Cryptology part 2 Keränen/ Teeriaho Ramk 2006

# 5. Public key cryptosystems

*Public Key Cryptosystems*



## ■ 5.1 History of public key cryptography

### ■ Kerchoff principle

*Motto: Auguste Kerckhoff 1835 - 1903:*

*"The security of a cryptosystem must depend only on the key and not on the secrecy of any other part of the system."*

Many cryptoalgorithms ( like DES ) were meant to be kept secret. When algorithms leaked out, safety was in danger. The architects of DES did not follow Kerckhoff's principle. In late 1970's first public key encryption methods were created. They obeyed Kerchoff's principle.

### Diffie's and Hellmann's lecture 1977

Two US scientists *Diffie and Hellman* presented in 1977 at a conference of cryptology two new ideas which made a small revolution if cryptology.

1) They sketched out an encryption system, where every user had two keys: a public and a private key. Messages should be encrypted with recipients public key, and the recipient should decrypt them using his private key. This is the basic idea of public key cryptography. Diffie and Hellman did not present a functioning algorithm. One of the first public key algorithms was presented by Rivest, Shamir and Adlemann one year later. It is called RSA.

2) Diffie and Hellman had also another great idea: a secure, simple and fast way of agreeing on a symmetric key for DES and other block ciphers.

This solution is called Diffie - Hellman key agreement protocol. It is used for example in PGP encryption software.

*Martin Edward Hellman (1945 -) is a cryptologist famous for his invention of public key cryptography in cooperation with Withfiedl Diffie (1944-). Diffie has worked since 1991 for Sun Microsystems laboratory and Hellman is an emeritus assistant professor of MIT (source: Wikipedia)*

### Existing public key algoritms

The first public key algorithm was published in 1978 by *Merkle and Hellman.* It was called *knapsack cryptosystem*. It is based of one of the *hard problems* of mathematics: *knapsack problem*. It was very popular in 1980's but popularity vanished, when matematicians found a solution to the knapsack problem.

A little later (1978) *Rivest , Shamir ja Adlemann* published **RSA - algorithm**, which still is a standard among PK algorithms. RSA is based on the difficulty of factoring large integers. If an effective way of factoring large integers is found, that will be the end of RSA .

In 1984 *Taher Elgamal* published a PK algorithm, which is based on DLP , *discrete logarithm problem in* $Z_p$. **Elgamal** is used in some verions of the popular PGP software.

PK algorihms are slow : only 1/50 of the speed of DES: The secure key size is 1024 bits for RSA and Elgamal.

In 1990's a new generation of PK algorithms were published based on cyclic groups on elliptic curves. . They are called **Elliptic Curve Cryptosystems** (ECC). Behind ECC exists the discrete logarithm problem on Elliptic Curves (ECDLP), which is mathematically more difficult than DLP in $Z_p$. The secure key length is in ECC about 180 bits, which is much shorter than in RSA or Elgamal.

## 5.2 RSA cryptosystem

**RSA - algorithm**

**1) Public keys**
User A generates two large primes p and q and publishes the product **n = pq**.
Then he chooses an integer e , where $1 < e < (p-1)(q-1)$ , and GCD(e , (p-1)(q-1)) = 1.

User A:s **public RSA key is the pair** **( n , e )**

2) **Decryption key of A is calculated as** $\mathbf{d = e^{-1} \bmod (p-1)(q-1)}$

3) When another user sends a message to A , he uses A's public keys for encryption

ciphertext is $\mathbf{c = m^e \bmod n}$

4) A decrypts the ciphertext using his private exponent d

$\mathbf{m = c^d \bmod n}$

## Proof that algorithm works

Assume GCD( m, n) = 1 , which means that m is an element of multiplication group $Z_n^*$ .

The keys e and d are chosen so that $e*d = 1 \mod (p-1)(q-1)$.
Product $(p-1)(q-1) = \varphi(n)$ , where $\varphi$ is Euler's totient function.

The fact that algorithm works, is due to the following chain of equalities.
In the end Euler's theorem is used ( $a^{\varphi(n)} = 1 \mod n$ )

$c^d \pmod n = (m^e)^d \pmod n = m^{e\,d} \pmod n = m^{1+k\,\varphi(n)} \pmod n$
$= m^1 (m^{\varphi(n)})^k \pmod n = m\ 1^k \pmod n = m \pmod n$

Proof in case GCD(m,n)≠ 1.

