Pi(x) tends to x/ln(x)
    TUTTI GLI ALTRI PRIMI SONO DISPARI
  - Furthermore, primes are distributed approximately uniformly over [2, x]



- Probability for a random odd p to be prime ≈ 2/ln(p)
  DI TROVARE UN PRIMO
  $P_R = \frac{\frac{x}{\ln x}}{\frac{x}{2}}$
  P DI UN DISPARI
  PROBABILITÀ DI TROVARE UN DISPARI
  - As we test only odd numbers
    $$P = x/\ln(\overset{x}{\cancel{p}})/\overset{x}{\cancel{p}}/2 = 2/\ln(\overset{x}{\cancel{p}})$$
  - Expected number of trials to find a prime p < x is ln(x)/2
    $\# TRIALS \approx \frac{\ln(x)}{2}$ LOGARITMICO!
    VA BENE, EFFICIENTE

PRIME NUMBERS THEOREM. Let Pi(x) be the number of primes less than x. The Prime Numbers Theorem tells us that for a very large x, Pi(x) tends to x/ln$x$. Furthermore, these primes are distributed approximately uniformly over [2, x].

The random generator generates odd numbers less than x. Odd numbers are x/2. It follows that the probability of finding a prime less than x is equal to (x/ln x)/(x/2) = 2/ln x. In other words, it is necessary to generate and test (ln x)/2 odd numbers before finding a prime.

EXAMPLE. Let us consider 1024-bit modulus RSA. It follows that p and q are 512-bits. The probability of generating one of these primes is P = 2/(512 ln 2) ≈ 1/177. In other words, we expect to test 177 odd numbers before we find one that is prime.

Primality tests

UNIVERSITÀ DI PISA

- Primality tests are computationally much easier than factorization
- Practical primality tests are probabilistic
  - At the question: “is p* prime?” they answer
    - p* is composed which is always a true statement
    - p* is prime, which is only true with a high probability
- Primality test
  - Fermat test
  - Miller-Rabin test

COMPLESSITA' NON RILEVANTE

True Primality proving algorithms are generally more computationally intensive than the probabilistic primality tests. Consequently, before applying one of these tests to a candidate prime n, the candidate should be subjected to a probabilistic primality test such as Miller-Rabin

Modular ops - complexity

k BIT

- Bit complexity of basic operations in $\mathbb{Z}_n$
  - Let n be on k bits ($n < 2^k$)
  - Let a and b be two integers in $\mathbb{Z}_n$ (on k-bits) → a, b DUE NUMERI RAPPRESENTABILI IN k BIT
    - Addition a + b can be done in time $O(k)$ LINEARE
    - Subtraction a – b can be done in time $O(k)$ "
    - Multiplication a × b can be done in $O(k^2)$ IL RESTO k^2
    - Division $b \times a^{-1}$ can be done in time $O(k^2)$
    - Inverse a^{-1} can be done in $O(k)$ EEA
    - Modular exponentiation a^n can be done in $O(k^3)$

Fast exponentiation

UNIVERSITÀ DI PISA

- How many multiplications to compute 2^{20}?
- Grade-school Algorithm requires
  - $2 \times 2 \times 2 \times \ldots \times 2$ => 19 multiplications $2^n \Rightarrow \#\,mul = O(n)$
- Square-and-Multiply Algorithm
  - $((2 \times (2^2)^2)^2)^2$ => 1 multiplications + 4 squares => 5 multiplications

DIFFICILE

$O(2^k)$

$(k-1)$ moltiplicazioni w x size

AL PIU' 2k MOLTIPLICAZIONI

FACILE E LINEARE $O(n)$

$$2^{20} = (2^{10})^2 = ((2^5)^2)^2 = ((2 \cdot 2^4)^2)^2 = ((2 \times (2^2)^2)^2)^2$$

$\boxed{a^x \bmod n}$ con numeri su k bit $[0, 2^k-1] \to$ ENCRYPTION $y = x^e \bmod n$

DECRYPTION $x = y^d \bmod n$

d è GRANDE → ALTRIMENTI POSSO USARE BRUTE FORCE

SIZEOF(mod n) = 2048 bit

In order to get convinced that exponentiation is efficient, let us consider the following example. How many multiplications do you need to compute 2^{20}?

