ELECS425F Computer and Network Security

Lec 04

Outline – Public Key Cryptography

- Drawbacks of symmetric key crypto.
- **P**ublic **K**ey **C**ryptography (PKC).
- Assessing security.
- One-way trapdoor functions.
- Knapsacks.
  - For encryption.
  - Super-increasing knapsacks.
  - Trapdoor knapsacks.

Outline – Public Key Cryptography

- RSA.
  - Encryption and decryption.
- Using RSA.
- Choosing p and q.
- Implementation considerations.
- Some comments on factoring.
- Finding and testing primes.
- Fast exponentiation.
- Assessing the security of RSA.

Drawbacks of symmetric key crypto

- So far we have studied **symmetric key cryptosystems** were the transmitter and receiver have the same key.
- They are also called
  - Private key cryptosystems
  - Secret key cryptosystems
  - Conventional cryptosystems

  but we will generally stick to the term symmetric.
- **Key management**, that is, generating good keys, distributing and storing them in a secure way, is a bottleneck in symmetric key cryptography.
- **Example:** For a network of N computer terminals, with pairwise secret keys, the total number of secret keys is N(N-1)/2. For N=100 there are 4950 keys, with 99 at each terminal and two copies of each across the network.

Drawbacks of symmetric key crypto

- Another significant drawback of symmetric key cryptography is that services such as **non-repudiation** cannot be achieved.
  - A **non-repudiation service** provides Alice protection against Bob later denying that some communication exchange took place. It allows the elements of the communication exchange to be linked with the transmitter.
  - In such a service Alice will have irrefutable evidence to support her claim.
- In symmetric key cryptography the secret information is shared between Alice and Bob, so whatever Alice can do Bob can do too.

Public Key Cryptography

- Diffie-Hellman (1976)
- Public Key Cryptography (PKC) provides solutions to both of the problems mentioned.
- In a PKC:
  - The encryption and decryption keys are different.
  - The decryption key cannot be deduced from the encryption key.
  - That is, keys come in pairs, (x, y), where x is the private and y is the public component of the key.

Public Key Encryption

- Each pair of keys satisfies the following two properties:
  - A plaintext encrypted with y can be decrypted with x. That is x determines the inverse of the encryption with key y.
  - Given a public key y it is computationally infeasible to discover the corresponding decryption key x.

Properties

- The first property is needed in all cryptosystems: Decryption is the inverse of encryption.
  - **$D_x(E_y(M)) = M$**
- In secret key cryptography **x** can be easily obtained from **y**.
- **Example:** In DES decryption sub-keys are the same as encryption sub-keys but applied in reverse order.
- Once **y** is known, **x** can be calculated and algorithms **E_y** and **D_x** are both known.

Properties

- In PKC, user Alice has a pair of keys.
  - Public component y_A generates a public transformation Ey_A for encryption.
  - Private component x_A generates a private transformation Dx_A for decryption.

Key Management is Much Easier!

Alice	y_A
Bob	y_B
Fred	y_F
Oscar	y_O
…	
…	

Public Key Directory

Scenario

- If Bob wants to send a message M to Alice, he checks the public directory to find Alice's public key, and forms the following cryptogram:

$$C = Ey_A(M)$$

- Alice can recover the message using her private key, as in

$$Dx_A(C) = Dx_A(Ey_A(M)) = M$$

Discussions

- Any Public Key Encryption System has to be at least Chosen-Plaintext Secure
  - Why?
- How can we construct a PKC which is resistant to such an attack?
- PKC's can be realized by using **trapdoor one-way** functions.

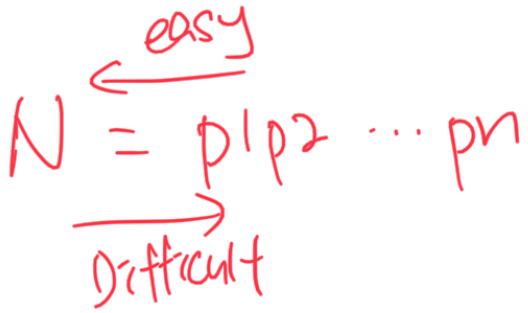
Trapdoor one-way functions

- A function f is called one-way if for all M finding f(M) is easy, but knowing f(M) it is hard to find M.

