6 Public Key Cryptography

6.1 Key Management

Keys
There are two main types of key used in cryptography:

- Static keys (long-term keys): these are keys which are to be in use for a long time period. The compromise of a static key is usually considered to be a major problem.
- Session keys (short-term keys): these are keys which have a short life-time, and are usually used to provide confidentiality for the given time period. The compromise of a session key should only result in the compromise of that session's secrecy and it should not affect the long-term security of the system.

A weak key is a key that, when used with a specific cipher, makes the cipher behave in some undesirable way.

Key Distribution
Possible approaches to key distribution:

- Physical distribution: this is not scalable and the security no longer relies solely on the key.
- Distribution using symmetric key protocols.
- Distribution using public key protocols.

If we have n users each of whom wish to communicate securely with each other then we would require $\frac{n(n-1)}{2}$ secret keys.
One solution to this problem is for each user to hold only one key with which they communicate with a central authority, so n users will only require n keys.
When two users wish to communicate they generate a session key which is only to be used for that message; this can be generated with the help of the central authority and a security protocol.

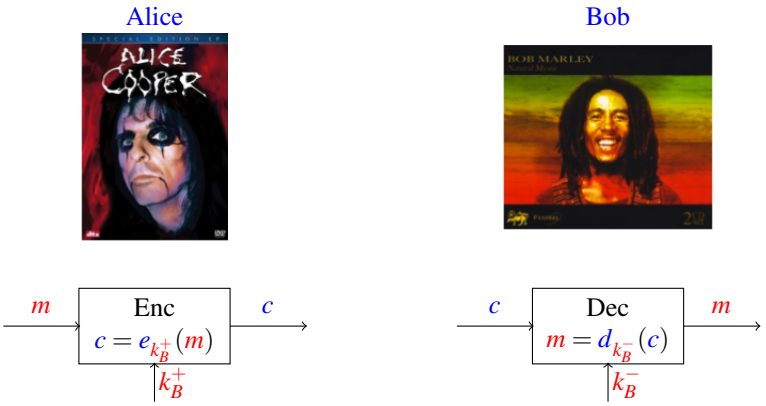
6.2 Public Key Cryptography

Public Key Cryptography
With symmetric cryptography we can already provide confidentiality, integrity and authentication, so why invent something new?

- Key distribution problem
- Solution: instead of having a "lock" with a key that can lock and unlock, we could use a lock which has a key that can only lock and a second key that can only unlock.

- Alice distributes locks and locking keys in public and only keeps the unlocking key private.
- Then, everybody can grab a lock and the locking key, put a message in a box, lock the box and send it to Alice.
- Only Alice can unlock the box and read the message.

Public Key Cryptography



Public key cryptography uses a different public key k^+ and private key k^- for encryption and decryption.
Here, Bob generates a key pair (k_B^+, k_B^-) and gives k_B^+ to Alice.

Public Key Cryptography

In public key cryptography, each user has a key pair, which consists of a public key (made public, used for encryption) and a private key (kept secret, used for decryption).

- Public key cryptography (asymmetric cryptography) realizes the idea described on the previous slide.
- The only (currently) known way to implement this idea in practice is to make use of (old) number theoretic problems.
- These problems lead to so-called one-way trapdoor functions.
  - These functions are easy to compute in one direction.
  - However, computing them in the other direction (computing the inverse) is very hard, without knowing some secret information.

Public Key Cryptography

The concept of public key cryptography was first thought of in 1976 in a paper by Diffie and Hellman: New Directions in Cryptography.
The first realisation of the concept appeared a few years later: (RSA).

In the same 1976 paper, Diffie and Hellman described a method for establishing a shared secret key over an insecure communication channel: Diffie-Hellman key exchange.
It turns out that a lot of these ideas had already been developed in the UK GCHQ, but were subject to an official secrets act:

- Around 1970, James H. Ellis conceived the principles of public key cryptography.
- In 1973, Clifford Cocks invented a solution resembling the RSA algorithm.
- In 1974, Malcolm J. Williamson developed the Diffie-Hellman key exchange.

6.3 One-Way Functions

One-Way Functions
A function $f : X \to Y$ is a one-way function iff:

- For all $x \in X$ it is very easy or efficient to compute $f(x)$.
- For almost all $y \in Y$, finding an $x \in X$ with $f(x) = y$ is computationally infeasible.

