Public Key Cryptography Notes

Introduction to Cryptology
MSc Program at UBa Cyber Crypto Center 2025
Jorge L. Villar

Last updated: Oct 21 10:53:19 2025

Contents

(Generated by lineprocx v2.97)

1 Public Key Encryption

1.1 Cryptography in the Multiuser Setting

In symmetric key cryptography every pair of communicating entities must share an independent secret key.

In a private communication network:

n users $\Rightarrow n(n-1)/2$ independent keys (each user securely stores $n-1$ independent keys)

Public Key Cryptography: Every user generates a key pair (pk, sk), publishes pk and keeps sk secret.

A single secret key per user for all communications...

...but the public keys must be reliably distributed (see below).

1.2 Public Key Encryption Syntax

In a public key encryption scheme there are three algorithms:

- A **Key Generation** algorithm, KeyGen, that given a security parameter λ (a positive integer) it produces a random key pair (pk, sk) of a suitable size.
- An **Encryption** algorithm, Enc, that given a public key pk and a message m, it produces a ciphertext c, possibly using some randomness.
- A **Decryption** algorithm, Dec, that given a secret key sk and a ciphertext c, it outputs a message m.

The public key typically contains some auxiliary information such the message space M_{pk} (or simply M) and the ciphertext space C_{pk} (or simply C), corresponding to the particular choice of λ and pk. The set of all possible key pairs produced by KeyGen for a particular value of λ is denoted as K_λ (or simply K).

Some of the algorithms are allowed fo fail (e.g., outputing a special rejection symbol $\perp$) if they receive improper inputs (like badly formed keys, messages or ciphertexts).

Correctness: $\forall (pk, sk) \in K_\lambda$, $\forall m \in M_{pk}$, $\mathrm{Dec}(sk, \mathrm{Enc}(pk, m)) = m$

When a user A wants to start receiving confidential information from other users, she produces a key pair (pk, sk) with KeyGen. Then she publishes pk and keeps sk secret as her long term key.

Now a user B can confidentially send a message $m \in M_{pk}$ by computing a ciphertext $c = \mathrm{Enc}(pk, m)$ and sending it to A by an insecure channel.

On reception, A will recover the message by running $m = \mathrm{Dec}(sk, c)$.

1.3 Man-in-the-Middle Attack

If the public keys are not reliably transmitted to the sender, an impersonation attack can efficiently break the security of a public key encryption scheme:

- An attacker C can generate its own keypair (pk', sk') and replace A's public key by pk'.
- Then, B sees pk' instead of pk as the public key of A, and she computes the ciphertext $c' = \mathrm{Enc}(pk', m)$ instead of c.
- C receives c' and decrypts it recovering B's message $m = \mathrm{Dec}(sk', c')$.

- If necessary, C can replace m by a different message m' and send to A the ciphertext $c'' = \text{Enc}(pk, m')$.
- On reception of c', A will recover $m' = \text{Dec}(sk, c'')$ instead of B's original message.

1.4 Public Key Certification

There exist mainly two ways to prevent public key replacement attacks:

- A trusted **Certification Authority** provides a way to verify that a given public key corresponds to a given identity. Then, any user B will check the validity of a public key before encrypting any message.
- A trusted **Key Generation Center** is able to compute key pairs (pk, sk) where the public key is an unambiguous representation of the identity of a user.

Systems relying on the latter approach are called **Identity Based Encryption** schemes. The most widely used approach is the former, in what is called a **Public Key Infrastructure**.

1.5 Hybrid Encryption

Known public key encryption schemes are far less efficient than the practical symmetric key encryption schemes. A hybrid solution can benefit from the good properties of both types of schemes.

In a hybrid encryption scheme, a sender B uses a public key encryption scheme to send a random session (short term) key k, that will be used to encrypt the message with an efficient symmetric key encryption scheme. Namely:

- B generates a random k and encrypts it with the public key pk of the recipient A:
  $c_0 = \text{Enc}(pk, k)$
- B encrypts the message with the symmetric key encryption scheme using the session key k:
  $c_1 = E_k(m)$
- B sends the ciphertext (c_0, c_1) to A.
- A parses the received ciphertext as (c_0, c_1).
- A decrypts the session key with its secret key sk:
  $k = \text{Dec}(sk, c_0)$
- A recovers the message m by using the symmetric key decryption algorithm:
  $m = D_k(c_1)$

Hybrid encryption schemes automatically support encryption of messages of arbitrary length from the mode of operation of the underlying symmetric key encryption scheme.

Therefore, the public key encryption scheme only needs to be able to encrypt short messages (since a symmetric key has only a few hundreds of bits).

1.6 Practical Constructions

1.6.1 ElGamal (1985)

ElGamal encryption scheme can be implemented on any cyclic finite group with an efficiently computable group operation, provided that some problems related to the discrete logarithm problem in the group are expected to be hard. The simplest example is using a large enough cyclic subgroup of the multiplicative group of a finite field.

The encryption scheme is an adaptation of a previously defined 2-party key agreement protocol.

1.6.1.1 Diffie-Hellman Key Agreement (1976)

In a 2-party key agreement protocol, two entities A and B interact sequentially sending some messages in such a way that at the end of the conversation both parties can derive a common key k, that is conjectured to be unknown to an attacker eavesdropping at the whole conversation.

This agreed secret key k can be used as the key in a symmetric encryption scheme, or in other symmetric key protocols, like a message authentication code.

In the Diffie-Hellman 2-party key agreement protocol, a finite cyclic group G of prime order q and a generator g are previously selected in a setup phase (e.g., they can be fixed in advance by a standarization body and shared among different pairs of users). Then, G and g are public parameters of the scheme.

Every party selects a nonzero random secret value $x \in Z_q^\times$ and computes the group element $y = g^x$.

The two parties, A and B, exchange the computed group elements $y_A = g^{x_A}$ and $y_B = g^{x_B}$.

Each party locally compute the shared secret key $k_{AB} = g^{x_A x_B}$. Namely $k_{AB} = y_A^{x_B} = y_B^{x_A}$.

It should be hard to compute the group element $g^{x_A x_B}$ only from G, g, g^{x_A} and g^{x_B}, for random $x_A, x_B \in Z_q^\times$.

Notice than in this version of the protocol, there are no long term keys. That is, to generate a second common key, an independent protocol execution must be performed. In addition, the protocol described above is susceptible to man-in-the-middle attacks.

However, there are ways to transform the basic scheme into a more sophisticated one that uses long term secret values to generate several independent common keys. Then, the long term secret values can be certified, preventing man-in-the-middle attacks.

1.6.1.2 The Basic Encryption Scheme

ElGamal encryption transforms Diffie-Hellman key agreement into a public key encryption scheme by using the public/secret information of the recipient as a long term key pair and the public/secret information of the sender as a one time key pair. The agreed secret key is then used as a one-time pad to encrypt the message.