If you use the grade-school algorithm you need 19 multiplications 2 x 2 x .... x 2. However, you can obtain the same result in a much faster way, namely in just five multiplications! We exploit the property that $2^{2x} = (2^x)^2$. So, applying this property to the quiz we get the following $2^{20} = (2^{10})^2 = ((2^5)^2)^2 = ((2\ 2^4)^2)^2 = ((2\ (2^2)^2)^2)^2 = ((2\ (2^2)^2)^2)^2$. This called the square-and-multiply algorithm.

Fast exponentiation

UNIVERSITÀ DI PISA

- RSA computes modular exponentiation
  - $a^x \bmod n$, where n is on k bits (i.e., $n \leq 2^k$)
- Grade-school Algorithm
  - requires $(x - 1)$ modular multiplications
    - If *x* is as large as n, which is exponentially large in k, the Grade-school Algorithm is inefficient
- Square-and-multiply Algorithm
  - requires up to 2k multiplications ($2 \times \log_2 x$)
  - Overall, can be done in $O(k^3)$

$\boxed{k = \log x}$

$O(n)$ moltiplicazioni nell'ordine di $O(n^2)$

Mar-21 The RSA Cryptosystem 22

Private parameter d has the same order of magnitude as Φ and, therefore, as n, in order to discourage brute force attack against the private key. Therefore d is in the order of 2^k. This means that the Grade-school requires 2^k multiplications, each costing $O(k^2)$. Therefore, the Grade-school algorithm is inefficient. In contrast, Square-and-Multiply requires 2k multiplications and therefore it is $O(k^3)$. So, it is efficient.

Fast exponentiation

- Square and multiply
  - Exponentiation by repeated squaring and multiplication
  - The exponentiation a^x mod n requires at most
    - $\log_2(x)$ multiplications and
    - $\log_2(x)$ squares
  - Proof
    - See next slide

Modulo reduction is performed at each multiplication-and-squaring round in order to keep the intermediate results small.

Average (#MUL + #SQ) Let us consider an exponentiation with a *k*-bit exponent. We have computed the maximum number of MUL and SQ. However, we can observe that #SQ is equal to the number of bits whereas #MUL depends on the value of the exponent (more precisely on its Hamming weight, i.e., the number of ones in its binary representation). It follows that on average we have #SQ = *k*, #MUL = 0.5*k* which, in total, makes **#MUL + #SQ = 1.5*k*.**

Example. If we consider **a k = 1024-bit** exponent we have that **Squaring-and-Multiplying requires, on average, #OPS = 1.5 x 1024 = 1536 multiplications**, whereas the **Grade-School Algorithm requires 2^{1024} multiplications**. However, remember that each SQ and MUL operates on 1024-bit numbers. This means that the number of multiplication in a CPU is much higher than 1536, but it is certainly doable on modern computers.

Fast exponentiation

$$a^x \bmod n = a^{\left(x_{k-1}2^{k-1}+x_{k-2}2^{k-2}+\cdots+x_2 2^2+x_1 2+x_0\right)} \bmod n \equiv$$

$$a^{x_{k-1}2^{k-1}} a^{x_{k-2}2^{k-2}} \cdots a^{x_2 2^2} a^{x_1 2} a^{x_0} \bmod n \equiv$$

$$\left(a^{x_{k-1}2^{k-2}} a^{x_{k-2}2^{k-3}} \cdots a^{x_2 2} a^{x_1}\right)^2 a^{x_0} \bmod n \equiv$$

$$\left(\left(a^{x_{k-1}2^{k-3}} a^{x_{k-2}2^{k-4}} \cdots a^{x_2}\right)^2 a^{x_1}\right)^2 a^{x_0} \bmod n \equiv$$

$$\ldots$$

$$\left(\left(\left(\left(a^{x_{k-1}}\right)^2 a^{x_{k-2}}\right)^2 \cdots a^{x_2}\right)^2 a^{x_1}\right)^2 a^{x_0} \bmod n$$

ALGORITHM

$c \leftarrow 1$
for (i = k-1; i >= 0; i --) {
  $c \leftarrow c^2$ mod n;
  if (x_i == 1)
    $c \leftarrow c \times a$ mod n;
}

COMMENT

- always k square operations
- at most k multiplications
  - *equal to the number of 1 in the binary representation of x*
- Modulo reduction is performed at each round in order to keep the intermediate results small.

