Advanced Cryptography
Lecture Notes

2017 Edition

Serge Vaudenay

EPFL
Lausanne, Switzerland
`http://lasec.epfl.ch`

Contents

Chapter 1

The Cryptographic Zoo

1.1 The Menagery

Cryptographic primitives. Cryptographic primitives are described by

- components (parameters, participants in protocols, algorithms, domains, etc);
- a functionality (describing what happens if all participants play their role in a honest manner);
- security properties (describing what shall *not* happen if some participants are malicious, this is typically not easy to formalize).

Confidentiality is addressed by *encryption*, may it be symmetric or not. If it is symmetric, the same key is used to encrypt and to decrypt. So, it must remain secret. If it is asymmetric, a key pair is generated and the encryption key can be publicly revealed.

Message authentication and integrity are addressed by *MAC (message authentication codes)* — with a symmetric key — or by *digital signatures* — with a key pair, the verifying key becoming public.

Probabilistic algorithms sometimes need to flip a coin to make a decision. For convenience, we write $\mathcal{A}(x;r)$ to say that $\mathcal{A}$ runs on input x with a prepared sequence of random coins r. The sequence r must be large enough for $\mathcal{A}$ to complete. In this notation, r is separated from the regular inputs by a semicolon.

To formally define what it means to say that a computation is "easy" or "hard", we commonly refer to the notion of a polynomially bounded algorithm. A computation is easy if it can be done by an algorithm which runs in a time $O(s^n)$ for some integer n, depending on a parameter s. Normally, this parameter s is called the *security parameter*. As "polynomially bounded" usually refers to a polynomial in terms of the input length, we provide s written in unary (we write it 1^s) to make sure that the length is s (and not $\log_2 s$). So, to be precise, we write $\mathcal{A}(1^s, x; r)$ but it is more convenient to take 1^s implicit and omit it from the notation.

Participants running the cryptographic primitives are probabilistic polynomially bounded (PPT) algorithms, in terms of the security parameter s. This also includes adversaries. We say we use the *computationally bounded adversarial model*. However, we may sometimes assume no complexity bound and use the *information theoretic adversarial model*.

Symmetric encryption schemes. The components of symmetric encryption schemes are: a key length (the security parameter), the plaintext domain (it can be messages of the same specified length, e.g. for block ciphers, or messages of variable length), the key domain, and a nonce domain if applicable (typically, for stream ciphers), two participants (a sender and a receiver), and three algorithms: a key generator (it is quite often implicit: it consists of picking a key in the key domain with uniform distribution), an encryption algorithm, and a decryption algorithm. The functionality specifies that for every message X, $\Pr[\mathsf{Dec}_K(\mathsf{Enc}_K(X)) = X] = 1$ over the distribution of K. The security must formalize the notion of *confidentiality*.

We distinguish several security models, depending on the goal of the adversary (e.g., to do a key recovery or to decrypt a target ciphertext) and on the capabilities of the adversary. The adversary can only collect ciphertexts (in a ciphertext only attack), collect plaintext/ciphertext pairs (in a known plaintext attack), play with an encryption black box and choose the plaintext to be encrypted (in a chosen plaintext attack), or even play with a decryption black box and choose the ciphertext to be decrypted (in a chosen ciphertext attack). Playing with the two black boxes can further be done adaptively or nor. Hence, we describe 6 types of capabilities for 2 possible goals, leading us to 12 security models! To have the highest security, we should protect against the weakest attacks, e.g. decryption under adaptive chosen plaintext / ciphertext attacks.

Message authentication codes (MAC). The description of a message authentication code is similar. Typically, a message X is sent by appending a tag $\mathsf{MAC}_K(X)$. To authenticate X, one sends $\mathsf{Auth}_K(X) = X\|\mathsf{MAC}_K(X)$. Upon reception, the same operation is performed and compared with the received tag. To verify $X\|t$, one executes $\mathsf{Check}_K(X,t)$ which checks that $t = \mathsf{MAC}_K(X)$ and produces X as an output. The security corresponds to the notions of *message authentication and message integrity*.

The goal of an adversary could be to recover the key (*key recovery*), to forge the valid tag of some random X (*universal forgery*), or to forge the valid tag of some particular message (*existential forgery*). Its capabilities could be to collect authenticated messages or to choose the message to be authenticated. The stronger security model is the resistance to existential forgeries under chosen message attacks.

Commitment schemes. A commitment scheme can be described by a single probabilistic function $\mathsf{Commit}(X;r)$ taking the input X and the coins r. The commitment protocol between a sender and a receiver uses only one input X on the sender side and one output X on the receiver side. It works in two phases: in the commitment phase, the sender with input X picks r and sends $c = \mathsf{Commit}(X;r)$ to the receiver; in the opening phase, the sender reveals X and r, the receiver checks that $c = \mathsf{Commit}(X;r)$ and outputs X. Security should capture the notion of a *hiding* commitment (i.e., the receiver has no clue about X before the opening phase) and of a *binding* commitment (i.e., the sender cannot open the commitment on two different values X). This should be equivalent to putting a document X in a safe c closed with a key r, then giving the safe to the receiver, then handing out the key r to open it.

Pseudorandom number generators (PRNG). A PRNG can be defined by an algorithm mapping a state (seed) to a new state (new seed) and a generated number. There exists several security notions. One of these is the notion of *unpredictability*: an adversary receiving a sequence of generated numbers cannot predict with good probability what will be the next generated number. Another notion is the one of *indistinguishability*: an adversary producing a bit given a sequence of number produces X, when the sequence consists of generated numbers, and Y, when the sequence consists of truly random numbers. The advantage of the adversary is $\Pr[X = 1] - \Pr[Y = 1]$. For indistinguishability, we need that all adversaries have a negligible advantage.[1]

Hash functions. A hash function can be used to construct a commitment scheme, a pseudorandom generator, a *key derivation function (KDF)*, or to expand the domain of a primitive (e.g., a signature scheme). Since there are so many ways to use hash functions, there are also many different security notions. We can consider resistance to *first preimage attacks* (given y, find x such that $H(x) = y$), to *second preimage attacks* (given x, find $x' \neq x$ such that $H(x) = H(x')$), and to *collision attacks* (find x and x' such that $x \neq x'$ and $H(x) = H(x')$).

Key agreement protocols. A key agreement protocol is an interactive protocol between two participants called Alice and Bob. The two algorithms use no input and produce one output K. The functionality is that both outputs are equal. The security implies that no adversary looking at the protocol messages can infer K.

[1]The precise meaning of what "negligible" means will be introduced later in the lecture.

Public-key cryptosystems. In a public-key cryptosystem, a key generator produces a key pair (K_p, K_s). An encryption algorithm is probabilistic. A decryption algorithm is deterministic. The functionality says that $\mathsf{Dec}_{K_s}(\mathsf{Enc}_{K_p}(X)) = X$ with probability 1. Security works like in the symmetric case, except that the minimal adversarial capabilities are chosen plaintext attacks, since the adversary can do the encryption by himself by using the public key.

Digital signature schemes. In a digital signature scheme, a key generator produces a key pair (K_p, K_s). A signing algorithm is probabilistic. A verifying algorithm is deterministic. The functionality says that $\mathsf{Ver}_{K_p}(X, \mathsf{Sig}_{K_s}(X)) = \mathsf{ok}$ with probability 1. Security formalizes the notion of *non-repudiation*: a signer who signed a document cannot later claim that he did not sign. This implies that signatures are *unforgeable*, otherwise, the signer can claim that the signature was forged. We have similar security models as for message authentication codes.

1.2 The Math Toolbox

Finite Abelian groups. We work with finite Abelian groups. I.e., finite sets with an operation such that the set is closed under the operation, the operation is associative, there exists a neutral element, all elements are invertible, and the operation is commutative. Examples are $\mathbf{Z}_n$, $\mathbf{Z}_p^*$, $\mathsf{GF}(q)^*$, and the elliptic curve $E_{a,b}(\mathbf{K})$ for a finite field $\mathbf{K}$.

Since there is a single operation, we have groups with additive notations (e.g., the neutral element is 0, and we consider multiplying an integer n with a group element a by $n.a = a + \cdots + a$) and groups with multiplicative notations (e.g., the neutral element is 1, and we consider raising a group element a to the power of an integer n by $a^n = a \times \cdots \times a$).

Groups can be constructed in many ways. Given a big group, we can consider smaller groups (subgroups) generated by some elements. We can make the product of groups, raise a group to some power, and make the quotient of an Abelian group by one of its subgroups.

The order of a group is its cardinality. The order of an element x is the order of the group it generates. It is also the smallest $n > 0$ such that $x^n = 1$ (with multiplicative notations). The group exponent is the smallest $n > 0$ such that $x^n = 1$ for every element x. The order of an element divides the exponent of the group. The Lagrange theorem implies that the exponent of the group divides the order of the group.

Rings. A commutative ring has two operations $+$ and $\times$. It must be a group for $+$. The multiplication must be associative, have a neutral element, be commutative. Furthermore, there must be a distributivity of multiplication over addition. Examples include $\mathbf{Z}$, $\mathbf{Z}_n$, $\mathbf{Z}[x]$, $\mathbf{Z}_p[x]$. Instead of subrings, we consider *ideals*. We can make the product of rings, raise a ring to some power, and make the quotient of a ring by an ideal.

Euclidean rings have a Euclidean division. For instance, $\mathbf{Z}$ and $\mathbf{K}[x]$ are Euclidean rings. Euclidean rings are principal rings. I.e., every ideal can be generated by a single element. In principal rings, all elements have a unique factorization into irreducible elements, up to multiplication by units and permutations.

Given a ring R, we consider the group R^* of elements which are invertible for the multiplication. This forms a group for the ring multiplication.

In $\mathbf{Z}$, a number p is prime if $p > 1$ and

$$\forall a, b \in \mathbf{Z} \quad p = ab \Longrightarrow |a| = 1 \text{ or } |b| = 1$$

In $\mathbf{K}[x]$, a polynomial $P(x)$ is irreducible if

$$\forall A(x), B(x) \in \mathbf{K}[x] \quad P(x) = A(x)B(x) \Longrightarrow \deg(A) = 0 \text{ or } \deg(B) = 0$$

Finite fields. A finite field is a finite ring in which every nonzero element is invertible. The Galois theorem says that finite fields have a cardinality which is the power of a prime number and that finite fields with same cardinality are isomorphic. Furthermore, given a prime power $q = p^n$, we can construct such field $\mathsf{GF}(q)$ by taking an irreducible monic (i.e., with leading coefficient 1) polynomial $P(x)$ of $\mathbf{Z}_p[x]$ of degree n then defining $\mathsf{GF}(q) = \mathbf{Z}_p[x]/(P(x))$. In practice, we will use either $\mathbf{Z}_p$ or $\mathsf{GF}(2^n)$.

The $\mathbf{Z}_n$ ring. In $\mathbf{Z}_n$, x is invertible if and only if $\mathsf{gcd}(x,n)=1$. The cardinality of $\mathbf{Z}_n^*$ is $\varphi(n)$ and its exponent is $\lambda(n)$. If the p_i's are prime and pairwise different, we have

$$\begin{aligned}\varphi(p_1^{\alpha_1}\dots p_r^{\alpha_r}) &= (p_1-1)p_1^{\alpha_1-1}\times\cdots\times(p_r-1)p_r^{\alpha_r-1}\\ \lambda(p_1^{\alpha_1}\dots p_r^{\alpha_r}) &= \mathsf{lcm}\left((p_1-1)p_1^{\alpha_1-1},\cdots,(p_r-1)p_r^{\alpha_r-1}\right)\end{aligned}$$

We know that for all $x\in\mathbf{Z}_n^*$, we have $x^{\varphi(n)} \bmod n = 1$ and $x^{\lambda(n)} \bmod n = 1$.

The $\mathbf{Z}_p$ field. $\mathbf{Z}_p$ is a field if and only if p is a prime. In that case, we know that $\mathbf{Z}_p^*$ is a cyclic group. I.e., there exists elements g (called generators) such that all elements can be written as a power of g in the group. We have that $x^{p-1} \bmod p = 1$ for all $x\in\mathbf{Z}_p^*$. When $p>2$, p is odd and the set $\mathsf{QR}(p)$ of quadratic residues of $\mathbf{Z}_p^*$ (i.e., the set of the square of all $\mathbf{Z}_p^*$ elements) is a group of order $\frac{p-1}{2}$. Actually, $x\in\mathbf{Z}_p^*$ is a quadratic residue if and only if $x^{\frac{p-1}{2}} \bmod p = 1$.

The Chinese Remainder Theorem. We state the following result:

Theorem 1.1. *If m and n are two relatively prime integers (i.e., $\mathsf{gcd}(m,n)=1$), then the ring $\mathbf{Z}_{mn}$ of residues modulo mn is isomorphic to the product ring $\mathbf{Z}_m\times\mathbf{Z}_n$. One isomorphism is the function mapping $x\in\{0,\dots,mn-1\}$ to the pair $(x \bmod m, x \bmod n)$.*

This simple fact has many important consequences:

- For every (a,b) pair, there exists a unique integer x (up to a multiple of mn) such that $x \bmod m = a$ and $x \bmod n = b$ at the same time. We can compute x by inverting f. One way consists of computing
$$x = \left(an(n^{-1} \bmod m) + bm(m^{-1} \bmod n)\right) \bmod (mn)$$
- The group of units of both rings have the same cardinality. Namely, $\varphi(mn)=\varphi(m)\varphi(n)$.

We stress that this holds for $\mathsf{gcd}(m,n)=1$.

Random variables. A random variable is a process X transforming some random seeds (e.g., coin flips) into an element of some set $\mathcal{Z}$. The *support* of X is the set of all possible $\mathcal{Z}$ elements which can be taken by X. The *distribution* of X is a function from a set including the support of X to $\mathbf{R}$, mapping a value x to the probability $\Pr[X=x]$ that X takes the value x. In this lecture, we concentrate on *discrete* random variables. This assumes that $\mathcal{Z}$ is enumerable.

Two random variables X and Y are called *independent* if for all x and y, we have $\Pr[X=x,Y=y]=\Pr[X=x]\Pr[Y=y]$.

We now consider random variables with a support in a vector space over the reals. The expected value of X is
$$E(X)=\sum_{\mathsf{seed}}\Pr[\mathsf{seed}]X(\mathsf{seed})=\sum_{x\in\mathsf{support}(X)}x\Pr[X=x]$$
(we recall that X transforms some seed into a value $X(\mathsf{seed})$ of $\mathsf{support}(X)$.) The variance of X is
$$V(X)=\left(E(X-E(X))^2\right)=E(X^2)-E(X)^2$$
The expected value is a linear operator. I.e., for all $\lambda,\mu\in\mathbf{R}$, we have $E(\lambda X+\mu Y)=\lambda E(X)+\mu E(Y)$. The variance is quadratic. I.e., for all λ, we have $V(\lambda X)=\lambda^2V(X)$. When X and Y are independent, we have $E(XY)=E(X)E(Y)$.

For a function f and a random variably X, $f(X)$ is a new random variable. We have
$$E(f(X))=\sum_{x\in\mathsf{support}(X)}f(x)\Pr[X=x]$$

When X is Boolean (i.e., its support is included in $\{0,1\}$), we have $E(X)=p$ and $V(X)=p(1-p)$ where $p=\Pr[X=1]$.

1.3 The Algorithmic Toolbox

Algorithms over big numbers. Assuming a binary representation, the addition of x and y can be done with complexity $O(\ell)$, where ℓ is the bitlength of the numbers. The multiplication can be done with complexity $O(\ell^2)$, as well as the Euclidean division. This includes the computation of $x \bmod y$, for instance. The *extended Euclid algorithm* computes from x and y two integers a and b such that $ax + by = \mathsf{gcd}(x, y)$. This is done with complexity $O(\ell^2)$.

Modular arithmetic. We consider $\mathbf{Z}_n$ where n has a bitlength ℓ and elements are represented as numbers between 0 and $n-1$. The addition in $\mathbf{Z}_n$ can be done with complexity $O(\ell)$. The multiplication with schoolbook algorithm is done in complexity $O(\ell^2)$.

The inversion of an invertible element is done with complexity $O(\ell^2)$, using the extended Euclid algorithm. Actually, $x \in \mathbf{Z}_n$ is invertible if and only if $\mathsf{gcd}(x, n) = 1$, so if and only if the algorithm fed with x and n returns some a such that $(ax) \bmod n = 1$.

The computation of $x^e \bmod n$ is done with complexity $O(\ell^2 \log e)$ using the schoolbook multiplication.

