Network Security

Elements of Applied Cryptography

Public key encryption

- **Public key cryptosystem**
- **RSA and the factorization problem**
- **RSA in practice**
- **Other asymmetric ciphers**

Asymmetric Encryption Scheme

Let us consider two families of algorithms representing **invertible transformations**:

$$\boxed{\begin{array}{l} \text{Encryption transformations}: \{E_e : e \in \mathcal{K}\}, E_e : \mathcal{M} \rightarrow \mathcal{C} \\ \text{Decryption transformations}: \{D_d : d \in \mathcal{K}\}, D_d : \mathcal{C} \rightarrow \mathcal{M} \end{array}}$$

such that:

I. $\forall\ e \in \mathcal{K}$, $\exists$ a unique $d \in \mathcal{K}$, such that D_d is the inverse of E_e

II. $\forall\ m \in \mathcal{M}$, $\forall\ c \in \mathcal{C}$, $E_e(m)$ and $D_d(c)$ are easy to compute

III. Known $e \in \mathcal{K}$ and $c \in C$, it is computationally infeasible to find the message $m \in \mathcal{M}$ such that $E_e(m) = c$

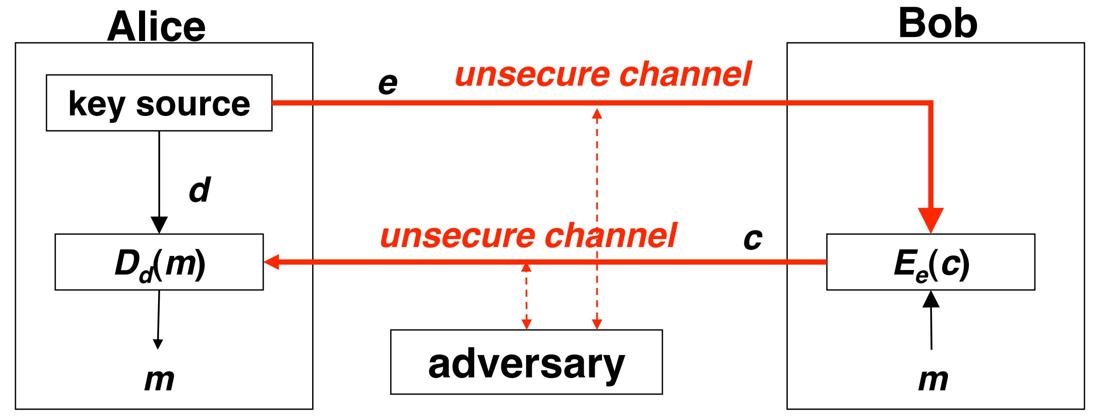
IV. Known $e \in \mathcal{K}$, it is computationally infeasible to determine the corresponding key d

Public key encryption

Because of properties III and IV,

- decryption key ***d*** MUST be kept **secret**
- encryption key ***e*** CAN be made **public** without compromising the security of the decryption key

2-party comm with asymmetric encryption



- The encryption key *e* can be sent on the ***same*** channel on which the ciphertext *c* is being transmitted
- It is necessary to ***authenticate*** public keys to achieve ***data origin authentication*** of the public keys themselves

Types of attack

Objectives of adversary

- **break the system**: recover plaintext from ciphertext
- **completely break the system**: recover the key

Types of attacks

- *No asymmetric cipher is perfect*
- Since the encryption keys are public knowledge, a passive adversary can always mount a **chosen-plaintext attack**
- A stronger attack is a **chosen-ciphertext attack** where an active adversary selects ciphertext of its choice, and then obtains by some means (from the victim) the corresponding plain-tex

A case study

THE RSA CRYPTOSYSTEM

Rivest Shamir Adleman (1978)

Key generation

1. Generate two **large**, **distinct primes** ***p*, *q*** (100÷200 decimal digits)
2. Compute $\boldsymbol{n = p \times q}$ and $\boldsymbol{\phi = (p-1)\times(q-1)}$
3. Select a **random number** $1 < \boldsymbol{e} < \phi$ such that **gcd(e, ϕ) = 1**
4. Compute the **unique** integer $1 < \boldsymbol{d} < \phi$ such that $ed \equiv 1 \bmod \phi$

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

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]** ($\boldsymbol{0 \le m < n}$)
3. Compute $\boldsymbol{c = m^e \bmod n}$
4. Send ***c*** to ***A***