A trapdoor one-way function is a one-way function $f : X \to Y$, but given some extra information, called the trapdoor information, it is easy to invert f i.e. given $y \in Y$, it is easy to find $x \in X$ such that $f(x) = y$.

One-Way Functions
Candidate one-way functions:
Multiplication:

- Given primes p and q, compute $N = pq$.
- This is very easy to compute, since we just multiply p and q.
- The inverse problem: given N find p and q (factoring).

Modular exponentiation:

- Given N and an element $a \in \mathbb{Z}_N$, compute $b \equiv a^m \pmod{N}$.
- This can be computed efficiently using squaring and multiplication.
- The inverse problem: given $N, a, b \in \mathbb{Z}_N$ find m such that $b \equiv a^m \pmod{N}$ (discrete logarithm problem).

6.4 Hard Problems

Hard Problems
Suppose you are given N but not p, q such that $N = pq$:

- Integer Factorisation Problem (IFP): Find p and q.
- RSA Problem (RSAP): Given $c \in \mathbb{Z}_N$ and integer e with $\gcd(e, \phi(N)) = 1$ find m such that $m^e \equiv c \pmod{N}$.
- Quadratic Residuosity Problem (QUADRES): Given a determine whether there is an x such that $a \equiv x^2 \pmod{N}$.
- Square Root Problem (SQROOT): Given a find x such that $a \equiv x^2 \pmod{N}$.

Hard Problems
Given an abelian group $(G, \otimes)$ and $g \in G$:

- Discrete Logarithm Problem (DLP):
  Given $y \in G$ find x such that $g^x = y$.
  The difficulty of this problem depends on the group G:
  - Very easy: polynomial time algorithm e.g. $(\mathbb{Z}_N, +)$
  - Rather hard: sub-exponential time algorithm e.g. $(GF(p), \times)$
  - Very hard: exponential time algorithm e.g. Elliptic Curve groups
- Diffie-Hellman Problem (DHP):
  Given $a = g^x$ and $b = g^y$ find $c = g^{xy}$.
- Decisional Diffie-Hellman Problem (DDH):
  Given $a = g^x$, $b = g^y$ and $c = g^z$, determine whether $z = xy$.

Hard Problems
IFP and DLP are believed to be computationally very difficult.
The best known algorithms for IFP and DLP are sub-exponential.
There is, however, no proof that IFP and DLP must be difficult.
Efficient quantum algorithms exist for solving IFP and DLP.
IFP and DLP are believed to be computationally equivalent.

Hard Problems
Some other hard problems:

- Computing discrete logarithms for elliptic curves.
- Finding shortest/closest vectors in a lattice.
- Solving the subset sum problem.
- Finding roots of non-linear multivariate polynomial equations.
- Solving the braid conjugacy problem.

6.5 Reductions

Reductions
We will reduce one hard problem to another, which will allow us to compare the relative difficulty of the two problems i.e. we can say:

Problem A is no harder than Problem B

Let A and B be two computational problems.
A is said to polytime reduce to B ($A \leq_P B$) if:

- There is an algorithm which solves A using an algorithm which solves B
- This algorithm runs in polynomial time if the algorithm for B does

Assume we have an efficient algorithm to solve problem B.
We then use this to give an efficient algorithm for problem A.

Reductions
Here we show how to reduce DHP to DLP i.e. we give an efficient algorithm for solving the DHP given an efficient algorithm for the DLP.
Given g^x and g^y we wish to find g^{xy}.
First compute y = DLP(g^y) using the assumed efficient algorithm.
Then compute $(g^x)^y = g^{xy}$
So DHP is no harder than DLP i.e. DHP $\leq_P$ DLP.
Remark: in some groups we can show that DHP is equivalent to DLP.

Reductions
Here we show how to reduce DDH to DHP i.e. we give an efficient algorithm for solving the DDH given an efficient algorithm for the DHP.
Given elements g^x, g^y and g^z, determine if $z = xy$
Using the assumed efficient algorithm to solve DHP, compute g^{xy} = DHP(g^x,g^y).
Then check whether $g^{xy} = g^z$.
So DDH is no harder than DHP i.e. DDH $\leq_P$ DHP.
Remark: in some groups we can show that DDH is probably easier than DHP.

Reductions
Here we show how to reduce SQROOT to IFP i.e. we give an efficient algorithm for solving SQROOT given an efficient algorithm for IFP.
Given $z = x^2 \pmod N$ we wish to compute x:

- Using the assumed efficient algorithm for IFP, find the prime factors p_i of N.
- Compute $\sqrt{z} \pmod{p_i}$ (can be done in polynomial time)
- Recover $\sqrt{z} \pmod N$ using CRT on the data $\sqrt{z} \pmod{p_i}$

We have to be a little careful if powers of p_i greater than one divide N.
So SQROOT is no harder than IFP i.e. SQROOT $\leq_P$ IFP.

