Sistemes de clau pública

Carlos Borrego

Carlos.Borrego@uab.cat

Departament d'Enginyeria de la Informació i de les Comunicacions
Universitat Autònoma de Barcelona

Criptografia i Seguretat

Contingut

Problem

OK, symmetric key encryption works, but...

how can I deliver a key to my communication partner knowing that the shared medium is **hostile**?

Contingut

Key exchange: Diffie-Hellman

- Alice chooses randomly some a and computes $A := g^a$.
- Bob chooses randomly some b and computes $B := g^b$.
- Alice sends Bob A. Bob sends Alice B.
- Alice computes $k := B^a = (g^b)^a = g^{a*b}$.
- Bob computes $k := B^a = (g^a)^b = g^{a*b}$.
- Now Alice and Bob can use k as their secret key to encrypt and decrypt messages.

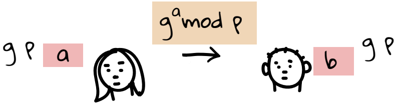
Key exchange: Diffie-Hellman

- Alice chooses randomly some a and computes $A := g^a$.
- Bob chooses randomly some b and computes $B := g^b$.
- Alice sends Bob A. Bob sends Alice B.
- Alice computes $k := B^a = (g^b)^a = g^{a*b}$.
- Bob computes $k := B^a = (g^a)^b = g^{a*b}$.
- Now Alice and Bob can use k as their secret key to encrypt and decrypt messages.

Key exchange: Diffie-Hellman

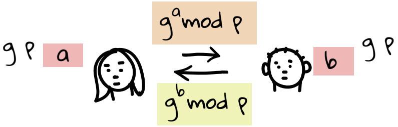
- Alice chooses randomly some a and computes $A := g^a$.
- Bob chooses randomly some b and computes $B := g^b$.
- Alice sends Bob A. Bob sends Alice B.
- Alice computes $k := B^a = (g^b)^a = g^{a*b}$.
- Bob computes $k := B^a = (g^a)^b = g^{a*b}$.
- Now Alice and Bob can use k as their secret key to encrypt and decrypt messages.

Key exchange: Diffie-Hellman



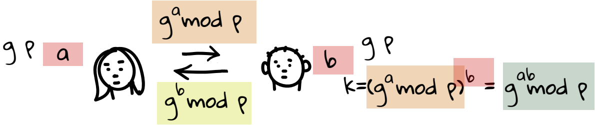
- Alice chooses randomly some a and computes $A := g^a$.
- Bob chooses randomly some b and computes $B := g^b$.
- Alice sends Bob A. Bob sends Alice B.
- Alice computes $k := B^a = (g^b)^a = g^{a*b}$.
- Bob computes $k := B^a = (g^a)^b = g^{a*b}$.
- Now Alice and Bob can use k as their secret key to encrypt and decrypt messages.

Key exchange: Diffie-Hellman



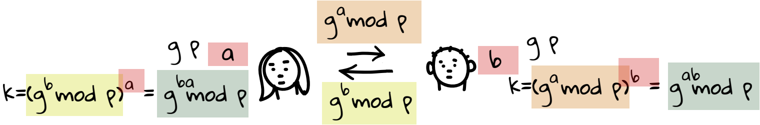
- Alice chooses randomly some a and computes $A := g^a$.
- Bob chooses randomly some b and computes $B := g^b$.
- Alice sends Bob A. Bob sends Alice B.
- Alice computes $k := B^a = (g^b)^a = g^{a*b}$.
- Bob computes $k := B^a = (g^a)^b = g^{a*b}$.
- Now Alice and Bob can use k as their secret key to encrypt and decrypt messages.

Key exchange: Diffie-Hellman



- Alice chooses randomly some a and computes $A := g^a$.
- Bob chooses randomly some b and computes $B := g^b$.
- Alice sends Bob A. Bob sends Alice B.
- Alice computes $k := B^a = (g^b)^a = g^{a*b}$.
- Bob computes $k := B^a = (g^a)^b = g^{a*b}$.
- Now Alice and Bob can use k as their secret key to encrypt and decrypt messages.

