Public Key Encryption

A case study

THE RSA CRYPTOSYSTEM

Rivest Shamir Adleman (1978)

UNIVERSITÀ DI PISA

Key generation

1. Generate two large, distinct primes p, q (100÷200 decimal digits)
2. Compute $n = p \times q$ and $\varphi(n) = (p-1)\times(q-1)$
3. Select a random number $1 < e < \varphi(n)$ such that $\gcd(e, \varphi(n)) = 1$
4. Compute the unique integer $1 < d < \varphi$ such that $ed \equiv 1 \bmod \varphi$
5. (d, n) is the *private* key
6. (e, n) is the *public* key

At the end of key generation, ***p*** and ***q*** must be destroyed

RSA encryption and decryption

UNIVERSITÀ DI PISA

Encryption. To generate c from m, Bob should do the following

1. Obtain *A*'s *authentic* public key (n, e)
2. Represent the message as an integer m in the interval $[0, n-1]$
3. Compute $c = m^e \bmod n$
4. Send c to A

Decryption. To recover m from c, Alice should do the following

1. Use the private key d to recover $m = c^d \bmod n$

RSA consistency

We have to prove that $D(d(E(e, m)) = m$, i.e.,

$c^d \equiv m^{de} \equiv m^{t \bullet \varphi(n)+1} \bmod n$, where t is some integer $\Rightarrow$

$m^{t \cdot \varphi(n)} \cdot m^1 \equiv (m^{\varphi(n)})^t \cdot m^1 \equiv m \bmod n$

The proof exploits the **Eulero's theorem**

$\forall$ integer $n > 1$, $\forall a \in \mathbb{Z}_n^*$, $a^{\varphi(n)} \equiv 1 \bmod n$ *where*

$\mathbb{Z}_n^* = \{ x \mid 1 < x < n, \gcd(x, n) = 1\}$

Example with artificially small numbers

Key generation

- Let $p = 47$ e $q = 71$
  $n = p \times q = 3337$
  $\varphi = (p-1) \times (q-1) = 46 \times 70 = 3220$
- Let $e = 79$
  $ed = 1 \bmod \varphi$
  $79 \times d = 1 \bmod 3220$
  $d = 1019$

Encryption

Let $m = 9666683$

Divide m into blocks $m_i < n$

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

Compute

$c_1 = 966^{79} \bmod 3337 = 2276$

$c_2 = 668^{79} \bmod 3337 = 2423$

$c_3 = 3^{79} \bmod 3337 = 158$

$c = c_1 c_2 c_3 = 2276\ 2423\ 158$

Decryption

$m_1 = 2276^{1019} \bmod 3337 = 966$

$m_2 = 2423^{1019} \bmod 3337 = 668$

$m_3 = 158^{1019} \bmod 3337 = 3$

$m = 966\ 668\ 3$

RSA

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

Modular ops - complexity

Bit complexity of basic operations in Z_n

- Let **n** be on **k** bits (**$n < 2^k$**)
- Let **a** and **b** be two integers in **Z_n** (on k-bits)
  - **Addition $a + b$** can be done in time **$O(k)$**
  - **Subtraction $a – b$** can be can be done in time **$O(k)$**
  - **Multiplication $a \times b$** can be done in **$O(k^2)$**
  - **Division $a = q \times b + r$** can be done in time **$O(k^2)$**
  - **Inverse a^{-1}** can be done in **$O(k^2)$**
  - **Modular exponentiation a^k** can be done in **$O(k^3)$**

How to encrypt/decrypt efficiently

- RSA requires *modular exponentiation* c^d **mod** ***n***
  - Let ***n*** have ***k*** bits in its binary representation, ***k = log n + 1***
- ***Grade-school*** algorithm requires ***(d-1)*** modular multiplications
  - ***d*** is as large as **n** which is exponentially large with respect to ***k***
  - The grade-school algorithm is inefficient
- ***Square-and-multiply*** algorithm requires up to **2k** multiplications thus the algorithm can be done in $O(k^3)$

How to encrypt/decrypt efficiently

- RSA requires *modular exponentiation* a^x **mod** ***n***
  - Let ***n*** have ***k*** bits in its binary representation, ***k = log n + 1***
- ***Grade-school*** algorithm requires ***(x-1)*** modular multiplications
  - If ***x*** is as large as **n,** which is exponentially large with respect to ***k*** ➔ the grade-school algorithm is inefficient
- ***Square-and-multiply*** algorithm requires up to **2k** multiplications thus the algorithm can be done in $O(k^3)$