Reductions
Here we show how to reduce IFP to SQROOT i.e. we give an efficient algorithm for IFP given an efficient algorithm for SQROOT.
Given $N = pq$ we wish to compute p and q:

- Compute $z = x^2$ for a random $x \in \mathbb{Z}_N^*$
- Compute $y = \sqrt{z} \pmod N$ using the assumed efficient algorithm for SQROOT.
  - There are four possible square roots, since there are two factors.
  - With fifty percent probability we have $y \neq \pm x \pmod N$
- Factor N by computing $\gcd(x-y, N)$.

So IFP is no harder than SQROOT i.e. IFP $\leq_P$ SQROOT.
So IFP and SQROOT are computationally equivalent: SQROOT $\equiv_P$ IFP.

Reductions
Here we show how to reduce RSAP to IFP i.e. we give an efficient algorithm for solving RSAP given an efficient algorithm for IFP.
Given $c = m^e \pmod N$ and the integer e, find m:

- Find the factorization of $N = pq$ using the assumed efficient algorithm.
- Compute $\phi(N)$ as $\phi(N) = (p-1)(q-1)$
- Using the XGCD compute $d = 1/e \pmod{\phi(N)}$
- Finally, recover $m = c^d \pmod N$

So RSAP is no harder than IFP i.e. RSAP $\leq_P$ IFP.
There is some evidence (although slight) that RSAP might be easier.

6.6 Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange
For Diffie-Hellman, we select a large prime $p(> 2^{400})$ and a generator g, then:

1. A chooses a random x such that $1 < x < p-1$.
2. $A \rightarrow B : g^x \pmod p$
3. B chooses a random y such that $1 < y < p-1$.
4. $B \rightarrow A : g^y \pmod p$
5. A computes $K = (g^y)^x \pmod p$.
6. B computes $K = (g^x)^y \pmod p$.
7. A and B now share the secret K.

Diffie-Hellman Key Exchange
A toy example ($p = 11$, $g = 5$).
Private keys:

- $x = 3$
- $y = 4$

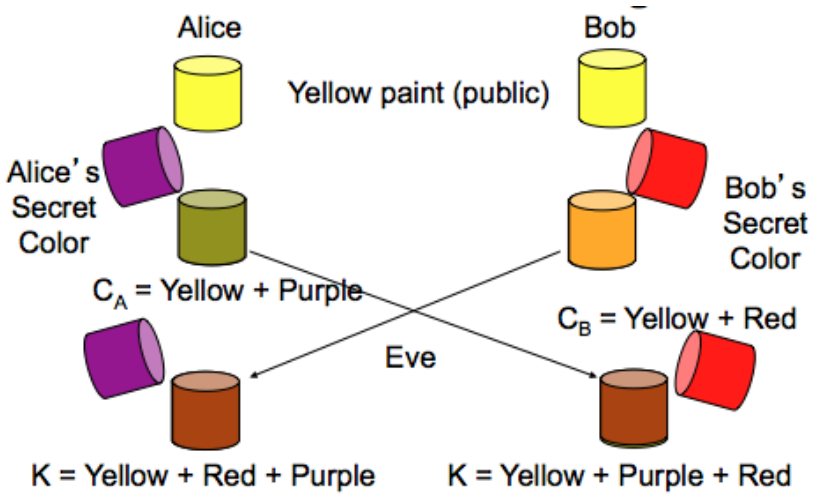
Public keys:

- $X = g^x \pmod p = 5^3 \pmod{11} = 125 \pmod{11} = 4$
- $Y = g^y \pmod p = 5^4 \pmod{11} = 625 \pmod{11} = 9$