The resulting encryption algorithm is probabilistic.

KeyGen(λ):
Select a cyclic group G of order q, where q is a λ-bit long prime, and a group generator g.
Choose a random $x \in Z_q^\times$ and compute the group element $y = g^x$.
Output $(pk, sk) = ((\text{param}, y), x)$, where $\text{param} = (G, g, q)$.

Enc(pk, m):
Parse $pk = (\text{param}, y)$.
Choose a random $r \in Z_q^\times$.
Output $c = (c_0, c_1) = (g^r, y^r m)$.

Dec(pk, sk, c):
Parse $pk = (\text{param}, y)$.
Parse $sk = x$.
Parse $c = (c_0, c_1)$.
Output $m = c_0^{-x} c_1$.

param is necessary in algorithm Dec, because it uses the group operation and the generator.

The security of ElGamal encryption is related to the amount of information an attacker can get about the group element g^{rx} from the group elements g, g^x and g^r.

In the original proposal, G is a prime order subgroup of the multiplicative group $Z_p^\times$, where p is a prime number such that q divides $p - 1$. For efficiency reasons the cofactor s in $p = sq + 1$ must be small, but for security reasons p must be large (due to known subexponential attaks against the discrete logarithm problem).

The scheme is directly generalizable to non-prime finite fields.

1.6.1.3 Hashed ElGamal

A serious limitation of the basic ElGamal scheme is that the message space is a multiplicative subgroup of a finite field. Thus, a proper encoding function from binary strings to group elements is needed.

An alternative way to solve the problem is converting the agreed secret key g^{rx} into a binary string by using a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$. This trick changes the message space to the binary strings of length λ.

KeyGen(λ):
Select a cyclic group G of order q, where q is a λ-bit long prime, and a group generator g.
Select a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$.
Choose a random $x \in Z_q^\times$ and compute the group element $y = g^x$.
Output $(pk, sk) = ((\text{param}, y), x)$, where $\text{param} = (G, g, q, H)$.

Enc(pk, m):
Parse $pk = (\text{param}, y)$.
Choose a random $r \in Z_q^\times$.
Output $c = (c_0, c_1) = (g^r, H(y^r) \oplus m)$.

Dec(pk, sk, c):
Parse $pk = (\text{param}, y)$.
Parse $sk = x$.
Parse $c = (c_0, c_1)$.
Output $m = H(c_0^x) \oplus c_1$.

The security of Hashed ElGamal encryption is related to the amount of information an attacker can get about the group element $H(g^{rx})$ from the group elements g, g^x and g^r.

1.6.1.4 Elliptic Curves

ElGamal encryption can be defined on any finite cyclic group such that some problems related to the discrete logarithm problem are considered hard.

A prime order subgroup of the group of points of an elliptic curve is a natural choice to implement ElGamal encryption, since the group operation can be efficiently implemented, and there is no known specific efficient algorithm solving the corresponding discrete logarithm problem on a random elliptic curve.

As before, an encoding function from binary strings to points in the subgroup is necessary. Alternatively, a hashed version of the encryption scheme can be used.

Since additive notation is commonly used for elliptic curves, ElGamal encryption is described in a slightly different way:

$\text{param} = (G, P, q)$, where P is a point generating the cyclic subgroup G of order q.
$sk = x \in Z_q^\times$ as before.
$pk = (\text{param}, Y)$, where $Y = xP$.
$\text{Enc}(pk, m) = (rP, rY + M)$, where M is now the message encoded as a point in G, and $r \in Z_q^\times$.
$\text{Dec}(pk, sk, C) = -xC_0 + C_1$.

1.6.2 RSA (1977)

The first proper public key encryption scheme was introduced by Rivest, Shamir and Adleman in 1977. Its security relies on the difficulty of factoring large integers, and uses two modular exponentiations which composition is the identity map.

The RSA modulus is the product of two random secret different primes p, q of the same binary length, $n = pq$. It is well known that for every integer x coprime with n, $x^{\phi(n)} \equiv 1 (\text{mod } n)$, where $\phi(n) = (p-1)(p-1)$ is Euler's totient function.

Therefore, if e, d are two integers such that $ed \equiv 1 (\text{mod } \phi(n))$ and x is coprime with n, then $x^{ed} \equiv x (\text{mod } n)$.

Notice that there exist efficient algorithms to compute modular exponentiations and modular inversions, and also to sample random prime numbers. Then, the above facts are the basis of the following public key encryption scheme:

In the following algorithms, residues modulo n are represented as integers between 0 and $n-1$, and similarly for residues modulo $\phi(n)$.

KeyGen(λ):
Select two different random prime numbers p, q of λ bits, and compute $n = pq$.
Choose a convenient public exponent e coprime with $\phi(n) = (p-1)(q-1)$ (typically e is a quite small prime, like $2^4 + 1$ or $2^{16} + 1$), and compute a secret exponent d as the modular inverse of e modulo $\phi(n)$.
Output $(pk, sk) = ((n, e), (p, q, d))$.

Enc(pk, m):
Parse $pk = (n, e)$.
Output $c = m^e \bmod n$.

Dec(pk, sk, c):
Parse $pk = (n, e)$.
Parse $sk = (p, q, d)$.
Output $m = c^d \bmod n$.

Actually, the decryption algorithm only requires the key components n and d, but in some optimized implementations, p and q are also used.

In most implementations of RSA, the public exponent e is fixed for all users, and the primes p, q are selected so that $p - 1$ and $q - 1$ are coprime with e (otherwise, they are resampled). Indeed, the complexity of the encryption algorithm (modular exponentiation) depends on the binary representation of the public exponent.

1.6.2.1 Deterministic Encryption

RSA encryption is deterministic, because for a given public key, all encryptions of the same message are equal. This lowers its security in the same way as in the ECB mode of operation of a block cipher. Therefore, in most practical applications a determistic public key encryption is combined with a randomized map (like a random padding or a block chaining mode of operation with a random initialization value).

However, the problem disappears if the public key deterministic encryption is only used to send encrypted random keys (as in a hybrid encryption scheme), because the probability of encrypting the same message twice is negligible.

1.6.3 Rabin (1979)

Rabin public key encryption is based on the squaring function modulo $n = pq$, for secret primes p, q.

KeyGen(λ):
Select two different random prime numbers p, q of λ bits such that $p, q \equiv 3 (\bmod 4)$, and compute $n = pq$.
Output $(pk, sk) = (n, (p, q))$.

Enc(pk, m):
Parse $pk = (n)$.
Output $c = m^2 \bmod n$.

Dec(pk, sk, c):
Parse $pk = (n)$.
Parse $sk = (p, q)$.
Compute $m_p = c^{(p+1)/4} \bmod p$, and $m_q = c^{(q+1)/4} \bmod q$.
Use the Chinese Remainder Theorem to compute the unique residue $m \in Z_n$ such that $m \equiv m_p (\bmod p)$ and $m \equiv m_q (\bmod q)$.
Output m.