How to encrypt and decrypt efficiently

UNIVERSITÀ DI PISA

Exponentiation by repeated squaring and multiplication: m^e **mod** n requires **at most** $\log_2(e)$ multiplications and $\log_2(e)$ squares

Let $e_{k-1}, e_{k-2}, \dots, e_2, e_1, e_0$, where $k = \log_2 e$, the binary representation of **e**

$$m^e \bmod n = m^{\left(e_{k-1}2^{k-1}+e_{k-2}2^{k-2}+\dots+e_2 2^2+e_1 2+e_0\right)} \bmod n \equiv$$
$$m^{e_{k-1}2^{k-1}} m^{e_{k-2}2^{k-2}} \cdots m^{e_2 2^2} m^{e_1 2} m^{e_0} \bmod n \equiv$$
$$\left(m^{e_{k-1}2^{k-2}} m^{e_{k-2}2^{k-3}} \cdots m^{e_2 2} m^{e_1}\right)^2 m^{e_0} \bmod n \equiv$$
$$\left(\left(m^{e_{k-1}2^{k-3}} m^{e_{k-2}2^{k-4}} \cdots m^{e_2}\right)^2 m^{e_1}\right)^2 m^{e_0} \bmod n \equiv$$
$$\left(\left(\left(\left(m^{e_{k-1}}\right)^2 m^{e_{k-2}}\right)^2 \cdots m^{e_2}\right)^2 m^{e_1}\right)^2 m^{e_0} \bmod n$$

```
c ← 1
for (i = k-1; i >= 0; i --) {
    c ← c² mod n;
    if (e_i == 1)
        c ← c × m mod n;
}
```

- always k square operations
- **at most** k modular multiplications (***equal to the number of 1 in the binary representation of e***)

Square and multiply

UNIVERSITÀ DI PISA

Exponentiation by repeated squaring and multiplication: a^x **mod** n requires **at most** $\log_2(x)$ multiplications and $\log_2(x)$ squares

Let $x_{k-1}, x_{k-2}, \dots, x_2, x_1, x_0$, where $k = \log_2 x$, the binary representation of x

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

...

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

```
c ← 1
for (i = k-1; i >= 0; i --) {
    c ← c² mod n;
    if (x_i == 1)
        c ← c × a mod n;
}
```

- always k square operations
- **at most** k modular multiplications (***equal to the number of 1 in the binary representation of e***)

Fast encryption with short public exponent

- RSA ops with public key exponent ***e*** can be speeded-up
  - Encryption
  - Digital signature verification
- The public key ***e*** can be chosen to be a very small value
  - e = 3 #MUL + #SQ = 2
  - e = 17 #MUL + #SQ = 5
  - e = 2^{16}+1 #MUL + #SQ = 17
  - **RSA is still secure**
- There is no easy way to accelerate RSA when the private exponent ***d*** is involved

How to find a large prime

repeat *p* ← randomOdd(x); **until** isPrime(*p*);	▪ **FACT**. On average (ln *x*)/2 odd numbers must be tested before a prime *p* < *x* can be found

- Primality tests **do not** try to factor the number under test
  - *probabilistic primality test* (Solovay-Strassen, Miller-Rabin) polynomial in **log n**
  - *true primality test* (O(n^{12}) in 2002))

On computing the public exponent

- Solution of **$d \cdot e \equiv 1 \bmod \varphi(n)$** with **$\gcd(e, \varphi(n)) \equiv 1$** can be done by means of the **Extended Euclidean Algorithm** (EEA)
  - Exponent ***d*** can be generated efficiently (polytime)
  - Condition $\gcd(e, \varphi(n)) \equiv 1$

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

Security of RSA

UNIVERSITÀ DI PISA

The RSA Problem (RSAP)

- **DEFINITION. The RSA Problem** (**RSAP**): recovering plaintext *m* from ciphertext *c*, given the public key (*n*, *e*)

RSA VS FACTORING

- **FACT. 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.***

RSA vs Factoring

UNIVERSITÀ DI PISA

- **THM**. Computing the decryption exponent ***d*** from the public key (***n, e***) is computationally equivalent to factoring ***n***
  - If the adversary could somehow factor ***n***, then he could subsequently compute the private key ***d*** efficiently
  - If the adversary could somehow compute ***d***, then it could subsequently factor ***n*** efficiently