Fast exponentiation – exercise

- Compute $r = a^{20}$
  - $x = 20 = 10100_2$
  - Step 0
    - $r_0 = a^1$
  - Step 1
    - $r_1 = (a^1)^2 = a^2 = a^{[10]_2}$
  - Step 2
    - $r_2 = (r_1)^2 = a^4 = a^{[100]_2}$
    - $r_2 = r_2 \cdot a = x^5 = a^{[101]_2}$
  - Step 3
    - $r3 = (r_2)^2 = a^{10} = a^{[1010]_2}$
  - Step 4
    - $r_4 = (r_3)^2 = a^{20} = a^{[10100]_2}$

Fast exponentiation

- Let k = 1024
- #MUL in the Grade-School Algorithm
  - #MUL = 2^{1024} multiplications
- #Ops in the Square-and-Multiply Algorithm
  - #SQ = k
  - #MUL = #(1's in the binary representation)
    - On average #MUL = 0.5K
  - #Ops = 1.5k = 1536 multiplications
  - Each multiplication is on 1024 bits

$1536 \ll 2^{1024}$

MOLTIPLICAZIONI PER COME INTEGERS, NON FATTIBILE DA CPU

Mar-21 The RSA Cryptosystem 26

AVERAGE (#MUL + #SQ).

Let us consider an exponentiation with a k-bit exponent. We have computed the maximum number of MUL and SQ. However, we can observe that #SQ is equal to the number of bits whereas #MUL depends on the value of the exponent (more precisely on its Hamming weight, i.e., the number of ones in its binary representation). It follows that on average we have #SQ = k, #MUL = 0.5k which, in total, makes #MUL + #SQ = 1.5k.

EXAMPLE.

If we consider a k = 1024-bit exponent we have that Squaring-and-Multiplying requires, on average, #OPS = 1.5 x 1024 = 1536 multiplications, whereas the Grade-School Algorithm requires 2^{1024} multiplications. However, remember that SQ/MUL operates on 1024-bit numbers. This means that the number of multiplication in a CPU is much higher than 1536, but it is certainly doable on modern computers.

RSA fast encryption with short public exponent

- RSA ops with public exponent e can be speeded-up
  - Encryption $y = x^e \bmod n$
  - Digital signature verification
- The public key e can be chosen to be a very small value
  - e = 3 11 #MUL + #SQ = 2
  - e = 17 10001 #MUL + #SQ = 5
  - e = 2^{16}+1 $\underbrace{10..01}_{16}$ #MUL + #SQ = 17
  - RSA is still secure

RAPPRESENTAZIONE BINARIA

MINIMO NUMERO DI 1 NELLA RAPPRESENTAZIONE

POSSIAMO SCEGLIERE e COME VOGLIAMO BASTA CHE $gcd(e, \phi) = 1$

SE SCELGO UN NUMERO PRIMO E' SICURAMENTE COPRIMO AD OGNI ALTRO NUMERO

RSA decryption

$x = y^d \bmod n$

- Assume a 2048-bit modulus (n) and a 32-bit CPU
- Decryption computing overhead
  - On average #MUL+#SQ = 1.5 × ~~1024~~ 2048 = 3072 long multiplications each of which involves 2048-bit operands
  - Single long-number multiplication
    - Each operand requires 2048/32 = 64 registers
    - Each long-number multiplication requires 64^2 = 4096 integer multiplications → CPU moltiplicatrice
    - Modulo reduction requires 64^2 = 4096 integer multiplications
    - In total 4096 + 4096 = 8192 integer multiplications for a single long multiplication
  - In total, 3072 × 8192 = 25.165.824 integer multiplications