If the factorization of n is provided, we can compute square roots of quadratic residues with complexity $O(\ell^3)$ (with schoolbook multiplication).

We can test the primality of an integer n of bitlength ℓ. If we use up to k iterations in the Miller-Rabin primality test algorithm, the probability of having an incorrect answer is bounded by 4^{-k}. Every iteration has a complexity of $O(\ell^3)$ (with schoolbook multiplication). A composite number is rejected with complexity $O(\ell^3)$ (with schoolbook multiplication). So, using the prime number theorem, we can generate random primes of length ℓ with complexity $O(\ell^4)$ (with schoolbook multiplication).

Birthday effect. Given a random function over a set of size N, we can find collisions with complexity $\sqrt{N}$ using the birthday paradox.

Generic attacks. For some encryption function based on a key of size n, we can do a key recovery of complexity $O(2^n)$ using *exhaustive search*. For a random hash function with range $\{0,1\}^n$, we can make a preimage attack with complexity $O(2^n)$. As already mentioned, collisions can be found with complexity $O(2^{\frac{n}{2}})$. Finally, for a message authentication code based on a key of size n, we can do a key recovery of complexity $O(2^n)$.

1.4 The Complexity Theory Toolbox

Membership problem. A *language* is a set of *words*, i.e., finite sequences of letters taken from a given alphabet. A membership problem is defined by a language L. An instance of the problem is a word x. The problem consists of deciding whether $x \in L$ or not. Languages in the class $\mathcal{NP}$ are of form

$$L = \{x; \exists w \quad R(x, w)\}$$

for some *predicate* R which can be evaluated in polynomial time. A value w such that $R(x, w)$ holds is a *witness* for x to be member of L. A problem is $\mathcal{NP}$-hard if solving it in polynomial time implies solving any problem in the class $\mathcal{NP}$.

Membership problems are problems consisting of computing one bit (i.e., whether the instance is in the language of not). We can consider problems consisting of computing several bits. For instance, the factoring problem consists of computing one non-trivial factor of the integer represented by the instance. The discrete logarithm problem consists, given g and y belonging to a group, in computing an integer x such that $g^x = y$. None of these problems are known to be $\mathcal{NP}$-hard. Nevertheless, they might by hard to solve.

The best algorithm to solve the factoring problem is the NFS algorithm. Factoring n takes

$$e^{O\left((\ln n)^{\frac{1}{3}} (\ln \ln n)^{\frac{2}{3}}\right)}$$

The best algorithm to solve the discrete logarithm problem in the group $\mathbf{Z}_p^*$ is index calculus. It works in complexity

$$e^{O\left((\ln p)^{\frac{1}{2}}(\ln \ln p)^{\frac{1}{2}}\right)}$$

Turing reduction. A problem (language) L_1 reduces to a problem (language) L_2 if there exists a polynomial-time oracle machine $\mathcal{A}^O$ solving L_1, given the oracle O assumed to solve L_2. That is, there exists an efficient algorithm to solve L_1 using as a subroutine an algorithm solving L_2 and with running time set to one unit. This notion of reduction is very useful to compare the difficulty of problems. Namely, if L_1 reduces to L_2, then L_1 is at most as hard to solve as L_2. That is, if we can solve L_2, then we can solve L_1 as well. Conversely, if L_1 is hard to solve, then L_2 is hard to solve as well.

The notion of reduction could be used to compare the complexity of two problems. Typically, we would compare the complexity of breaking a cryptosystem to the complexity of some well-known computational problem such as integer factoring.

Chapter 2

Cryptanalysis (Public-Key)

In this chapter, we review some case studies about situations where things can become badly insecure with public-key cryptography. We also start a systematic study of security analysis, to try to assess the difficulty of breaking security.

2.1 RSA

The so-called *textbook-RSA* cryptosystem [49] works as follows (see Fig. 2.1):

- for key generation, we generate two different prime numbers p and q, compute $N = pq$ and $\varphi(N) = (p-1)(q-1)$. Then, we pick some e such that $\mathsf{gcd}(e, \varphi(N)) = 1$ and compute $d = e^{-1} \bmod \varphi(N)$ using the extended Euclid algorithm. The public key is (e,N) and the secret one is (d,N).
- for encrypting a number $x \in \mathbf{Z}_N$, we compute $y = x^e \bmod N$.
- for decrypting a number $y \in \mathbf{Z}_N$, we compute $x = y^d \bmod N$.

For signature, we sign y by computing $x = y^d \bmod N$ and we check that x is a valid signature of y by checking $y = x^e \bmod N$. Interestingly, y can be extracted from x in the RSA case, so we could have a signature with *message recovery* (see Fig. 2.2). To assess the security of RSA, we essentially consider two problems:

- the RSA decryption problem: given an RSA public key (e,N) and a ciphertext y, compute x such that $y = x^e \bmod N$.

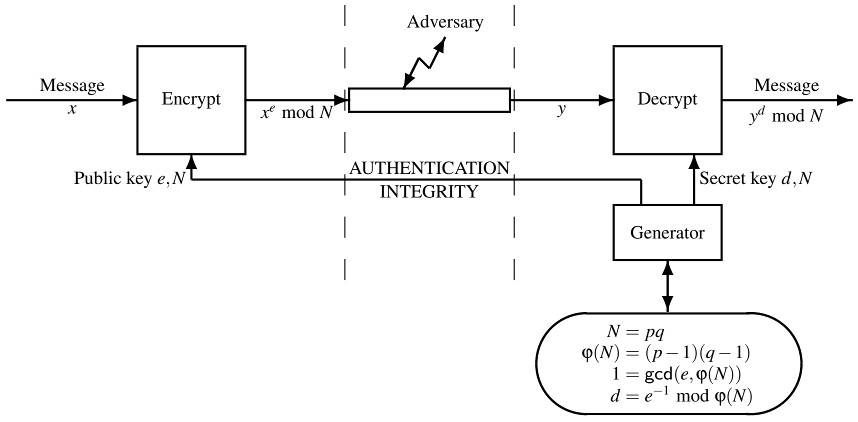


Figure 2.1: Textbook RSA Encryption

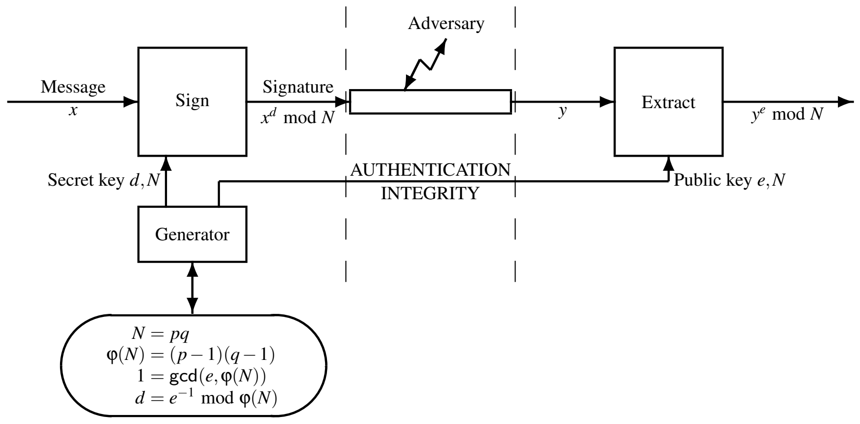


Figure 2.2: Textbook RSA Signature

- the RSA key recovery problem: given an RSA public key (e,N), find a number d such that for all $x \in \mathbf{Z}_N^*$ we have $x^{ed} \bmod N = x$.

We will compare them with some problems from number theory:

- the RSA factoring problem: given an RSA modulus N, find the factors p and q.
- the RSA order problem: given an RSA modulus N, compute $\varphi(N)$, the order of $\mathbf{Z}_N^*$.
- the RSA exponent multiple problem: given an RSA modulus N, find an integer k which is a positive multiple of $\lambda(N)$.

As for the last problem, we recall that the set of all k's such that $\forall x \in \mathbf{Z}_N^* \quad x^k \bmod N = 1$ is an ideal of the ring $\mathbf{Z}$ and that $\lambda(N)$ is the smallest positive such k. Since $\mathbf{Z}$ is a principal ring, this ideal is generated by $\lambda(N)$. Consequently, k is a multiple of $\lambda(N)$ if and only if $\forall x \in \mathbf{Z}_N^* \quad x^k \bmod N = 1$.

We can show, using Turing reductions, that the three above problems from number theory are equivalent to the RSA key recovery problem and that the RSA decryption problem reduces to the RSA key recovery problem. However, these two problems are not known to be equivalent although both are believed to be hard to solve.

RSA decryption reduces to RSA key recovery. This is essentially trivial: assuming that we have an oracle solving the RSA key recovery problem, given an instance (e,N,y) of the RSA decryption problem, we can submit (e,N) to the oracle and get d such that for all $y \in \mathbf{Z}_N^*$, $y^{ed} \bmod N = y$. So, by taking $x = y^d \bmod N$, we obtain that $x^e \bmod N = y$. This is just a complicated way to say that if we can recover the secret key, then we can apply the decryption algorithm to decrypt y!

RSA key recovery reduces to the RSA order problem. Assuming that we have an oracle which can compute $\varphi(N)$ from the RSA modulus N, given an RSA public key (e,N), we can first get $\varphi(N)$ using the oracle, then compute $d = e^{-1} \bmod \varphi(N)$. Clearly, for all $x \in \mathbf{Z}_N^*$, we have $x^{ed} \bmod N = x$. So, this solves the RSA key recovery problem.

The RSA exponent multiple problem reduces to RSA key recovery. Given an oracle which computes d from (e,N), the number $k = ed - 1$ satisfies $x^k \bmod N = 1$ for all $x \in \mathbf{Z}_N^*$. So, we can solve the RSA exponent multiple problem by taking a valid e. I.e., if we take a random e and that by any chance we have $\gcd(e, \varphi(N)) = 1$, we solve the problem. It is not guaranteed what happens when e is not valid since we don't know that the oracle returns (if it returns anything) in that case. What we could do is to iterate on random e's and compute the lcm of all obtained k's. Since eventually we will have a good e, it will return a solution k and the lcm will be another solution.

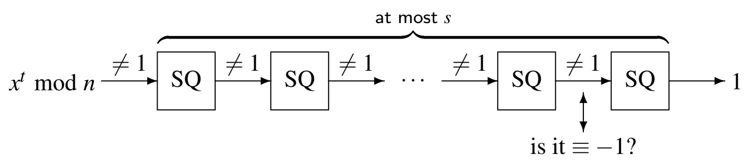


Figure 2.3: Factoring N using $\lambda(N)$

The RSA order problem reduces to RSA factoring. Given an oracle computing p and q from N, it is clear that we can compute $\varphi(N) = (p-1)(q-1)$.

RSA factoring reduces to the RSA order problem. Conversely, given an oracle computing $\varphi(N)$ from N, we notice that $p+q = N-\varphi(N)+1$ and $pq = N$. So, the quadratic equation $X^2-(N-\varphi(N)+1)X+N = 0$ over $\mathbf{R}$ has p and q as roots. Since it is easy to solve these equations in $\mathbf{R}$, we solve the RSA factoring problem.

RSA factoring reduces to the RSA exponent multiple problem. This is the most tricky reduction. Assuming an oracle giving an exponent multiple k from N, we factor N as follows: first, we write $k = 2^s t$ for some integers s and t such that t is odd (i.e., we iteratively divide by 2, s times in total, until the result t becomes odd). We know that for all $x \in \mathbf{Z}_N^*$, if we square iteratively s times the residue $x^t \bmod N$, we must obtain 1. We pick $x \in \mathbf{Z}_N - \{0\}$ at random. If $\gcd(x,N) \neq 1$, we find either p or q by some incredible chance and can stop. Otherwise, we deduce that $x \in \mathbf{Z}_N^*$. We compute $y = x^t \bmod N$. If $y = 1$, this is bad luck and we try again. Otherwise, we iteratively square y until $y^2 \bmod N = 1$. If $y \equiv -1 \pmod{N}$, this is bad luck and we try again. Otherwise, y is a square root or 1 which is neither 1 nor -1. So, $(y-1)(y+1)$ is a multiple of $N = pq$ such that neither $y-1$ nor $y+1$ is a multiple of N. So, $\gcd(y-1,N)$ is either p or q and we solve the factoring problem (see Fig. 2.3).

To prove that this works, we define s_p and s_q such that $\frac{p-1}{2^{s_p}}$ and $\frac{q-1}{2^{s_q}}$ are odd, then $s' = \max(s_p,s_q)-1$. Since k is a multiple of $\lambda(N) = \mathsf{lcm}(p-1,q-1)$, it is a multiple of $p-1$, so a multiple of 2^{s_p} as well. So, $s_p \leq s$. Similarly, $s_q \leq s$. Hence, $0 \leq s' < s$. The mapping $x \mapsto x^{2^{s'}t}$ over $\mathbf{Z}_p^*$ is a group homomorphism. Let H_p be the set of images of this function. Clearly, H_p is a subgroup of $\{1,-1\}$. If $s' \geq s_p$, this is $H_p = \{1\}$. Otherwise, for $s' = s_p - 1$, we know that a non-quadratic residue x modulo p would map to -1, so $H_p = \{1,-1\}$. We define H_q similarly. Without loss of generality, we assume that $s_p \geq s_q$. So, $H_p = \{1,-1\}$. Then we consider the mapping $x \mapsto x^{2^{s'-1}t}$ over $\mathbf{Z}_N^*$. This is a group homomorphism onto a group H which is isomorphic to $H_p \times H_q$ due to the Chinese remainder theorem. If $s_p = s_q$, we have $H_q = \{1,-1\}$. So, H contains four elements, including 1 and -1, and two "interesting" ones. (I.e., equal to 1 modulo either p or q but not both, and equal to -1 modulo either p or q but not both.) Otherwise, for $s_p > s_q$, we have $H_q = \{1\}$. So, H contains two elements, including 1 and an "interesting" one. In both cases, half of the element are "interesting". I.e., they are non-known square roots of 1. Since the mapping $x \mapsto x^{2^{s'-1}t}$ is homomorphic, it is balanced from $\mathbf{Z}_N^*$ to H. Hence, mapping a random element x gives an "interesting" element of H with probability $\frac{1}{2}$. So, the above produce works with probability at least $\frac{1}{2}$ in one iteration. By iterating enough, it works, eventually.

To conclude, RSA key recovery is equivalent to RSA factoring and to computing $\varphi(N)$ or any multiple. RSA decryption reduces to this but may be simpler. The equivalence is an open research problem.

RSA engineering. The textbook-RSA cryptosystem looks nice in textbooks. But using it in practice is not a piece of cake. Actually, we first have to realize that messages are not integers in practice, so we need some formatting rules. Then, there are usage and implementation issues. For instance, broadcasting a message to several users (each receiving the encryption of that message with his key) is insecure if the encryption exponent e is small. In general, there are many problems related to small e's or d's. In addition to this, implementation may leak some information through *side channels*.

Side channels can have various forms. For instance, devices provided with external power leak how much power they use over time. When stressed, devices can make computation errors, and the type of error may leak some information. The execution time may also leak some information. Finally, formatting rules added by protocols may also leak. We will see some leakage examples later.

RSA ISO standard. The ISO/IEC 9796 standard is an RSA signature standard providing *message recovery*, but suffering from some vulnerabilities. To sign a message, we apply an invertible formatting rule to transform it into a number, then sign that number using textbook RSA signature. When applying the textbook RSA extraction to the signature, we recover the number and can invert the formatting rule to recover the message.

The formatting rule looks like a cook recipe. What is important for the cryptanalysis to follow is to know that given a four-byte message $m = m_4m_3, m_2m_1$ such that $m_1 = 66$ in hexadecimal and the most significant bit of $S(m_4)$ is 1 for some byte permutation S, then formatting the message will lead to the number

$$x(m) \times \Gamma$$

for the constant $\Gamma = 1 + 2^{64} + 2^{128} + \cdots + 2^{k-64}$ (where k is the modulus bitlength, assumed to be a multiple of 64) and

$$x(m) = S(m_4)m_4S(m_3)m_3S(m_2)m_2\mathtt{2266}$$

Actually, the ISO standard requires that a single bit of $x(m) \times \Gamma$ is flipped. However, we will ignore it in what follows.