Factoring

UNIVERSITÀ DI PISA

- **FACTORING**.
  - Given ***n* > 0**, find its prime factorization; that is, write $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ where $\boldsymbol{p_i}$ are pairwise distinct primes and each $\mathbf{e_i} \geq \mathbf{1}$**,**
- **Primality testing vs. factoring**
  - Deciding whether an integer is composite or prime seems to be, in general, much easier than the factoring problem
- **Factoring algorithms**
  - Brute force
  - Special purpose
  - General purpose
  - Elliptic Curve
  - Factoring on Quantum Computer (for the moment only theorethical)

Factoring algorithms

UNIVERSITÀ DI PISA

- **Brute Force**
  - Unfeasible if *n* large and *p* len = *q* len
- **General purpose**
  - The running time depends solely on the size of n
    - Quadratic sieve
    - General number field sieve
- **Special purpose**
  - The running time depens on certain properties
    - Trial division
    - Pollard's rho algorithm
    - Pollard's *p* -1 algorithm
- **Elliptic curve algorithm**

Running times

UNIVERSITÀ DI PISA

Trial division: $O\left(\sqrt{n}\right)$

Quadratic sieve: $O\left(e^{\left(\sqrt{\ln(n)\bullet\ln\ln(n)}\right)}\right)$

General number field sieve: $O\left(e^{\left(1.923\times\sqrt[3]{\ln(n)\bullet(\ln\ln(n))^2}\right)}\right)$

Security of RSA

UNIVERSITÀ DI PISA

RSAP and e-th root

- A possible way to decrypt $c = m^e \bmod n$ is to compute the e-th root of c
- **THM**. Computing the e-th root is a computationally easy problem iff n is prime
- **THM**. If n is composite the problem of computing the e-th root is *equivalent* to factoring

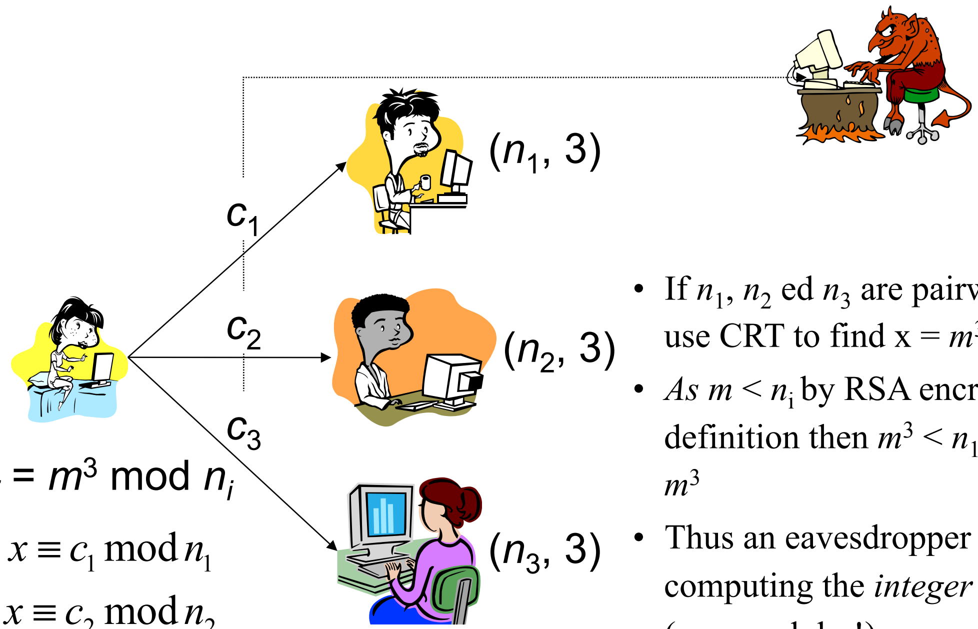
Security of RSA

- **Factoring vs totally breaking RSA**
  - A possible way to completely break RSA is to obtain φ
- **THM**. Knowing φ is computationally equivalent to factoring
  - **PROOF**.
    1. Given p and q, s.t. $n = pq$, computing φ is immediate.
    2. Let φ be given.
       a. From $\varphi(n) = (p-1)(q-1) = n - (p+q) + 1$, determine $x_1 = (p+q)$.
       b. From $(p - q)^2 = (p + q)^2 - 4n$, determine $x_2 = (p - q)$.
       c. Finally, $p = (x1 + x2)/2$ and $q = (x1 - x2)/2$.

Security of RSA

- A possible way to completely break RSA is an exhaustive attack to the private key *d*
  - This attack could be more difficult than factoring because (according to the choice for *e*) *d* can be much greater than *p* and *q*.