Key exchange: Diffie-Hellman



- Alice chooses randomly some a and computes $A := g^a$.
- Bob chooses randomly some b and computes $B := g^b$.
- Alice sends Bob A. Bob sends Alice B.
- Alice computes $k := B^a = (g^b)^a = g^{a*b}$.
- Bob computes $k := B^a = (g^a)^b = g^{a*b}$.
- Now Alice and Bob can use k as their secret key to encrypt and decrypt messages.

Key exchange: Diffie-Hellman. Primivite Roots

The simplest and the original implementation of the protocol uses the multiplicative group of integers modulo p, where p is prime, and g is a primitive root modulo p.

In modular arithmetic, a number g is a primitive root modulo n if every number a coprime to n is congruent to a power of g modulo n.

2 is a primitive root mod 5, because for every number a relatively prime to 5, there is an integer z such that $2^z \equiv a$.
All the numbers relatively prime to 5 are 1, 2, 3, 4, and each of these (mod 5) is itself (for instance 2 (mod 5) = 2) :

- $2^0 = 1,\ 1 \pmod 5 = 1$, so $2^0 \equiv 1$
- $2^1 = 2,\ 2 \pmod 5 = 2$, so $2^1 \equiv 2$
- $2^3 = 8,\ 8 \pmod 5 = 3$, so $2^3 \equiv 3$
- $2^2 = 4,\ 4 \pmod 5 = 4$, so $2^2 \equiv 4$.

For every integer relatively prime to 5, there is a power of 2 that is congruent.

4 is not a primitive root mod 5, because for every number relatively prime to 5 (again, 1, 2, 3, 4) there is not a power of 4 that is congruent. Powers of 4 (mod 5) are only congruent to 1 or 4. There is no power of 4 that is congruent to 2 or 3:

- $4^0 = 1,\ 1 \pmod 5 = 1$
- $4^1 = 4,\ 4 \pmod 5 = 4$
- $4^2 = 16,\ 16 \pmod 5 = 1$
- $4^3 = 64,\ 64 \pmod 5 = 4$

and the pattern continues...

Key exchange: Diffie-Hellman Example

- Alice and Bob publicly agree to use a modulus $p = 23$ and base $g = 5$
- Alice chooses *a* secret integer $a = 4$, then sends Bob $A = g^a \ mod \ p$:

  $A = 5^4 \ mod \ 23 = 4$
- Bob chooses a secret integer $b = 3$, then sends Alice $B = g^b \ mod \ p$:

  $B = 5^3 \ mod \ 23 = 10$
- Alice computes $k = B^a \ mod \ p$:

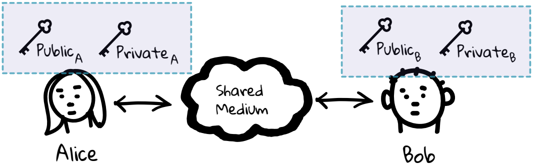
  $k = 10^4 \ mod \ 23 = 18$
- Bob computes $k = A^b \ mod \ p$:

  $k = 4^3 \ mod \ 23 = 18$
- Alice and Bob now share a secret (the number 18)

Contingut

Asymmetric Key Scheme

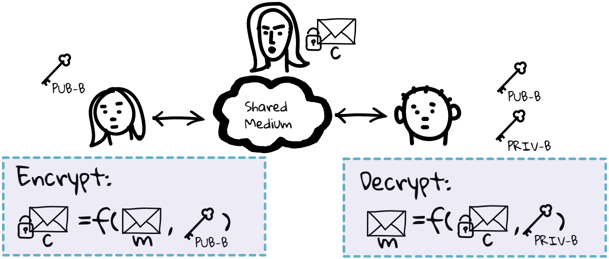
Asymmetric encryption schemes rely on two **keys** for each user.



These keys have the property that what is **encrypted** with one of them will be **decrypted** with the other.

Asymmetric Key Scheme: Confidentiality

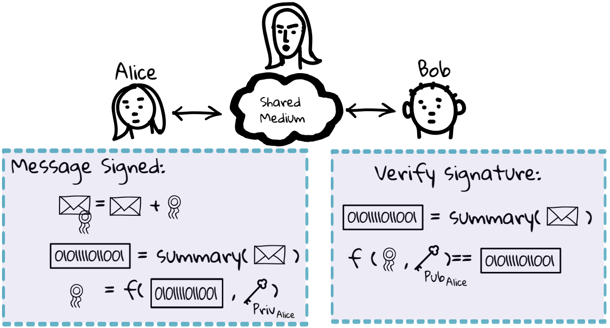
Public-key cryptography, or asymmetric cryptography, is a cryptographic system that uses **pairs** of keys: public keys, which may be disseminated widely, and private keys, which are known only to the owner.