  ↳ SA PRENDO DI NON AVER HARDWARE SPECIALIZED COMPONENTS

RSA decryption

- ‘70s-’80s: only hardware implementation
- Today, an RSA decryption takes ≈100 μs on high-speed hw
- End ‘80s, software implementation becomes possible
- Today, 2048-bit RSA takes ≈10 ms on a 2 GHz CPU
  - Throughput = 2048 × 100 = 204.800 bit/s
  - ≈ 3 orders of magnitude slower than symmetric encryption

RSA Fast decryption

- There is no easy way to accelerate RSA when the private exponent d is involved
  - sizeof(d) = sizeof(n) to discourage brute force attack
    - It can be shown that sizeof(d) ≥ 0.3 sizeof(n)
- One possible approach is based on the Chinese Remainder Theorem (CRT)
  - We do not prove the theorem
  - We just apply it

Fast RSA decryption by CRT

UNIVERSITÀ DI PISA

- Problem
  - Compute $y \equiv x^d \pmod n$ efficiently

  k BIT → $O(k^3)$

- The method
  1. Transformation of the problem in the CRT domain
     1. Compute $x_p \equiv x \pmod p$
     2. Compute $x_q \equiv x \pmod q$
  2. Exponentiation in the CRT domain
     1. $y_p \equiv x_p^{d_p} \bmod p$, where $d_p \equiv d \bmod (p-1)$ ← $O\left(\frac{k}{2}\right)^3 = \frac{k^3}{8}$
     2. $y_q \equiv x_q^{d_q} \bmod q$, where $d_q \equiv d \bmod (q-1)$ ← $O\left(\frac{k}{2}\right)^3 = \frac{k^3}{8}$

VANTAGGIO, $\overset{k}{n} = \overset{k/2}{p} \times \overset{k/2}{q}$

$$\frac{k^3}{8} + \frac{k^3}{8} = \frac{k^3}{4}$$

SPEEDUP = 4

1) SIDE EFFECT = OPERO MODULO p E MODULO q

p e q DEVONO ESSERE CONSERVATI DA QUALCHE PARTE E NON POSSONO ESSERE CANCELLATI

↳ VANNO PROTETTI

Fast RSA decryption by CRT

- The method (cont.ed)
  3. Inverse transformation in the problem domain
     1. $y \equiv [q \cdot c_p]y_p + [p \cdot c_q]y_q \bmod n$ where
        - $c_p \equiv q^{-1} \bmod p$ and
        - $c_q \equiv p^{-1} \bmod q$

POSSONO ESSERE PRE - CALCOLATI

Fast RSA decryption by CRT

- Comments
  - With reference to step 2, as sizeof(p) = sizeof(q), d_p, d_q, y_p, y_q have about half the bit length of n
    - This leads to a speedup = 4
  - With reference to step 3, expressions in square brackets can be precomputed
    - Then, the reverse transformation requires two modular multiplications and one modular addition

Fast RSA decryption by CRT

- Complexity of CRT-based RSA decryption
  - Step 1 and step 3 are negligible
  - Step 2
    - Let n length is t bits, then all quantities in step 2 are on t/2 bits
    - By applying the Square-and-multiply algorithm
      - #SQ+#MUL = 2 × (1.5 t/2) = 1.5 t
        - » The #operations is the same as without CRT, however, each operation involve t/2-bit operands instead of t-bit operand so its time is $(t/2)^2$
      - As multiplication complexity is quadratic, the total speed up is a factor of 4
- The method is subject to fault-injection attack

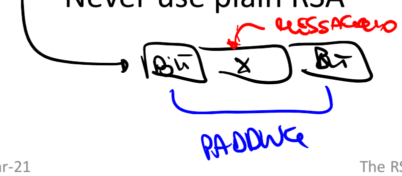
Richiede di interagire risultato con l'encryption

The RSA Cryptosystem

RSA IN PRACTICE

RSA in practice

- Schoolbook/plain RSA is insecure
  - RSA is deterministic
    - A given pt is always mapped into a specific ct
  - PT values 0 and 1 produce CT equal to 0 and 1
  - Small exponent and small pt might be subject to attacks
  - RSA is malleable
- Padding is a solution to all these problems
  - Never use plain RSA