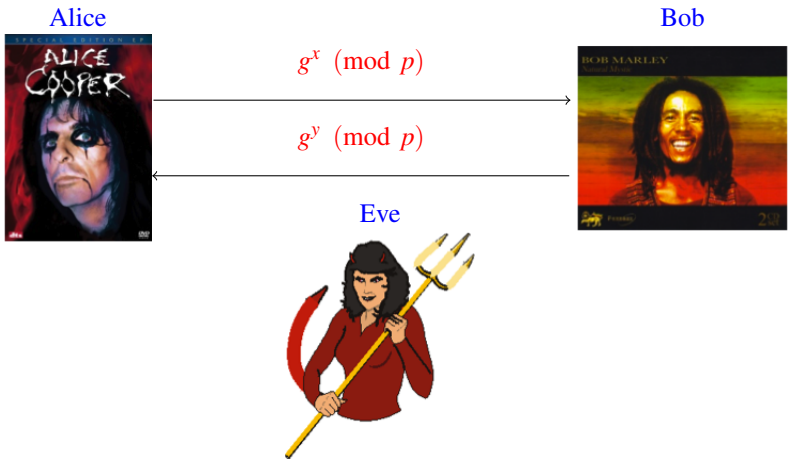
Shared secret:

- $Y^x \pmod p = 9^3 \pmod{11} = 729 \pmod{11} = 3$
- $X^y \pmod p = 4^4 \pmod{11} = 256 \pmod{11} = 3$

Diffie-Hellman Key Exchange
This is analagous to paint mixing (Simon Singh):

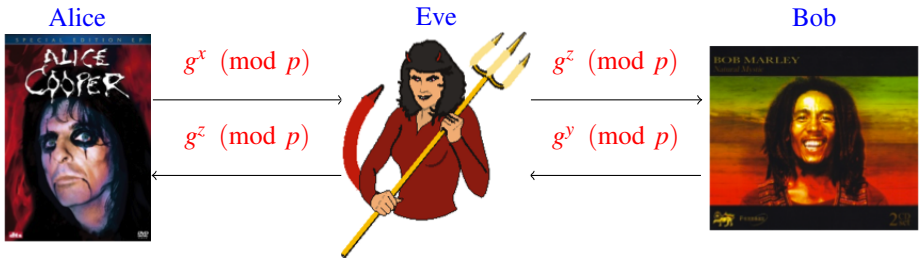


Diffie-Hellman Key Exchange
Consider an eavesdropper Eve.



Diffie-Hellman Key Exchange
Bob and Alice use $g^{xy} \pmod p$ as a shared key.
Eve can see $g^x \pmod p$ and $g^y \pmod p$
Note $g^x g^y = g^{x+y} \neq g^{xy} \pmod p$.
If Eve can find x or y, system is broken.
If Eve can solve the discrete log problem, then she can find x or y.
This is a difficult problem - modular exponentiation is a one-way function.

Diffie-Hellman Key Exchange
This key exchange is susceptible to a man in the middle attack:



Eve shares secret $g^{xz} \pmod p$ with Alice and secret $g^{yz} \pmod p$ with Bob.
Alice and Bob do not know that Eve exists.

6.7 Asymmetric Ciphers

Public Key Encryption
The basic idea of public key encryption is:

Message	+	Bob's Public Key	=	Ciphertext
Ciphertext	+	Bob's Private Key	=	Message

Anyone with Bob's public key can send Bob a secret message.
But only Bob can decrypt the message, since only Bob has the private key.
All one needs to do is look up Bob's public key in some directory.

Digital Signatures
If a user encrypts data with their private key, then anyone can decrypt the data using the user's public key:

- This approach does not provide confidentiality.
- However, anyone receiving encrypted data can be sure that it was encrypted by the owner of the key pair.
- This is the basis for a digital signature.
- This allows us to obtain authentication and non-repudiation.

The basic idea of using public key cryptography for digital signatures is:

$$\begin{aligned} \text{Message} + \text{Alice's Private Key} &= \text{Signature} \\ \text{Message} + \text{Signature} + \text{Alice's Public Key} &= \text{Yes/No} \end{aligned}$$

Only Alice can have encrypted the message, since only Alice has the private key.

Digital Signatures
There are two distinct types of signature scheme:

1. Schemes with Message Recovery.
   - Allow the original message to be recovered from the signature.
   - Signature validation does not require the original message.
   - Only practical with small messages.
   - Message must contain some redundancy otherwise we cannot determine if the signature is valid.
2. Signature Schemes with Appendix.
   - Require the original message as part of the validation algorithm.

A digital signature scheme with message recovery can be turned into a scheme with appendix by using a hash function.

Asymmetric Ciphers
There are a number of widely used asymmetric ciphers:

- RSA (Rivest, Shamir & Adleman)
  - Most widely known and used asymmetric cipher.
  - Both the private and public keys can be used for encryption.

  - Varying key sizes (512 bits, 1024 bits, ...).