RSA: low exponent attack



$c_i = m^3 \bmod n_i$

$$\begin{cases} x \equiv c_1 \bmod n_1 \\ x \equiv c_2 \bmod n_2 \\ x \equiv c_3 \bmod n_3 \end{cases}$$

- If n_1, n_2 ed n_3 are pairwise coprime, use CRT to find x = $m^3 \bmod n_1 n_2 n_3$
- *As* $m < n_i$ by RSA encryption definition then $m^3 < n_1 n_2 n_3$, then $x = m^3$
- Thus an eavesdropper recovers m by computing the *integer cube root* of x (non modular!)

RSA in practice - padding

- We have described "schoolbook RSA"
- RSA implementation may be insecure
  - RSA is deterministic
  - PT values x = 0, x = 1 produce CT equal to 0 and 1
  - Small PT might be subject to attacks
  - RSA is malleable
- Padding is a possible solution
  - Optimal Asymmetric Encryption Padding (OAEP)
  - Public Key Cryptography Standard #1 (PKCS #1)

RSA is malleable

- RSA malleability is based on the homo-morphic property of RSA
- Attack
  - The attacker replaces CT = y mod n by CT' = $s^e \cdot y$ mod n, with s some integer
  - The receiver decrypts CT': $(s^e \cdot y)^d = s^{ed} \cdot x^{ed} = s \cdot x \bmod n$
  - By operating on the CT the adversary manages to multiply PT by s
  - **EX**. Let x be an amount of money. If $s = 2$ then the adversary doubles the amount
  - Possible solution: introduce redundancy: ex. $x \,||\, x$

RSA – Homomorphic property

- Let m_1 and m_2 two plaintext messages
- Let c_1 and c_2 their respective encryptions
- Observe that

$$(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod{n}$$

- In other words, the CT of the product $m_1 m_2$ is the product of CTs $c_1 c_2$ mod n

RSA in practice - PKCS #1

- Parameters
  - M = message
  - | M | = message len in bytes
  - k = | n | modulus len in bytes
  - | H | = hash function output len in bytes
  - L = optional label ("" by default)

RSA in practice - PKCS #1

- Padding
  1. Generate a string *PS* = 00…0; *PS* len = *k* – | *M* | - 2 |*H*| - 2 (*PS* len may be zero)
  2. *DB* = Hash(*L*) || *PS* || `0x01` || *M*
  3. *seed* = random(); *seed* len = | H |
  4. *dbMask* = MGF (*seed*, *k* - | *H* | - 1) (*)
  5. *maskedDB* = *DB* **xor** *dbMask*
  6. *seedMask* = MGF(*maskedDB*, | *H* |)
  7. *maskedSeed* = *seed* **xor** *seedMask*
  8. *EM* = `0x00` || *maskedSeed* || *maskedDB* (**)

(*) *MGF mask generation function (e.g., SHA-1)*
(**) *EM is the padded message*

RSA in practice

- ***RSA is substantially slower than symmetric encryption***
  – RSA is used for the transport of symmetric-keys and for the encryption of small quantities
- ***Recommended size of the modulus***
  – 512 bit: marginal security
  – 768 bit: recommended
  – 1024 bit: long-term security

RSA in practice

Selecting primes p and q

– ***p*** and ***q*** should be selected so that factoring ***n = pq*** is computationally infeasible, therefore