It is obvious in this case, that GCD(m,n) is either p or q, because n has no other non-trivial divisors.
Wihtout a loss of generality we assume that GCD(m,n) = p. This means that m is either of form a p, where GCD(a,n) = 1 or m is a power of p.
We need a following lemma:

Lemma: If p and q are primes, then $p^q = p \mod q$

Proof: According to Fermat's theorem, $p^{q-1} = 1 \mod q$ . This means that
$p^{q-1} = 1 + k\,q$ for some integer k. Multiplying by p we get
$p^q = p + k\,pq$ . Hence
$p^q = p \pmod q$ m.o.t

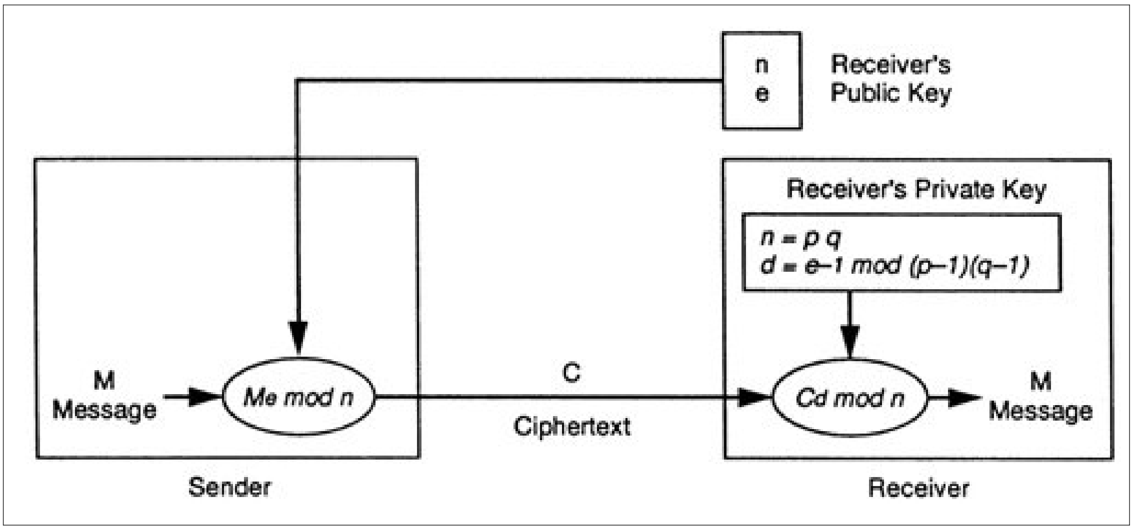
Next we proof that if m = ap, where GCD(a,n) = 1, then $m^{ed} = m \pmod n$.
To prove $(ap)^{ed} = ap \pmod n$ , it is enough to prove that $p^{ed} = p \pmod n$ , because we already have proved that equation $a^{ed} = a \pmod n$ is ture for a.

Now $p^{ed} = p^{1+k(p-1)(q-1)} = p^{1+kpq-kq-kp+k} = p^{kpq-kq}\, p^{-kp+k+1}$
$= (p^{q})^{kp-k}\, p^{-kp+k+1} = p^{kp-k}\, p^{-kp+k+1}$ ( previous lemma)
$= p^{kp-k-kp+k+1} = p$ (mod n) , which completes the proof in case m = ap.

If m = $p^u$ , where u ≥2 is an integer, then using the previous result:

$$m^{ed} = (p^u)^{ed} = (p^{ed})^u = p^u = m \pmod{n}$$

**RSA principle**




## ■ Example of RSA encryption

1. Alice chooses primes p = 1987 and q = 2309 and public exponent e = 33227

```
p = 1987;  q = 2309;
{n = p * q, e = 33227}    (* A:s public keys *)

{4587983, 33227}
```

2. Alice checks the validity of e (condition GCD(e, $\varphi$(n) = 1)

```
φ = (p - 1) (q - 1)  (* see part 1 *)
GCD[e, φ]

4583688
```

```
1
```