- DSA (Digital Signature Algorithm) - a variant of the ElGamal Signature Scheme.
  - Digital Signature Standard (DSS) - NIST standard.
  - Key size 512 to 1024 bits.
  - Only the private key can be used for encryption, i.e., this cipher can only be used for creating digital signatures.

6.8 RSA

RSA
Rivest, Shamir, Adleman (1978): A Method for Obtaining Digital Signatures and Public Key Cryptosystems.
Key generation: Generate two large primes p and q of at least 512 bits.

- Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$
- Select a random integer e, $1 < e < \phi(N)$, where $\gcd(e, \phi(N)) = 1$.
- Using the extended Euclidean algorithm compute the unique integer d, $1 < d < \phi(N)$ with $ed \equiv 1 \pmod{\phi(N)}$.

Public key = (e,N) which can be published.
Private key = (d,N) which needs to be kept secret.

RSA
Encryption: if Bob wants to encrypt a message for Alice, he does the following:

- Obtains Alice's authentic public key (e,N).
- Represents the message as a number $0 < m < N$.
- Computes $c = m^e \pmod{N}$.
- Sends the ciphertext c to Alice.

Decryption: to recover m from c, Alice uses the private key (d,N) to recover $m = c^d \pmod{N}$.

RSA
Recall that $ed \equiv 1 \pmod{\phi(N)}$, so there exists an integer k such that:

$$ed = 1 + k\phi(N)$$

If $\gcd(m,p)=1$:

- By Fermat's Little Theorem we have $m^{p-1} \equiv 1 \pmod{p}$.

- Taking $k(q-1)-$th power and multiplying by m yields:

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

If $\gcd(m,p) = p$, then $m \equiv 0 \pmod{p}$ and $(*)$ is valid again.
Hence, in all cases $m^{ed} \equiv m \pmod{p}$ and by a similar argument we have $m^{ed} \equiv m \pmod{q}$.
Since p and q are distinct primes, the CRT leads to:

$$c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)+1} = m \pmod{N}$$

RSA: Toy Example
Choose primes $p = 7$ and $q = 11$.
Key Generation:

- Compute N =77 and $\phi(N) = (p-1)(q-1) = 6 \times 10 = 60$.
- Choose e = 37, which is valid since gcd(37,60) = 1.
- Using the extended Euclidean algorithm, compute d = 13 since $37 \times 13 \equiv 481 \equiv 1 \pmod{60}$.
- Public key = (37,77) and private key = (13,77).

Encryption: suppose m = 2 then:

$$c \equiv m^e \pmod{N} \equiv 2^{37} \pmod{77} \equiv 51$$

Decryption: to recover m compute:

$$m \equiv c^d \pmod{N} \equiv 51^{13} \pmod{77} \equiv 2$$

RSA
The security of RSA relies on the difficulty of finding d given N and e.
RSAP can be reduced to IFP, since if we can find p and q, then we can compute d.
Therefore, if factoring is easy we can break RSA.

- Currently 768-bit numbers are the largest that have been factored.
- For medium term security, best to choose 1024-bit numbers.

RSA
Assume for efficiency that each user has:

- The same modulus N
- Different public/private exponents (e_i, d_i)

Suppose user one wants to find user two's d_2:

- User one computes p and q since they know d_1
- User one computes $\phi(N) = (p-1)(q-1)$
- User one computes $d_2 = (1/e_2) \pmod{\phi(N)}$

So each user can then find every other users key.

RSA
Now suppose the attacker is not one of the people who share a modulus.
Suppose Alice sends the message m to two people with public keys:

- (N, e_1),(N, e_2), i.e. $N_1 = N_2 = N$.

Eve can see the messages c_1 and c_2 where:

- $c_1 = m^{e_1} \pmod{N}$
- $c_2 = m^{e_2} \pmod{N}$

RSA
Eve can now compute:

- $t_1 = e_1^{-1} \pmod{e_2}$
- $t_2 = (t_1 e_1 - 1)/e_2$

Eve can then compute the message from:

$$\begin{aligned} c_1^{t_1} c_2^{-t_2} &= m^{e_1 t_1} m^{-e_2 t_2} \pmod{N} \\ &= m^{1+e_2 t_2} m^{-e_2 t_2} \pmod{N} \\ &= m^{1+e_2 t_2 - e_2 t_2} \pmod{N} \\ &= m^1 = m \pmod{N} \end{aligned}$$

RSA: Example
Take the public keys as:

- $N = N_1 = N_2 = 18923$
- $e_1 = 11$, $e_2 = 5$

Take the ciphertexts as:

- $c_1 = 1514$, $c_2 = 8189$
- The associated plaintext is $m = 100$

Then $t_1 = 1$ and $t_2 = 2$
We can now compute the message from: $c_1^{t_1} c_2^{-t_2} = 100 \pmod{N}$

RSA
Modular exponentiation is computationally intensive.
Even with the square-and-multiply algorithm, RSA can be quite slow on constrained devices such as smart cards.
Choosing a small public exponent e can help to speed up encryption.
Minimal possible value: $e = 3$

- 2 does not work since $\gcd(2, \phi(N)) = 2$
- However, sending the same message encrypted with 3 different public keys then breaks naive RSA.

RSA
Suppose we have three users:

- With public moduli N_1, N_2 and N_3
- All with public exponent $e = 3$

Suppose someone sends them the same message m
The attacker sees the messages:

- $c_1 = m^3 \pmod{N_1}$
- $c_2 = m^3 \pmod{N_2}$
- $c_3 = m^3 \pmod{N_3}$

Now the attacker, using the CRT, computes the solution to: $X = c_i \pmod{N_i}$
To obtain $X \pmod{N_1 N_2 N_3}$

RSA
So the attacker has: $X \pmod{N_1 N_2 N_3}$
But since $m^3 < N_1 N_2 N_3$ we must have $X = m^3$ over the integers.
Hence $m = X^{1/3}$.
This attack is interesting since we find the message without factoring the modulus.
This is evidence that breaking RSA can be easier than factoring.

RSA
Choosing a small private key d results in security weaknesses.

- In fact, d must have at least $0.3\log_2 N$ bits

The Chinese Remainder Theorem can be used to accelerate exponentiation with the private key d.

- Based on the CRT we can replace the computation of $c^{d \pmod{\phi(N)}} \pmod{N}$ by two computations $c^{d \pmod{p-1}} \pmod{p}$ and $c^{d \pmod{q-1}} \pmod{q}$ where p and q are 'small' compared to N.

RSA: Example
Take $N_1 = 323$, $N_2 = 299$, $N_3 = 341$
The attacker sees $c_1 = 50$, $c_2 = 268$, $c_3 = 1$ and wants to determine the value of m
The attacker computes via CRT: $X = 300763 \pmod{N_1 N_2 N_3}$
The attacker computes, over the integers: $m = X^{1/3} = 67$
Lessons

- Plaintexts should be randomised before applying RSA.
- Very small exponents should be avoided for RSA encryption.
  - Recommended value: $e = 65537 = 2^{16} + 1$
  - Runtime for encryption: 17 modular multiplications.
- Runtime of RSA: fast encrypt/slow decrypt

Raw RSA
Raw RSA is not semantically secure.
To make it semantically secure we use a padding scheme to add randomness and redundancy.
Note: Some old padding schemes (e.g. PKCS#1.5) are now considered weak.
Bellare and Rogaway have a scheme called OAEP (Optimal Assymetric Encryption Padding) which adds this randomness and redundancy.
OAEP is now used in the PKCS standards and hence in most Internet protocols.

RSA
The OAEP padding scheme is as follows.
Let r denote a random value and m the plaintext s.t. $(m||0||r) < N$
Let G and H be pseudo-random functions (e.g. crypto hash functions).
We define the padding scheme: $\text{OAEP}(m||0||r) = (G(r) \oplus (m||0))||(H(m||0 \oplus G(r)) \oplus r)$
Encryption is performed by: $c = (\text{OAEP}(m||0||r))^e \pmod{N}$
Decryption is performed by: $(m||z||r) = \text{OAEP}^{-1}(c^d \pmod{N})$
Followed by verification that $z = 0$.
If $z = 0$, the decrypted message is m.
Otherwise, the ciphertext was forged, and the decrypted value should be ignored.

6.9 ElGamal

ElGamal
ElGamal (1985): A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.

- Domain Parameter Generation: Generate a "large prime" p (> 512 bits) and generator g of the multiplicative group $\mathbb{Z}_p^*$,

- Key Generation: Select a random integer a, $0 < a < p-1$ and compute $h \equiv g^a \pmod{p}$.
- Public key = (p, g, h) which can be published.
- Private key = a which needs to be kept secret.