- Example: Given n primes p1, p2, … , pn, it is easy to find their product

$$N = p_1 p_2 \ldots p_n.$$

- Given a large number N it is difficult to find its prime factors.



Trapdoor one-way functions

- A **trapdoor** is a piece of knowledge which makes it easier to find **M** from **f(M)**.

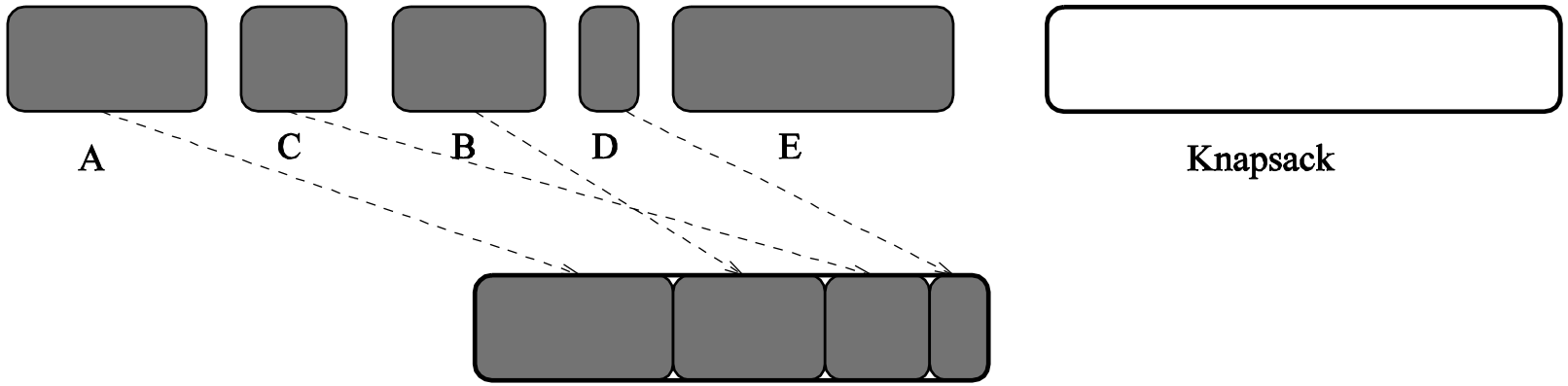
- A **trapdoor one-way function** is a function which looks like a one-way function but is equipped with a secret **trapdoor**. If this secret door is known, the inverse can be easily calculated.

Trapdoor one-way functions

- A trapdoor one-way function can be used to realize a PKC:
  - $\mathbf{E_y}$ is easy to compute (from the public component of the key **y**) but is hard to invert.
  - Knowledge of **x**, which is the private component of the key (i.e. it is the trapdoor), allows easy computation of $\mathbf{D_x}$ (inverting $\mathbf{E_y}$).
- We are going to look at two examples of building trapdoors in one-way functions. The first one has failed (knapsacks) and the second one has provided us with the most important PK cryptosystem (RSA).

Knapsacks (背包問題)

- The **knapsack problem** is derived from the notion of packing an odd assortment of packages into a container:



- *Find a subset of packages that fills the container.*
- In general, there is no efficient way of finding the subset (NP-hard): You have to try all possibilities。

- Alternatively:
  - We are given an ordered set of n positive integers a_i, $1 \leq i \leq n$, and a target number T.
  - The ordered set of positive integers is referred to as a *cargo vector*.
  - The knapsack problem is to find a subset $S \subseteq \{1,\ldots n\}$ such that

$$\sum_{i \in S} a_i = T$$

- This is also called the **subset-sum problem**. This is known to be a *difficult* problem as there is no algorithm more efficient than exhaustive search in the general case.