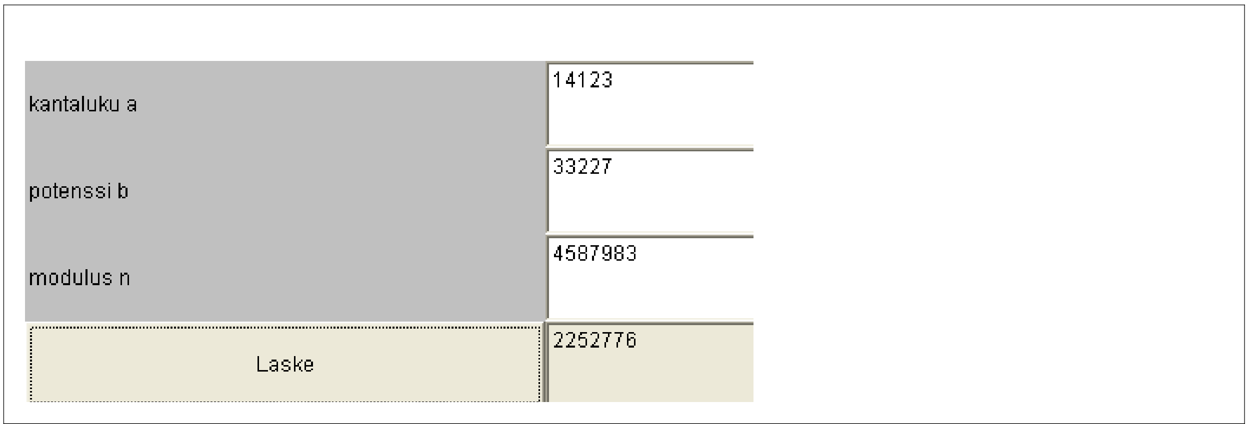
Alice calculates herself a edcryption key d , which is the invers of e  mod (p–1)(q–1).
(In *Mathematica* *you can do it with PowerMod* )

```
d = PowerMod[e, -1, (p - 1) * (q - 1)]

657611
```

Another user  Bob sends  Alice a message, which is encoded to integer m = 14123. Bob encrypts the message using  Alice's public keys.

```
m = 14123;
c = PowerMod[m, e, n]

2252776
```

In Web-address http://tl.ramk.fi/~jouko.teeriaho/krypto/krypto.html you find a **Java applet which performs powermod, if you do not have access to *Mathematica software.***




Alice decrypts the message using  d

```
opened = PowerMod[c, d, n]

14123
```

Result is the original message

## ■ 5.3 RSA security

To be able to calculate decryption key d = $e^{-1}$ mod (p-1)(q-1) , on has to factorize n.

Factorization of large integers is one of the "hard problems of mathematics ", which have been explored for centuries.

Even today the best algorithms are so slow, that it is not possible to factorize a 700 bit integer with large prime factors in a reasonable time.

On *RSA - challenges* web-site RSA Security challenges the public to try factorization . Today the next 212 digit integer it the next unfactorized challenge number. Anyone who succeeds , gets 30 000 $ reward.

7403756347956171282804679609742957314259318888923128908493623263897276503402826627689199641962511784399589433050212758537011896809828673317327310893090055250511687706329907239638078671008609696253793465056379635

In RSA a 1024 - bit modulus n is regarded to quarantee security. (However top secret files need 2048 bit keys.)

## ■ 5.4 About choosing RSA - keys

1) Primes p and q should not be too close to each other, because in that case an attack called *Wien's attack* can be used to factorize n.

2) On the other hand p and q should not be too far from each other. If either of them is small, it helps factorization.

3) Public exponent e should not be too small

4) Safe choices for p and q are regarded to be *strong primes*, which means that both p-1 and q-1 have large prime factors.

5) In some RSA versions the public exponent e is the same for all users, and only n is unique for all users.

# ■ 5.5 RSA in detail

For application of RSA you need to know some basic operations

## ▪ How to calculate decryption key d

Calculation of d is done using extended Euclid's algoritm:

These are presented in detail in part1 (mathematics of cryptology) of this material

In ***Mathematica* - software** function **ExtendedGCD[a,b**] returns the GCD of a and b as a linear combination of a and b .

```
ExtendedGCD[1745, 3137]

{1, {471, -262}}
```