To break the standard (or, actually, the variant of it with no bit flip), we prepare many messages m of the above form and factor $x(m)$. (Since $x(m)$ has a bitlength of 64, this is easy.) Then, we only keep messages m such that $x(m)$ has no prime factor larger than 2^{16}. With a pool of a few hundred of such messages, it is likely that we find four messages m_g, m_h, m_i, m_j such that $x(m_g) \times x(m_h) = x(m_i) \times x(m_j)$. Consequently, if the σ's denote the signature of these messages, we obtain that $\sigma_g \times \sigma_h \equiv \sigma_i \times \sigma_j \pmod{N}$. So, we can make an existential forgery under chosen message attack: we just query the signatures $\sigma_g, \sigma_h, \sigma_i$ and we construct the signature σ_j.

This attack was presented in [17]. It was later extended to the full ISO signature standard [18].

Attack on broadcast RSA with low exponent. Assuming n users having an RSA public key (e, N_i), $i = 1, \ldots, n$ with same e and e so low that $e \leq n$, if someone broadcasts the message x (i.e., sends $y_i = x^e \bmod N_i$ to the ith user, $i = 1, \ldots, n$), then an adversary can easily decrypt x. Indeed, he can compute $y = x^e \bmod N$ for $N = N_1 \cdots N_n$ using the Chinese remainder theorem. Then, since x must be lower than all N_i's, we have $x^e < N$. So, $y = x^e$ over $\mathbf{Z}$. Now, we can use one's favorite algorithm to extract eth roots to y over $\mathbf{Z}$ to obtain x. This attack is due to Håstad [34]. It can be extended when the e's are different but all small.

Attack on related messages. There are extensions of the previous attack when several messages (with a known algebraic relation between them) are all encrypted with the same RSA public key. For instance, if a message x is concatenated with a counter (e.g., because the protocol requires messages to be numbered) and sent several times with a different counter, we can recover x. Typically, we can extract x from $y = x^e \bmod N$ and $y' = (x+1)^e \bmod N$ when e is small. The idea is essentially the same as the Euclid algorithm: we consider the ideal polynomials (in z) spanned by $z^e - y$ and $(z+1)^e - y'$. This is a pair of polynomials generating the ideal. By linear combination, we can reduce this pair into another equivalent pair where one polynomial is unchanged and the degree of the other is lowered. Typically, if the polynomial $P(z)$ with lowest degree starts has leading monomial αz^d and the other $Q(z)$ has $\beta z^{d'}$, we replace the latter by $Q(z) - \frac{\beta}{\alpha} z^{d'-d} P(z) \bmod N$. We iterate this reduction until we obtain a pair with a polynomial of form $\alpha z - \beta$, yielding the solution $x = \frac{\beta}{\alpha} \bmod N$. This attack was proposed by Coppersmith, Franklin, Patarin, and Reiter [21].

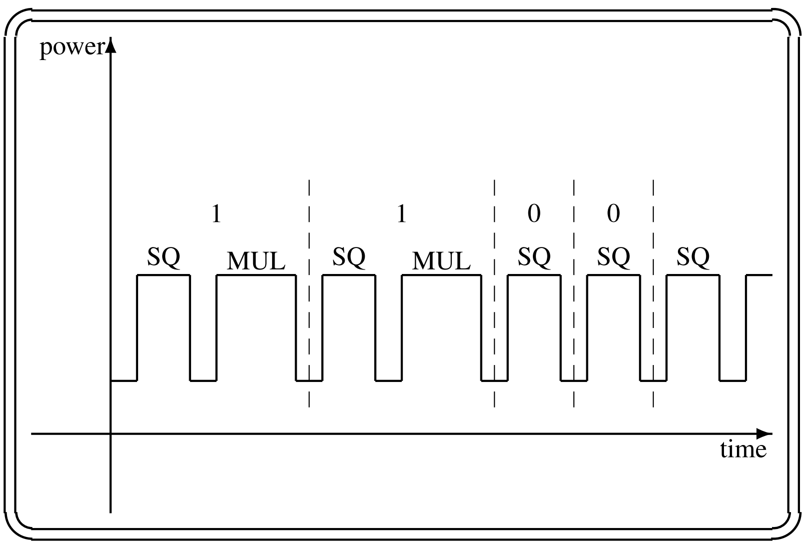


Figure 2.4: Simple Power Analysis

Attacks on low exponents. There are other problems related to low e's. Actually, the Coppersmith algorithm [19, 20] can be used to solve modulo N a polynomial equation of degree e when a root is known to be lower than $N^{\frac{1}{e}}$. This can be used to decrypt a message when $\frac{2}{3}$ of the plaintext bits are already known and $e = 3$. For instance, using a standard of form $\mathsf{Enc}(x) = (\mathsf{pattern}\|x)^3 \bmod N$ with x over ℓ bits and N larger than 3ℓ bits, we can write the equation $y = (2^\ell \mathsf{pattern} + x)^3 \bmod N$ and solve it with the Coppersmith algorithm.

There are other insecurity cases when d is short. For instance, for d of 64 bits, the Wiener algorithm [59] computes d from e and N.

Power analysis. Using the square-and-multiply algorithm, an RSA-decryption device just scans all the bits of d. For every bit, it is doing a squaring operation. If the bit is 1, it is doing an extra multiplying operation. In some implementations, these operations are done by an arithmetic coprocessor which is using more power than the microprocessor alone. Furthermore, squaring is typically faster than multiplying. So, when looking at the power consumption over time, we can see the square and multiply operations over time (see Fig. 2.4). We deduce all bits of d. This power analysis works for some smartcards, since they use external power sources. The smartcard industry has to address these potential problems by having countermeasures to smoothen the power consumption, of other decryption algorithms.

There are several possible attacks based on power consumption or on the time variation of computations. (See Kocher [36, 37].)

Differential fault analysis. When RSA decryption is implemented using the Chinese remainder theorem, the device computes $y^d \bmod p$, $y^d \bmod q$, and reconstruct $y^d \bmod N$ using CRT. If the device is stressed (by heating, increasing the power voltage, the clock frequency, etc) at some point it starts making errors. If there is only one computation error, it is likely to be done during either $y^d \bmod p$ or $y^d \bmod q$. An adversary who feeds the device with $y = x^e \bmod N$ for some random x will get some x' which is equal to x modulo either p or q but not both. Hence, $\mathsf{gcd}(x - x', N)$ is a prime factor of N and we can deduce p and q. This attack was presented by Boneh, DeMillo, and Lipton [13]. To defeat that, smartcards should have sensors to disconnect when some external stress is detected.

A protocol side channel in PKCS#1v1.5. The PKCS#1v1.5 standard imposes that plaintext messages shall start with 0002 in hexadecimal. Hence, for a k-byte long modulus, the plaintext is between $2 \times 256^{k-2}$ and $3 \times 256^{k-2}$. An adversary who has got a ciphertext y can try to submit $s^e y \bmod N$ to the server for

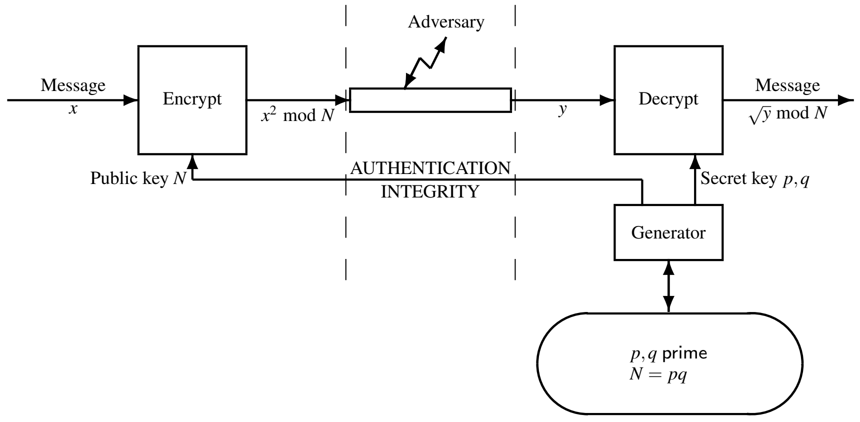


Figure 2.5: Textbook Rabin Cryptosystem

some chosen s. The server will decrypt and accept it as a valid message only when $sx \bmod N$ is in this interval. This can be used as an oracle to query whether $sx \bmod N$ is in this interval for some chosen s. Bleichenbacher [12] made this observation and derived an algorithm which, by using such oracle, is able to fully decrypt y into x. The algorithm was improved by Bardou *et al.* [3].

2.2 Rabin

The so-called *textbook-Rabin* cryptosystem [47] works as follows (see Fig. 2.5):

- for key generation, we generate two different prime numbers p and q, compute $N = pq$ and $\varphi(N) = (p-1)(q-1)$.
- for encrypting a number $x \in \mathbf{Z}_N$, we compute $y = x^2 \bmod N$.
- for decrypting a number $y \in \mathbf{Z}_N$, we compute $x = \sqrt{y} \bmod N$.

With this description, it is not really a cryptosystem because the $\sqrt{y} \bmod N$ operation is ambiguous. Actually, there are four square roots of y and it is not clear which one to take for the decryption. A technique to address this problem is to impose some redundancy in the plaintext (e.g., that there are 64 special bit positions all equal to 0). Since it is unlikely that another square root will satisfy this redundancy, we can decrypt non-ambiguously.

To assess the security of the Rabin cryptosystem, we essentially consider two problems:

- the Rabin decryption problem: given a Rabin public key N and a ciphertext y, compute one x such that $y = x^2 \bmod N$ (we do not consider the redundancy check here).
- the Rabin key recovery problem: given an Rabin public key N, factor N.

We can show that both are equivalent. Clearly, factoring N allows to compute square roots. So, the Rabin decryption problem reduces to the Rabin key recovery problem. Conversely, if we have an oracle solving the Rabin decryption problem, upon input N, we can pick $x \in \mathbf{Z}_N^*$ at random then submit $y = x^2 \bmod N$ to the oracle who will return x' such that $x^2 \equiv (x')^2$. Since x is a random square roots of y and that the oracle has no information on which one it is, we have that $x = \pm x' \bmod N$ with probability $\frac{1}{2}$. In the other cases, we deduce that $\mathsf{gcd}(x - x', N)$ is a non-trivial factor of N, so we can factor N.

On the one hand, we could favor the Rabin cryptosystem as opposed to RSA because the decryption problem is known to be equivalent to factoring, whereas RSA decryption may be easier than factoring. However, the proof of equivalence can also be viewed as a chosen ciphertext attack which breaks the Rabin cryptosystem. This is a pretty paradoxical situation where knowing that decryption is as hard as key recovery also leads to a devastating chosen ciphertext attack!

When introducing plaintext redundancy to avoid decryption ambiguity, the equivalence no longer holds, and nor does the attack. This continues the paradoxical situation... So, it seems that in order to have a better security, decryption should not be as hard as key recovery!

2.3 Diffie-Hellman

The so-called textbook Diffie-Hellman key agreement protocol [24] works as follows. We assume a standard cyclic group (such as $\mathbf{Z}_p^*$, a subgroup of it, an elliptic curve, etc) which is generated by some element g. Alice has a secret key $x \in \mathbf{Z}$ and a public key $X = g^x$. Bob has a secret key $y \in \mathbf{Z}$ and a public key $Y = g^y$. They both exchange X and Y and compute $K = g^{xy}$: Alice computes $K = Y^x$ and Bob computes $K = X^y$. The final key shared by Alice and Bob is K.

If an adversary — Eve — can interfere with the communication, she can perform a *man-in-the-middle attack*. It consists in running protocols independently with Alice and Bob, then ending up with sharing a key K_1 with Alice and a key K_2 with Bob. The protocol is supposed to resist to passive attacks: i.e., a passive Eve cannot infer K given g, X, and Y.

To assess the security of the protocol, we consider first the two following problems:

- the Diffie-Hellman problem: given (g, X, Y) in a given group, where $X, Y \in \langle g \rangle$, compute $K = g^{X^y}$ where $Y = g^y$.
- the discrete logarithm problem: given (g, y) in a given group, where $Y \in \langle g \rangle$, compute y such that $Y = g^y$.

Clearly, the Diffie-Hellman problem reduces to the discrete logarithm. However, the converse is still an open problem.

It must be stressed that the discrete logarithm problem is not always hard. Actually, in the group $\mathbf{Z}_n$, which is cyclic, with additive notations, computing the discrete logarithm of Y in basis g means finding y such that $Y = gy \bmod n$. This is clearly easy to solve by using the extended Euclid algorithm.

If n is a smooth number, i.e., if all its prime factors are less than a bound B which is small, then the discrete logarithm in a group of order n can be solved with $O(\sqrt{B} \log n)$ group operations by using the Pohlig-Hellman algorithm. So, the hardness implies a large prime factor in the order of the group. Consequences to cryptography were explored by van Oorschot and Wiener [42].

The Pohlig-Hellman algorithm [45] works as follows: in a group of order $n = p_1^{\alpha_1} \times \cdots \times p_r^{\alpha_r}$ where the p_i's are pairwise different primes and the α_i's are non-negative integers, we compute the logarithm of y in basis g

```
1: for $i = 1, \ldots, r$ do
2:     $g' \leftarrow g^{n/p_i^{\alpha_i}}$
3:     $g'' \leftarrow g'^{p_i^{\alpha_i - 1}}$
4:     $y' \leftarrow y^{n/p_i^{\alpha_i}}$
5:     $x_i \leftarrow 0$
6:     for $j = 0$ to $\alpha_i - 1$ do
7:         $y'' \leftarrow y'^{p_i^{\alpha_i - j - 1}}$
8:         compute the discrete logarithm $u$ of $y''$ in the subgroup of order $p_i$ which is spanned by $g''$ (next
           algorithm)
9:         $y' \leftarrow y' / g'^{u.p_i^j}$
10:        $x_i \leftarrow x_i + u.p_i^j$
11:    end for
12: end for
13: reconstruct and yield $x$ such that $x \equiv x_i \pmod{p_i^{\alpha_i}}$
```

Essentially, for each i we do α_i discrete logarithms in a group of order p_i. The idea is that for each i, by raising y and g to the power $n/p_i^{\alpha_i}$, we end up in a group of order $p_i^{\alpha_i}$ where the new y has the same logarithm in the new basis, modulo $p_i^{\alpha_i}$. Then, we recover all "basis-p_i digits" of the logarithm from the least significant to the most significant. If some digits are known, we divide y by g raised to the known part power, then raise the remainder to some power of p_i so that we end up in a group of order p_i, to compute the next digit. The final reconstruction is done by applying the Chinese Remainder Theorem.

To compute a logarithm in a group of prime order p, we apply the Baby-step Giant-step algorithm by Shanks [55]:

```
Precomputation
 1: let $\ell = \lceil \sqrt{B} \rceil$ be the size of a "giant step"
 2: for $i = 0, \ldots, \ell - 1$ do
 3:     insert $(g^{i\ell}, i)$ into a hash table
 4: end for
Computation
 5: for $j = 0, \ldots, \ell - 1$ do
 6:     compute $z = yg^{-j}$
 7:     if we have a $(z, i)$ in the hash table then
 8:         yield $x = i\ell + j$ and stop {we get $yg^{-j} = g^{i\ell}$}
 9:     end if
10: end for
```

Essentially, we store all "giant steps" $g^{i\ell}$ in the table and make "baby steps" yg^{-j} from y until we reach one value of the table. This algorithm has a complexity bounded by $O(\sqrt{p})$ group operations. So, the Pohlig-Hellman algorithm has a complexity bounded by $O((\alpha_1 + \cdots + \alpha_r)\sqrt{\max_i p_i})$. Since the sum of the α_i's is bounded by $\log_2 n$ and p_i is bounded by B, we obtain $O(\sqrt{B} \log n)$.

The decisional Diffie-Hellman problem. We consider another problem, relative to a generator selector Gen depending on some security parameter s:

- the decisional Diffie-Hellman problem: given (g, X, Y, K) in the group generated by $g \leftarrow \mathsf{Gen}(s)$, give an algorithm $\mathcal{A}$ that outputs a bit. We define

$$\mathsf{Adv}(\mathcal{A}) = \Pr_{\mathsf{exp}_1}[\mathcal{A}(g, X, Y, K) = 1] - \Pr_{\mathsf{exp}_0}[\mathcal{A}(g, X, Y, K) = 1]$$

  where exp_b is the experiment consisting of

```
1: take $g \leftarrow \mathsf{Gen}(s)$
2: generate $X, Y, K$ uniformly in $\langle g \rangle$
3: if $b = 1$, replace $K$ by the solution to the Diffie-Hellman problem with input $(g, X, Y)$
```

Intuitively, the decisional Diffie-Hellman problem consists of deciding whether a value K is the solution to the Diffie-Hellman problem (g, X, Y) or something independent. Clearly, this reduces to the Diffie-Hellman problem.