RSA malleability

- Malleability
  - A crypto scheme is said to be malleable if the attacker is capable of transforming the ciphertext into another ciphertext which leads to a known transformation of the plaintext
    - The attacker does not decrypt the ciphertext but (s)he is able to manipulate the plaintext in a predictable manner

RSA Malleability

- The sender
  - Transmits $y = x^e \bmod n$
- The adversary
  - Intercepts y
  - Chooses s s.t. $\gcd(s, n) = 1$
  - Computes and forwards $y' = s^e \cdot y \bmod n$
- The receiver
  - Decrypts y', $x' = y'^d = (s^e \cdot y)^d = s^{ed} \cdot y^d = s \cdot x \bmod n$
    - The attacker manages to multiply x by s

RSA Padding

- Padding intuition
  - It embeds a random structure into the plaintext before encryption
- Padding in RSA
  - Optimal Asymmetric Encryption Padding (OAEP)
    - Specified and standardized in PKCS#1 (Public Key Cryptography Standard #1)

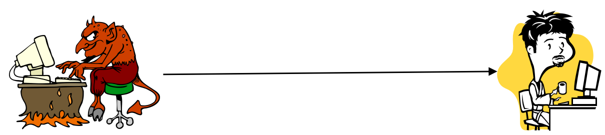
An intuitive but inefficient example of padding is x || x.

RSA malleability

- More in general, RSA malleability descends from the homomorphic property
  - Let x_1 and x_2 two plaintext messages
  - Let y_1 and y_2 their respective encryptions
  - Then, $y \equiv (x_1 \cdot x_2)^e \equiv {x_1}^e {x_2}^e \equiv y_1 \cdot y_2 \bmod n$
  - That is, the CT of the product is the product of the CTs

Adaptive chosen-ciphertext attack

- The problem
  - Bob decrypts any ciphertext except a given ciphertext y
  - The attacker wants to determine the plaintext corresponding to y

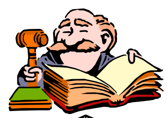


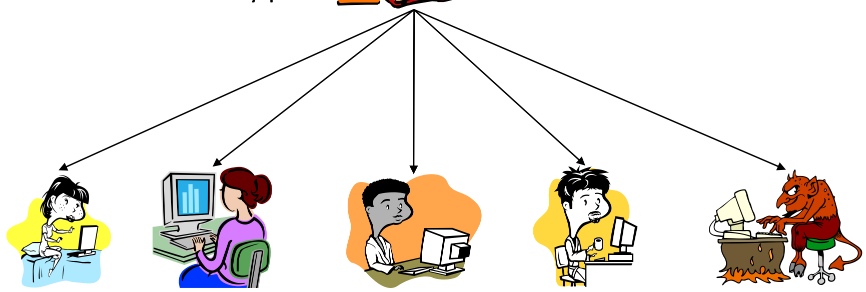
Adaptive chosen-ciphertext attack

- The attack
  - The adversary selects an integer s, s.t. $\gcd(s, n) = 1$, and sends Bob the quantity $y' \equiv s^e \cdot y \bmod n$
  - Upon receiving y', as $y' \neq y$, Bob decrypts y', producing $x' \equiv s \cdot x \bmod n$, and returns x' to the adversary
  - The adversary determines x, by computing $x \equiv x' \cdot s^{-1} \bmod n$
- Countermeasure
  - The attack can be contrasted by using padding
  - Bob returns x' iff it has a structure coherent with padding

Common modulus attack

The server uses a
common modulus n for
all key pairs





(n, e_1) (n, e_2) (n, e_3) (n, e_4) (n, e_5)