In such a system, any person can encrypt a message using the receiver's public key, but that encrypted message can only be decrypted with the receiver's private key.

Asymmetric Key Scheme: Authentication/Integrity

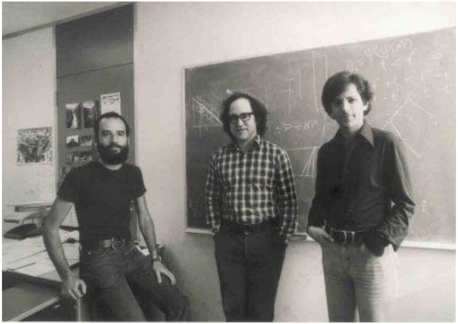
A message is signed by summarising the message using a hash function, for example. Then, the summary is encrypted using the sender's private key.



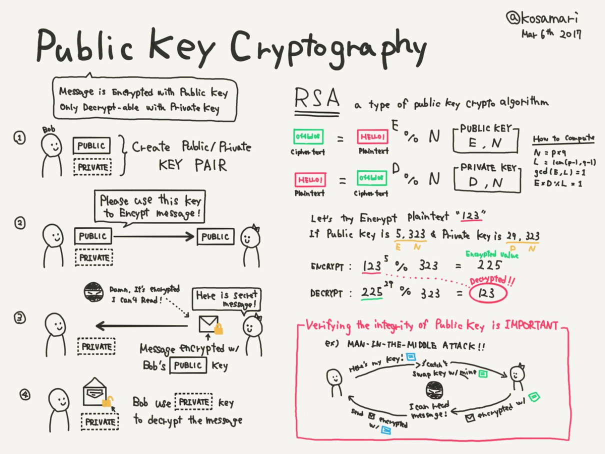
The recipient of the message can **verify** the integrity of the message by computing the summary of the received message and compare it to the result of decrypting with the sender's public key the signature.

Contingut

RSA (Rivest-Shamir-Adleman)



RSA (visual)


@kosamari
Mar 6th 2017
Public Key Cryptography
Message is Encrypted with Public key
Only Decrypt-able with Private key
1
Bob
PUBLIC
PRIVATE
Create Public/Private KEY PAIR
2
Please use this key to Encypt message!
PUBLIC
PRIVATE
PUBLIC
3
Damn, It's encrypted I can't Read!
Here is secret message!
PRIVATE
Message encrypted w/ Bob's PUBLIC Key
4
PRIVATE
Bob use PRIVATE key to decrypt the message
RSA a type of public key crypto algorithm
Cipher text = Plaintext E % N
Plaintext = Cipher text D % N
PUBLIC KEY E , N
PRIVATE KEY D , N
How to Compute
N = p*q
gcd(E,L) = 1
E*D%L = 1
Let's try Encrypt plaintext "123"
If Public key is 5, 323 & Private key is 29, 323
ENCRYPT : 123^5 % 323 = 225
Encrypted value
DECRYPT : 225^29 % 323 = 123
Decrypted!!
Verifying the integrity of Public Key is IMPORTANT
ex) MAN-IN-THE-MIDDLE ATTACK!!
Here's my key!
I can read message!

RSA (recipe)

Algorithm Key generation for RSA public-key encryption

SUMMARY: each entity creates an RSA public key and a corresponding private key.
Each entity A should do the following:

1. Generate two large random (and distinct) primes p and q, each roughly the same size.
2. Compute $n = pq$ and $\phi = (p-1)(q-1)$. (See Note 8.5.)
3. Select a random integer e, $1 < e < \phi$, such that $\gcd(e, \phi) = 1$.
4. Use the extended Euclidean algorithm (Algorithm 2.107) to compute the unique integer d, $1 < d < \phi$, such that $ed \equiv 1 \pmod{\phi}$.
5. A's public key is (n, e); A's private key is d.