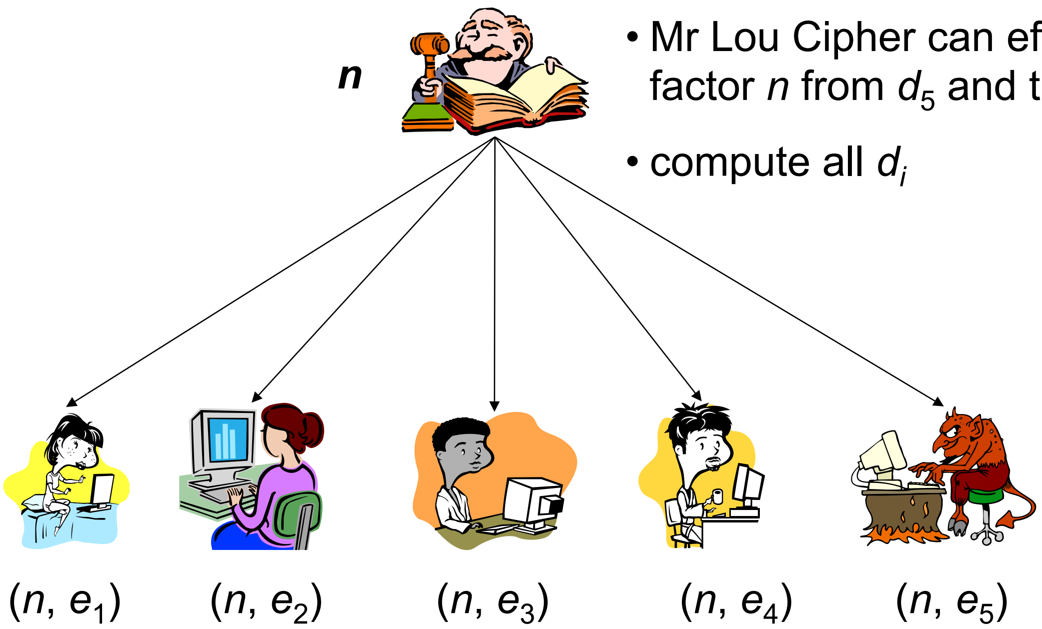
– ***p*** and ***q*** should be ***sufficiently large*** and about the ***same bitlenght*** (to avoid the elliptic curve factoring algorithm)

– ***p - q should be not too small***

RSA in practice

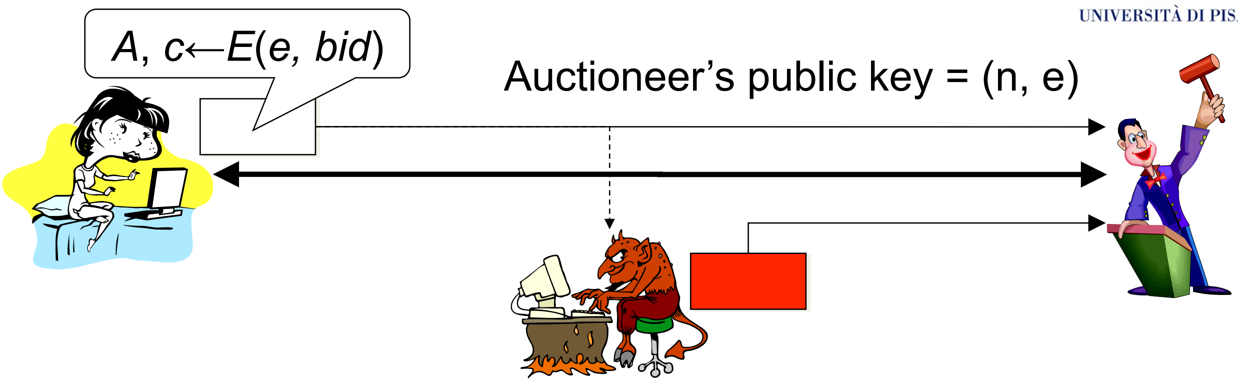
- Exponent *e* should be small or with a small number of 1's
  - **e = 3**
    [1 modular multiplication + 1 modular squaring]
    subject to small encryption exponent attack
  - **e = 2^{16} + 1 (Fermat's number)**
    [1 modular multiplication + 16 modular squarings]
    resistant to small encryption exponent attacks
- Decryption exponent *d* should be roughly the same size as *n*
  - Otherwise, if *d* is small, it could be possible to obtain *d* from the public information (*n*, *e*) or from a brute force attack

Common modulus attack



- Mr Lou Cipher can efficiently factor *n* from d_5 and then
- compute all d_i

Chosen-plaintext attack



The adversary encrypts all possible bids (e.g, 2^{32}) until he finds a **b** such that *E*(e, b) = *c*

Thus, the adversary sends a bid containing the minimal offer to win the auction: b' = b + 1

Salting is a solution: $r \leftarrow$ random(); $c \leftarrow E(e, r \,||\, bid)$

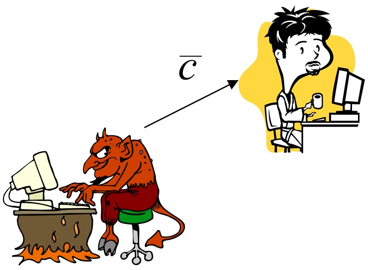
Homomorphic property of RSA

- Let m_1 and m_2 two plaintext messages
- Let c_1 and c_2 their respective encryptions
- Observe that

$$(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod{n}$$

  - In other words, the ciphertext of the product $m_1 m_2$ is the product of ciphertexts $c_1 c_2$ mod *n*