There are exactly 4 different square roots of a valid ciphertext (i.e., $c = m^2 \bmod n$), provided that the

ciphertext is coprime with n. Namely, the 4 roots are obtained by applying the Chinese Remainder Theorem to the four pairs (m_p, m_q), $(m_p, -m_q)$, $(-m_p, m_q)$, $(-m_p, -m_q)$. Therefore, some redundancy must me added to m in order to recognize which of the four square roots is the correct decryption.

Any (probabilistic) algorithm computing a square root of c modulo $n = pq$, for a random modulus n and a random quadratic residue c, with some probability ϵ can be converted into another algorithm computing the prime factors of n with probability at least $\epsilon/2$. Thus, decrypting Rabin cryptosystem without the secret key is not easier than directly factoring the modulus.

1.6.4 Paillier (1999)

Paillier encryption scheme uses arithmetic modulo n^2, where $n = pq$ is an RSA modulus.

KeyGen(λ):
Select two different random prime numbers p, q of λ bits, and compute $n = pq$.
Choose $g \in Z_{n^2}$ such that $g^n \equiv 1 (\text{mod } n)$, but $g^x \not\equiv 1 (\text{mod } n)$ for any smaller x.
Output $(pk, sk) = ((n^2, g), (p, q))$.

Enc(pk, m):
Parse $pk = (n^2, g)$.
Choose a random $r \in Z_n^\times$.
Output $c = r^n g^m \bmod n^2$.

Dec(pk, sk, c):
Parse $pk = (n^2, g)$.
Parse $sk = (p, q)$.
Compute $s = c^{\phi(n)} \bmod n^2$.
Compute $s_1 = (s-1)/n$.
Compute $t = g^{\phi(n)} \bmod n^2$.
Compute $t_1 = (t-1)/n$.
Output $m = s_1 (t_1)^{-1} \bmod n$.

Fixing $g = n + 1$ makes both the encryption and decryption mode efficient. Indeed, $(n+1)^m \equiv 1 + mn (\text{mod } n^2)$, and in decryption $t_1 = \phi(n) \bmod n = n - p - q + 1$.

Observe that an attacker capable to invert the $RSA(n, n)$ encryption function (i.e., taking $e = n$) would be able to decrypt a Paillier ciphertext.

1.6.5 Regev (2009)

Some modern proposals of public key encryption schemes are based on hard computational problems defined over lattices.

KeyGen(λ):
Select parameters q, n, σ such that q is a prime between λ^2 and $2\lambda^2$, $n = (1+\epsilon)(\lambda+1)\log q$ for some positive $\epsilon, \sigma^2 = o(1/(\lambda \log^2 \lambda))$, e.g., $\sigma^2 = 1/(\lambda \log^3 \lambda)$.
Select a random $n \times \lambda$ matrix A in Z_q, and a column vector $\mathbf{s} \in Z_q^\lambda$.
Compute $\mathbf{b} = A\mathbf{s} + \mathbf{x} \bmod q$, where the n components of $\mathbf{x}$ are sampled from a discrete Gaussian distribution of variance σ^2.
Output $(pk, sk) = ((\text{param}, A, \mathbf{b}), (\mathbf{s}))$, where $\text{param} = (\lambda, q, n, \sigma)$.

Enc(pk, m):
Parse $pk = (\text{param}, A, \mathbf{b})$.
Choose a random binary row vector $\mathbf{r} \in \{0,1\}^n$.
Output $c = (\mathbf{r}A \bmod q, \mathbf{r}\mathbf{b} + m(q-1)/2 \bmod q)$, where $m \in \{0, 1\}$.

Dec(pk, sk, c):
Parse $pk = (\text{param}, A, \mathbf{b})$.
Parse $sk = (\mathbf{s})$.

Parse $c = (\mathbf{c_1}, c_2)$.
Compute $s = c_2 - \mathbf{c_1 s} \bmod q$.
Set t to the nearest element in $\{0, (q-1)/2\}$ to s.
Output $m = 0$ if $t = 0$. Otherwise, output $m = 1$.

Correctness is not guaranteed with probability one, but the error probability can be made negligible.

The noise $\mathbf{x}$ added in the key generation is critical: if it is too small, the public key reveals the secret key. If it is too large, wrong decryption occurs with noticeable probability.

The efficiency of the scheme is not good, because it encrypts only one bit in a large ciphertext. But the scheme is conjectured to remain secure even when the attacker has access to a quantum computer.

Observe that the matrix A defines a lattice, and the vector $\mathbf{b}$ is near the (secret) lattice point $A\mathbf{s}$. Therefore, an attacker capable to find the nearest lattice point to a given point would be able to compute the secret key $\mathbf{s}$.

1.7 One-Way Functions

The concept of public key cryptography implies the existence of functions that can be evaluated efficiently, but they are hard-to-invert, that is finding a preimage of a given random element in their range. For instance, given a public key and a message, it is easy to compute the corresponding ciphertext, but given the ciphertext and the public key, it is infeasible to recover the message.

Furthermore, if some auxiliary information is given (the secret key) the preimage computation becomes easy. This implies that the one-way function has a "trapdoor".

Definition 1 *A function $f : \{0,1\}^* \to \{0,1\}^*$ is called **one-way** if:*

1. *There exists an efficient algorithm A that evaluates f on all possible inputs, that is, there exists a polynomial P_A such that for all $\lambda \in Z^+$ and all $x \in \{0,1\}^\lambda$, $A(x) = f(x)$ and the running time of A is upper bounded by $P_A(\lambda)$.*
2. *No efficient algorithm is able to invert f. More precisely, for any algorithm B, any polynomial P_B and any $c \in Z^+$, either there exist $\lambda \in Z^+$ and $x \in \{0,1\}^\lambda$ such that the running time of B on input $f(x)$ exceeds P_B, or the probability that $B(f(x)) \in f^{-1}(f(x))$ for a random $x \in \{0,1\}^\lambda$ only exceeds λ^{-c} at finitely many values (or no value) of $\lambda \in Z^+$.*

The previous definition can be simplified if we introduce some notations:

Definition 2 *An algorithm A is called **polynomial time** if there exists a polynomial P_A such that on any input x of length λ the running time of A is upper bounded by $P_A(\lambda)$.*

Definition 3 *A function $\epsilon : Z^+ \to R$ is called **negligible** if for any $c \in Z^+$, $|\epsilon(\lambda)| > \lambda^{-c}$ only at finitely many values (or no value) of $\lambda \in Z^+$. We will write $\epsilon \in$ **negl**, or $\epsilon(\lambda) \in$ **negl**(λ).*

Now, in the definition of one-way function, A and B are polynomial time algorithms and the probability that $B(f(x)) \in f^{-1}(f(x))$ is a negligible function in λ.