We say that the decisional Diffie-Hellman problem relative to Gen is hard if for any probabilistic polynomial-time algorithm $\mathcal{A}$, we have that $\mathsf{Adv}(\mathcal{A})$ is negligible in s. This means that for all n, $\mathsf{Adv}(\mathcal{A}) = O(s^{-n})$ as s grows to $+\infty$.

There are some groups for which this new hardness assumption does not hold. Among them, we have those for which the discrete logarithm problem is easy, but there are others. For instance, when p is an odd prime, $\mathbf{Z}_p^*$ does not satisfy this hardness assumption. Indeed, we can define $\mathcal{A}(g, X, Y, K)$ as producing 1 if and only if the property $\left(\frac{K}{p}\right) = -1$ holds at the same time as the property $\left(\frac{X}{p}\right) = \left(\frac{Y}{p}\right) = -1$. That is, K is not a quadratic residue if and only if both X and Y are not quadratic residues. In exp_1, we know that if either X or Y is a quadratic residue, then its logarithm is even, so the solution to the Diffie-Hellman problem is always a quadratic residue. So, $\mathcal{A}$ always outputs 1 in this experiment. In exp_0, K is independent from (X, Y) so $\mathcal{A}$ output 1 with probability $\frac{1}{2}$. Thus, $\mathsf{Adv}(\mathcal{A}) = \frac{1}{2}$. This is not negligible!

For the (supposedly) hard cases, we will consider a large subgroup of prime order of $\mathbf{Z}_p^*$, or of an elliptic curve. In the $\mathbf{Z}_p^*$ case, one way to define Gen is as follows:

```
1: pick a random prime $q$ of size $s$
2: pick a random number $p$ of size $\mathsf{poly}(s)$ such that $q | p - 1$
```

$$\begin{array}{rcl}
\textbf{Alice} & & \textbf{Bob} \\
\text{pick } x \in \mathbf{Z}_q^*, X \leftarrow g^x & \xrightarrow{\quad X \quad} & \text{if } X \notin \langle g \rangle - \{1\}, \text{abort} \\
\text{if } Y \notin \langle g \rangle - \{1\}, \text{abort} & \xleftarrow{\quad Y \quad} & \text{pick } y \in \mathbf{Z}_q^*, Y \leftarrow g^y \\
K \leftarrow \mathsf{KDF}(Y^x) & & K \leftarrow \mathsf{KDF}(X^y) \\
& (K = \mathsf{KDF}(g^{xy})) &
\end{array}$$

Figure 2.6: The Diffie-Hellman Key Agreement Protocol

```
3: start again until $p$ is prime
4: pick a random $h$ of $\mathbf{Z}_p^*$
5: set $g = h^{\frac{p-1}{q}} \bmod p$
6: if $g = 1$, start again with a new $h$
```

Other man-in-the-middle attacks. We could refine the man-in-the-middle attack to make sure that $K_1 = K_2$ and that Eve can have it as well. A trivial way consists, for Eve, in sending the public key 1 to both Alice and Bob. Clearly, we end up with $K_1 = K_2 = 1$. An easy way to avoid this attack is to check that the public keys are not equal to 1.

A more subtle attack works when the order of the group has small factors. For instance, if the order of the group is $2w$, Eve can receive X from Alice and send X^w to Bob, receive Y from Bob and send Y^w to Alice. The final key for Alice and Bob is $K = g^{xyw}$. We have X' and Y' living in the subgroup of the square roots of 1. The group is generated by g^w. So, Eve can compute the logarithm of X' in basis g^w (which is a bit ξ) and raise $Y^{w\xi}$ to obtain K.

More generally, if the order is bw and b is smooth, Eve can proceed the same way. $X' = X^w$ will be in a subgroup of order b, which is smooth, so she will be able to compute its logarithm in basis g^w, obtain ξ (which is now a residue modulo b), and raise $K = Y^{w\xi}$. To avoid these problems, we could mandate that the group has a prime order.

Making the Diffie-Hellman protocol secure. Another problem could be that K has a weird distribution depending on how the group is represented. To avoid that, we should consider K as a seed for a key derivation function KDF.

Finally, we consider the following Diffie-Hellman protocol: a parameter g generates a group of prime order q. Alice selects her secret key $x \in \mathbf{Z}_q^*$ and takes her public key $X = g^x$. Bob selects his secret key $y \in \mathbf{Z}_q^*$ and takes his public key $Y = g^y$. Alice and Bob check that the received public keys X and Y are in the group but not equal to 1. Alice and Bob compute $X^y = Y^x = g^{xy}$ then $K = \mathsf{KDF}(g^{xy})$.

One property of this protocol is that if Alice is honest and Y is selected independently of X, then Y^x is uniformly distributed in the group except 1. If Bob is honest, then X^y is uniformly distributed in the group except 1.

2.4 ElGamal

We assume a cyclic group generated by some g. The *ElGamal* cryptosystem [26] works as follows: (see Fig. 2.7):

- for key generation, we pick an integer x as a secret key and compute the public key $y = g^x$.
- for encrypting a group element m, we pick an integer r and compute the ciphertext $(u,v) = (g^r, my^r)$.
- for decrypting (u,v), we compute $m = vu^{-x}$.

We note that encryption is probabilistic. Indeed, running it multiple times will produce many different ciphertexts which all decrypt to the same message.

To assess the security of the ElGamal cryptosystem, we essentially consider two problems:

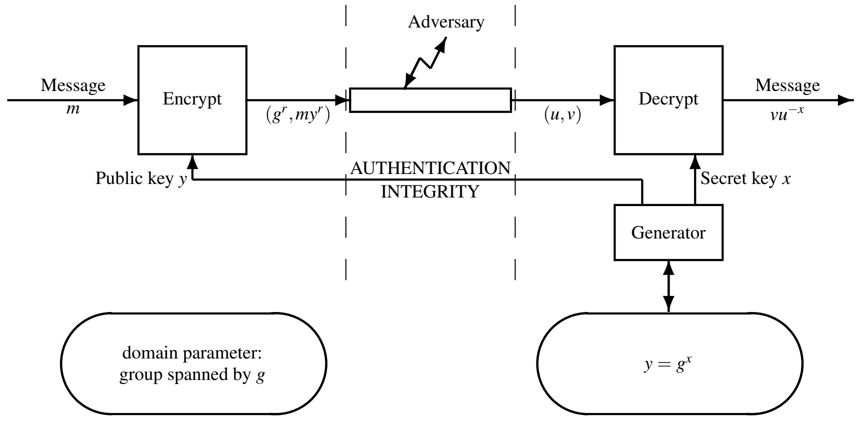


Figure 2.7: Textbook ElGamal Cryptosystem

- the ElGamal decryption problem: given an ElGamal public key y and a ciphertext (u,v), compute m such that $v = my^r$ for some r such that $u = g^r$.
- the ElGamal key recovery problem: given an ElGamal public key y, find x such that $y = g^x$.

Clearly, the ElGamal key recovery problem is equivalent to the discrete logarithm problem.

We can also show that the ElGamal decryption problem is equivalent to the Diffie-Hellman problem. Indeed, given a Diffie-Hellman solving oracle, we decrypt (u,v) for key y as follows: we compute $X = u$ and $Y = y$ and submit (g,X,Y) to the oracle to get $K = g^{rx}$. Then, we just divide v by K to obtain m. So, ElGamal decryption reduces to the Diffie-Hellman problem.

Conversely, given an ElGamal decryption oracle, we solve the Diffie-Hellman problem (g,X,Y) by setting $u = X$, $y = Y$, picking v at random in $\langle g \rangle$, sending (g,y,u,v) to the oracle to get $m = vu^{-x}$. Then, we set $K = v/m$ and it solves the Diffie-Hellman problem. So, the Diffie-Hellman problem reduces to ElGamal decryption. Therefore, both problems are equivalent.

ElGamal signature. The ElGamal digital signature scheme [26] works in the cyclic group $\mathbf{Z}_p^*$ generated by some g. It works as follows (see Fig. 2.8):

- for key generation, we pick an integer x as a secret key and compute the public key $y = g^x$.
- to sign a message M, we pick $k \in \mathbf{Z}_{p-1}^*$ at random and the signature is (r,s) with $r = g^k \bmod p$ and $s = \frac{H(M)-xr}{k} \bmod (p-1)$, where H is a hash function.
- to verify that (r,s) is a valid signature for M, we check that $0 \le r < p$ and that $y^r r^s \equiv g^{H(M)} \pmod{p}$.

Clearly, the key recovery problem is equivalent to the discrete logarithm problem in the same group. There exists a security result further saying that making existential forgeries under chosen message attack is hard, *on average* over the random choice of the parameters (p,g), and in the *random oracle model*, provided that the discrete logarithm problem is hard [46]. We will explain the random oracle model in an upcoming chapter. Unfortunately, security is only guaranteed for the average case: we will see that there are indeed some unfortunate choices of p and g which could make the signature scheme weak.

First, we have to stress that the condition $0 \le r < p$ in the signature verification is important. If we miss it, we can easily make universal forgeries. For that, we first pick $r_{p-1}, s \in \mathbf{Z}_{p-1}^*$ at random. Then, we set $r_p = g^{\frac{H(M)}{s}} y^{-\frac{r_{p-1}}{s}} \bmod p$. By using the Chinese remainder theorem, we can find r such that $r \bmod (p-1) = r_{p-1}$ and $r \bmod p = r_p$ at the same time. So, we easily see that (r,s) is a valid signature for M, except that r is of order p^2 instead of p. So, we really have to check that $0 \le r < p$.

Next, we see an unfortunate choice for p and g which was found by Bleichenbacher [11]. We have to assume that $p - 1 = bw$ with b smooth (e.g., we could take $b = 2$ since p is odd), and that we know

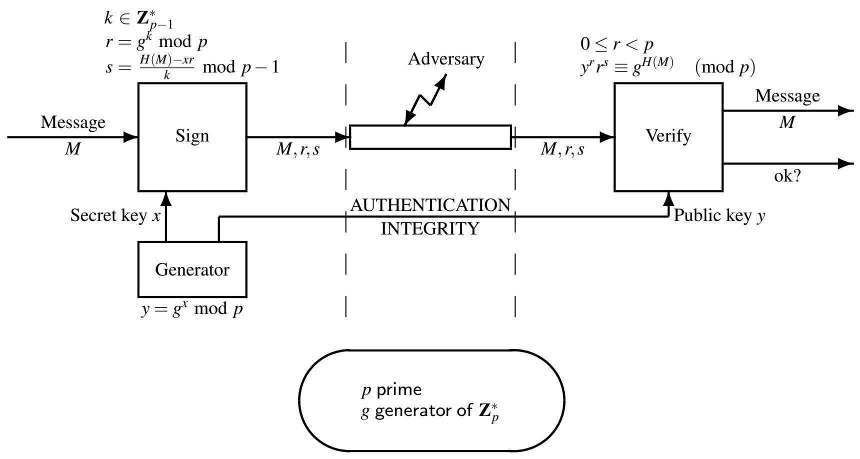


Figure 2.8: Textbook ElGamal Signature

some relation $g^{1/t} \bmod p = cw$ for some integers t and c. As an example, whenever b generates $\mathbf{Z}_p^*$ and $p \bmod 4 = 1$, we can take $g = b$, $t = \frac{p-3}{2}$, and $c = 1$. Indeed,

$$(cw)^t \equiv \left(\frac{p-1}{g}\right)^{\frac{p-1}{2}-1} \equiv -g\frac{(-1)^{\frac{p-1}{2}}}{g^{\frac{p-1}{2}}} \equiv g \pmod p$$

Once we have these two assumptions $p-1 = bw$ and $g^{1/t} \bmod p = cw$, we make a universal forgery for M by setting $r = cw$, finding the discrete logarithm z of y^{cw} in basis g^{cw}, i.e., $y^{cw} = g^{cwz}$, and taking $s = t(H(M) - cwz) \bmod (p-1)$. We clearly have $0 \le r < p$ and

$$y^r r^s \equiv y^{cw}(cw)^{t(H(M)-cwz)} \equiv y^{cw} g^{H(M)-cwz} \equiv g^{H(M)} \pmod p$$

So, (r,s) is a valid signature for M!

Chapter 3

Cryptanalysis (Conventional)

In this chapter we review some notions of cryptanalysis for block ciphers. More precisely, we describe differential and linear cryptanalysis. We apply it to DES reduced to 8 rounds. Then, we present some theory on the analysis with the notion of distinguisher. We discuss about the optimal one and see how to analyze the security of block ciphers with the notion of decorrelation.

3.1 Block Ciphers

One technique for symmetric encryption is based on *block ciphers*. This treats messages by *blocks* of fixed length, e.g., ℓ bits. Formally, a block cipher is a deterministic algorithm taking as input a plaintext block $x \in \{0,1\}^\ell$ and a secret key K and returning $y = C_K(x)$, a ciphertext block $y \in \{0,1\}^\ell$. It comes with another deterministic algorithm denoted by C^{-1} such that $C_K^{-1}(C_K(x)) = x$ for all x and K. So, for each K, C_K is a permutation of the set $\{0,1\}^\ell$.

The *perfect cipher* has $2^\ell!$ possible keys and is such that every possible permutation over $\{0,1\}^\ell$ has a key defining it. In terms of security, this is the best block cipher that we can dream of. Unfortunately, it is by far impractical as the key would be way too long to be representable. Indeed, we know the Stirling formula

$$n! \sim \sqrt{2\pi n} n^n e^{-n}$$

which implies that $\log_2(n!)$ can be approximated by $n \log_2 n$ when n is large. So, the most efficient binary representation of the keys requires $\log_2(2^\ell!)$ bits for a key, which is approximately $\ell 2^\ell$. For $\ell = 64$, which is nowadays considered as a too short block length, we obtain that we need more than one million of Petabytes to store a single key.

Instead of using the perfect cipher, we can still try to make ciphers look like the perfect one for the given usage. For instance, if the cipher is meant to be used only once, it is fair enough to require that for any x, the random variable $C_K(x)$, defined over the random choice of the key K, is uniformly distributed.

If the cipher is meant to be used only twice, we can simply require that for any x_1, x_2 with $x_1 \neq x_2$, $(C_K(x_1), C_K(x_2))$ is uniformly distributed among all pairs (y_1, y_2) satisfying $y_1 \neq y_2$. This is the notion of *pairwise independent permutation*.

This generalizes to *n-wise independent permutations*: for all $x_1, \ldots, x_n$ which are pairwise different, the tuple $(C_K(x_1), \ldots, C_K(x_n))$ is uniformly distributed among all $(y_1, \ldots, y_n)$ of pairwise different ciphertext blocks. If a cipher satisfies this criterion and if an adversary gets to learn no more than n pairs (x_i, y_i), then what he sees has the same distribution as what he would see if C was the perfect cipher. So, the cipher would ideally look like the perfect one, up to n samples.

3.2 Differential Cryptanalysis

Differential cryptanalysis was invented by Eli Biham and Adi Shamir. In 1990 [8], it was used to break some ciphers looking like DES. In 1992 [9], an attack was proposed (with a complexity too high for the

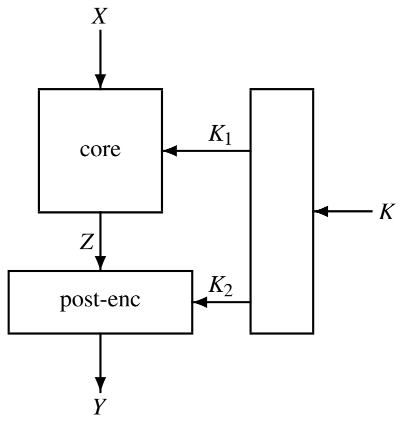


Figure 3.1: Splitting a Block Cipher for Differential Cryptanalysis

technology of that time) against DES. In 1993 [10], it was observed that any slight variant of DES would be subject to a more efficient and actually practical attack. Then, in 1994, Don Coppersmith (one of the designers of DES), released a technical report [16] showing that DES was built to resist to this type of attack. Indeed, this report showed that the technique of differential cryptanalysis was already taken into account by the DES designers in the 70's, even though it was not publicly known.