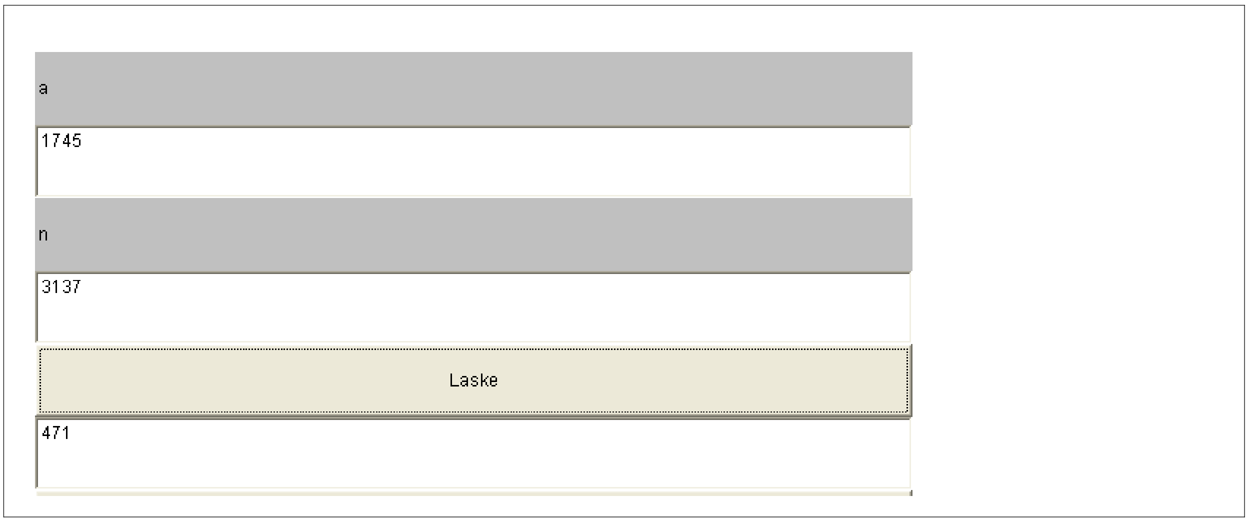
The result means that

1) GCD(1745,3137) = 1
2) => 1745 has an inverse mod 3137
3) 1 = 471*1745 - 262* 3137 (linear combination)
4) => 471*1745 mod 3137 = 1
=> 471 is the inverse required

*Mathematica* - function **PowerMod** can also be used in form PowerMod[a , -1 , n ]

```
PowerMod[1745, -1, 3137]

471
```

**In Java** there is a **BigInteger** - class, which has **inverse mod n** as a standard method . .
On the page http://tl.ramk.fi/~jouko.teeriaho/krypto/krypto.html you find a Java applet, which you can use for calculation of inverses.
Example of using Java - applet




- **How to encode message strings to a sequnce of large integers ?**

Easy way:
If the length of public key n is N bits, divide the binary message into blocks of length N-1 ( or shorter). Treat these bit blocks as binary numbers and transform them as decimal numbers.

Example: Let n = 4587983 and e = 33227 be RSA -public keys.
1. Calculate the length on n in bits.

```
n = 4587983;  IntegerDigits[n, 2]
Length[%]

{1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1}
```

```
23
```

Because n is 23 bit integer , we can divide the message into 22 bit blocks before encryption.
In the following we encrypt message "helsinki" using 22 bit blocks

Transformation to ASCII - codes

```
m1 = ToCharacterCode["helsinki"]

{104, 101, 108, 115, 105, 110, 107, 105}
```

From ASCII to 8 - bit bytes

```
m2 = IntegerDigits[m1, 2, 8]

{{0, 1, 1, 0, 1, 0, 0, 0}, {0, 1, 1, 0, 0, 1, 0, 1},
 {0, 1, 1, 0, 1, 1, 0, 0}, {0, 1, 1, 1, 0, 0, 1, 1},
 {0, 1, 1, 0, 1, 0, 0, 1}, {0, 1, 1, 0, 1, 1, 1, 0},
 {0, 1, 1, 0, 1, 0, 1, 1}, {0, 1, 1, 0, 1, 0, 0, 1}}
```

From 8 - bit bytes to bit stream

```
m3 = m2 // Flatten

{0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1,
 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1,
 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1}
```

From bit stream to 22 bit blocks

```
m4 = Partition[m3, 22]

{{0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1},
 {0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0}}
```

From 22 bit blocks to decimal numbers.

```
Map[Function[x, FromDigits[x, 2]], m4]

{1710427, 472726}
```

Now we encrypt the message with RSA

```
e = 33227;
c = PowerMod[m, e, n]

{1437592, 1265474}
```

## Another way of encoding the message to large integers

1. Encode the message characters to its characer codes
2. Divide the obtained sequence into blocks with length k , which is the
largest integer with $256^k < n$.
3. Interprete the blocks as the integer digits of numbers in base 256 and transform the blocks into corresponding large decimal numbers.
4. Use RSA - encryption algorithm

Example:

Keys are now

```
Prime[6000]

59359
```

```
p = 104729; q = 59359;
{n = p * q, e = 33227}  ;
d = PowerMod[e, -1, (p - 1) * (q - 1)];
```

Message m is transformer to ASCII codes

```
m1 = ToCharacterCode["helsinki"]

{104, 101, 108, 115, 105, 110, 107, 105}
```

In 8 bit ASCII there are 256 characters.
So 256 is the suitable base for the number system.