The previous function can be seen as the function family $\{f_\lambda : \{0,1\}^\lambda \to \{0,1\}^*\}_{\lambda \in Z^+}$. The definition of one-way function can be extended to more general function families:

Definition 4 *A function family $\{f_i : X_i \to Y_i\}_{i \in I}$, where I is a set of indices that is the disjoint union of $\{I_\lambda\}_{\lambda \in Z^+}$, is called **one-way** if:*

1. *There exists a polynomial time algorithm A such that for any $i \in I$ and $x \in X_i$, $A(i, x) = f_i(x)$.*

2. *There exists polynomial time algorithms that for any $\lambda \in Z^+$ it produces a uniformly distributed element $i \in I_\lambda$, and for any $i \in I_\lambda$ it produces a uniformly distributed element in X_i.*

3. *For any polynomial time algorithm B the probability that $B(i, f_i(x)) \in f_i^{-1}(f_i(x))$ for random $i \in I_\lambda$ and $x \in X_i$ is a negligible function in λ.*

The indexes of a one-way functions are typically related to the public keys of an encryption scheme. For example, in RSA cryptosystem the encryption function's domain and range depends on the public key. Therefore, the set I would be the set of RSA public keys, and the corresponding functions would be $f_{(n,e)} : Z_n \to Z_n$ such that $f_{(n,e)}(x) = x^e \bmod n$.

It is an open problem to find a one-way function family (or even proof its existence), but some well-known families (like the previous RSA example) are conjectured to be one-way. Indeed, with a precise definition of security of a public key encryption scheme, it can be shown that the existence of a secure deterministic public key encryption scheme implies the existence of one-way functions.

1.7.1 Amplification

Given a function family $\{f_i : X_i \to Y_i\}_{i \in I}$ that cannot be inverted with probability greater than a given bound, one can define another family g with a much smaller bound in the probability of inversion, by considering an n-fold version of f:
$g_i(x_1, ..., x_n) = (f_i(x_1), ..., f_i(x_n))$
Indeed, any polynomial time algorithm B_g inverting g with probability ϵ_g can be used to build another polynomial time algorithm B_f that inverts f with a much greater probability ϵ_f as follows:

- On the input of (i, y), where $y = f_i(x)$ for random $i \in I_\lambda$ and $x \in X_i$, B_f chooses a random $j \in \{1, ..., n\}$ and sets $y_j = y$ and $y_k = f_i(x_k)$ for a random $x_k \in X_i$, for all $k \neq j$.
- Next, B_f runs $B_g(i, y_1, ..., y_n)$, obtaining the output $(x_1, ..., x_n)$.
- If $f_i(x_j) = y$ then B_f outputs x_j. Otherwise, B_f repeats the previous steps until a predefined iteration limit t is reached. In that case, the algorithm fails (for instance, it outputs a random element in X_i).

Notice that when B_f fails inverting f, B_g failed inverting the vector $\mathbf{y}$ in all t iterations. If the vectors were independent, the failure would occur with probability $(1 - \epsilon_g)^t$. Then it would be $\epsilon_f = 1 - (1 - \epsilon_g)^t$, which is exponentially close to 1 for a large enough t. However, this analysis is not correct because the vectors $\mathbf{y}$ are correlated (actually, all of them have at least one component equal to the input y). But this correlation becomes negligible when n is large enough.

A further step in the direction of proving the existence of one-way functions is finding a weakened version of the definition.

Definition 5 *A function family $\{f_i : X_i \to Y_i\}_{i \in I}$, where I is a set of indices that is the disjoint union of $\{I_\lambda\}_{\lambda \in Z^+}$, is called **weakly one-way** if:*

1. *There exists a polynomial time algorithm A such that for any $i \in I$ and $x \in X_i$, $A(i, x) = f_i(x)$.*

2. *There exists a positive polynomial Q such that for any polynomial time algorithm B the probability that $B(i, f_i(x)) \in f_i^{-1}(f_i(x))$ for random $i \in I_\lambda$ and $x \in X_i$ is greater than $1 - 1/Q(\lambda)$ only for finitely many values of λ.*

What is required in this weakened version is that any efficient algorithm trying to invert f fails with a non negligible probability, which seems to be a very relaxed statement. However, the two notions of one-wayness are indeed equivalent (from the existential point of view):

Theorem 1 *A (strong) one-way function family exists if and only if a weak one-way function family exists.*

Proof. Trivially, every strong one-way function family is also weakly one-way. Two show the other implication it suffices to prove that a n-fold version of the weak one-way function family is strongly one-way, if n is large enough. For simplicity, we will omit the index i, because it takes the same fixed value along the proof.

Let us call B_f^1 to a single iteration of the algorithm B_f described above (i.e., taking $t = 1$). The algorithm B_g used into it is any polynomial time algorithm trying to invert the n-fold version of f.

Let Bad be the set of $x \in X$ such that B_f^1 can invert $f(x)$ with probability less than a threshold α, and call Good to its complement. This threshold will be fixed later. Let $\beta = \text{Prob}(x \in \text{Bad})$ for a uniformly distributed x, that is, β is the fraction of elements of X lying on the set Bad. Observe that β is a function of α.

Let us denote by $\text{Fail}(B_f|x \in X)$ the probability that B_f fails to invert f on $f(x)$ for a random $x \in X$. By the weak one-wyness of f,

$1/Q(\lambda) < \text{Fail}(B_f|x \in X) = \text{Fail}(B_f|x \in \text{Bad})\beta + \text{Fail}(B_f|x \in \text{Good})(1-\beta) \leq \beta + \text{Fail}(B_f|x \in \text{Good}) \leq \beta + (1-\alpha)^t$.

Taking $t = \lambda/\alpha$ the term $(1-\alpha)^t$ becomes exponentially small, and we have shown that $\beta > 1/2Q(\lambda)$.

Now, we can use B_f^1 to find an upper bound to the probability that B_g inverts the n-fold version of f on the image of a random vector in X^n. Let us call $\text{Succ}(B_g)$ to this probability.

$\text{Succ}(B_g) = \text{Succ}(B_g \text{ and } \mathbf{x} \in X^n \setminus \text{Good}^n) + \text{Succ}(B_g \text{ and } \mathbf{x} \in \text{Good}^n) \leq \text{Succ}(B_g \text{ and } \mathbf{x} \in X^n \setminus \text{Good}^n) + (1-\beta)^n \leq \text{Succ}(B_g \text{ and } \mathbf{x} \in X^n \setminus \text{Good}^n) + (1 - 1/2Q(\lambda))^n$.

The second term becomes exponentially small if we take $n = 2\lambda Q(\lambda)$.