- **Example:** $(a_1,a_2,a_3,a_4)=(2,3,5,7)$ and T=7.
  Solutions: (1,0,1,0), (0,0,0,1).

- In general we may have more than one solution.

Knapsacks for encryption

- Knapsacks are first implemented as **block ciphers**. For a block size of n bits we require a **key** or **cargo vector**, of positive integers a_i, $1 \leq i \leq n$, referred to as weights.

  a = $(a_1, a_2, a_3, ..., a_n)$

- Then to encrypt a message block of n bits:

  X = $(x_1, x_2, x_3, ..., x_n)$

  we find the number

$$T = \sum_{i=1}^{n} x_i a_i$$

and write it in a binary representation.

- **To decrypt**: The receiver knows T and all a_i, so goes through all possible plaintext blocks to find a block that satisfies the condition.

- In this system:
  - **Encryption is easy** and requires n additions. ☺
  - **Decryption is difficult**, even if the key is known. ☹

- Hence, the resulting cryptosystem is not useful.

- To construct an acceptable cryptosystem, Merkle & Hellman (1977) used a special case of the knapsack problem, involving *super-increasing knapsacks*, where decryption is easy ***if*** you have the key.

Super-increasing knapsacks

- A knapsack is super-increasing if each element of the cargo vector is greater then the sum of the preceding elements.
- **Example**: **a**=(1,2,4,8)

  Given T=14 say, it is easy to find $X=(x_1,x_2,x_3,x_4)$ such that $x_1+2x_2+4x_3+8x_4=14$.

i	a_i	Total	Include?
4	8	14	1
3	4	6	1
2	2	2	1
1	1	0	0

X=0111

- At each step the current target value with the largest unchecked element a_i of the cargo vector **a**. If the current target value is larger that element of the cargo vector must be included element (i.e. x_i=1) otherwise it cannot be (i.e. x_i=0).
- Example: **a**=(171,197,459,1191,2410), T=3798.
- What is X=(x_1,x_2,x_3,x_4,x_5)?

i	a_i	Total	Include?
5	2410	3798	1
4	1191	1388	1
3	459	197	0
2	197	197	1
1	171	0	0

X=01011

- As described the super-increasing knapsacks could be used as a symmetric key cryptosystem.

- But super-increasing knapsacks cannot be used directly for public key cryptography, because making the encryption key public *allows everyone to decipher* **X**!

- The idea of Merkle and Hellman (1977) was to start with an easy knapsack and then *disguise* it to *look difficult* for people without knowledge of the trapdoor.

- The trapdoor will only be known by the person who wants to decrypt the cryptogram. It will be their private key.

Trapdoor knapsacks

- Alices chooses a super-increasing knapsack, $a=(a_1,a_2,\ldots,a_n)$, as her **private key**.
- Then she chooses an integer **m** > $(a_1 + a_2+\ldots+a_n)$ called the **modulus**, and a random integer **w**, called the **multiplier**, which is relatively prime to **m**. Relatively prime means the largest common factor of **w** and **m** is 1. This condition implies there exists an **inverse** of **w modulo m**.
  - Note that if **m** is prime, then **w** can be any number.
- Alice's public key is **b** where

  b=$(b_1,b_2,\ldots,b_n)$ and $b_i=w*a_i \bmod m$

- Alice's secret key is **(a,m,w)**.
- **To encrypt**: Bob sends a message **X** to Alice by computing

$$T = \sum_{i=1}^{n} x_i b_i$$

- **To decrypt**: Alice receives T and …
  - Uses the inverse of **w**, $\mathbf{w^{-1}}$, to calculate

    $R = w^{-1}T \bmod m$.
  - Uses the easy knapsack **a** to find **X**, since **R=a.X**

Example

$a = (1, 4, 9, 15, 65, 130, 261)$

$\gcd(w, m) = 1$ $m = 541$, $w \Rightarrow 113$

$b = (1\times113, 4\times113, \dots 261\times113) \bmod 541$

$= (113, 452, 476, 72, 114, 312, 83, 279)$

$X = 11111101 \rightarrow$ [1 is element exist, vice versa]