Differential cryptanalysis is a key recovery attack with *chosen* plaintexts. First, it requires to split the block cipher into three elements: a key schedule transforms K into K_1 and K_2; X is processed by the core encryption using K_1; then the result Z is processed by the post-encryption using K_2. This yields $Y = C_K(X)$ (see Fig. 3.1). Second, we must find a deviant property of the core encryption of the form $\Pr[Z' - Z = b | X' - X = a]$ is large, when X and X' are random, Z resp. Z' is the core encryption of X resp. X', and a and b are constants. Here, we use the XOR $\oplus$ as a notion of difference. I.e., $Z' - Z = Z' \oplus Z$ and $X' - X = X' \oplus X$. To find this deviant property, we will use heuristics (see below). Third, we isolate some verifiable information based on Y and Y' and a piece of information κ of K_2. This is a predicate $R(\kappa, \pi(Y,Y'))$ which is true whenever $Z' \oplus Z = b$ and κ is correct, and which is exceptionally true otherwise. The function π is used to compress Y and Y' to the required information needed in order to evaluate R. Finally, we run the attack based on statistics.

```
Precomputation:
 1: initialize $\mathsf{SubCandidate}_u$ to empty set for all $u$
 2: for all $u$ and all $\kappa$ such that $R(\kappa,u)$, insert $\kappa$ in $\mathsf{SubCandidate}_u$
Collection phase:
 3: collect $n$ pairs $((x,y),(x',y'))$ of plaintext-ciphertext pairs, with $x' = x \oplus a$
Analysis phase:
 4: initialize counters $m_\kappa$ to 0
 5: for each pair $((x,y),(x',y'))$ do
 6:     compute $u = \pi(y,y')$
 7:     for all $\kappa \in \mathsf{SubCandidate}_u$ increment $m_\kappa$
 8: end for
 9: sort all possible $\kappa$ in decreasing order of $m_\kappa$
Search phase:
10: for each sorted $\kappa$, exhaustively look for $K$
```

In the precomputation phase, we prepare some tables $\mathsf{SubCandidate}_u$ to quickly yield all possible κ such that $R(\kappa,u)$ holds. In the collection phase, we collect pairs of pairs (x,y) and (x',y') such that $y = C_K(x)$, $y' = C_K(x')$, and $x' \oplus x = a$. This is done by chosen plaintext attack. Then, during the analysis phase, we compute $u = \pi(y,y')$ and increment the counter of each key in $\mathsf{SubCandidate}_u$. Then, we can look at the score of all candidates and sort them by decreasing score. Finally, the search phase will treat each κ in the sorted list as the potential value corresponding to K. The idea is that with enough samples, the highest score will be made by the correct value.

Given a function f mapping p bits to q bits, we define a function DP^f by

$$\mathsf{DP}^f(a,b) = \Pr_X[f(X \oplus a) = f(X) \oplus b]$$

for $a \in \{0,1\}^p$, $b \in \{0,1\}^q$, and X uniformly distributed in $\{0,1\}^p$. This is the *differential probability*. Clearly, we have the following properties:

- $\mathsf{DP}^f(0,b) = 1$ if and only if $b = 0$;
- $\sum_b \mathsf{DP}^f(a,b) = 1$ for all a;
- $2^p \times \mathsf{DP}^f(a,b)$ is an even integer.

The last property comes from the fact that the number of x such that $f(x \oplus a) = f(x) \oplus b$ must be even: if x satisfies the relation, then $x \oplus a$ as well, so all these x's come in pairs. Clearly, the deviant property which is used in differential cryptanalysis can be expressed by $\mathsf{DP}^{C'_{K_1}}(a,b)$ being high.

To find the deviant property, we write the block cipher as a computation circuit, we look at the propagation of differences of plaintexts X and X' in this circuit, and we follow some heuristics. Clearly, if we have a linear gate M mapping an input X to an output Y, if two inputs are within a difference of ΔX, the resulting outputs will be within a difference of $\Delta Y = M \times \Delta X$. This can be applied to a duplicate gate mapping X to $M \times X = (X,X)$, so $M = (1\ 1)^t$,[1] or to a XOR gate, mapping (X,Y) to $M \times (X,Y) = X \oplus Y$, so $M = (1\ 1)$. When crossing a non-linear gate, we look at a plausible difference transform (by studying the differential properties of that gate) and we do the heuristic approximation that the difference propagation through all non-linear gates will be independent. So, we approximate $\mathsf{DP}^{C_K}(a,b)$ by the product of the probabilities that these propagations hold.

For the differential cryptanalysis for DES reduced to 8 rounds (instead of 16), we find a deviant property with a probability close to $2^{-13.4}$. We can further show that κ has 30 bits and that each key pair increases the score of 2^{10} counters m_κ. We assume that the selection of these counters look like random. So, each counter (for a wrong value) is incremented with probability $p_2 \approx \frac{2^{10}}{2^{30}} = 2^{-20}$ by each pair, and that the counter for the correct value κ is incremented with probability $p_1 = 2^{-13.4}$. The final score of this value will be np_1 on average, where n is the number of pairs. Typically, the distance to the expected value will be of order $\sqrt{np_1}$. This comes from the total score being the sum of n independent, identically distributed, random boolean variables with expected value p_1. So, the expected value of the sum is np_1 and the standard deviation of the sum is $\sqrt{np_1(1-p_1)} \approx \sqrt{np_1}$. Similarly, the expected value of m_κ for a bad κ will be within a distance of $\sqrt{np_2}$ to np_2. Clearly, $p_1 \gg p_2$. So, $np_1 - np_2 \approx np_1$ and $\sqrt{np_1} \gg \sqrt{np_2}$. Hence, whenever $\sqrt{np_1} \ll np_1$, we can separate the good counter from the bad ones and deduce κ (see Fig. 3.2). The condition for this to be the case is thus that $n \gg 1/p_1$.

3.3 Linear Cryptanalysis

In 1990 [28, 57], Henri Gilbert and his colleagues invented a way to break FEAL, a block cipher looking like DES. This inspired Mitsuru Matsui to develop *linear cryptanalysis* in 1993 [39], then to successfully apply it to DES in 1994 [40]. His attack is a key recovery *known* plaintext attack requiring 2^{43} known plaintexts.

Like for differential cryptanalysis, it first requires to split the block cipher into three elements: a key schedule transforms K into K_1 and K_2; X is processed by the core encryption using K_1; then the result Z is processed by the post-encryption using K_2 (see Fig. 3.3). This yields $Y = C_K(X)$. Second, we must find a deviant property of the core encryption of the form "$\left|\Pr[a \cdot X = b \cdot Z] - \frac{1}{2}\right|$ is large", when X is random, Z is the core encryption of X, a and b are constants, and $x \cdot y$ is the modulo 2 dot product between the vectors x and y. Third, we isolate some way to compute $a \cdot X \oplus b \cdot Z$ from X, Z, and a piece of information κ of K_2. This is a function $P(\kappa, \pi(X,Y))$ which is equal to $a \cdot X \oplus b \cdot Z$ whenever κ is correct, and which is uniformly distributed otherwise. Finally, we run the attack based on statistics.

[1] where $(1\ 1)^t$ denotes the transposed matrix of $(1\ 1)$

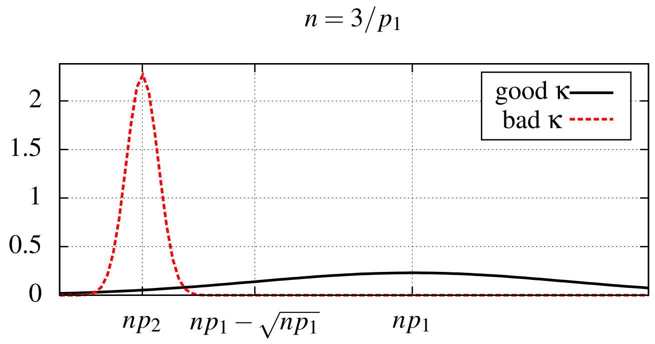


Figure 3.2: Probability Density of a Good and a Bad Counter in Differential Cryptanalysis

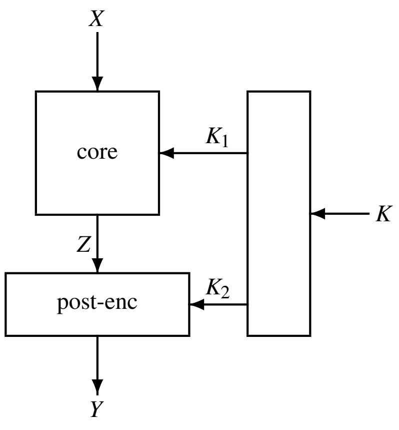


Figure 3.3: Splitting a Block Cipher for Linear Cryptanalysis

```
Collection phase:
1: for all possible $u = \pi(x,y)$ do
2:     initialize a counter $n_u$ to zero
3: end for
4: collect $n$ plaintext-ciphertext pairs $(x,y)$
5: for each $(x,y)$ do
6:     compute $u = \pi(x,y)$
7:     increment $n_u$
8: end for
Analysis phase:
9: for all possible $\kappa$ do
10:     compute $m_\kappa = \sum_{u \text{ s.t. } P(\kappa,u)=0} n_u$
11: end for
12: sort all $\kappa$ in decreasing order of $|m_\kappa - \frac{n}{2}|$
Search phase:
13: for each sorted $\kappa$ exhaustively look for $K$
```

In the collection phase, we just count how many pairs (x,y) give $\pi(x,y) = u$, for each u. Then, for each κ we compute m_κ which is how many times $P(\kappa, \pi(x,y))$ is equal to 0. Then, we can look at the score of all candidates and sort them by decreasing distance to $\frac{n}{2}$. Finally, the search phase will treat each κ in the sorted list at the potential value corresponding to K. The idea is that with enough samples, the highest score will be made by the correct value.

Given a function f mapping p bits to q bits, we define a function LP^f by

$$\mathsf{LP}^f(a,b) = \left(2\Pr_X[a \cdot X = b \cdot f(X)] - 1\right)^2$$

for $a \in \{0,1\}^p$, $b \in \{0,1\}^q$, and X uniformly distributed in $\{0,1\}^p$. This is the *linear probability*. Clearly, the deviant property which is used in linear cryptanalysis can be expressed by $\mathsf{LP}^{C'_{K_1}}(a,b)$ being high.

To find the deviant property $\Pr[a \cdot X = b \cdot Z]$ far from $\frac{1}{2}$, we proceed in a way which is the *dual* of what we did for differential cryptanalysis: we write the block cipher as a computation circuit, set the output mask b, and follow the computation backward to see what input mask to set. If we have a linear gate M mapping X to $Y = MX$, we know that

$$b \cdot Y = b \cdot (MX) = (M^t b) \cdot X = a \cdot X$$

when $a = M^t b$. So, an output mask b to M corresponds to an input mask $M^t b$. When crossing a non-linear gate S with output mask b, we look at the possible masks a making $\Pr[a \cdot X = b \cdot S(X)]$ far from $\frac{1}{2}$. We obtain $b \cdot S(X) = (a \cdot X) \oplus B$ for a biased bit B. When piling up all equations, the final relation around the core encryption looks like

$$(a \cdot X) \oplus (b \cdot Z) = \mathsf{bit}(K) \oplus \bigoplus_{i=1}^{n} B_i$$

for some Boolean function $\mathsf{bit}(K)$ of the key K and some biases bits B_i corresponding to the non-linear gates. To measure the bias of a random bit B, we define

$$\mathsf{LP}(B) = (2\Pr[B = 0] - 1)^2$$

Then, we make the *heuristic* assumption that all B_i's are independent[2] and apply the following result:

Lemma 3.1 (Piling-up Lemma). *Given some independent random bits $B_1, \ldots, B_n$, we have*

$$\mathsf{LP}(B_1 \oplus \cdots \oplus B_n) = \mathsf{LP}(B_1) \times \cdots \times \mathsf{LP}(B_n)$$

[2]This is of course not true, but in cryptanalysis, we often make some approximations to be able to make estimate, and we only care if the final implementation works: if we do recover the secret key, we do not care whether our mathematics analysis was formally correct or not!

Proof. To prove this, we observe that $\mathsf{LP}(B) = \left(E\left((-1)^B\right)\right)^2$ and apply the properties of independent variables. $\square$

It is interesting to see how differential and linear cryptanalysis are dual of each other. On one case, we were doing the computation forward on differences, applying the linear transforms M, computing DP's. On the other case, we were doing the computation backward on masks, applying the transposed linear transforms M^t, computing LP's. Unsurprisingly, there is a nice link between DP's and LP's. Actually, one is the *discrete Fourier transform* of the other, which is expressed by the following result.

Theorem 3.2. *If f is a function mapping p bit to q bits, we have*

$$\mathsf{DP}^f(a,b) = 2^{-q}\sum_{\alpha,\beta}(-1)^{a\cdot\alpha\oplus b\cdot\beta}\mathsf{LP}^f(\alpha,\beta)$$

and

$$\mathsf{LP}^f(\alpha,\beta) = 2^{-p}\sum_{a,b}(-1)^{a\cdot\alpha\oplus b\cdot\beta}\mathsf{DP}^f(a,b)$$

Proof. We first observe that

$$\mathsf{LP}^f(\alpha,\beta) = \left(E\left((-1)^{(\alpha\cdot X)\oplus(\beta\cdot f(X))}\right)\right)^2 = E\left((-1)^{(\alpha\cdot(X\oplus Y))\oplus(\beta\cdot(f(X)\oplus f(Y)))}\right)$$

where X and Y are independent and uniformly distributed in $\{0,1\}^p$. Then, we compute

$$\sum_{\alpha,\beta}(-1)^{a\cdot\alpha\oplus b\cdot\beta}\mathsf{LP}^f(\alpha,\beta) = E\left(\sum_{\alpha,\beta}(-1)^{(\alpha\cdot(a\oplus X\oplus Y))\oplus(\beta\cdot(b\oplus f(X)\oplus f(Y)))}\right)$$

Given X and Y, the inner sum over α and β is always zero, except if $X\oplus Y = a$ and $f(X)\oplus f(Y) = b$, in which case the sum is 2^{p+q}. So,

$$\begin{aligned}\sum_{\alpha,\beta}(-1)^{a\cdot\alpha\oplus b\cdot\beta}\mathsf{LP}^f(\alpha,\beta) &= 2^{p+q}E\left(1_{X\oplus Y=a,f(X)\oplus f(Y)=b}\right)\\ &= 2^q\mathsf{DP}^f(a,b)\end{aligned}$$

which gives the first equation. To obtain the second, we compute the right-hand side of the equation and replace DF^f by the expression we have just got:

$$\begin{aligned}\sum_{a,b}(-1)^{a\cdot\alpha\oplus b\cdot\beta}\mathsf{DP}^f(a,b) &= 2^{-q}\sum_{a,b}(-1)^{a\cdot\alpha\oplus b\cdot\beta}\sum_{\alpha',\beta'}(-1)^{a\cdot\alpha'\oplus b\cdot\beta'}\mathsf{LP}^f(\alpha',\beta')\\ &= 2^{-q}\sum_{\alpha',\beta'}\mathsf{LP}^f(\alpha',\beta')\sum_{a,b}(-1)^{a\cdot(\alpha\oplus\alpha')\oplus b\cdot(\beta\oplus\beta')}\end{aligned}$$

The inner sum is zero except for $\alpha=\alpha'$ and $\beta=\beta'$, for which it is 2^{p+q}. So, this expression is equal to $2^p.\mathsf{LP}^f(\alpha,\beta)$. $\square$

We could do a complexity analysis of the linear attack method. What we would obtain is that the required number of samples to find the correct κ with good probability has of order of magnitude $1/\mathsf{LP}^{C'_{K_1}}(a,b)$. Again, this is a result similar to the one of differential cryptanalysis where it was the inverse of $\mathsf{DP}^{C'_{K_1}}(a,b)$.

3.4 Hypothesis Testing in Cryptography

In cryptography, we are often concerned about *distinguishing* if some random samples follow a given distribution P_0 or a given distribution P_1. Concretely, we have a random source generating independent samples $x_1,\ldots,x_q$ following the same distribution P. Then, an algorithm $\mathcal{A}$ called a *distinguisher* analyzes

$x_1,\ldots,x_q$ and tries to guess whether $P = P_0$ or $P = P_1$. I.e., $\mathcal{A}(x_1,\ldots,x_q)$ is a bit. The ability to distinguish P_0 from P_1 is measured by the notion of *advantage*: we define

$$\mathsf{Adv}_{\mathcal{A}}(P_0,P_1) = \Pr[\mathcal{A}(x_1,\ldots,x_q) = 1|P = P_1] - \Pr[\mathcal{A}(x_1,\ldots,x_q) = 1|P = P_0]$$

We say that P_0 and P_1 are (q,ε)-indistinguishable if for any $\mathcal{A}$ limited to q samples, we have $|\mathsf{Adv}_{\mathcal{A}}(P_0,P_1)| \leq \varepsilon$.