Now, for any $j \in \{1, ..., n\}$ we can write

$\text{Succ}(B_g \text{ and } x_j \in \text{Bad}) \leq n\ \text{Succ}(B_f^1 \text{ and } x_j \in \text{Bad}) \leq n\ \text{Succ}(B_f^1 | x_j \in \text{Bad}) < \alpha n$.

Therefore, by the union bound, $\text{Succ}(B_g \text{ and } \mathbf{x} \in X^n \setminus \text{Good}^n) \leq \alpha n^2$.

The proof is almost concluded, because for any algorithm B_g trying to invert the n-fold version of f and any constant $c \in Z^+$, we can take $\alpha = 1/(2\lambda^c n^2)$, so that $\text{Succ}(B_g) < \lambda^{-c}$. In other words, $\text{Succ}(B_g) \in \mathbf{negl}(\lambda)$.

The particular choice of α implies that $n = 2\lambda Q(\lambda)$ and $t = 2\lambda^{c+1} n^2$.

1.7.2 Trapdoor One-Way Permutations

Public key encryption is tightly related to the concept of one-way function family, but the existence of a decryption algorithm implies a trapdoor mechanism that allows the efficient inversion of the one-way function family, given some auxiliary information like the secret key.

The definition of a trapdoor one-way permutation (i.e., a bijective map from and to the same set) families adds this functionality:

Definition 6 *A permutation family $\{f_i : X_i \to X_i\}_{i \in I}$, where I is a set of indices that is the disjoint union of $\{I_\lambda\}_{\lambda \in Z^+}$, is called **trapdoor one-way** if:*

1. *There exists a polynomial time algorithm A such that for any $i \in I$ and $x \in X_i$, $A(i, x) = f_i(x)$.*
2. *There exists polynomial time algorithms that for any $\lambda \in Z^+$ it produces a uniformly distributed element $i \in I_\lambda$ and a trapdoor information t_i, and for any $i \in I_\lambda$ it produces a uniformly distributed element in X_i.*
3. *There exists a polynomial time algorithm that on the input of $i \in I$, the corresponding t_i and $f_i(x)$ for any $x \in X_i$, it outputs x.*
4. *For any polynomial time algorithm B the probability that $B(i, f_i(x)) = x$ for random $i \in I_\lambda$ and $x \in X_i$ is a negligible function in λ.*

Some of the function families derived from the previously defined public key encryption schemes are conjectured to be trapdoor one-way families.

1.8 Hardcore Predicates

Even if a function f is one-way, it cannot be said that the image $f(x)$ hides x. For instance, if f is one-way, so is the function $g(w,x) = (w, f(x))$, even when w is completely exposed in the image of the function. The notion of one-wayness only implies that a substantial part of the preimage x, but not the entire x, is hidden in $f(x)$. Then, there is the need to know which part of x is actually hidden in $f(x)$.

Given a one-way function f, we say that a map $h : \{0,1\}^* \to \{0,1\}$ (also called a predicate) is hardcore for f if no useful information of $h(x)$ is leaked from $f(x)$. More precisely:

Definition 7 *The (balanced) predicate $h : \{0,1\}^* \to \{0,1\}$ is **hardcore** for a one-way function f if*

- *There exists a polynomial time algorithm A computing $h(x)$ from x.*
- *For any polynomial time algorithm B the probability that the difference $Prob[B(f(x)) = h(x)] - 1/2$ for a random $x \in X_\lambda$ is a negligible function of λ.*

The difference between the success probability of algorithm B and $1/2$ is the advantage of B in guessing $h(x)$ with respect to a lazzy algorithm that simply ignores $f(x)$ and flips a coin to decide its output.

We assume that h is a balanced predicate, that is, $\text{Prob}[h(x) = 1] - 1/2$ for a random $x \in X_\lambda$ is a negligible function of λ.

Proving that h is a hardcore predicate for f is usually done by explicitly building a polynomial time algorithm B_f that inverts f with a non-negligible probability from any other polynomial time algorithm B_h that computes $h(x)$ from $f(x)$ with a non-negligible advantage. A partial proof applying only to perfect B_h (that is, computing the correct $h(x)$ with probability one) is usually easy to obtain, but extending it to the general case often requires far more work.

1.8.1 Goldreich-Levin Hardcore Predicate

Given any one-way function f it can be modified to obtain a new one $g(w,x) = (w, f(x))$ and a hardcore predicate for it, called the Goldreich-Levin predicate, $h(w,x) = w \cdot x = w_1x_1 + ... + w_\lambda x_\lambda \mod 2$.

A partial proof of this result, working only for perfect algorithms B_h is almost trivial: Just run $B_h(e_j, y)$ for $j = 1, ..., \lambda$, where $y = f(x)$ is the input of B_f and e_j has a single one-bit in the j-th position. If B_h is perfect, then $B_h(e_j, f(x)) = h(e_j, x) = x_j$, that is, the j-th bit of x.

However, if the success probability of B_h is close to $1/2$, then the probability that all the λ calls to B_h give the correct answer will intuitively drop exponentially to $2^{-\lambda}$. Therefore, a more involved strategy would be necessary to build a general proof.

1.8.2 Other Known Hardcore Predicates

Some well-known candidates to one-way functions have hardcore predicates that are just some of the bits of x. For instance, all bits of x are hardcore predicates for the RSA-based one-way function, or the most significant bits of the discrete-log based one-way function are also hardcore.

For example, let us consider the least significant bit of the RSA one-way function family $f_{(n,e)}(x) = x^e \mod n$

We define the least significant bit of an element of Z_n as $\text{LSB}(x) = 1$ if x is an even integer and $\text{LSB}(x) = 0$ if x is an odd integer, assuming that x is represented as an integer in $\{0, ..., n-1\}$. We define the sequence $x_i = 2^i x \mod n$, for $i = 1, ..., \lambda$.

Observe that $y_i = f_{(n,e)}(x_i) = x_i^e = 2^{ie}y \bmod n$, where $y = f_{(n,e)}(x) = x^e \bmod n$. Then, an algorithm B_{RSA} that tries to learn x from y can call a perfect algorithm B_{LSB} on inputs $y_1, ..., y_\lambda$. It is easy to see that $\mathrm{LSB}(x_1) = 0$ if and only if $0 \leq x < n/2$, $\mathrm{LSB}(x_2) = 0$ if and only if $0 \leq x < n/4$ or $n/2 < x < 3n/4$, and so on. Therefore, x can be easily retrieved from the LSB values given by B_{LSB}.

As before, generalizing this argument for a non-perfect B_{LSB} is not a trivial task.

1.8.3 Public Key Encryption from a Hardcore Predicate

There is a (quite inefficient) generic public key encryption scheme based on any hardcore predicate h of a trapdoor one-way permutation family f.

KeyGen(λ):
Sample a random index $i \in I_\lambda$ along with its corresponding trapdoor t_i.
Output $(pk, sk) = (i, t_i)$.