An adaptive chosen-ciphertext attack



- Bob decrypts ciphertext except a given ciphertext *c*
- Mr Lou Cipher wants to determine the ciphertext corresponding to *c*

- Mr Lou Cipher selects *x* at random, s.t. gcd(*x*, *n*) =1, and sends Bob the quantity $\overline{c} = cx^e \bmod n$
- Bob decrypts it, producing $\overline{m} = (\overline{c})^d = c^d x^{ed} = mx \pmod{n}$
- Mr Lou Cipher determine *m* by computing $m = \overline{m}x^{-1} \bmod n$

The attack can be contrasted by imposing structural constraints on *m*

Hybrid systems

- An asymmetric cipher is subject to the chosen-plaintex attack
- An asymmetric cipher is three orders of magnitude slower than a symmetric cipher

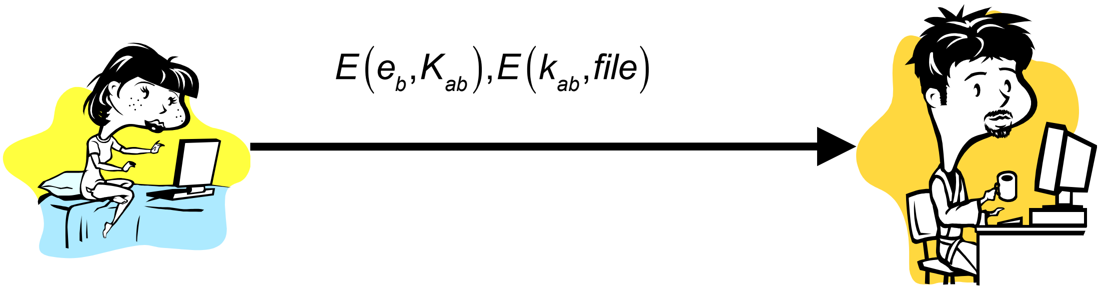
therefore

- An asymmetric cipher is often used in conjunction with a symmetric one so producing an *hybrid system*

Hybrid systems

UNIVERSITÀ DI PISA

Alice confidentially sends Bob a file *file*



- File *file* is encrypted with a symmetric cipher
- Session key is encrypted with an asymmetric cipher
- Alice needs an *authentic* copy of Bob's public key

Public-key encryption

OTHER PUBLIC KEY CRYPTO-SYSTEMS

Other asymmetric cryptosystems

Discrete Logarithm Systems

- Let p be a prime, q a prime divisor of p–1 and $g \in [1, p-1]$ has order q
- Let x be the *private key* selected at random from $[1, q-1]$
- Let y be the corresponding *public key* $y = g^x \bmod p$
- **Discrete Logarithm Problem (DLP)**
- Given (p, q, g) and y, determine x

ElGamal encryption scheme

- **Encryption**
  - select ***k*** randomly
  - $c1 = g^k \bmod p$, $c_2 = m \times y^k \bmod p$
  - send (c_1, c_2) to recipient
- **Decryption**
  - $c_1^x = g^{kx} \bmod p = y^k \bmod p$
  - $m = c_2 \times y^{-k} \bmod p$
- **Security**
  - An adversary needs $y^k \bmod p$. The task of calculating $y^k \bmod p$ from **(g, p, q)** and **y** is equivalent to **DHP** and thus ***based*** on **DLP** in $\mathbb{Z}_p$

ElGamal in practice

UNIVERSITÀ DI PISA

- Prime p and generator g can be common system-wide
- Prime p size
  - 512-bit: marginal
  - 768-bit: recommended
  - 1024-bit or larger: long-term
- Efficiency
  - Encryption requires two modular exponentiations
  - Message expansion by a factor of 2
- Security
  - Different random integers k must be used for different messages