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

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

RSA proof

Proof. We have to prove that $\boldsymbol{D_d(E_e(m)) = m}$, i.e.,

$c^d \equiv m^{de} \equiv m^{k\phi+1} \bmod n, \forall k \in \mathbb{Z} \equiv (m^{\phi})^k \times m \bmod n \equiv$

(applying Eulero's Theorem)

$\equiv m \bmod n$

Eulero's Theorem

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

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

Example with artificially small numbers

Key generation

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

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

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

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

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

$m = 966\ 668\ 3$

How to encrypt/decrypt efficiently

- Let **a** and **b** be two **k**-bit integers
  - **Addition a + b** can be done in time **O(k)**
  - $\mathbf{a \times b}$ can be done in $\mathbf{O(k^2)}$
- Let **c** be an (at most) **2k**-bit integer
  - **c mod a** can be done in $\mathbf{O(k^2)}$
- Let **d** be a **k**-bit integer
  - $\mathbf{a \times b}$ **mod d** can be done in $\mathbf{O(k^2)}$

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 ϕ which is exponentially large with respect to ***k***
  - The grade-school algorithm is inefficient
- *Square-and-multiply* algorithm requires **2*r*** modular multiplications where ***r*** is the number of bits in the binary representation of ***d***
  - As $r \leq k$ then the algorithm can be done in $\mathbf{O}(k^3)$

How to encrypt and decrypt efficiently

Exponentiation by repeated squaring and multiplication: m^e **mod n** requires **at most 2log$_2$(e)** multiplications and **2log$_2$(e)** divisions

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

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

$$m^{e_{k-1}2^{k-1}} m^{e_{k-2}2^{k-2}} \cdots m^{e_2 2^2} m^{e_1 2} m^{e_0} \bmod n \equiv$$

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

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

$$\left(\left(\left(\left(m^{e_{k-1}}\right)^2 m^{e_{k-2}}\right)^2 \cdots m^{e_2}\right)^2 m^{e_1}\right)^2 m^{e_0} \bmod n$$

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

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

How to find a large prime

```
repeat
    $b \leftarrow$ randomOdd();
until isPrime($b$);
```

On average (log x)/2 odd numbers must be tested before a prime $b < x$ 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))

- Given **e**, ***d*** can be computed efficiently by means of the extended Euclid algorithm
- It follows that keys can be generated efficiently (polytime)

Factoring

- **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 $\mathbf{e_i \geq 1}$,

- **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 a theorethical construct)

Factoring algorithms

- **Brute Force**
  - Unfeasible if *n* large and $|p| = |q|$
- **General purpose**
  - the running times depend solely on the size of *n*
  - Quadratic sieve
  - General number field sieve
- **Special purpose**
  - the running times depend on certain properties of *n* (lead to the introduction of **strong primes**)
  - Trial division
  - Pollard's rho algorithm
  - Pollard's $p - 1$ algorithm
- **Elliptic curve algorithm**

Running times

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

The RSA Problem (RSAP)

- **DEFINITION. The RSA Problem** (**RSAP**): recovering plaintext *m* from ciphertext *c*, given the public information (*n*, *e*)
- **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 the RSA and the integer factorization problems are computationally equivalent, although no proof of this is known.

Security of RSA

RSAP from yet another viewpoint…

- A possible way to decrypt ***c* = *m*^e^ mod *n*** is to compute the ***e*-th root** of *c*
  - Computing the *e*-th root is a computationally easy problem iff *n* is prime
  - If *n* is not prime the problem of computing the *e*-th root is *equivalent* to factoring

Security of RSA

Relationship between Factoring and totally breaking RSA

- **A possible way to completely break RSA is to discover $\Phi(n)$**
- **Computing $\Phi(n)$ is computationally equivalent to factoring *n***
  - Given **p** and **q**, s.t. ***n* =*pq***, computing **Φ(n)** is immediate.
  - Let $\Phi(n)$ be given.
    From $\Phi(n) = (p-1)(q-1) = n - (p+q) + 1$, determine $\mathbf{x_1 = (p+q)}$.
    From $(p - q)^2 = (p + q)^2 - 4n$, i.e., $(p - q)(p+q) = (p + q)^2 - 4n$, determine $\mathbf{x_2 = (p - q)}$.
    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***.