ElGamal
Encryption: Bob encrypts a message for Alice as follows:

- Obtains Alice's authentic public key (p, g, h).
- Represents the message as an integer m where $0 \leq m \leq p-1$.
- Generates a random ephemeral key k, with $0 < k < p-1$.
- Computes $c_1 = g^k \pmod{p}$ and $c_2 = mh^k \pmod{p}$.
- Sends the ciphertext $c = (c_1 || c_2)$ to Alice.

Decryption: to recover the message, Alice does the following:

- Uses the private key a to compute $c_1^{p-1-a} \pmod{p} \equiv c_1^{-a} \equiv g^{-ak}$.
- Recovers m by computing $(c_1^{-a})c_2 \equiv m \pmod{p}$.

Proof that decryption works: $(c_1^{-a})c_2 \equiv g^{-ak} m g^{ak} \equiv m \pmod{p}$.

ElGamal: Toy Example
Toy example:
Select prime $p = 17$.
Generator $g = 6$.
Alice chooses the private key $a = 5$ and computes:

$$g^a \pmod{p} \equiv 6^5 \pmod{17} \equiv 7$$

Alice's public key is $(p = 17, g = 6, h = 7)$, which can be published.
Alice's private key is $a = 5$ which she keeps secret.

ElGamal: Toy Example
To encrypt the message $m = 13$, Bob selects a random integer $k = 10$ and computes:

$$c_1 = g^k \pmod{p} = 6^{10} \pmod{17} = 15$$
$$c_2 = mh^k \pmod{p} = (13 \times 7)^{10} \pmod{17} = 9$$

Bob then sends the ciphertext $(c_1 || c_2)$ to Alice.
To decrypt, Alice first computes:

$$c_1{}^{p-1-a} \pmod{p} \equiv 15^{11} \pmod{17} \equiv 9 \pmod{17}$$

and recovers m by computing:

$$m \equiv 9 \times 9 \pmod{17} \equiv 13 \pmod{17}$$

6.10 Rabin Cryptosystem

Rabin Cryptosystem
Rabin (1979): Digitalized Signatures and Public Key Functions as Intractable as Factorization.
Breaking RSA has not been proven to be equivalent with IFP.
SQROOT and IFP are computationally equivalent.
The Rabin Cryptosystem is based on the difficulty of extracting square roots modulo $N = pq$.
The Rabin Cryptosystem was the first provably secure scheme.
Despite all its advantages over RSA it is not widely used in practice.

Rabin Cryptosystem
Key Generation:

- Generate two large primes p and q of roughly the same size.
- Compute $N = pq$.

Public key = N which can be published.
Private key = (p,q) which needs to be kept secret.
Note: we always take $p \equiv q \equiv 3 \pmod 4$, since this makes extracting square roots easy and fast.

Rabin Cryptosystem
Encryption: to encrypt a message Bob does the following:

- Obtain Alice's authentic public key N.
- Represent the message as an integer m where $0 \leq m \leq N-1$.
- Compute the ciphertext c as $c = m^2 \pmod N$

Decryption: to recover the plaintext m from c Alice computes:

$$m = \sqrt{c} \pmod N$$

Notice at first sight this uses no private information, but we need the factorization to be able to find the square root.

Rabin Cryptosystem
The decryption operation in the Rabin Cryptosystem hides a small problem:

- N is the product of two primes p and q.
- Therefore there are four possible square roots modulo N.
- On decryption obtain four possible plaintexts.
- Need to add redundancy to plaintext to decide which one to take.

Rabin encryption is very, very fast.
Decryption is made faster by the special choice of p and q.

Rabin Cryptosystem: Toy Example
Alice chooses the primes $p = 127$ and $q = 131$.
Note that both primes are congruent to 3 modulo 4.
Alice then computes $N = pq = 16637$.
Suppose Bob wants to encrypt the message $m = 4410$, then he computes the ciphertext as:

$$c = m^2 \pmod{N} = 16084$$

Rabin Cryptosystem: Toy Example
To decrypt the ciphertext c, Alice computes $\sqrt{16084} \pmod{16637}$ by computing:

- $\sqrt{16084} \pmod{127} \equiv \pm 16084^{32} \pmod{127} \equiv \pm 35 \pmod{127}$
- $\sqrt{16084} \pmod{131} \equiv \pm 16084^{33} \pmod{131} \equiv \pm 44 \pmod{131}$