$T = 1\times113 + 452\times113 + \dots 83\times0 + 279\times113$

$= 1848$

$R = 113^{-1} \bmod 541$ [Extend Euclidean Alogritm]

$\rightarrow r=0,\ q_2 = n_1^{-1} \bmod n_2,\ b_2 = n_2^{-2} \bmod n_1$

$= 158$

$R = 158\times1848 \bmod 541 = 385$

$$
\begin{aligned}
R &= w^{-1}T \pmod m \\
&= w^{-1}\sum_{i=1}^{n} b_i x_i \pmod m \\
&= w^{-1}\sum_{i=1}^{n} (wa_i)x_i \pmod m \\
&= \sum_{i=1}^{n} (w^{-1}wa_i)x_i \pmod m \\
&= \sum_{i=1}^{n} a_i x_i \pmod m \\
&= a.X
\end{aligned}
$$

Brute force attack

- For those who do not know the secret trapdoor, decryption requires an exhaustive search through all 2^n possible **X**.

- **Example**:
- Earlier we had: **a**=(171,197,459,1191,2410)
- Let w = 2550 and m = 8443.

  b=(5457, 4213, 5316, 6013, 7439) is the public cargo vector.

Example: PKC using a trapdoor knapsack

- Secret key **a**=(2, 5, 10, 21).
- Trapdoor: **m**=39>38 (sum of weights).
  - Random w, **w**=15.
  - gcd algorithm, gcd(39,15)=3≠1
  - Another w, **w**=11.
  - gcd(39,11)=1
  - Inverse of 11 mod 39 = 32

- Using **w**=11 and **m**=39 disguise **a**.
  b_1=2*11=22 (mod 39)
  b_2=5*11=16 (mod 39)
  b_3=10*11=32 (mod 39)
  b_4=21*11=36 (mod 39)
- Public key: **b**=(22, 16, 32, 36).

- Secret key: **a**=(2, 5, 10, 21).

(w,m)=(11, 39)

(remember also that $\mathbf{w}^{-1}$=32 mod 39)

- To decrypt a message T=48…
  - Find R = 48*32 mod 39 = 15
  - Use the cargo vector **a** to decrypt R and find X=(0, 1, 1, 0).

  - We see this agrees with the public key version too (16+32=48).
  - In useful sized examples it would be not trivial to find the solution using just the public key, although it is easy to check the answer using it.

Multiple layers and the fall of knapsacks

- The disguising process can be repeated a number of times on the cargo vector to create more and more difficult knapsack problems **(w_1, m_1)**, **(w_2, m_2)**,... . The result is, in general, not equivalent to a single **(w, m)** transformation.
- Adleman (1982) broke the knapsack cryptosystem by taking a public cargo vector and finding a pair **(w', m')** that would convert it back to a super-increasing cargo vector sufficient for decrypting encrypted messages.
- Merkle was confident enough that the multiple layers were still secure to offer a reward of $1000 to anyone who could break a multiple layer knapsack.
- In 1984, Brickell announced the destruction of a knapsack system, with 40 iterations and a hundred weights (elements of the cargo vector), in about one hour of Cray-1 time.
  - Merkle gave him the money ☺

PKC progress…

- Since 1976, many PKC have been proposed. Many of these are broken and proven to be insecure.
- Among the ones that are considered secure, many are impractical because of either large key or large message expansion.
- PKC algorithms are, however, all slow and unsuitable when fast encryption is needed. However, they can provide high security and can be used when low processing rates are acceptable or when high security is needed.
- Remember that a PKC can be used for:
  - confidentiality (secrecy).
  - authentication (digital signatures).
- Two algorithms that can easily be used for both *secrecy and authentication* are RSA and ElGamal.
  - We are going to look at RSA next.

RSA

Trapdoor one-way function

- The RSA Public--Key Cryptosystem (**R**ivest, **S**hamir and **A**dleman (1978)) is the most popular and versatile PKC.