Security of RSA: relation to factoring

- **The problem of computing the RSA decryption exponent *d* from the public key (*n, e*) and the problem of factoring *n* are computationally equivalent**
  - 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

Adaptive chosen-ciphertext attack

- A ***chosen-ciphertext attack*** is one where the adversary selects the ciphertext and is then given the corresponding plaintext.
  - One way to mount such an attack is for the adversary to gain access to the equipment used for decryption (but not the decryption key, which may be securely embedded in the equipment). The objective is then to be able, without access to such equipment, to deduce the plaintext from (different) ciphertext.
- An ***adaptive chosen-ciphertext*** attack is a chosen-ciphertext attack where the choice of ciphertext may depend on the plaintext received from previous requests

Adaptive chosen-ciphertext attack

Homomorphic property of RSA

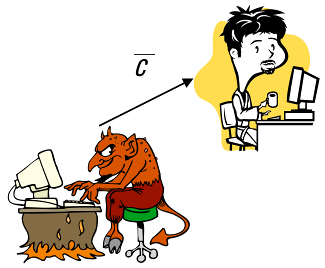
- Let m_1 and m_2 two plaintext messages
- Let c_1 and c_2 their respective encryptions
- Observe that

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

- In other words, *the ciphertext of the product is equal to the product of the ciphertexts*

An adaptive chosen-ciphertext attack …

…based on the homomorphic property of RSA



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

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

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

Hybrid systems

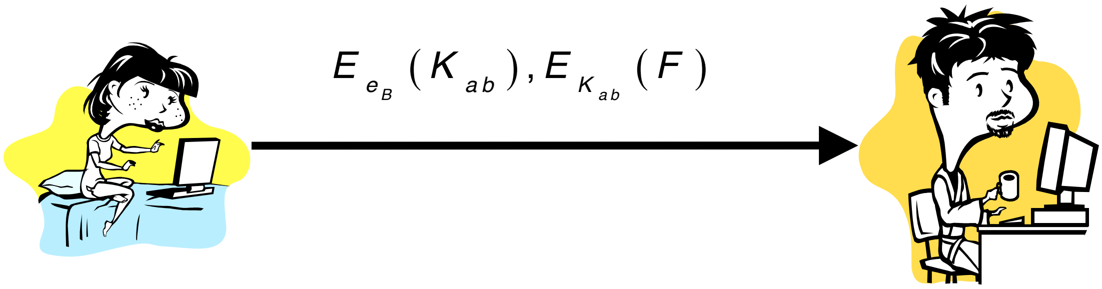
- An asymmetric cipher is subject to the chosen-plaintex attack
- An asymmetric cipher is three orders of magnitude slower than a symmetric cipher