Using the CRT, we obtain four possible square roots:

$$s \equiv \pm 4410 \text{ or } \pm 1616$$

Therefore, the possible messages are:

$$s \equiv 4410 \text{ or } 12227 \text{ or } 1616 \text{ or } 15021$$

6.11 Other Public Key Algorithms

Other Public Key Algorithms

Encryption algorithm	Security depends on
Goldwasser-Micali encryption	QUADRES
Blum-Goldwasser encryption	SQROOT
Chor-Rivest encryption	Subset sum problem
XTR	DLP
NTRU	Closest vector problem in lattices
Schnorr signature	DLP
Nyberg-Rueppel signature	DLP
Digital Signature Algorithm (DSA)	DLP
Elliptic Curve DSA (ECDSA)	DLP in elliptic curves

6.12 Public Key Cryptography in Practice

Public Key Cryptography in Practice
Main drawback of public key cryptography is the inherently slow speed.

- A few schemes are faster, but require huge keys, so are impractical.

Therefore, public key schemes are not used directly for encryption.
Instead, they are used in conjunction with secret key schemes.

- Encryption is performed by secret key schemes (e.g. AES).
- Key agreement is performed by public key schemes (e.g. RSA or Diffie-Hellman).

Public Key Cryptography in Practice
In secret key schemes key sizes have increased from 56-64 bits to 128 bits to provide sufficient security.

- Keys of 128 bits are large enough to thwart any practical attack, as long as the cipher does not have weakness due to its design.

In public key schemes, considerably longer keys are required (e.g. 1024 bits for RSA, 512 bits for ElGamal).

- Keys are not uniformly selected from all the possible keys with the same length.
- The number of keys is (slightly) smaller than the number of values of the same length as the keys.
- The key inherits information due to the properties of the cipher.

Public Key Cryptography in Practice
Public key cryptography provides a tool for secure communication between parties by letting them trust messages encrypted or signed by the already known public keys of the other parties.
However, no algorithmic scheme can solve the original trust problem of accepting the identity of a party that you never met.
The usual face-to-face identification is by a trusted third party (e.g. a friend) who presents the two parties to each other.
Such a presentation protocol is also required for cryptographic protocols.
The presenting party in the cryptographic environment is called a certification authority (CA).
The management of the CAs requires a public key infrastructure (PKI).

Public Key Cryptography in Practice
During face-to-face presentation, the presenter gives the relation between the name and the face of the presented party, together with some side information (e.g. they are a friend).
For cryptographic use the certification authority should give the relation between the public key and the identity of the owner.
This information should be transmitted authenticated from the CA to the receiver, e.g. signed under the widely known public key of the CA.
Thus, the receiver should only verify the signature of the CA, rather than communicate with the CA to verify every new key.
Such a CA signature is called a certificate.

Public Key Cryptography in Practice
A certificate includes:

1. The CA name
2. Sequential number of the certificate
3. The public key of the user
4. The identity of the user
5. Date
6. Last validation date
7. Signature of the CA on all the above

Public Key Cryptography in Practice
It might happen that the secret key of some user becomes known to another due to theft, factoring, or other reasons.
Therefore, CAs maintain Certificate Revocation Lists (CRLs) or blacklists of cancelled certificates which must not be trusted.
Users can ask to add their old certificates to the blacklists if they suspect that their secret keys became known.
The last validation date field in the certificate ensures that the blacklists will not have to keep such certificates for more than a selected time, as after the last validation date, the certificates become invalid anyway, and the user should select another key instead.

Public Key Cryptography in Practice
The X.509 standard defines a tree hierarchy of CAs.
Each CA has some 'parent', who signs and certifies the CA's public key.
The only required widely known public key is the key of the root CA.
All other public keys of CA can be verified using certificates.
Then, a receiver verifies a certificate of another user by verifying the certificate of the CA first, and then verifying the signature of the CA on the certificate of the user.
In turn, verification of the certificate of the CA is performed by verifying the certificate of the parent CA and the signature of the parent CA, etc.
Only the public key of the root CA should not be verified, as it is widely known.

Public Key Cryptography in Practice
The drawback of the X.509 hierarchy is that every user must trust all the CAs.
In the PGP hierarchy every user is also a CA, and users can select which CAs they trust, and which they do not trust.

- As a CA, a user signs certificates to their recognised friends; this does not mean that the friend is trustworthy.
- Each user then asks for certificates from many other users, and collects as many as desired.

- When a user needs to prove their identity, they publish (or send) the certificates they have collected to the other user, and the other user verifies them.
- The receiver can decide to trust certificates signed by some CAs unconditionally, and not trust certificates signed by other CAs.