Enc(pk, m):
Parse $pk = (i)$.
Parse m as the bit sequence $(m_1, ..., m_n)$.
Sample a random $x_0 \in X_i$.
Compute the sequence $x_j = f_i(x_{j-1})$ for $j = 1, ..., n$.
Compute the bit sequence $b_j = h(x_{j-1})$ for $j = 1, ..., n$.
Output $c = (m_1 \oplus b_1, ..., m_n \oplus b_n, x_n)$.

Dec(pk, sk, c):
Parse $pk = (i)$.
Parse $sk = (t_i)$.
Parse $c = (c_1, ..., c_n, x_n)$.
Compute the sequence $x_{n-j} = f_i^{-1}(x_{n-j+1})$ for $j = 1, ..., n$ using the trapdoor t_i.
Compute the bit sequence $b_j = h(x_{j-1})$ for $j = 1, ..., n$.
Output $m = (c_1 \oplus b_1, ..., c_n \oplus b_n)$.

1.9 Homomorphic Encryption

Some public key encryption schemes respect some group operations both in the message space and the ciphertext space. Let us assume that $+$ is a group operation in M and $*$ is a group operation in C.

Definition 8 *A public key encryption scheme (KeyGen,Enc,Dec) is called (weakly)* ***homomorphic*** *with respect to the operations $(M, +)$ and $(C, *)$ if*
$\forall (pk, sk) \in K$, $\forall m_1, m_2 \in M_{pk}$, $Dec(sk, Enc(pk, m_1) * Enc(pk, m_2)) = m_1 + m_2$.

This notion can be strengthened for probabilistic encryption schemes:

Definition 9 *A public key encryption scheme (KeyGen,Enc,Dec) is called (strongly)* ***homomorphic*** *with respect to the operations $(M, +)$ and $(C, *)$ if there exists an efficient algorithm Rerand such that*
$\forall (pk, sk) \in K$, $\forall m_1, m_2 \in M_{pk}$, *the vectors $(c_1, c_2, Rerand(pk, c_1 * c_2))$ and $(c_1, c_2, Enc(pk, m_1 + m_2)$, where $c_1 = Enc(pk, m_1)$ and $c_2 = Enc(pk, m_2)$, are identically distributed random variables.*

This definition means that a ciphertext for $m_1 + m_2$ can be obtained directly from the ciphertexts c_1, c_2, without the knowledge of the messages and in an indistinguishable way (that is, the randomness in the new ciphertext is independent of the randomness contained in c_1 and c_2). Typically, $\mathrm{Rerand}(pk, c)$ outputs $\mathrm{Enc}(pk, 0) * c$, where 0 is the neutral element in $(M, +)$.

For deterministic encryption schemes, the two notions are equivalent, with $\mathrm{Rerand}(pk, c) = c$, since the ciphertexts contain no randomness.

Proposition 1 *RSA is homomorphic with respect to the product modulo n.*

Proof. $\text{Enc}((n,e), m_1 m_2) = (m_1 m_2)^e \bmod n = m_1^e m_2^e \bmod n = \text{Enc}((n,e), m_1)\text{Enc}((n,e), m_2) \bmod n$.

Proposition 2 *ElGamal is strongly homomorphic with respect to the group operation in G and in $G \times G$.*

Proof. $\text{Rerand}(pk, \text{Enc}(pk, m_1)\text{Enc}(pk, m_2)) = \text{Enc}(pk, 1)\text{Enc}(pk, m_1)\text{Enc}(pk, m_2) = (g^{r_0}, y^{r_0})(g^{r_1}, y^{r_1} m_1)$ $(g^{r_2}, y^{r_2} m_2) = (g^{r_0+r_1+r_2}, y^{r_0+r_1+r_2} m_1 m_2)$. On the other hand, $\text{Enc}(pk, m_1 m_2) = (g^{r_3}, y^{r_3} m_1 m_2)$. The two ciphertexts are identically distributed, conditioned to the values of r_1 and r_2, because r_0 and r_3 are uniformly distributed.

Proposition 3 *Paillier is strongly homomorphic with respect to addition modulo n and multiplication modulo n^2.*

Proof. $\text{Rerand}(pk, Enc(pk, m_1)\text{Enc}(pk, m_2)) = \text{Enc}(pk, 0)Enc(pk, m_1)\text{Enc}(pk, m_2) = r_0^n r_1^n g^{m_1} r_2^n g^{m_2} = (r_0 r_1 r_2)^n g^{m_1+m_2}$. On the other hand, $\text{Enc}(pk, m_1 + m_2) = r_3^n g^{m_1+m_2}$. The two ciphertexts are identically distributed, conditioned to the values of r_1 and r_2, because r_0 and r_3 are uniformly distributed.

1.9.1 Applications

Homomorphic encryption has many applications, being homomorphic talling in electronic voting the most intuitive one.

In an electronic voting for a referendum (each individual vote can contain one of "yes" or "no"), the bulletin board sets up a public key encryption scheme with additive homomorphic properties, and keeps secretly the decryption key. Every voter encrypts 1 (for "yes") or 0 (for "no") and sends the resulting ciphertext to the board. In the end of the election, all the votes (ciphertext) are operated to obtain a ciphertext of the sum of all votes. The resulting ciphertext is decrypted to obtain the number of positive votes.

This is just a toy example of a electronic voting system, because there is no verification mechanism to avoid voting more than once (e.g., by encrypting a number greater than one or a negative number), there is no way to prevent voting in a way related to other voter (e.g., rerandomizing other's vote as the own vote), to mention a few examples of all possible attacks.

2 Digital Signatures

Digital signatures are the public key analogous to message authentication codes in symmetric cryptography. They provide at once message integrity, authentication and not repudiation by securely combining a secret key with the message, and making the signature verification publicly available.

In addition, digital signatures can be used to avoid impersonation attacks like the man-in-the-middle attack, by adding signatures of the users' public keys by a trusted authority (see below).

Digital signatures were invented in the same seminal paper where Diffie and Hellman (1976) introduced the public key cryptography concept.

2.1 Digital Signature Syntax

In a digital signature scheme there are three algorithms:

- A **Key Generation** algorithm, KeyGen, that given a security parameter λ (a positive integer) it produces a random key pair (pk, sk) of a suitable size.

- An **Signature** algorithm, Sig, that given a secret key sk and a message m, it produces a signature s, possibly using some randomness.
- A **Verification** algorithm, Ver, that given a public key pk, a message m and a signature s, it outputs 1 if s is a valid signature for pk and m, or 0 otherwise.

The public key typically contains some auxiliary information such the message space M_{pk} (or simply M) and the signature space S_{pk} (or simply S), corresponding to the particular choice of λ and pk. The set of all possible key pairs produced by KeyGen for a particular value of λ is denoted as K_λ (or simply K).