In the theory of *hypothesis testing*, $\mathcal{A}$ is testing the null hypothesis

$$H_0 : \quad P = P_0$$

against the alternate hypothesis

$$H_1 : \quad P = P_1$$

The frequentist approach studies two types of errors:

- the type I error: $\alpha = \Pr[\mathcal{A}(x_1,\ldots,x_q) = 1|P = P_0]$, the error made by $\mathcal{A}$ thinking that the distribution is P_1 when it is actually P_0;
- the type II error: $\beta = \Pr[\mathcal{A}(x_1,\ldots,x_q) = 0|P = P_1]$ the error made by $\mathcal{A}$ thinking that the distribution is P_0 when it is actually P_1.

The Bayesian approach rather considers that both hypotheses have a probability π_0 resp. π_1 and studies the probability of error

$$P_e = \alpha\pi_0 + \beta\pi_1$$

In the typical case that we will use in this course, we have $\pi_0 = \pi_1 = \frac{1}{2}$, so

$$\mathsf{Adv}_{\mathcal{A}}(P_0,P_1) = 1 - 2P_e = 1 - (\alpha + \beta)$$

When limited to $q = 1$ sample a natural way to distinguish P_0 from P_1 is to take a decision based on whether $P_0(x) \leq P_1(x)$: if this inequality holds, then it is more likely that $P = P_1$ so we can output 1. This strategy is actually optimal as we can show. First, we can assume without loss of generality that $\mathcal{A}$ is deterministic. So, $\mathcal{A}$ is characterized by the set $\mathcal{A}^{-1}(1)$ of values of x producing the output 1. We have

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A}}(P_0,P_1) &= \sum_{x\in\mathcal{A}^{-1}(1)} (P_1(x) - P_0(x)) \\
&\leq \sum_{x:P_0(x)\leq P_1(x)} (P_1(x) - P_0(x)) \\
&= \frac{1}{2}\sum_x |P_1(x) - P_0(x)|
\end{aligned}$$

with equality when $\mathcal{A}^{-1}(1) = \{x : P_0(x) \leq P_1(x)\}$, which corresponds to the above natural strategy. Given some real functions f_0 and f_1, we define the *statistical distance* (or L_1 distance) between f_0 and f_1 by

$$d(f_0,f_1) = \frac{1}{2}\sum_x |f_1(x) - f_0(x)|$$

We obtain the following result:

Theorem 3.3. *For any $\mathcal{A}$ limited to $q = 1$ sample, we have $\mathsf{Adv}_{\mathcal{A}}(P_0,P_1) \leq d(P_0,P_1)$. The equality is reached for the algorithm producing 1 if and only if $P_0(x) \leq P_1(x)$.*

Proof. We have already proven the inequality. To study the equality case, we have to see what it implies in the proof for inequality. Clearly, equality implies that

$$\sum_{x\in\mathcal{A}^{-1}(1)} (P_1(x) - P_0(x)) = \sum_{x:P_0(x)\leq P_1(x)} (P_1(x) - P_0(x))$$

which means that $\mathcal{A}^{-1}(1)$ contains all x such that $P_0(x) < P_1(x)$ and maybe some extra x such that $P_0(x) = P_1(x)$. □

Given a distribution P on values x and an integer q, we define $P^{\otimes q}$ a distribution on q-tuples of values $(x_1,\ldots,x_q)$ by

$$P^{\otimes q}(x_1,\ldots,x_q) = P(x_1)\cdots P(x_q)$$

We can see the general case as a particular case of the $q = 1$ one: q samples can be considered as one sample of a q-tuple! So, we obtain the following result [2]:

Theorem 3.4. *For any $\mathcal{A}$ limited to q independent samples, we have $\mathsf{Adv}_{\mathcal{A}}(P_0,P_1) \leq d(P_0^{\otimes q},P_1^{\otimes q})$. The equality is reached for the algorithm producing* 1 *if and only if $P_0(x_1)\cdots P_0(x_q) \leq P_1(x_1)\cdots P_1(x_q)$.*

One remaining question is the following: how large must be q so that $d(P_0^{\otimes q},P_1^{\otimes q})$ is significant for cryptanalysis? We can easily show by induction that $d(P_0^{\otimes q},P_1^{\otimes q}) \leq qd(P_0,P_1)$. So, we need at least $q > 1/d(P_0,P_1)$, but it is not guaranteed that this would be enough. In what follows, we want to have a more precise estimate, based on some notions from the theory of large deviations.

Given a sample vector $x = (x_1,\ldots,x_q)$, we define the *observed distribution* (which is sometimes called a *type*) P_x by $P_x(y) = \frac{1}{q}\#\{i : x_i = y\}$. Given two distributions P_0 and P_1, we define the *Kullback-Leibler divergence*

$$D(P_0\|P_1) = \sum_{x\in\mathsf{Supp}(P_0)} P_0(x)\log\frac{P_0(x)}{P_1(x)}$$

where the log is in basis 2. Although this is not symmetric, this is very similar to a notion of distance: it is non-negative and equal to 0 if and only if $P_0 = P_1$. We define

$$\Pi = \{P : D(P\|P_1) \leq D(P\|P_0)\}$$

the set of distributions which are "closer" to P_1 than to P_0. We can easily see that the strategy from the above theorem outputs 1 if and only if the observed distribution of $x = (x_1,\ldots,x_q)$ is in Π. Indeed,

$$D(P_x\|P_1) - D(P_x\|P_0) = \sum_{y\in\mathsf{Supp}(P_x)} P_x(y)\log\frac{P_0(x)}{P_1(x)} = \frac{1}{q}\sum_{i=1}^{q}\log\frac{P_0(x_i)}{P_1(x_i)} = \frac{1}{q}\log\frac{P_0(x_1)\cdots P_0(x_q)}{P_1(x_1)\cdots P_1(x_q)}$$

We can then use a result by Sanov [51] to prove the following result.

Theorem 3.5 **([2]).** *Let $\mathsf{BestAdv}_q(P_0,P_1)$ be the largest advantage of a distinguisher between P_0 and P_1 limited to q queries. We have*

$$1 - \mathsf{BestAdv}_q(P_0,P_1) \doteq 2^{-qC(P_0,P_1)}$$

where

$$C(P_0,P_1) = -\inf_{0<\lambda<1}\log\sum_{x\in\mathsf{Supp}(P_0)\cap\mathsf{Supp}(P_1)} P_0(x)^{1-\lambda}P_1(x)^{\lambda}$$

is the Chernoff information between P_0 and P_1 and $f(q) \doteq g(q)$ means that $f(q) = g(q)e^{o(q)}$ when $q \to +\infty$.

This results generalizes to distinguishers with more elaborate hypotheses.

Theorem 3.6. *Consider the two following hypotheses:*

H_0: $x_1,\ldots,x_q$ are i.i.d. and follow the distribution P_0

H_1: $x_1,\ldots,x_q$ are i.i.d. and follow the same distribution taken in $\{P_1,\ldots,P_d\}$

Let $\mathsf{BestAdv}_q(H_0,H_1)$ be the largest advantage of a distinguisher between H_0 and H_1 limited to q queries. We have

$$1 - \mathsf{BestAdv}_q(H_0,H_1) \doteq 2^{-q\min_{1\leq i\leq d}C(P_0,P_i)}$$

The following distinguisher has advantage $\mathsf{BestAdv}_q(H_0,H_1)$:

```
input: $x_1,\ldots,x_q$
1: compute $P$, the type of $x_1,\ldots,x_q$
2: let $i$ be $\arg\min_{0\leq i\leq d} D(P\|P_i)$
3: if $i > 0$, output 1, otherwise, output 0
```

Note that checking $\arg\min_{0\leq i\leq d} D(P\|P_i) > 0$ is equivalent to checking

$$\min_{1,\leq i\leq d} \sum_{j=1}^{q} \log \frac{P_0(x_j)}{P_i(x_j)} \leq 0$$

Finally, we note that when a distribution P_1 tends towards a distribution P_0 with the same support, then we have

$$C(P_0,P_1) \sim \frac{1}{8\ln 2} \sum_x \frac{(P_1(x)-P_0(x))^2}{P_0(x)}$$

This can be used to compute the Chernoff information. When P_0 is the uniform distribution, we observe that it is proportional to the (squared) Euclidean distance between P_0 and P_1.

To see the link with differential and linear cryptanalysis, let us specify these two attack methods in a very schematic way. We use a random permutation C (which used to be the core encryption) and use a deviant property based on $p = \mathsf{DP}^C(a,b) = \Pr[C(x\oplus a) = C(x)\oplus b]$. When this is really a deviant property, we can distinguish C from an ideal cipher C^* (for which $\mathsf{DP}^{C^*}(a,b) \ll p$). So, let us focus at the following problem: the adversary collects some samples of $Z = C(x\oplus a)\oplus C(x)$. He wants to distinguish if Z is uniformly distributed (null hypothesis) or if it follows the distribution $\Pr[Z=b] = p$ and $\Pr[Z=c] = \frac{1-p}{n-1}$ for all $c \neq b$ (alternate hypothesis). We denote $\beta = \frac{1-p}{n-1}$. By computing the Chernoff information between the two distributions, we obtain that it is equivalent to $\frac{p}{\ln 2}$, asymptotically. So, the required number of samples to reach a good advantage is of order $1/p = 1/\mathsf{DP}^C(a,b)$. We can further describe the best distinguisher which is based on the number of occurrences $\#b$ of $Z=b$. We (easily) see that the likelihood ratio $R\leq 1$ is equivalent to $\#b \geq q\frac{\ln(n\beta)}{\ln(\beta/p)}$. With q of order $\frac{1}{p}$, this is equivalent to $\#b > 0$. So, the best distinguisher just observes if any sample Z is equal to b.

For linear cryptanalysis, we take the same approach and define $Z = (a\cdot x)\oplus(b\cdot C(x))$. The approximation says that Z has a probability to be able to deduce some key bit with probability $\frac{1}{2}(1+\varepsilon)$. Since this key bit is uniformly distributed, we have that Z follows a distribution P_β defined by $P_\beta(0) = \frac{1}{2}(1+(-1)^\beta\varepsilon)$, for $\beta \in \{0,1\}$. Note that $\mathsf{LP}^C(a,b) = \varepsilon^2$, no matter β. Now, we want to distinguish whether Z follows the uniform distribution (null hypothesis) from that it follows some distribution P_β (alternate hypothesis). By computing the Chernoff information between the uniform distribution and P_β, we obtain approximately $\varepsilon^2/(8\ln 2)$. So, the required number of samples to reach a good advantage is of order $1/\varepsilon^2 = 1/\mathsf{LP}^C(a,b)$. We can further describe the best distinguisher which is based on the number of occurrences #0 of $Z=0$. We easily see that the likelihood ratio $R_\beta \leq 1$ is equivalent to

$$(-1)^\beta \frac{q}{\#0} \leq (-1)^\beta \frac{\log(1-(-1)^\beta\varepsilon)}{\log\frac{1-(-1)^\beta\varepsilon}{1+(-1)^\beta\varepsilon}}$$

Given that $\varepsilon = o(1)$, the distinguisher based on the composite hypothesis yields 1 if and only if $|\#0 - \frac{q}{2}| \geq \frac{\varepsilon}{4}q$. So, the best distinguisher just observes if any sample Z is equal to 0 and checks that the number of such samples is far from $\frac{q}{2}$. The analysis also makes the threshold precise, as for what is considered to be "far".

3.5 Decorrelation

We assume that a distinguisher is given access to an oracle implementing some random function from a set A to a set B. We know that either it has the distribution of a random function F or a distribution of an ideal random function F^*. For instance, F is a random cipher C (set up with a random key) on the set A (and $B=A$) and F^* is the perfect cipher C^* over A. As another example, F is a function defined by a MAC with a random key and F^* is a uniformly distributed random function. We assume that the distinguisher is limited with the number of queries q that he can make. The distinguisher is not limited in complexity.

Given a random function F from A to B and an integer q, we define a (huge) real matrix $[F]^q$ in which rows have an index corresponding to a tuple $x = (x_1,\ldots,x_q)$ of q inputs and columns have an index corresponding to a tuple $x = (y_1, \ldots, y_q)$ of q outputs. The element $[F]^q_{x,y}$ at position (x, y) is the real number $\Pr[F(x_1) = y_1, \ldots, F(x_q) = y_q]$. Decorrelation to the order q is the distance between $[F]^q$ and $[F^*]^q$.

It is convenient to define the distance in terms of a matrix norm. Matrix norms are norms (i.e., $\|M\|$ is always positive, equal to 0 if and only if $M = 0$, $\|\lambda M\| = |\lambda| \times \|M\|$, and $\|M+M'\| \leq \|M\| + \|M'\|$) with the additional property that $\|MM'\| \leq \|M\| \times \|M'\|$. For instance, the ∞-norm over the vectors $\|v\|_\infty = \max_y |v_y|$ induces a companion matrix-norm

$$|||M|||_\infty = \max_{\|v\|_\infty \leq 1} \|Mv\|_\infty = \max_x \sum_y |M_{x,y}|$$

This norm bridges the theory of decorrelation with the theory of best non-adaptive distinguishers as the following result shows. A distinguisher is non-adaptive if it prepares all its queries at once. Namely, it does not adapt a query x_i based on the response from previous queries.

Theorem 3.7 ([58]). *For any random functions F and G, the best advantage of a non-adaptive distinguisher between F and G, limited to q queries, is equal to $\frac{1}{2}|||[F]^q - [G]^q|||_\infty$.*

Proof. A non-adaptive distinguisher can be assumed to prepare the q queries $x_1, \ldots, x_q$ before making any query. Then, he obtains a vector $(Y_1, \ldots, Y_q)$ of random variables defined by $Y_i = F(x_i)$ in the F case and $Y_i = G(x_i)$ in the G case. So, this reduce to distinguish the distributions of $(F(x_1), \ldots, F(x_q))$ and $(G(x_1), \ldots, G(x_q))$. We know from Th. 3.3 that the best advantage is half of the statistical distance between the two distributions, hence

$$\mathsf{Adv} = \frac{1}{2} \sum_{y_1,\ldots,y_q} \left|\Pr[F(x_1) = y_1, \ldots, F(x_q) = y_q] - \Pr[G(x_1) = y_1, \ldots, G(x_q) = y_q]\right|$$

The best advantage over the choice of $x_1, \ldots, x_q$ is

$$\mathsf{Adv} = \frac{1}{2} \max_{x_1,\ldots,x_q} \sum_{y_1,\ldots,y_q} \left|\Pr[F(x_1) = y_1, \ldots, F(x_q) = y_q] - \Pr[G(x_1) = y_1, \ldots, G(x_q) = y_q]\right|$$

which is $\frac{1}{2}|||[F]^q - [G]^q|||_\infty$. □

To compute the advantage of a distinguisher which can be adaptive, we use the norm

$$\|M\|_a = \max_{x_1} \sum_{y_1} \cdots \max_{x_q} \sum_{y_q} |M_{((x_1,\ldots,x_q),(y_1,\ldots,y_q))}|$$

This is indeed a matrix norm [58]. Just like the previous theorem, we can prove the following result.

Theorem 3.8 ([58]). *For any random functions F and G, the best advantage of a distinguisher between F and G, limited to q queries, is equal to $\frac{1}{2}\|[F]^q - [G]^q\|_a$.*

Decorrelation enjoys the following property.

Theorem 3.9 ([58]). *If C_1 and C_2 are independent random permutations, to be compared with a uniformly distributed random permutation C^*, for any matrix norm, we have that*

$$\|[C_2 \circ C_1]^q - [C^*]^q\| \leq \|[C_1]^q - [C^*]^q\| \times \|[C_2]^q - [C^*]^q\|$$

Proof. We first observe that $[C_2 \circ C_1]^q = [C_1]^q \times [C_2]^q$.

Then, we notice that $[C^*]^q$ has an absorbing property. Indeed, $[C_1]^q \times [C^*]^q$ is equal to $[C^* \circ C_1]^q$. Since C^* and C_1 are independent, in the group of permutations, and that C^* is uniformly distributed, $C^* \circ C_1$ and C^* have the same distribution. So, $[C^* \circ C_1]^q = [C^*]^q$ from which we deduce $[C_1]^q \times [C^*]^q = [C^*]^q$. We similarly show that $[C^*]^q \times [C_2]^q = [C^*]^q$.