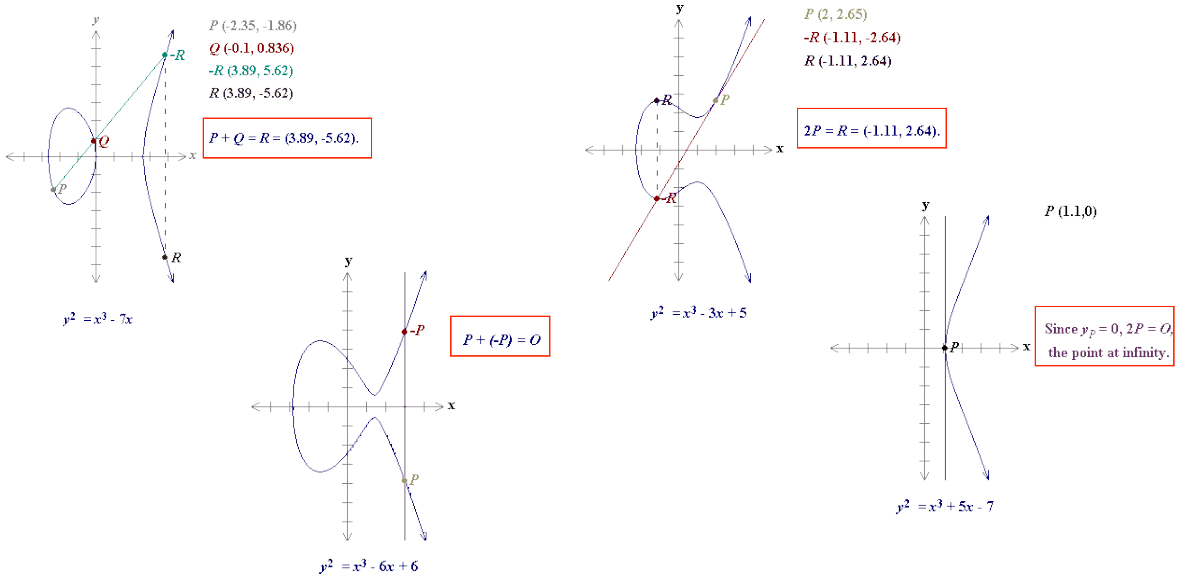
Ellyptic Curve Cryptography

UNIVERSITÀ DI PISA

- Let p and $\in \mathbb{F}_p$
- Let E be an elliptic curve defined by
  $y^2 = x^3 + ax + b$ (mod p) where a, $b \in \mathbb{F}_p$ and $4a^3+27b^2=0$
- Example. E: $y^2 = x^3 + 2x + 4$ (mod p)
- The set of points $E(\mathbb{F}_p)$ with *point at infinity* ∞forms an additive Abelian group

Elliptic curves

Geometrical approach



Elliptic Cryptography (ECC)

- Algebric Approach
  - Elliptic curves defined on finite field define an Abelian finite field
- **Elliptic curve discrete logarithm problem**
  - **Given points G and Q such that Q=kG, find the integer *k***
  - **No sub-exponential algorithm to solve it is known**
- ECC keys are smaller than RSA ones

Ellyptic Curve Cryptography

- Let *P* have order ***n*** then the cyclic subgroup generated by ***P*** is ***G = <P, 2P,…, (n – 1)P>***
- ***p***, ***E***, ***P*** and n are the *public parameters*
- Private key ***d*** is selected at random in [1, *n*–1]
- Public key is ***Q =dP***

Ellyptic Curve Cryptography

- Encryption
  - A message ***m*** is represented as a point ***M***
  - $C_1 = kP$; $C_2 = M + kQ$
  - send $(C_1; C_2)$ to recipient
- Decryption
  - $dC_1 = d(kP) = kQ$
  - $M = C_2 - dC_1$
- Security
  - The task of computing *kQ* from the domain parameters, ***Q***, and $C_1=kP$, is the **ECDHP**

Comparison among crypto-systems

	Security level (bits)				
	80 (SKIPJACK)	112 (3DES)	128 (AES small)	192 (AES medium)	256 (AES large)
DL parameter q EC parameter n	160	224	256	384	512
RSA modulus n DL modulus p	1024	2048	3072	8192	15360

- Private key operations are more efficient in EC than in DL or RSA
- Public key operations are more efficient in RSA than EC or DL if small exponent *e* is selected for RSA

PHYSICAL ATTACKS

Physical attacks

UNIVERSITÀ DI PISA

- Embedded systems change the threat model
  - The adversary may physically attack the system
    - E.g.: smart meter
  - The system is even given to the adversary
    - E.g.: a bank or telco smart card
  - The adversary physically interfere with the system
  - Main attacks
    - Fault injection
    - Time analysis
    - Power analysis

CRT and RSA optimization

UNIVERSITÀ DI PISA

- **Chinese Remainder Theorem** allows us to compute RSA more efficiently
- **Problem**: Compute $m = c^d \pmod{n}$
  1. Compute $m_1 = c^d \pmod{p}$ and $m_2 = c^d \pmod{q}$
  2. Compute $\mathbf{m = a_1 m_1 q + a_2 m_2 p}$
     where $\mathbf{a_1}$ and $\mathbf{a_2}$ are properly computed coefficients
- **Advantage.**
  - $E_1 = c^d \pmod{p} = c^{(d \bmod p-1)} \pmod{p}$,
  - While d is on k bits, **p–1** is on **k/2** bits
  - Thus, multiplication takes $\mathbf{O(k^2/4)}$