therefore

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

Hybrid systems

Alice confidentially sends Bob a file *F*



$$E_{e_B}(K_{ab}), E_{K_{ab}}(F)$$

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

The RSA cryptosystem

RSA IN PRACTICE

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***

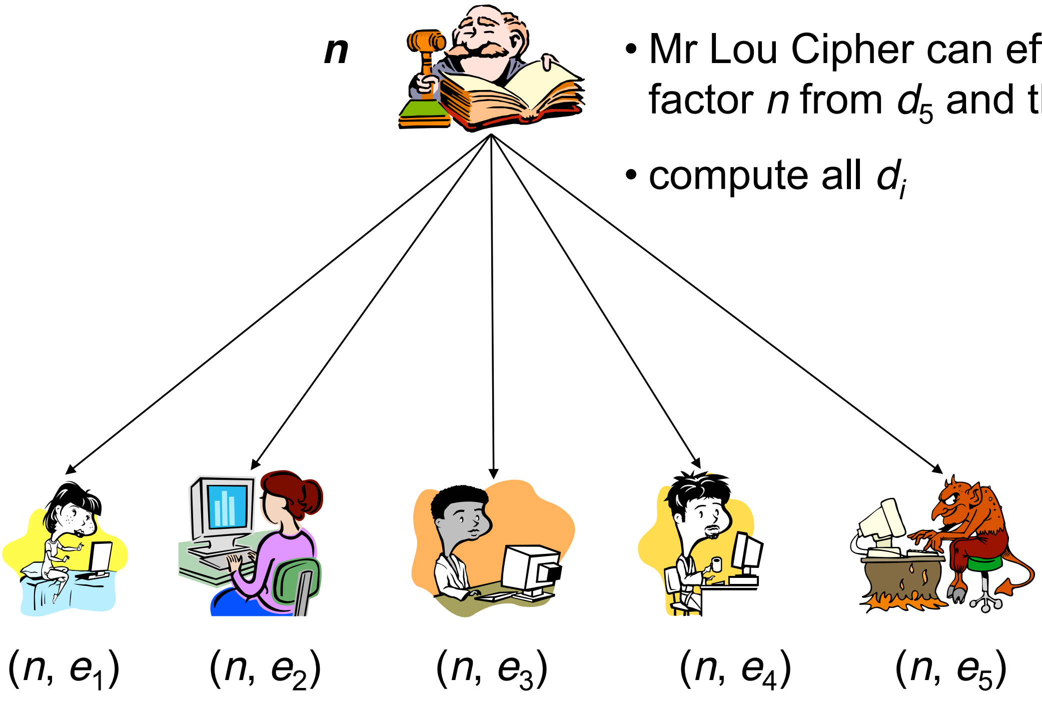
RSA in practice

- **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

The RSA Cryptosystem

WELL KNOWN ATTACKS AGAINST RSA

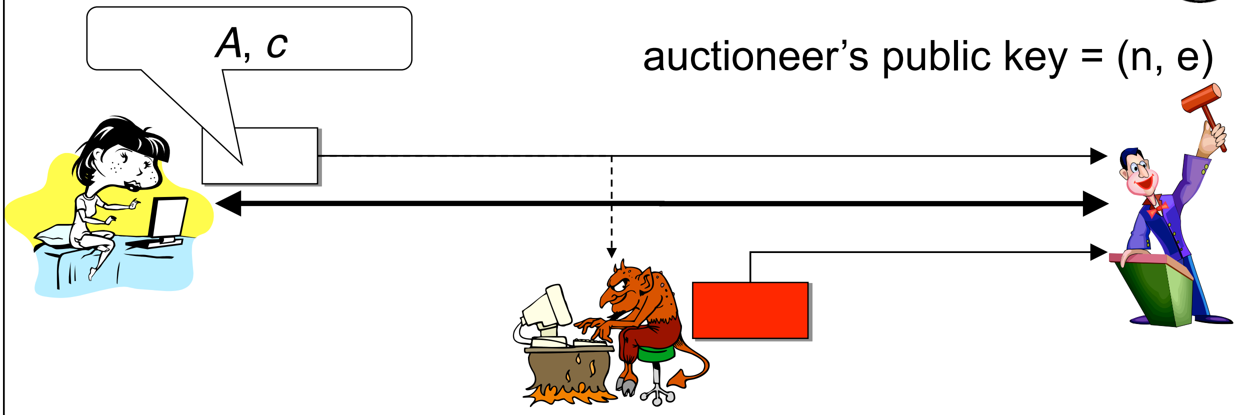
Common modulus attack



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

Small message space attack

auctioneer’s public key = (n, e)



The adversary encrypts all possible bids (2^{32}) until he finds an offer Θ such that $E_e(\Theta) \equiv c$

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

Salting is a solution

Chinese Remainder Theorem

Suppose that $m_1, m_2, \ldots, m_r$ are pair wise relatively prime positive integers, and suppose that $a_1, a_2, \ldots a_r$ are integers. Then the system of congruence's $x \equiv a_i \bmod m_i$ $(1 \le i \le r)$ has a unique solution modulo $M = m_1 \times m_2 \times \ldots \times m_r$ which is given by

$$x = \sum_{i=1}^{r} a_i M_i y_i \bmod M$$

$$\text{where } M_i = \frac{M}{m_i} \text{ and } y_i = M_i^{-1} \bmod m_i \text{ for } 1 \le i \le r$$

CRT optimization for performance

- **Task**
- We have to compute ***m* = *c*^{d} (mod *n*)**
- **Facts**
  - **Fact 1**. CRT allows us to compute ***x* mod *pq*** from ***x* mod *p*** and ***x* mod *q***.
  - **Fact 2**. Little Fermat's theorem states that **a^{p-1} mod *p* = 1.**