- It is the *de facto* standard for PKC.
- It supports *secrecy and authentication* and can be used to produce *digital signatures*.
- RSA uses the knowledge that it is *easy* to find primes and multiply them together to construct composite numbers, but it is *difficult* to factor a composite number.

The Euler phi Function

$$\phi(n) = \left|\{x : 1 \le x \le n \quad and \quad \gcd(x,n) = 1\}\right|$$

- $\phi(2) = |\{1\}| = 1$
- $\phi(3) = |\{1,2\}| = 2$
- $\phi(4) = |\{1,3\}| = 2$
- $\phi(5) = |\{1,2,3,4\}| = 4$
- $\phi(6) = |\{1,5\}| = 2$

- $\phi(37) = 36$
- $\phi(21) = (3–1)\times(7–1) = 2\times6 = 12$

The RSA algorithm

1. Choose two primes p and q. Compute n = pq and m=ϕ(n)= (p-1)(q-1).
   - ϕ(n) is Euler's totient function: It is the number of positive integers less than n that are relatively prime to n.
2. Choose e, $1 \leq e \leq m - 1$, such that gcd(e,m)=1.
3. Finds d such that ed=1 mod m.
   - This is possible because of the choice of e.
   - d is the multiplicative inverse of e modulo m and can be found using the Extended Euclidean algorithm.
4. The **Public key** is **(e, n)**.

   The **Private key** is **(d, n)**.

Encryption and decryption

- Let **X** denote a plaintext block, and **Y** denote the corresponding ciphertext block.
- Let **(z_A, Z_A)** denote the private and public components of Alice's key.

- If Bob want to encrypt a message **X** for Alice. He uses Alice's public key and forms the cryptogram:

$$\mathbf{Y = E_{Z_A}(X) = X^e \bmod n}$$

- When Alice wants to decrypt **Y**, she uses the private component of her key **z_A=(d,n)** and calculates

$$\mathbf{X=D_{z_A}(Y) = Y^d \bmod n}$$

- **X** and **Y** are both integers in {0, 1, 2, …, n-1}.

Encryption and decryption

- Choose two primes p=47 and q=71 ⇨ n = pq = 3337.
- Choose e such that it is relatively prime to $\phi(n) = 46x70 = 3220$.
  - e.g. e = 79.

gcd (46, 70) => 1

- Compute $d \equiv e^{-1} \bmod \phi(n)$ using extended Euclidean algorithm.
  - $d \equiv 79^{-1} \pmod{3220} = 1019$
- Public key (n, e) = (3337,79)
- Private key (n, d) = (3337,1019)

- Encrypt 688 ⇨ $688^{79} \bmod 3337 = 1570$
- Decrypt 1570 ⇨ $1570^{1019} \bmod 3337 = 688$

- Why does this work out?

Correctness of RSA

- **Euler's Theorem**: Let n be a composite. Then $a^{\phi(n)} \equiv 1 \pmod{n}$ for any integer a which is ***relatively prime*** to n.

P & Q PRIME 1
N = PQ 2
ED ≡ 1 MOD (P-1)(Q-1) 3
C = M^E MOD N 4
M = C^D MOD N 5

$$C^D \equiv (M^E)^D \equiv M^{k\phi(N)+1} \equiv (M^{\phi(N)})^k M \equiv M \pmod{N}$$

(4) (3) (1)

$E \cdot D = 1 \bmod (p-1)(q-1)$

$= k(p-1)(q-1) + 1$

An RSA public key directory

User	(n,e)
Alice	(85,23)
Bob	(117,5)
Fred	(4757,11)

- An important property of the RSA algorithm is that encryption and decryption are the same function: both exponentiation modulo n.

$$E_{Z_A}(D_{Z_A}(X)) = X$$

- Example:
  - First decrypt: $2^{13} = 41 \bmod 143$
  - Then encrypt: $41^{37} = 2 \bmod 143$

- This is the basis of using RSA for authentication.