Some of the algorithms are allowed fo fail (e.g., outputing a special rejection symbol $\perp$) if they receive improper inputs (like badly formed keys, messages or signatures).

Correctness: $\forall (pk, sk) \in K_\lambda$, $\forall m \in M_{pk}$, $\mathrm{Ver}(pk, m, \mathrm{Sig}(sk, m)) = 1$.

When a user A wants to be able to sign messages for other users, she produces a key pair (pk, sk) with KeyGen. Then she publishes pk and keeps sk secret as her long term key.

Now A can compute a digital signature for a message $m \in M_{pk}$ by computing $s = \mathrm{Sig}(sk, m)$.

Any other user B can verify the signature s on m with A's public key by checking whether $\mathrm{Ver}(pk, m, s) = 1$.

Security of a signature scheme refers to the hardness of generating a valid signature without knowing the secret key in any realistic attack scenario, like knowing some valid pairs message/signature from the target signer.

A signature can authenticate some extra information like place, time and purpose of the signature, as in conventional handwritten signatures, by appending in an unambiguous way the extra information to the original message.

Dealing with arbitrary length documents typically imply the use of a cryptographically secure hash function. Indeed, the chosen hash function is part of the specification of the signature scheme, and what is essentially signed is not the document itself but its hash value. This technique is referred to as the "hash-and-sign" paradigm.

2.2 Practical Constructions

2.2.1 Plain RSA (1977) and FDH (1993)

From the very close analogy between the syntax of encryption schemes and signatures, some public key encryption schemes can be transformed into digital signature schemes by using Dec as the signature algorithm and using Enc in the verification of the signature. This is precisely the case of RSA, as proposed in the seminal paper.

KeyGen(λ): (same as in the encryption scheme)
Select two different random prime numbers p, q of λ bits, and compute $n = pq$.
Choose a public exponent e coprime with $\phi(n) = (p-1)(q-1)$ (typically e is a quite small prime), and compute a secret exponent d as the modular inverse of e modulo $\phi(n)$.
Output $(pk, sk) = ((n, e), (p, q, d))$.

Sig(pk, sk, m):
Parse $pk = (n, e)$.
Parse $sk = (p, q, d)$.
Output $s = m^d \bmod n$.

Ver(pk, m, s):
Parse $pk = (n, e)$.
Output 1 if $m = s^e \bmod n$, and 0 otherwise.

The scheme is clearly insecure because an attacker can generate an arbitrary pair $(m, s) = (x^e \bmod n, x)$ for a random x, that is accepted by the verification algorithm. Even when the resulting message m would

be nonsense, this kind of attack must be prevented. Indeed, signature schemes are sometimes used to sign random values instead of structured messages with high redundancy.

The plain RSA signature can be improved by adding some redundancy (e.g., padding) to the message before signing it. However, a better solution is applying the hash-and-sign paradigm to the plain RSA signature, because this solves two problems at once: preventing the descibed attack and allowing messages of arbitrary length to be signed.

The Full Domain Hash (FDH) signature scheme combines any one-way trapdoor permutation and a secure hash function to build a signature scheme. In the case of RSA, the resulting scheme is the following:

KeyGen(λ): (same as in the encryption scheme)
Select two different random prime numbers p, q of λ bits, and compute $n = pq$.
Choose a public exponent e coprime with $\phi(n) = (p-1)(q-1)$ (typically e is a quite small prime), and compute a secret exponent d as the modular inverse of e modulo $\phi(n)$.
Choose a hash function $H : \{0,1\}^* \to Z_n^\times$.
Output $(pk, sk) = ((n, e, H), (p, q, d))$.

Sig(pk, sk, m):
Parse $pk = (n, e, H)$.
Parse $sk = (p, q, d)$.
Output $s = H(m)^d \bmod n$.

Ver(pk, m, s):
Parse $pk = (n, e, H)$.
Output 1 if $H(m) = s^e \bmod n$, and 0 otherwise.

Observe that the previously described attack is no longer possible, since now the attacker must give a preimage of $x^e \bmod n$ by the hash function, which seems to be an infeasible task. However, the only known security proofs are in the Random Oracle idealized model (so, they are no actual security proofs, but just heuristic security arguments).

2.2.2 ElGamal (1984) and Pointcheval-Stern (1996)

ElGamal signature scheme is based on the difficulty of computing discrete logarithms on some cyclic groups.

KeyGen(λ):
Choose a prime number p of λ bits.
Choose a generator g of the multiplicative group $Z_p^\times$.
Choose a random $x \in Z_{p-1}$ and compute $y = g^x \bmod p$.
Output $(pk, sk) = ((\text{param}, y), x)$, where $\text{param} = (g, p)$.

Sig(pk, sk, m):
Parse $pk = (\text{param}, y)$.
Parse $sk = (x)$.
Choose a random $k \in Z_{p-1}^\times$ and compute $r = g^k \bmod p$.
Compute $t = (m - xr)k^{-1} \bmod p - 1$.
If $t = 0$, repeat the procedure by choosing a new random k.
Output $s = (r, t)$.

Ver(pk, m, s):
Parse $pk = (\text{param}, y)$.
Parse $s = (r, t)$.
Output 1 if $g^m \equiv r^t y^r \pmod p$, and 0 otherwise.

This signature scheme is vulnerable in the same way as the basic RSA signature. Indeed, for arbitrary values of $\alpha \in Z_{p-1}^\times$ and $\beta \in Z_{p-1}$, the pair $s = (r, t)$, where $r = y^\alpha g^\beta \bmod p$, $t = -r\alpha^{-1} \bmod p - 1$, is a valid signature for the message $m = t\beta \bmod p$.

Following the hash-then-sign paradigm, Pointcheval and Stern (1996) added the use of a hash function to this basic ElGamal signature scheme to prevent the previous forgery attack. A security proof in the Random Oracle Model exists for the modified scheme.

2.2.3 DSA (1991)

DSA signature is a variant of ElGamal signature scheme that was standarized as the Digital Signature Standard (DSS) in 1994.

KeyGen(λ):
Choose a prime number p of λ_1 bits (typically λ_1 is taken between 6λ and 10λ).
Choose a prime number q of λ bits that divides $p-1$.
Choose a generator g of a cyclic subgroup of order q in the multiplicative group $Z_p^\times$.
Choose a secure hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$.
Choose a random $x \in Z_q$ and compute $y = g^x \bmod p$.
Output $(pk, sk) = ((\text{param}, y), x)$, where $\text{param} = (p, g, q, H)$.

Sig(pk, sk, m):
Parse $pk = (\text{param}, y)$.
Parse $sk = (x)$.
Choose a random $k \in Z_q^\times$ and compute $r = (g^k \bmod p) \bmod q$.
If $r = 0$, repeat the procedure by choosing a new random k.
Compute $t = (H(m) + xr)k^{-1} \bmod q$.
If $t = 0$, repeat the procedure by choosing a new random k.
Output $s = (r, t)$.