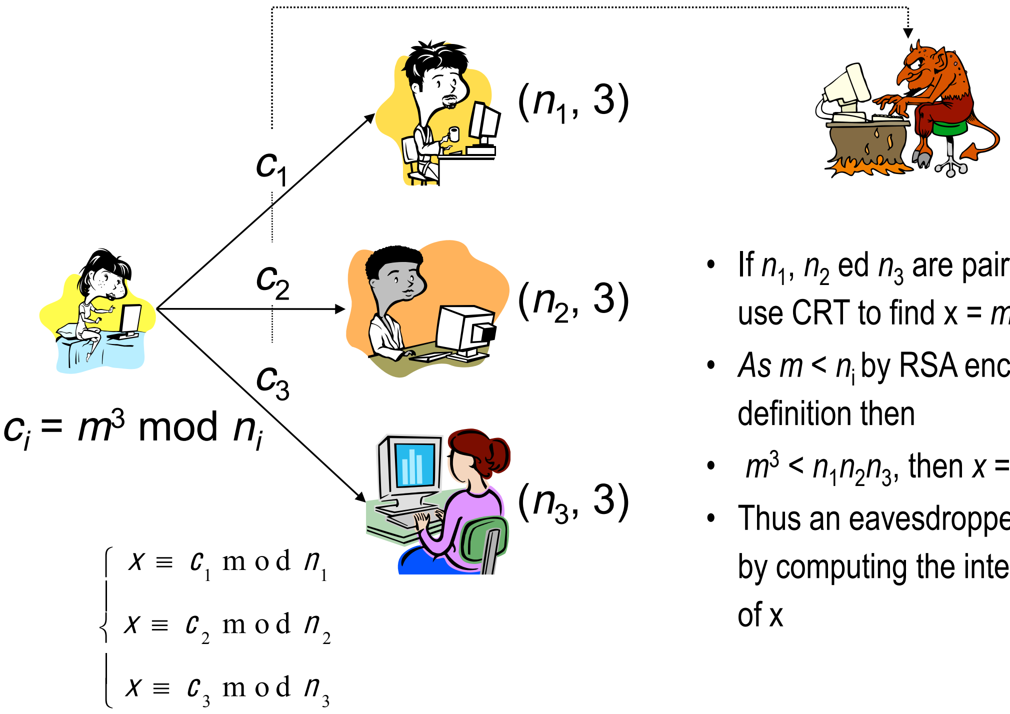
CRT optimization for performance

- **Algorithm**
- Compute $\boldsymbol{m_1 = c^d}$ **(mod *p*)** and $\boldsymbol{m_2 = c^d}$ **(mod *q*).**
- By **Fact 2**, compute $\mathbf{m_1 = c^{d \bmod p-1}}$ **(mod p)** and $\mathbf{m_2 = c^{d \bmod q-1}}$ **(mod q)**.
- Using CRT (Fact 1) compute
- $a_1 = q^{-1} \bmod p$; $a_2 = p^{-1} \bmod q$
- $m = a_1 m_1 q + a_2 m_2 p$
- **Advantage**
- $\boldsymbol{E_1 = c^d}$ **(mod *p*) = c** $^{\mathbf{(d \bmod p - 1)}}$ **(mod p)**, while ***d*** is on ***k*** bits, **p–1** is on **k/2** bits. Thus, multiplication takes $\mathbf{O(k^2/4)}$

A fault-injection attack

- ***The fault-injection attack exploits an RSA implementation with CRT.***
- **The attack**
  - Cause an **hw fault** while computing $\mathbf{m_1}$ which produces $\mathbf{m'_1}$ and thus $\mathbf{m' = a_1 m'_1 q + a_2 m_2 p}$
  - It follows that $\mathbf{m - m' = a_1(m'_1 - m_1)q}$
  - Thus, $\mathbf{gcd(m - m', n) = q}$ which can be efficiently computed with the Euclide's algorithm
- **Practical considerations**
  - **causing hw fault**: tamper with computing circuitry
  - **scenario**: embedded systems, ***physical possession***
  - **countermeasures**: checking results (*performance*)

RSA: low exponent attack



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

- If n_1, n_2 ed n_3 are pairwise coprime, use CRT to find x = $m^3 \bmod n_1n_2n_3$
- *As* $m < n_i$ by RSA encryption definition then
- $m^3 < n_1n_2n_3$, then $x = m^3$
- Thus an eavesdropper recovers m by computing the integer cube root of x

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 mod p**. The task of calculating **y^k mod p** from **(g, p, q)** and **y** is equivalent to **DHP** and thus ***based*** on **DLP** in $\mathbb{Z}_p$

ElGamal in practice

- 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

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

Ellyptic Curve Cryptography

- Let P have order $\boldsymbol{n}$ then the cyclic subgroup generated by $\boldsymbol{P}$ is $\langle\infty\boldsymbol{.\ P, 2P,\ldots, (n-1)P}\rangle$
- $\boldsymbol{p}$, $\boldsymbol{E}$, $\boldsymbol{P}$ and n are the *public parameters*
- Private key $\boldsymbol{d}$ is selected at random in $[1, n-1]$
- Public key is $\boldsymbol{Q = dP}$

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