Security is based on the hardness of **factorization**. Given $n = p.q$, no known efficient algorithm to recover p and q.

RSA (encrypt decrypt)

Algorithm RSA public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key (n, e).
   (b) Represent the message as an integer m in the interval $[0, n-1]$.
   (c) Compute $c = m^e \bmod n$ (e.g., using Algorithm 2.143).
   (d) Send the ciphertext c to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Use the private key d to recover $m = c^d \bmod n$.

Security is based on the hardness of **factorization**. Given $n = p.q$, no known efficient algorithm to recover p and q.

RSA (toy example)

- Key generat on:
  - Choose two prime numbers, for example $p=5$, $q=11$
  - Calculate n so that $n = p \times q = 55$
  - Calculate ϕ as ϕ = (p-1) x (q-1) = 40
  - Choose an e such that it must be coprime with ϕ, for example e=7
  - Choose d as the inverse of e modulo ϕ, that is, $e \times d = 1 \ (mod\ \phi)$. Since $7 \times 23 = 161 = 4 \times 40 + 1$, then d = 23
  - Public Key: (e, n)
  - Private Key: (d, n)

RSA (toy example)

- Message Creation:
  - Choose a toy message, for example ("Hello"). Let's start with "H" (8). Our plain text (m = 8)
- Encryption:
  - Generate the encrypted text: $c = m^e \pmod{n} = 8^7 \pmod{55} = 2$
- Decryption:
  - Decrypt the encrypted text: $m = c^d \pmod{55} = 2^{23} \pmod{55} = 8$

Factorisation example

```
import java.util.ArrayList;
import java.util.List;
import java. util. Scanner;

public class PrimeFactorsEffective {
    public static List<Double> primeFactors(double numbers) {
        double n = numbers;
        List<Double> factors = new ArrayList<Double>();
        for (double i = 2; i <= n / i; i++) {
            while (n % i == 0) {
                factors.add(i);
                n /= i;
            }
        }
        if (n > 1) {
            factors.add(n);
        }
        return factors;
    }

    public static void main(String[] args) {
                double n;
        Scanner readme = new Scanner(System.in);
        System.out.println("Enter p");
        double p = readme.nextDouble();
                System.out.println("Enter q");
        double q = readme.nextDouble();
                n = p * q;

                long startTime = System.nanoTime();
                System.out.println("Primefactors of " + n);
        for (Double d : primeFactors(n)) {
            System.out.println(d);
        }

                long endTime = System.nanoTime();
                long duration = (endTime - startTime);
                System.out.println("Factor time:"+ duration);

                startTime = System.nanoTime();
                n = p * q;
                endTime = System.nanoTime();
                duration = (endTime - startTime);
                System.out.println("Multiplication time:"+ duration);

    }
}
```

Función de Euler $\phi(n)$

- El Indicador o Función de Euler $\phi(n)$ nos entregará el número de elementos del CRR.
- Podremos representar cualquier número n de estas cuatro formas:
  - a) n es un número primo.
  - b) n se representa como $n = p^k$ con p primo y k entero.
  - c) n es el producto $n = p*q$ con p y q primos.
  - d) n es un número cualquiera, forma genérica:

$$n = p_1^{e1} * p_2^{e2} * \ldots * p_t^{et} = \prod_{i=1}^{t} p_i^{ei}$$

Función $\phi(n)$ de Euler cuando n = p

Caso 1: n es un número primo

Si n es primo, $\phi(n)$ será igual a CCR menos el 0.

$$\boxed{\phi(n) = n - 1}$$

Si n es primo, entonces CRR = CCR - 1 ya que todos los restos de n, excepto el cero, serán primos entre sí.

Ejemplo

CRR(7) = {1,2,3,4,5,6} seis elementos

$\therefore \phi(7) = n - 1 = 7-1 = 6$

$\phi(11) = 11-1 = 10$; $\phi(23) = 23-1 = 22$

Esta expresión se usará en los sistemas de cifra de ElGamal y DSS.