Now, we have

$$([C_1]^q - [C^*]^q) \times ([C_2]^q - [C^*]^q) = [C_2 \circ C_1]^q - [C^*]^q$$

by expanding the product, thanks to the absorbing property of $[C^*]^q$. Due to the matrix norm multiplicative property, we have

$$\|[C_2 \circ C_1]^q - [C^*]^q\| \leq \|[C_1]^q - [C^*]^q\| \times \|[C_2]^q - [C^*]^q\|$$

□

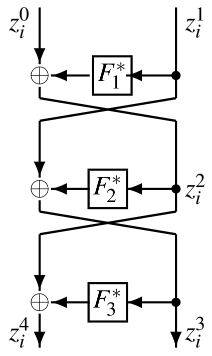


Figure 3.4: Proof of the Luby-Rackoff Theorem

We can apply this result on the Luby-Rackoff Theorem.

Theorem 3.10 (Luby-Rackoff 1986 [38]). *Let F_1^*, F_2^*, F_3^* be three independent round functions with uniform distributions from the set of $\frac{\ell}{2}$-bit strings to itself. We consider the 3-round Feistel scheme $C = \Psi(F_1^*, F_2^*, F_3^*)$ to be compared with the ideal cipher C^*. For all distinguisher limited to q queries, the advantage to distinguish C from C^* is bounded by $q^2.2^{-\frac{\ell}{2}}$.*

Proof. We split an input x_i into $x_i = (z_i^0, z_i^1)$. Similarly, we split and output $y_i = (z_i^4, z_i^3)$. We further define $z_i^2 = z_i^0 \oplus F_1^*(z_i^1)$. Clearly, x_i maps to y_i if and only if $z_i^3 = z_i^1 \oplus F_2^*(z_i^2)$ and $z_i^4 = z_i^2 \oplus F_3^*(z_i^3)$ (see Fig. 3.4). Let E be the event that for $i = 1, \ldots, q$, we have $z_i^3 = z_i^1 \oplus F_2^*(z_i^2)$ and $z_i^4 = z_i^2 \oplus F_3^*(z_i^3)$. We obtain that $[C]_{x,y}^q = \Pr[E]$.

Let $\mathcal{Y}$ be the set of all $y = (y_1, \ldots, y_q)$ such that for all $i \neq j$, y_i and y_j define some z_i^3 and z_j^3 such that $z_i^3 \neq z_j^3$. If we take a uniformly distributed random y, the probability that it is not in $\mathcal{Y}$ is $\Pr[\exists i < j\ z_i^3 = z_j^3]$. This is bounded by $\frac{q(q-1)}{2}$ times $\Pr[z_i^3 = z_j^3] = 2^{-\frac{\ell}{2}}$. So, we have

$$\Pr[y \in \mathcal{Y}] \geq 1 - \varepsilon$$

with $\varepsilon = \frac{q(q-1)}{2} 2^{-\frac{\ell}{2}}$.

Let x be arbitrary and let $y \in \mathcal{Y}$ be arbitrary but in $\mathcal{Y}$. We define the event E^2 that all z_i^2 are pairwise different. Just like above, we have $\Pr[E^2] \geq 1 - \varepsilon$. Then, we have

$$[C]_{x,y}^q = \Pr[E] \geq \Pr[E, E^2] = \Pr[E|E^2]\Pr[E^2]$$

If the z_i^2 are pairwise different, the $F_2^*(z_i^2)$ are uniform and independent. Since we also know that the z_i^3 are pairwise different, the $F_3^*(z_i^3)$ are also uniform and independent. Hence $\Pr[E|E^2] = 2^{-\ell q}$. We deduce

$$[C]_{x,y}^q \geq (1-\varepsilon)2^{-\ell q} = (1-\varepsilon)[F^*]_{x,y}^q$$

Therefore, we have found a set $\mathcal{Y}$ such that $\Pr[y \in \mathcal{Y}] \geq 1 - \varepsilon$ and $[C]_{x,y}^q \geq (1-\varepsilon)[F^*]_{x,y}^q$ for all $y \in \mathcal{Y}$. By applying Lemma 3.11 below, we deduce that the best advantage to distinguish C (denoted by F in Lemma 3.11) from F^* limited to q queries is bounded by $2\varepsilon = q(q-1)2^{-\frac{\ell}{2}}$.

In Lemma 3.12, we show that the best advantage to distinguish C^* from F^* is bounded by $q(q-1)2^{-\ell}$. For $q \leq 2^{\frac{\ell}{2}}$, the sum is bounded by $q2^{-\frac{\ell}{2}}$. So, the best advantage to distinguish C from C^* limited to q queries is bounded by $q^2 2^{-\frac{\ell}{2}}$. For q larger, this bound is larger than 1 so the advantage is also bounded by this. □

The lemma below is inspired by Patarin's "*H* coefficient technique" [43].

Lemma 3.11. *Let F be a random function from a set $\mathcal{M}_1$ to a set $\mathcal{M}_2$. We let X be the subset of $\mathcal{M}_1^q$ of all $(x_1, \ldots, x_q)$ with pairwise different entries. We let F^* be a uniformly distributed random function from $\mathcal{M}_1$ to $\mathcal{M}_2$. We assume there exists a subset $\mathcal{Y} \subseteq \mathcal{M}_2^q$ and two positive numbers ε_1 and ε_2 such that*

- $\Pr[y \in \mathcal{Y}] \geq 1 - \varepsilon_1$ *for* $y \in \mathcal{M}_2^q$ *random*
- $\forall x \in \mathcal{X} \quad \forall y \in \mathcal{Y} \quad [F]^q_{x,y} \geq (1-\varepsilon_2)[F^*]^q_{x,y}$.

Then the best advantage to distinguish F from F^* limited to q queries is bounded by $\varepsilon_1 + \varepsilon_2$.

Proof. We know that for all $x \in \mathcal{X}$ and $y \in \mathcal{M}_2^q$ we have $[F^*]^q_{x,y} = p_0$ for the constant $p_0 = (\#\mathcal{M}_2)^{-q}$.

Without loss of generality, the best distinguisher is deterministic. Let x_i be its ith query and y_i the response from the oracle. (Note that x_i can depend on $y_1, \ldots, y_{i-1}$.) Let A be the set of all $y_1, \ldots, y_q$ making the distinguisher output 1. We assume without loss of generality that $x \in \mathcal{X}$ (if a query repeats, we can replace it by an arbitrary new one and substitute the answer to the previously known answer of the repeating query). The advantage is

$$\mathsf{Adv} = \sum_{y \in A} \left([F^*]^q_{x,y} - [F]^q_{x,y} \right)$$

For $y \in \mathcal{Y}$, we have $[F^*]^q_{x,y} - [F]^q_{x,y} \leq \varepsilon_2 [F^*]^q_{x,y}$. Otherwise, we use $[F^*]^q_{x,y} - [F]^q_{x,y} \leq [F^*]^q_{x,y}$. So,

$$\mathsf{Adv} \leq \varepsilon_2 \sum_{y \in A, y \in \mathcal{Y}} [F^*]^q_{x,y} + \sum_{y \in A, y \notin \mathcal{Y}} [F^*]^q_{x,y} \leq \varepsilon_2 \sum_{y \in \mathcal{Y}} [F^*]^q_{x,y} + \sum_{y \notin \mathcal{Y}} [F^*]^q_{x,y} \leq \varepsilon_2 + \Pr[y \notin \mathcal{Y}] \leq \varepsilon_1 + \varepsilon_2$$

□

It is interesting to look at the structure of the $[C^*]_{x,y}$ matrix. We note by $\mathsf{Part}(x)$ the partition of $\{1, \ldots, q\}$ such that i and j are in the same class if and only if $x_i = x_j$. When $\mathsf{Part}(x) \neq \mathsf{Part}(y)$, we have $[C^*]_{x,y} = 0$. When $\mathsf{Part}(x) = \mathsf{Part}(y)$ and there are exactly m classes, then $[C^*]_{x,y} = \frac{1}{2^\ell (2^\ell - 1) \cdots (2^\ell - m + 1)}$. Contrarily, if each class in $\mathsf{Part}(x)$ is a subset of a class of $\mathsf{Part}(y)$, we have $[F^*]_{x,y} = 2^{-m\ell}$. Otherwise, $[F^*]_{x,y} = 0$. Below, we bound the distance between $[C^*]_{x,y}$ and $[F^*]_{x,y}$.

Lemma 3.12. *Let F^* be a uniformly distributed function from a set $\mathcal{M}$ to itself. Let C^* be a uniformly distributed permutation on $\mathcal{M}$. The best advantage to distinguish C^* from F^* limited to q queries is bounded by $\frac{q(q-1)}{|\mathcal{M}|}$.*

Proof. We let $\mathcal{Y}$ be the set of all $y = (y_1, \ldots, y_q) \in \mathcal{M}^q$ with pairwise different entries. Clearly, $\Pr[y \in \mathcal{Y}] \geq 1 - \frac{q(q-1)}{2|\mathcal{M}|}$. For $x \in \mathcal{M}^q$ with pairwise different entries and $y \in \mathcal{Y}$, we have

$$\frac{[C^*]^q_{x,y}}{[F^*]^q_{x,y}} = \prod_{i=1}^q \left(1 - \frac{i-1}{|\mathcal{M}|}\right) = \Pr[y \in \mathcal{Y}] \geq 1 - \frac{q(q-1)}{2|\mathcal{M}|}$$

So, by applying the previous lemma, we obtain that the best advantage is bounded by $\frac{q(q-1)}{|\mathcal{M}|}$. □

The Luby-Rackoff Theorem is not so usable in this form since we don't have uniformly distributed functions F_i^*. If we have some independent functions F_1, F_2, F_3 such that $\frac{1}{2}\|[F_i]^n - [F_i^*]^n\|_a \leq \varepsilon$, we obtain

$$\begin{aligned}
\frac{1}{2}\|[\Psi(F_1,F_2,F_3)]^n - [C^*]^n\|_a \quad \leq \quad & \frac{1}{2}\|[\Psi(F_1,F_2,F_3)]^n - [\Psi(F_1^*,F_2,F_3)]^n\|_a + \\
& \frac{1}{2}\|[\Psi(F_1^*,F_2,F_3)]^n - [\Psi(F_1^*,F_2^*,F_3)]^n\|_a + \\
& \frac{1}{2}\|[\Psi(F_1^*,F_2^*,F_3)]^n - [\Psi(F_1^*,F_2^*,F_3^*)]^n\|_a + \\
& \frac{1}{2}\|[\Psi(F_1^*,F_2^*,F_3^*)]^n - [C^*]^n\|_a
\end{aligned}$$

Each of the first three terms in the sum can be considered as the advantage of a distinguisher between F_i and F_i^*, respectively, so they can be bounded by ε. We thus obtain

$$\frac{1}{2}\|[\Psi(F_1,F_2,F_3)]^n - [C^*]^n\|_a \leq 3\varepsilon + n^2.2^{-\frac{\ell}{2}}$$

Now, we can use the amplification result and obtain the following theorem.

Theorem 3.13 **([58]).** *Let $F_1,\ldots,F_{3r}$ be $3r$ independent round functions such that $\frac{1}{2}\|[F_i]^n-[F_i^*]^n\|_a\leq\varepsilon$. We consider the $3r$-round Feistel scheme $C=\Psi(F_1,\ldots,F_{3r})$ to be compared with the ideal cipher C^*. For all distinguisher limited to q queries, the advantage to distinguish C from C^* is bounded by $\frac{1}{2}\left(6\varepsilon+2q^2.2^{-\frac{\ell}{2}}\right)^r$.*

Proof. We note that C is the product of r independent 3-round Feistel ciphers for which we have just proven that the decorrelation of order q was bounded by $6\varepsilon+2q^2.2^{-\frac{\ell}{2}}$. We apply Th. 3.9 to deduce that the decorrelation of C is bounded by $\left(6\varepsilon+2q^2.2^{-\frac{\ell}{2}}\right)^r$. We conclude by the equivalence between best advantage and decorrelation (Th. 3.8). □

If we wanted to apply this to DES, we would have $\ell=64$. Even in some ideal case with $n\leq 2^{15}$ and $\varepsilon=0$, we obtain a distinguisher with advantage bounded by 2^{-7} for 18 rounds. This is not a good security result.

However, we could apply the theorem with $q=2$ and obtain interesting results. Namely, every distinguisher limited to two queries has an advantage bounded by $\frac{1}{2}\left(6\varepsilon+8.2^{-\frac{\ell}{2}}\right)^r$. Applying this to linear and differential distinguishers with a single iteration, we deduce that for every a and b, $E(\mathsf{DP}^C(a,b))$ and $E(\mathsf{LP}^C(a,b))$ are low. Namely, we have the following result.

Theorem 3.14 **([58]).** *For $a\neq 0$ and $b\neq 0$, we have*

$$\begin{aligned}E(\mathsf{DP}^C(a,b)) &\leq \frac{1}{2^\ell-1}+\frac{1}{2}|||[C]^2-[C^*]^2|||_\infty\\ E(\mathsf{LP}^C(a,b)) &\leq \frac{1}{2^\ell-1}+2|||[C]^2-[C^*]^2|||_\infty\end{aligned}$$

So, decorrelation theory can already be used to show that there is no good $E(\mathsf{DP}^C(a,b))$ and $E(\mathsf{LP}^C(a,b))$ for differential or linear cryptanalysis.

Proof. We write

$$E(\mathsf{DP}^C(a,b))=2^{-\ell}\sum_{x_1,x_2,y_1,y_2}1_{x_2\oplus x_1=a,y_2\oplus y_1=b}[C]^2_{x,y}$$

So, we (easily) deduce that $E(\mathsf{DP}^{C^*}(a,b))=\frac{1}{2^\ell-1}$.

We consider the non-adaptive distinguisher picking x_1 and x_2 of difference a then querying x_1 and x_2 to obtain y_1 and y_2 and producing 1 if and only if $y_2\oplus y_1=b$. Clearly, the advantage is $E(\mathsf{DP}^C(a,b))-\frac{1}{2^\ell-1}$. Due to the equivalence between advantage and decorrelation, we have

$$E(\mathsf{DP}^C(a,b))-\frac{1}{2^\ell-1}\leq\frac{1}{2}|||[C]^2-[C^*]^2|||_\infty$$

For the LP, we use the Fourier transform:

$$\begin{aligned}E(\mathsf{LP}^{C^*}(\alpha,\beta)) &= 2^{-\ell}\sum_{a,b}(-1)^{a\cdot\alpha\oplus b\cdot\beta}E(\mathsf{DP}^{C^*}(a,b))\\ &= 2^{-\ell}+2^{-\ell}\sum_{a,b\neq 0}(-1)^{a\cdot\alpha\oplus b\cdot\beta}\frac{1}{2^\ell-1}\\ &= \frac{1}{2^\ell-1}\end{aligned}$$

then

$$\begin{aligned}E\left(\mathsf{LP}^C(a,b)\right) &= E\left(\left((-1)^{a\cdot x\oplus b\cdot C(x)}\right)^2\right)\\ &= E\left((-1)^{a\cdot x_1\oplus b\cdot C(x_1)}(-1)^{a\cdot x_2\oplus b\cdot C(x_2)}\right)\\ &= E\left((-1)^{a\cdot(x_1\oplus x_2)\oplus b\cdot(C(x_1)\oplus C(x_2))}\right)\\ &= E\left(2\times 1_{a\cdot(x_1\oplus x_2)=b\cdot(C(x_1)\oplus C(x_2))}-1\right)\\ &= 2\left(2^{-2\ell}\sum_{x_1,x_1,y_1,y_2}1_{a\cdot(x_1\oplus x_2)=b\cdot(y_1\oplus y_2)}[C]^2_{x,y}\right)-1\end{aligned}$$

so

$$\begin{aligned}
E\left(\mathsf{LP}^{C}(a,b)\right)-\frac{1}{2^{\ell}-1} &= E\left(\mathsf{LP}^{C}(a,b)\right)-E\left(\mathsf{LP}^{C^{*}}(a,b)\right)\\
&= 2\times 2^{-2\ell}\sum_{x_1,x_1,y_1,y_2} 1_{a\cdot(x_1\oplus x_2)=b\cdot(y_1\oplus y_2)}\left([C]^2_{x,y}-[C^*]^2_{x,y}\right)\\
&\leq 2\max_{x_1,x_2}\sum_{y_1,y_2}\left|[C]^2_{x,y}-[C^*]^2_{x,y}\right|\\
&= 2|||[C]^2-[C^*]^2|||_{\infty}
\end{aligned}$$

□

Chapter 4

The Power of Interaction

An essential cryptographic protocol is the notion of *interactive proof*. Typically, a client would prove his credentials to a server. Here, the client plays the role of a prover and the server is a verifier. Ideally, his credential should not leak from the protocol, even to the verifier who could be malicious. This is the notion of *zero-knowledge protocol*. In this chapter, we formalize the notions of interaction, proof, zero-knowledge, and provide building blocks.