Using RSA

- The RSA system can be used for:

1. **Confidentiality**: To hide the content of a message **X**, A sends $\mathbf{E_{Z_B}(X)}$ to B.
2. **Authentication**: To ensure integrity of a message **X**,
   - Alice *signs* the message by using her decryption key to form $D_{z_A}(X)$ and sends $(X, D_{z_A}(X)) = (X, S)$ to Bob.
   - When Bob wants to *verify the authenticity* of the message:
     - He computes $X' = E_{z_A}(S)$.
     - If X'=X the message is accepted as authentic and from Alice.
   - Both **message integrity** and **sender authenticity** are verified.
     - This is true because even one bit change to the message can be detected, and because D_{z_A} is known only to Alice.
   - This method is inefficient. We will see later that Alice may compute a hash value of **X** and then apply D_{z_A} to the result.
   - We will talk about Digital Signatures later anyway.

use own private key to "sign" the encrypted message

Choosing p and q

prime
p · q ⇒ n easy
n → p · q difficult

- The main conditions on p and q are:
  - They must be at least 100 decimal digits long (about 330 bits).
  - They must be of similar size, say both 100 digits.

- To choose each number the user does the following:
  - Randomly choose a random number b which is 100 digits long, or whatever length is appropriate.
  - Checks to see if b is prime. Usually using a probabilistic primality testing algorithm.
  - If b is not prime, choose another value for b.

Finding primes

- An algorithm to generate all primes does not exist.

- However, given a number n, there exist efficient algorithms to *check whether it is prime or not*. Such algorithms are called **primality testing algorithms**.
- As mentioned earlier, prime generation for RSA is basically just a matter of guessing and testing.

- Primality Testing: Deterministic algorithms for proving primality are non-trivial and are only advisable on high performance computers. Probabilistic tests allow an *educated guess* as to whether a candidate number is prime or not.
  - This means that the probability of the guess being wrong can be made arbitrarily small.

Lehman's test

- Let **n** be an odd number. For any number **a** define

$$e(a,n) = a^{\frac{n-1}{2}} \bmod n$$

$$G = \{e(a,n) : a \in Z_n^*\}$$

where Z_n^*={1,2,…,n-1}.

- **Example**: n=7

$3^3=1$, $2^3=1$, $3^3=6$, $4^3=1$, $5^3=6$, $6^3=6$ → G={1,6} → only 2 distinct value

$e(a,n) = a^{\frac{n-1}{2}} = a^3$, $a \in Z_n^*$

If $G=\{1, n-1\}$

Lehman's test

- **Lehman's theorem**:

  If n is odd, $G=\{1,n-1\}$ if and only if n is prime.

Example: n=15 isn't prime: (n-1)/2=7

$2^7=8 \bmod 15$, $3^7=12 \bmod 15$

→ $n=15$

$e(a,n) = a^{\frac{n-1}{2}} = a^7$

$G=\{1,14\}$

→ $2^7=8 \neq 14$

∴ 15 is not prime

Example: n=13 (prime): (n-1)/2=6

$2^6=-1 \bmod 13$, $3^6=1 \bmod 13$, $4^6=1 \bmod 13$,

$5^6=-1 \bmod 13$, $6^6=-1 \bmod 13$, $7^6=-1 \bmod 13$,

$8^6=-1 \bmod 13$, $9^6=1 \bmod 13$, $10^6=1 \bmod 13$,

$11^6=1 \bmod 13$, $12^6=1 \bmod 13$.

Lehman's test

- Thus, we have the following test:

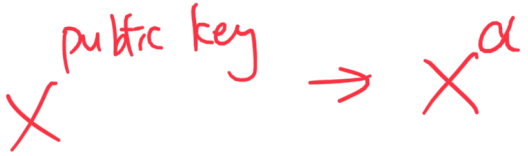
if (gcd(a,n) >1) *return*('composite')
else
  if ($a^{(n-1)/2}$=1) *or* ($a^{(n-1)/2}$=-1)
    return('prime_witness')
  else
    return('composite')

If for a given **n** the test returns prime_witness for 100 randomly chosen **a**, then the probability of **n** not being not prime (i.e. being a composite disguised as a prime) is negligible.