Función $\phi(n)$ de Euler cuando $n = p*q$

Caso 3: $n = p*q$ (con p y q primos)

$$\phi(n) = \boxed{\phi(p*q) = \phi(p)*\phi(q) = (p-1)(q-1)}$$

De los $p*q$ elementos del CCR, restaremos todos los múltiplos de $p = 1*p, 2*p, \ldots (q-1)*p$, todos los múltiplos de $q = 1*q, 2*q, \ldots (p-1)*q$ y el cero.

$$\phi(p*q) = p*q - [(q-1) + (p-1) + 1] = \underbrace{p*q - q - p + 1}_{(p-1)(q-1)}$$

Esta expresión se usará en el sistema de cifra RSA.

Teorema de Euler

Dice que si mcd (a,n) = 1 $\Rightarrow$ $a^{\phi(n)} \bmod n = 1$

Ahora igualamos $a * x \bmod n = 1$ y $a^{\phi(n)} \bmod n = 1$

$$\therefore \quad a^{\phi(n)} * a^{-1} \bmod n = x \bmod n$$

$$\therefore \quad x = a^{\phi(n)-1} \bmod n$$

El valor x será el inverso de a en el cuerpo n

Nota: Observe que se ha *dividido* por a en el cálculo anterior. Esto se puede hacer porque mcd (a, n) = 1 y por lo tanto hay un único valor inverso en el cuerpo n que lo permite.

RSA (demonstration)

Proof that decryption works. Since $ed \equiv 1 \pmod{\phi}$, there exists an integer k such that $ed = 1 + k\phi$. Now, if $\gcd(m, p) = 1$ then by Fermat's theorem (Fact 2.127),

$$m^{p-1} \equiv 1 \pmod{p}.$$

Raising both sides of this congruence to the power $k(q-1)$ and then multiplying both sides by m yields

$$m^{1+k(p-1)(q-1)} \equiv m \pmod{p}.$$

On the other hand, if $\gcd(m, p) = p$, then this last congruence is again valid since each side is congruent to 0 modulo p. Hence, in all cases

$$m^{ed} \equiv m \pmod{p}.$$

By the same argument,

$$m^{ed} \equiv m \pmod{q}.$$

Finally, since p and q are distinct primes, it follows that

$$m^{ed} \equiv m \pmod{n},$$

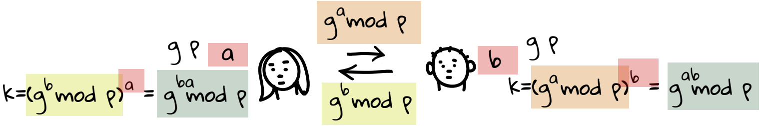
and, hence,

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Contingut

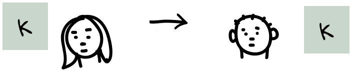
Xifratge basat en el logaritme discret: ElGamal

ElGamal és un criptosistema de clau pública basat en Diffie-Hellman i el problema del logaritme discret.

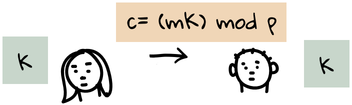


Xifratge basat en el logaritme discret: ElGamal

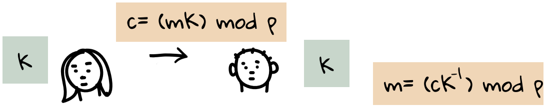
Xifratge basat en el logaritme discret: ElGamal



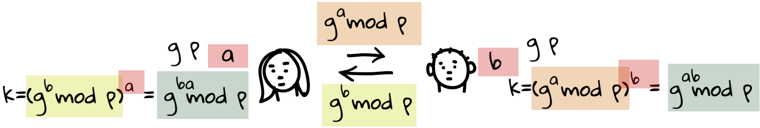
Xifratge basat en el logaritme discret: ElGamal



Xifratge basat en el logaritme discret: ElGamal



Diffie-Hellman: generació d'una clau compartida simètrica



- Alice tria de manera aleatòria un a i calcula $A := g^a$.
- Bob tria de manera aleatòria b i calcula $B := g^b$.
- Alice envia a Bob A. Bob envia a Alice B.
- Alice calcula $k := B^a = (g^b)^a = g^{a*b}$.
- Bob calcula $k := B^a = (g^a)^b = g^{a*b}$.
- Ara Alice i Bob poden fer servir k com el seu secret per xifrar i desxifrar.