CRT and RSA optimization

- **Chinese Remainder Theorem** allows us to compute RSA (decryption, signing) more efficiently
- **Problem**: Compute $y = x^d \pmod{n}$
  1. Compute $x_p = x \bmod p$ and $x_q = x \bmod q$
  2. Compute $y_p = x_p^{d \bmod (p-1)} \bmod p$ and $y_q = x_q^{d \bmod (q-1)} \bmod q$
  3. Compute $y = a_p y_p q + a_q y_q p$ where a_p and a_q are properly (pre-)computed coefficients
- **Advantage.**
  – Computation of y_p and y_q is the most demanding
  – It requires #MUL+#SQ = 1.5*t*, on average
  – Each squaring/multiplication involves *t*/2-bit operands ➔ multiplication/ squaring takes $O(k^2/4)$
  – Thus **the total speedup obtained through CRT is a factor of 4**.

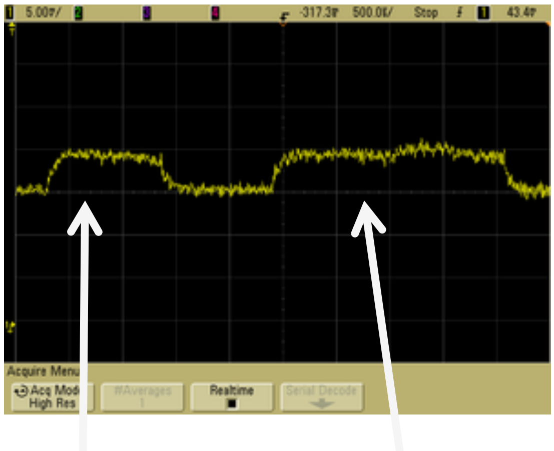
A fault-injection attack against CRT-based RSA

- **Attack intuition:** by injecting a fault the adversary is able to factorize **n**
- **The attack**
  – Cause an **hw fault** while computing y_p which produces y'_p and thus $y' = a_p y'_p q + a_q y_q p$
  – It follows that $y - y' = a_p(m'_p - m_p)q$
  – Thus, $\gcd(y - y', n) = q$ *which can be efficiently computed with the Euclide's algorithm*
- **Practical considerations**
  – **causing hw fault**: tamper with computing circuitry
  – **countermeasures**: checking results (10% *slow down*)

Power Analysis

- Power analysis is a **side channel** attack in which the attacker studies the power consumption of a cryptographic hardware device
  - smart card, tamper-resistant "black box", or integrated circuit
- The attack is non-invasive
- **Simple power analysis (SPA)** involves *visual examination* of graphs of the current used by a device over time.
  - Variations in power consumption occur as the device performs different operations.

Power Analysis of RSA

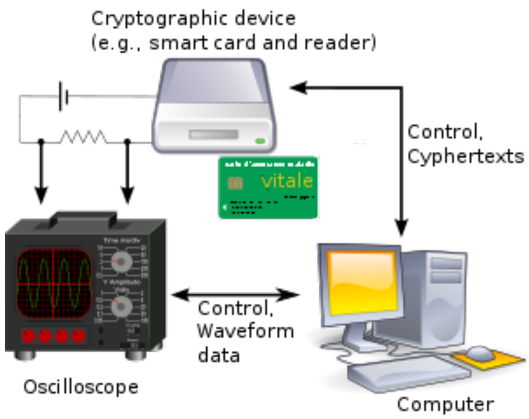


Key bit = 0
No multiplication

Key bit = 1
multiplication

Power Analysis

- **Differential power analysis (DPA)** involves statistically analyzing power consumption measurements from a cryptosystem.
  - DPA attacks have signal processing and error correction properties which can extract secrets from measurements which contain too much noise to be analyzed using simple power analysis.



Timing attack

UNIVERSITÀ DI PISA

- A **timing attack** is a **side channel** attack in which the attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms
  - Execution time depends on inputs (e.g., key!)
  - Precise measurement of time
  - Attack is application dependent
  - E.g., square-and-multiply for exp mod n
    - time depends on number of “1” in the key
    - Statistical analysis of timings with same key and different inputs