Lehman's test

- Not too many numbers must be tested before a prime can be found. This is justified because of the known distribution of primes.
- Techebycheff showed that the percentage of positive integers less than **n** which are primes is 1/ln(n). $P(\mathbb{R}^+ \leq n) = \frac{1}{\ln(n)}$
- Thus if we randomly choose an **odd** integer less than **n**, the chance that it is a prime is about 2/ln(n).
- For $n=10^{100}$ this is about 1/15.

Fast exponentiation



- Exponentiation can be performed by repeated multiplication. In general, we can use the *square and multiply* technique.
- To calculate X^{α}:
  1. Write α in base two:

     $\alpha = \alpha_0 2^0 + \alpha_1 2^1 + \alpha_2 2^2 + \ldots \alpha_{n-1} 2^{n-1}$;
  2. Calculate X^{2^i}, $1 \leq i \leq n-1$.
  3. Use $X^{\alpha} = (X^{2^0})^{\alpha_0} * (X^{2^1})^{\alpha_1} \ldots (X^{2^{n-1}})^{\alpha_{n-1}}$ and

     Multiply the X^{2^i} for which α_i is not zero.

- **A partial example**: N=179, e=73.
  X=2 → Y=2^{73} mod 179.
  73=64+8+1=$2^6+2^3+2^0$

  Y=$2^{64+8+1}=2^{64}*2^8*2^1$
  This is only a partial example because we haven't looked at calculated the elements of the last line.

Precomputation:

$X^2=X*X$

$X^4= X^{2^2}=X^{2*}X^2$

…

$X^{2^{n-1}}= X^{2^{n-2}}* X^{2^{n-2}}$

This is a total of n-1 multiplications, all mod N

- **Example**: N=1823, n=$\log_2 1823$=11.
  - Calculate Y=5^{375} mod N
- Precomputation:

$Y = 5^{375} \bmod 1823$

X^1	5	X^2	25	X^4	625
X^8	503	X^{16}	1435	X^{32}	1058
X^{64}	42	X^{128}	1764	X^{256}	1658
X^{512}	1703	X^{1024}	1639		

375=256+64+32+16+4+2+1.

5^{375}=5*25*625*1435*1058*42*1658

= 591 mod 1823

$5^{375} = 5^{256+64+32+16+4+2+1} = 5^{256} \cdot 5^{64} \cdot 5^{32} \cdot 5^{16} \cdot 5^4 \cdot 5^2 \cdot 5^1$

Caution!

Common modulus attack: Consider a group of users who's public keys consists of the same modulus and different exponents.

- If an intruder intercept two cryptograms where
  - They are encryptions of the same message with different keys.
  - The two encryption exponents do not have any common factor. Then the attacker can find the plaintext.

- Let us consider why:

- The enemy knows e_1, e_2, N, Y_1 and Y_2, and furthermore that
  $Y_1 = X^{e_1} \bmod N$ and $Y_2 = X^{e_2} \bmod N$

- Since e_1 and e_2 are relatively prime, the Extended Euclidean algorithm can be used to find **a** and **b** such that
  $ae_1 + be_2 = 1$.

  But then $Y_1^{a} Y_2^{b} = X^{ae_1} * X^{be_2} = X^{ae_1+be_2} = X$

- **Using a common modulus among a number of participants is not advisable!**

Summary

- Drawbacks of symmetric key crypto.
- **P**ublic **K**ey **C**ryptography (PKC).
- One-way trapdoor functions.
- Knapsacks.
- RSA.
- Finding and testing primes.
- Fast exponentiation.
- Assessing the security of RSA.

- RSA.
  - Encryption and decryption.
- Using RSA.
- Choosing p and q.
- Implementation considerations.
- Some comments on factoring.
- Finding and testing primes.
- Fast exponentiation.
- Assessing the security of RSA.