**Block size**

Suitable block size is the largest value of k satisfying $256^k < n$ ( here n = 6216608711)

```
256^4 < n
256^5 < n

True
```

```
False
```

Now $256^4 < n$, but $256^5 > n$, which means that a suitable block size is 4.

```
ToCharacterCode["Helsinki"]      (* kirjainten koodit *)

{72, 101, 108, 115, 105, 110, 107, 105}
```

Above character codes are used as the integer digits of two 256 - base numbers.

Blocks "hels" and " inki" are transformed to integers:

```
m = {72 * 256^3 + 101 * 256^2 + 108 * 256 + 115,
  105 * 256^3 + 110 * 256^2 + 107 * 256 + 105}

{1214606451, 1768844137}
```

Encryption using recipients A public keys:

```
c = PowerMod[m, e, n]

{5251775299, 2106055825}
```

A decrypts the message

```
opened = PowerMod[c, d, n]

{1214606451, 1768844137}
```

This is the original message

```
FromCharacterCode[Flatten[IntegerDigits[opened, 256]]]

Helsinki
```

## ■ 5.6 Elgamal

V. 1985 T. Elgamal published a cryptoalgorithm, which was based on discrete logarithm problem.
*( DLP : How to solve x from* $a^x = y \mod p$ *)*

### ▪ Principle ElGamal

The base paramters of the system are a large prime $p$ and a primitive element $a$ of $Z_p^*$ . Integers $p$ and $a$ are public.

**Keys:** Each user A chooses a random integer $x_A$ with $1 < x_A < p–1$.
The public key of A $y_A$ is obtained with
$y_A \equiv a^{x_A} \mod p$.

**Encryption:** Let m be a message from A to B. The message is first encoded to integers of $Z_p$. Assume B:s private key is $x_B$ and public key $y_B \equiv a^{x_B} \mod p$.
1) A chooses a random $k$ with $1 < k < p - 1$.
2) A calculates key $K \equiv y_B{}^k \mod p$ .
3) A encrypts the message M to a pair of integers ( $C_1$ , $C_2$ ) , where
$$C_1 \equiv a^k \mod p \quad \text{and} \quad C_2 \equiv K\,M \mod p .$$
The encryption function $e$ is a mapping $e$: $M \to (C_1, C_2)$ , which doubles the length og the message.

**Decryption**:
1) B calculates $K$ by $K \equiv C_1{}^{x_B} \mod p$ .
( $K \equiv y_B{}^k \equiv a^{x_B\,k} \equiv (a^k)^{x_B} \equiv C_1{}^{x_B} \mod p$ )
2) Then B obtains M from equation $C_2 \equiv K\,M$ mod p with
$$M \equiv K^{-1}\,C_2 \mod p$$

### ▪ Example of ElGamal

Let $p = 71$ and $a = 7$ .
Let private key of B be $x_B = 26$ . Then B's public key is $y_B \equiv a^{x_B} \mod p$ .

```
yB = Mod[7^26, 71]

3
```

A sends B a message $M = 30$ and chooses $k = 6$.
The A calculates $K \equiv y_B{}^k \mod p$ .

```
K = Mod[3^6, 71]

19
```

Next $M$ is cipher to a pair
$(C_1, C_2) \equiv (a^k \mod p, KM \mod p)$.

```
{Mod[7^6, 71], Mod[19 × 30, 71]}

{2, 2}
```

B receives a cipher ( 2 , 2 ).
Decryption:
1) First B calculates $K \equiv {C_1}^{x_B} \mod p$.

```
Mod[2^26, 71]

19
```

The B needs the inverse of $K = 19$ mod p

```
PowerMod[19, -1, 71]

15
```

2) Original message is $M \equiv K^{-1} C_2 \mod p$.

```
Mod[15 × 2, 71]

30
```

## ■ 5.7 Elliptic Curve Cryptosystems  ECC

RSA and Elgamal need  1024 bit - even 2048 bit keys. It makes them slow and also lots of memory is required. For example in smart cards memory is limited.

Most promising new alternatives are  **Elliptic Curve Cryptosystems (ECC)**,  which can be used for encryption, authentication, digital signatures.... ECC is based on  discrete logarithm problem of cyclic groups on  elliptic curves.

In ECC secyrty level of 1024  RSA is accieved with only 180 bit keys.   A company called Certicom (*www.certicom.com*)  is a provider of ECC systems.

*Some cryptographers are still suspicious about ECC, because they think that we do not know enough about the mathematical properties of elliptic curves. Maybe that is why RSA is still more popular.*