Ver(pk, m, s):
Parse $pk = (\text{param}, y)$.
Parse $s = (r, t)$.
Compute $u = (g^{H(m)} y^r)^{t^{-1}} \bmod p$.
Output 1 if $r \equiv u \pmod q$, and 0 otherwise.

There exists some security proofs in the Random Oracle Model that require some extra assumptions. The proofs become simpler (but still in the Random Oracle Model) if $H(m)$ is replaced by $H(r, m)$.

2.3 Identification Schemes

The goal of an identification schemes is to verify the identity of a party involved in a protocol. For instance, the party can show that she knows the secret key sk corresponding to a public key pk (without revealing sk).

From a digital signature, it is easy to build an identification scheme. Namely, the owner of the secret key sk can sign a random message generated by the verifier with the key. Then the verifier can check whether the signature is valid with the corresponding public key pk. No one can impersonate the owner of sk, unless the signature scheme is insecure.

However, some identification schemes can be converted into digital signature schemes.

A special family of 3-moves protocols, called Sigma protocols, can be used to identify a party (called the Prover) to another one (called the Verifier). Assuming that the prover's public key pk is known to the verifier, the prover starts the protocol sending some committing information a. Then, the verifier sends a challenge c to the prover, who answers with a response t. Finally, the verifier checks whether the conversation (a, c, t) is a convincing one for the public key pk.

Correctness: If the prover's private input is (pk, sk) and the verifier's private input is (pk) and both parties follow the protocol honestly, then the verifier accepts the conversation with probability 1.

Soundness: If the prover's private input is just (pk) then the probability that a honest verifier accepts the

conversation is noticeably less than 1. A sequential repetition of the protocol can amplify the soundness to force the acceptance probability to a negligible function of λ.

Zero knowledge: (Informal) Whatever a honest verifier can compute after the protocol, can also be computed without interacting with a honest prover (implying that the verifier learns nothing about sk in a protocol execution).

2.3.1 Schnorr Identification Scheme

A typical example of sigma protocol is Schnorr's identification scheme, which is based on the hardness of computing discrete logarithms in a group G:

KeyGen(λ): (run by the prover)
Choose a prime number q of λ bits.
Choose a (cyclic) group G and a generator g.
Choose a random $x \in Z_q$ and compute the group element $y = g^x \in G$.
Output $(pk, sk) = ((\text{param}, y), x)$, where param $= (G, g, q)$.

The prover P sends pk to the verifier V and both start the 3-moves conversation:

$P(pk, sk) \leftrightarrow V(pk)$:
P samples a random $r \in Z_q$ and computes the commitment $a = g^r$.
P sends a to V.
V choses a random challenge $c \in Z_q$, and sends it back to P.
P computes the response $t = r + cx$ using the secret key.
P sends t to V.
V accepts the conversation if $g^t = ay^c$.

If a (possibly dishonest) prover P can make a honest verifier V accept a conversation (a, c, t) with probability greater than $1/q$, then there exists a commitment a such that P can compute t_1 and t_2 for some different c_1 and c_2 such that both (a, c_1, t_1) and (a, c_2, t_2) are accepted by V. But this implies that P can compute x from the two conversations. Indeed:

$g^{t_1} = ay^{c_1}$ and $g^{t_2} = ay^{c_2} \Rightarrow g^{t_1 - t_2} = y^{c_1 - c_2} \Rightarrow x = (t_1 - t_2)(c_1 - c_2)^{-1} \bmod q$.

Therefore, a prover that does not know sk can only convince the verifier with a probability at most $1/q$, unless it is able to solve the discrete logarithm problem in G.

Finally, the honest conversation (a, c, t) can easily be generated by V, without interacting with P, by computing $a = y^{-c} g^t$ for randomly chosen $c, t \in Z_q$.

2.3.2 Fiat-Shamir Conversion

A sigma protocol can be converted into a non-interactive identification scheme by replacing the challenge c by the hash of the public information pk and the commitment a. Then the prover only computes (a, t) with the (interactive) sigma protocol but using as challenge $c = H(pk, a)$.

The equivalence between the interactive and the non-interactive protocols can only be proven in the Random Oracle Model. But the non-interactive protocol can be turned easily into a signature scheme by adding the message m to the argument of the hash function. That is, the challenge is now computed as $c = H(pk, a, m)$, and the signature is the pair $s = (a, t)$

2.3.3 Schnorr Signature (1990)

Applying Fiat-Shamir transformation to Schnorr's identification scheme we obtain Schnorr's signature scheme:

KeyGen(λ):
Choose a prime number q of λ bits.
Choose a (cyclic) group G and a generator g.

Choose a secure hash function $H : \{0,1\}^* \to Z_q$.
Choose a random $x \in Z_q$ and compute the group element $y = g^x \in G$.
Output $(pk, sk) = ((\text{param}, y), x)$, where $\text{param} = (G, g, q, H)$.

Sig(pk, sk, m):
Parse $pk = (\text{param}, y)$.
Parse $sk = (x)$.
Choose a random $r \in Z_q$ and compute $a = g^r$.
Compute $c = H(y, a, m)$, and $t = r + cx$.
Output $s = (a, t)$.

Ver(pk, m, s):
Parse $pk = (\text{param}, y)$.
Parse $s = (a, t)$.
Compute $c = H(y, a, m)$.
Output 1 if $g^t = ay^c$, and 0 otherwise.

2.4 Certificates

As mentioned before, to avoid impersonation attacks like the man-in-the-middel attack the public keys must be unambiguously linked to the corresponding identities. The typical way to do it in practice is to concentrate the trust into a single certification authority (CA) that signs the public key/identity pairs, along with some extra information about the validity period and the intended use of the public keys.

A public key certificate consists of a structure containing the owner identity, the associated public key, some issuage information (like the issuer identity and the issuing time), the public key expiry date and the public key intended usage (public key encryption algorithms or public key signature algorithms for which the key is valid). This structure is signed with the secret key of the CA, so that any user that trusts it (and knows the public key of the CA) can verify the integrity of the certificate, and the validity of the public key contained in it.

A hierarchy of certification authorities can be based on the same model. A root CA can issue certificates for the intermediate CAs, by signing their signing public keys. The intermediate CAs can certify other CAs' public keys, or they can directly certify the public keys of the final users. Then, to verify the validity of a particular public key, the chain of trust from the root CA to the final user is used, and every certificate in the chain is verified with the public key of the issuer.

In some cases, a CA can enforce security by asking a user to prove in zero-knowledge that she knows the corresponding secret key. For instance, Schnorr's identification scheme can be used to prove knowledge of $x \in Z_q$ such that the public key $((G, g, q), y)$ satisfies $y = g^x$ without revealing x.