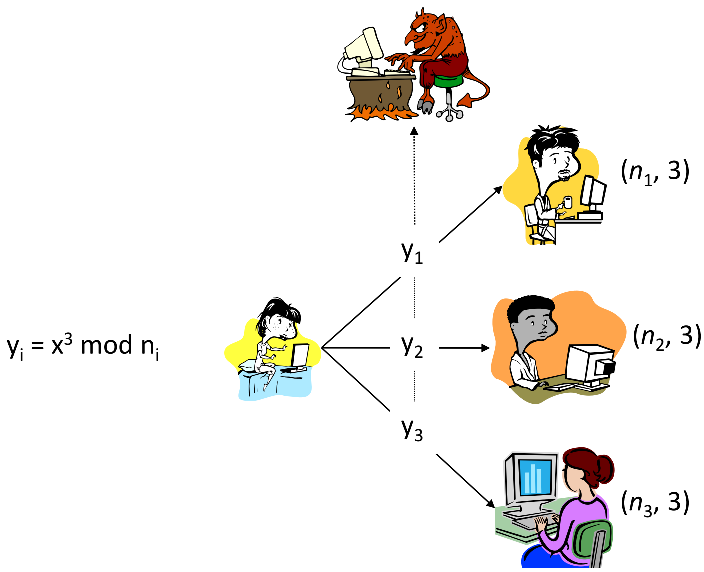
Mr Lou Cipher can efficiently factor n from d_5
(FACT 1) and then compute all d's

Cinese Remainder Theorem

- CHINESE REMAINDER THEOREM. If the integers $n_1, n_2, \ldots, n_k$ are pairwise relatively prime, then the system of simultaneous congruences
  - $x \equiv a_1 \pmod{n_1}$
  - $x \equiv a_2 \pmod{n_2}$
  - ...
  - $x \equiv a_k \pmod{n_k}$

  has a unique solution modulo $n = n_1 n_2 \cdots n_k$.
- GAUSS'S ALGORITHM. The solution x to the simultaneous congruences in the Chinese remainder theorem (Fact 2.120) may be computed as $x = \sum_{i=1}^{k} a_i N_i M_i \bmod \mathrm{n}$ where $N_i = n/n_i \bmod n_i$ and $M_i = {N_i}^{-1} \bmod n$
- These computations can be performed in $O((\lg n)^2)$ bit operations.

Low Exponent Attack



Mar-21 The RSA Cryptosystem 45

Proof for a generic value of e.
Assume that moduli are relatively primewith respect to each other. This is higly likely because, otherwise an adversary could factorise them by computing MCD.

For the CRT, there exists a single value $x < n = n_1 n_2 n_3 \ldots n_e$ that solves $x = m^e \bmod n$. As $m < n_i$ by definition, then, for all i, $m^e < n$. Therefore we can calculate m from x by computing the e-th continuous (not modular) square that is «easy».

Therefore a low exponent must be avoide when you want to send the same message to several destinations. A possible solution consists in appending (padding) a different salt for each different destination. So doing x becomes $x_i = x \,||\, salt_i$.

Low Exponent Attack

- If n_i are pairwise coprime, use CRT to compute $z = x^3 \bmod n_1 n_2 n_3$ that solves

$$\begin{cases} z \equiv y_1 \bmod n_1 \\ z \equiv y_2 \bmod n_2 \\ z \equiv y_3 \bmod n_3 \end{cases}$$

- According to RSA encryption definition $x < n_i$ then $x^3 < n_1 n_2 n_3$ and $z = x^3$
- Therefore x is the integer cube root of z
  - Not a modular root then "easy"

RSA in practice

UNIVERSITÀ DI PISA

- Selecting primes p and q
  - p and q should be selected so that factoring n = pq is computationally infeasible, therefore
  - p and q should be sufficiently large and about the same bit lenght (to avoid the elliptic curve factoring algorithm)
  - p – q should be not too small

- **Why p – q should not be small**
- Let us suppose that **|p – q|** is small. Then, **(p + q)/2** is close to **SQRT(n).**
- Let $(p + q)^2/4 - n = (p - q)^2/4$. Notice that i) the right side of the equality is a perfect square; and, ii) **(p+q)/2** is greater than **SQRT(n)**. Therefore, we can search for a number **z** larger than **SQRT(n)** s.t. $z^2 - n$ is equal to a perfect square w^2, i.e, $z^2 - n = w^2$.
- Then, from **z** and **w** we compute **p = z + w** e **q = z – w** and then we verify these values on a ciphertext **c**.
- This attack may be very efficient

- An efficient attack but more complex is possible when **(p – 1)** e **(q – 1)** have large common factors. As both **(p – 1)** and **(p – 1)** are even, the best choice is that **(p – 1)/2** and **(q – 1)/2** are relatively prime.

The RSA Cryptosystem

RSA SECURITY

Attacks

- Protocol attacks
- Mathematical attacks
- Side-channel attacks

Protocol attacks

- Based on malleability of RSA
- Avoidable by padding

Mathematical attacks

- The RSA Problem (RSAP)
  - Recovering plaintext x from ciphertext y, given the public key (n, e)
- RSA VS FACTORING
  - If p and q are known, RSAP can be easily solved
  - RSAP $\leq_p$ FACTORING
    - FACTORING is at least as difficult as RSAP or, equivalently, RSAP is not harder than FACTORING
      - It is widely believed that RSAP and Factoring are computationally equivalent, although no proof of this is known.

FACTORING.
Given n > 0, find its prime factorization; that is, write $n = \Pi_i(p_i^{e_i})$ where p_i are pairwise distinct primes and each $e_i \geq 1$.
If one can factorize n, then he can completely break RSA and thus solve RSAP. For the moment, nobody has proven that breaking RSA necessarily requires the ability of factoring n, although this conjecture is considered very plausible.

Mathematical Attacks

- THM (FACT 1) Computing the decryption exponent d from the public key (n, e) is computationally equivalent to factoring n
  - Proof
    - If factorization of n is known, then it id possible to compute the private key d efficiently
    - (It can be proven that) if d known, then it is possible to factor n efficiently

Mathematical Attacks

- RSAP vs e-th root
  - A possible way to decrypt $y = x^e \bmod n$ is to compute the modular e-th root of c
- THM (FACT 2) Computing the e-th root is a computationally easy problem iff n is prime
- THM (FACT 3) If n is composite the problem of computing the e-th root is equivalent to factoring

Mathematical Attacks

- THM - Knowing ϕ is computationally equivalent to factoring
  - PROOF.
    - Given p and q, s.t. n =pq
      - Computing ϕ is immediate.
    - Given ϕ
      - From $\phi = (p-1)(q-1) = n - (p+q) + 1$, determine $x1 = (p+q)$.
      - From $(p - q)^2 = (p + q)^2 - 4n = x_1^2 - 4n$, determine $x2 = (p - q)$.
      - Finally, $p = (x1 + x2)/2$ and $q = (x1 - x2)/2$.

Mathematical Attacks

UNIVERSITÀ DI PISA

- Exhaustive Private Key Search
  - This attack must be more difficult than factoring n
  - The bit length of private exponent d must be the same as the bit length of n
    - sizeof(p) ≈ sizeof(q)
    - sizeof(d) >> sizeof(p) AND sizeof(d) >> sizeof(q)

Factoring

- Primality testing vs. factoring
  - FACT 5 – To decide whether an integer is composite or prime seems to be, in general, much easier than the factoring problem

Factoring

- Factoring algorithms
  - Special purpose algorithms
    - Tailored to perform better when the integer n being factored is of special form
      - Running time depends on certain properties of factors of n
    - Examples
      - Trial division, Pollard's rho alg., Pollard's p – 1 alg., elliptic curve alg., and special number sieve
  - General purpose algorithms
    - Running time depends on n
    - Examples
      - Quadratic sieve and general number field sieve

Factoring

UNIVERSITÀ DI PISA

- Factoring algorithms
  - No algorithm can factor all integers in polynomial time
    - Neither the existence nor non-existence of such algorithms has been proven, but it is generally suspected that they do not exist
    - Peter Shor discovered a quantum algorithm that is polynomial (1994)
  - There are sub-exponential algorithms
    - For computers, the best algorithm is General Number Field Sieve (GNFS)

Factoring

- Length of the modulus
  - RSA sparked much interest in the old problem of integer factorization
    - Factoring methods improved considerably during ’80s and ‘90s
  - Advisable modulus length
    - Until recently, 1024-bit was a default
      - Nowadays factorization within 10-15 years or even earlier
    - Modulus in the range 2048-4096 bit for long term security