4.1 Interactive Proofs

We consider

- an **alphabet** Z, i.e., a set of **letters**;
- the set Z^* of finite strings made of elements in Z, i.e. the set of all **words**;
- the subsets of Z^* are called **languages**, i.e. sets of words.

Given a language L and a word x, we consider the problem of deciding whether or not x belongs to L. This is the **membership problem**.

Languages for which the membership problem can be decided by a deterministic algorithm within a time bounded by a polynomial in terms of $|x|$, the length of the string x, are called $\mathcal{P}$ languages.

Sometimes, we will consider x as a *statement* and L be the language of statements which are true. True statement may be proven by a *proof* w which will be called a **witness**. Given a predicate $R(x,w)$ checking whether w is a correct proof for x, the language L is defined by

$$L = \{x \in Z^*; \exists w \in Z^* \quad R(x,w)\}$$

(For convenience, proofs are encoded into a word so that we can also assume that the witness is a word.)

Languages such as the above, where R can be evaluated in a time bounded by some polynomial in terms of $|x|$, and where the witness must have a length also bounded by a polynomial, are called $\mathcal{NP}$ languages. The complement of an $\mathcal{NP}$ language is called a co-$\mathcal{NP}$ language. It is known that

$$\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$$

i.e., any $\mathcal{P}$ language is both an $\mathcal{NP}$ language and a co-$\mathcal{NP}$ language. This is illustrated on Fig. 4.1. A big open question in complexity is to wonder if $\mathcal{P} = \mathcal{NP}$ or not. There is an inclusion, but it is not known if all $\mathcal{NP}$ language can be recognized in polynomial time or if some of these languages do not have any polynomial-time algorithm to decide membership. Another open question is to wonder if $\mathcal{NP} = \text{co-}\mathcal{NP}$ or not. I.e., for languages for which membership can be checked with a witness in polynomial time, can we always check non-membership with a witness as well? Note that if $\mathcal{P} = \mathcal{NP}$ then $\mathcal{P} = \mathcal{NP} = \text{co-}\mathcal{NP}$.

We already used the notion of Turing reduction but there is another notion due to Karp. We say that a language L_1 reduces to a language L_2 if there exists a function f computable by a deterministic polynomial-time algorithm such that for all words x, $x \in L_1$ is equivalent to $f(x) \in L_2$. Compared to the Turing reduction, this means that the oracle for L_2-membership can be invoked only once.

There exist languages L which are $\mathcal{NP}$**-hard**. This means that for each $L' \in \mathcal{NP}$, L' reduces (in the sense of Karp) to L. There even exist $\mathcal{NP}$**-hard** languages in the class $\mathcal{NP}$ itself. These languages are called $\mathcal{NP}$**-complete**. For example, assuming a way to encode Boolean terms on Boolean variables in the form of a word, the language SAT of encoded terms that can evaluate to "true" by at least one assignment of the variables is $\mathcal{NP}$-complete [22]. Consequently, $\mathcal{P} = \mathcal{NP}$ is equivalent to $\mathsf{SAT} \in \mathcal{P}$.

Next, we define an interactive machine as follows.

Definition 4.1. *An* interactive machine *is an algorithm $\mathcal{A}$ taking as input some x, a list of incoming messages $m_1, \ldots, m_n$ of variable length, and a (long enough) sequence of random coins r and computing an outgoing message $\mathcal{A}(x, m_1, \ldots, m_n; r)$. The tuple $(x, m_1, \ldots, m_n; r)$ is called the* partial view *of $\mathcal{A}$.*

We assume a special symbol in the alphabet. Messages ending with this symbol are called terminal messages. *We assume that if m_n is a terminal message, then $\mathcal{A}(x, m_1, \ldots, m_n; r)$ is a terminal message as well.*

If $\mathcal{A}(x, m_1, \ldots, m_n; r)$ is a terminal message, $(x, m_1, \ldots, m_n; r)$ is called the final view *of $\mathcal{A}$.*

A pair of interactive machines $(\mathcal{A}, \mathcal{B})$ (with $\mathcal{A}$ called the initiator) is called an **interactive system**. An experiment $\mathsf{exp} = \left(\mathcal{A}(r_A) \xleftrightarrow{x} \mathcal{B}(r_B)\right)$ is characterized by an input x and the coins r_A and r_B for each participant. It consists of iteratively defining

$$\begin{aligned} a_i &= \mathcal{A}(x, b_1, \ldots, b_{i-1}; r_A) \\ b_j &= \mathcal{B}(x, a_1, \ldots, a_j; r_B) \end{aligned}$$

for $i = 1, \ldots, n_A$, where n_A is the smallest i such that a_i is a terminal message, and $j = 1, \ldots, n_B$, where n_B is the smallest j such that b_j is a terminal message. Namely, $\mathcal{A}$ initiates the interaction with the message $a_1 = \mathcal{A}(x; r_A)$ to $\mathcal{B}$. Then, $\mathcal{B}$ sends the message $b_1 = \mathcal{A}(x, a_1; r_B)$ to $\mathcal{A}$. Then $\mathcal{A}$ carries on with $a_2 = \mathcal{A}(x, b_1; r_A)$ and so on. We define the outputs of both participants $\mathsf{Out}_{\mathcal{A}}(\mathsf{exp}) = a_{n_A}$ and $\mathsf{Out}_{\mathcal{B}}(\mathsf{exp}) = b_{n_B}$, and the final views $\mathsf{View}_{\mathcal{A}}(\mathsf{exp}) = (x, b_1, \ldots, b_{n_A - 1}; r_A)$ and $\mathsf{View}_{\mathcal{B}}(\mathsf{exp}) = (x, a_1, \ldots, a_{n_B}; r_B)$.

We are now ready to define an interactive proof.

Definition 4.2. *Given a language L over an alphabet Z, an* interactive proof system *is an interactive system $(\mathcal{P}, \mathcal{V})$, where $\mathcal{P}$ is called a* prover *and $\mathcal{V}$ is called a* verifier, *such that there exists a polynomial P and some real numbers α and β such that $0 \leq \beta < \alpha \leq 1$ and*

- (termination*) for any x and every coins, the experiment $\mathcal{P} \overset{x}{\leftrightarrow} \mathcal{V}$ makes $\mathcal{V}$ terminates within a complexity bounded by $P(|x|)$;*

- (α-completeness*) for any $x \in L$, the experiment $\mathcal{P} \overset{x}{\leftrightarrow} \mathcal{V}$ makes $\mathcal{V}$ output "*accept*" with probability at least α (the probability is taken over the random coins);*

- (β-soundness*) for any $x \notin L$ and any interactive machine $\mathcal{P}^*$, the experiment $\mathcal{P}^* \overset{x}{\leftrightarrow} \mathcal{V}$ makes $\mathcal{V}$ output "*accept*" with probability at most β (the probability is taken over the random coins).*

This means that a prover $\mathcal{P}$ can convince a verifier $\mathcal{V}$ that $x \in L$, with probability at least α, and that no malicious prover $\mathcal{P}^*$ can convince the verifier when this is not true, with probability larger than β. We note that we assume no complexity bound on $\mathcal{P}$ or $\mathcal{P}^*$. We often consider $\alpha = 1$ in which case we say we have *perfect completeness*.

It is trivial to see that languages in $\mathcal{P}$ and $\mathcal{NP}$ have an interactive proof system: for a language in $\mathcal{P}$, we just consider a prover doing nothing and a verifier running the verifying predicate defining the language by himself. For a language in $\mathcal{NP}$, we just consider a prover finding the witness w then sending it to the verifier and the verifier checking that this is a correct witness. The protocol is as follows:

$$\begin{array}{lcl}
\textbf{Prover} & & \textbf{Verifier} \\
 & x & \\
\text{find } w & \xrightarrow{w \text{ (terminal)}} & \\
 & \xleftarrow{\text{accept (terminal)}} & \text{if } R(x,w)=1
\end{array}$$

It can be much more complicated to see if languages in co-$\mathcal{NP}$ have an interactive proof system. One non-trivial example is the Goldwasser-Micali-Rackoff proof GMR85 [31] for non-quadratic residuosity. Here, we consider words encoding a pair (n,v) of integers and the language

$$L = \{(n,v) \text{ integers}; v \in \mathbf{Z}_n^*, v \notin \mathsf{QR}(n)\}$$

We recall that

$$\mathsf{QR}(n) = \{y \in \mathbf{Z}_n^*; \exists x \quad y = x^2 \bmod n\}$$

To construct a proof system we consider the following verifier:

```
1: pick $r \in_U \mathbf{Z}_n^*$, $e \in_U \{0,1\}$, compute $y = v^e r^2 \bmod n$ and send $y$
2: receive $f$. If $\gcd(v,n) = 1$ and $e = f$, output the terminal message "accept", otherwise, output the
   terminal message "reject"
```

The prover is defined by

```
1: receive $y$, solve the equation $y = x^2 \bmod n$, if it is solvable, output the terminal message $f = 1$, other-
   wise, output the terminal message $f = 0$
```

The protocol runs as follows:

$$\begin{array}{lcl}
\textbf{Prover} & & \textbf{Verifier} \\
 & (n,v) & \\
 & & \text{pick } r,\ e = 0 \text{ or } 1 \\
\text{solve } y = x^2 \bmod n & \xleftarrow{y} & y = v^e r^2 \bmod n \\
f = \left\{ \begin{array}{ll} 0 & \text{if solvable} \\ 1 & \text{otherwise} \end{array} \right. & \xrightarrow{f \text{ (terminal)}} & \\
 & \xleftarrow{\text{accept (terminal)}} & \text{if } e = f \text{ and } \gcd(v,n) = 1
\end{array}$$

Termination and perfect completeness are trivial. To prove $\frac{1}{2}$-soundness, we consider an arbitrary prover $\mathcal{P}^*$ receiving y and sending f as a function of n,v,y. We assume that $(n,v) \notin L$. If $v \notin \mathbf{Z}_n^*$, it is clear that the verifier always rejects. If now $v \in \mathbf{Z}_n^*$, since $(n,v) \notin L$, we can write $v = w^2 \bmod n$ for some w. So, the distribution of $y = (w^e r)^2 \bmod n$ is uniform in $\mathsf{QR}(n)$, no matter the value of e. Hence, f is independent from e. Thus, $\Pr[e = f] = \frac{1}{2}$.

Soundness amplification. For simplicity, we consider perfect completeness. I.e., $\alpha = 1$. In the case of the GMR85 protocol, it may be unsatisfactory to have a proof in which a prover could cheat with probability $\frac{1}{2}$. To solve that, we can amplify the soundness by *sequential composition*. Namely, we could construct a new interactive proof in which we sequentially run the previous proof n times and accept only if all executions accepted. We could show that the new soundness probability would become β^n.

Amplification works very well for sequential composition but there are tricky things if we consider *parallel composition*, i.e., if we run the n executions in parallel. As for interactive proofs as we defined them, it works, but for slightly different notions of interactive proofs (e.g., variants in which the prover is computationally bounded), it does not. So, we must be careful when considering parallel composition of interactive systems.

As an example, we define the DD game of Bellare, Impagliazzo, and Naor [5]. A verifier commits to a random bit e, then a prover commits to a random bit e', then both open their commitment and the verifier accepts the "proof" if $e \neq e'$:

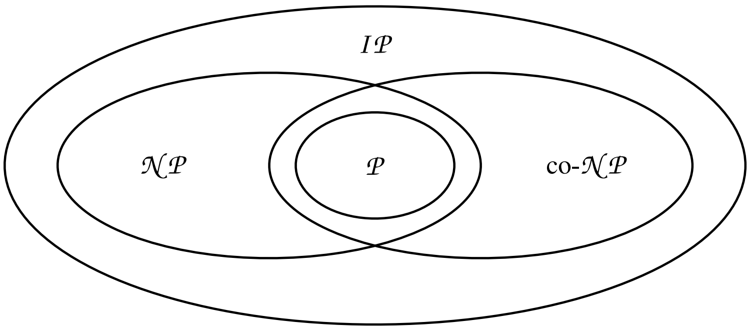


Figure 4.1: Complexity Classes of Languages

Prover		**Verifier**
		pick r, $e = 0$ or 1
pick r', $e' = 0$ or 1	$\xleftarrow{\quad y \quad}$	$y = \mathsf{commit}(e; r)$
$y' = \mathsf{commit}(e'; r')$	$\xrightarrow{\quad y' \quad}$	
	$\xleftarrow{\quad r,e \quad}$	
	$\xrightarrow{\quad r',e' \quad}$	check $y' = \mathsf{commit}(e'; r')$
	$\xleftarrow{\text{accept (terminal)}}$	if $e \neq e'$

If the prover is computationally bounded and the commitment is hiding and binding, there is no way to prove with probability significantly larger than $\frac{1}{2}$. So, we could think that for two parallel composition of this protocol, there is no way to prove with probability larger than $\frac{1}{4}$. However, this is not the case as the following strategy shows. The prover just repeats the two parallel commitments of the verifier in the opposite order and win with probability $\frac{1}{2}$:

Prover		**Verifier**
		pick r_1, r_2, $e_1, e_2 = 0$ or 1
	$\xleftarrow{\quad y_1,y_2 \quad}$	$y_i = \mathsf{commit}(e_i; r_i)$
set $y'_1 = y_2$, $y'_2 = y_1$	$\xrightarrow{\quad y'_1,y'_2 \quad}$	
	$\xleftarrow{\quad r_1,e_1,r_2,e_2 \quad}$	
	$\xrightarrow{\quad r_2,e_2,r_1,e_1 \quad}$	check

So, soundness amplification is not so trivial for parallel composition.

The class of languages with an interactive proof. We define $\mathcal{IP}$, the class of languages for which there exists an interactive proof. There is a famous theorem from 1992, due to Shamir [54], saying that $\mathcal{IP}$ corresponds to the class $\mathcal{PSPACE}$ of languages for which there is a deterministic algorithm deciding on membership or not which run with bounded space complexity, i.e. a polynomially bounded number of memory cells. Intuitively, this class includes the exhaustive search algorithm and others.

Theorem 4.3. *$\mathcal{IP} = \mathcal{PSPACE}$.*

So, the class $\mathcal{IP}$ is much larger than $\mathcal{NP}$ and co-$\mathcal{NP}$. This is depicted on Fig. 4.1.

When considering $\mathcal{NP}$ languages with an interactive proof, we said that the proof is trivial: the prover finds a witness (e.g., by exhaustive search), gives it to the verifier, and the verifier can check that it is a valid witness. For cryptographic application, this interactive proof is not satisfactory. Ideally, we would like the prover to prove the existence of the witness without revealing it, and without revealing anything that the verifier could not find by himself. This it the next notion to study: zero-knowledge.

4.2 Zero-Knowledge

We define a notion corresponding to interactive proofs where the verifier learns no information except the membership status of the input x.

Definition 4.4. *An interactive proof system* $(\mathcal{P}, \mathcal{V})$ *is* $*$-zero-knowledge *if for any p.p.t. interactive machine* $\mathcal{V}^*$ *there exists a p.p.t. algorithm* $\mathcal{S}$ *(called a* simulator*) such that for any* $x \in L$

$$\mathsf{View}_{\mathcal{V}^*}\left(\mathcal{P}(r_P) \overset{x}{\leftrightarrow} \mathcal{V}^*(r_V)\right)$$

and $\mathcal{S}(x;r)$ *produce* $*$*-identical distributions.*

There are three notions of zero-knowledge, depending on the notion of identical distributions (the $*$ in the definition):

- $*$=*perfect*: $*$-identical really means identical!
- $*$=*statistical*: $*$-identical means that the statistical distance is negligible in terms of $|x|$, i.e., any adversary has a negligible advantage.[1]
- $*$=*computational*: $*$-identical means any p.p.t. distinguisher has a negligible advantage.

As an example, we consider the following proof by Goldwasser-Micali-Rackoff (GMR89) [32] for the language of quadratic residues:

$$L = \{(n,v) \text{ integers}; v \in \mathsf{QR}(n)\}$$

1. the prover finds s such that $v = s^2 \bmod n$, picks $r \in \mathbf{Z}_n^*$, and sends $x = r^2 \bmod n$ to the verifier;
2. the verifier picks a random $e \in \{0,1\}$ and sends it to the prover;
3. the prover sends $y = s^e r \bmod n$;
4. the verifier accepts if $\mathsf{gcd}(n,v) = 1$ and $y^2 \equiv v^e x \pmod{n}$.

$$\begin{array}{rcl}
\textbf{Prover} & & \textbf{Verifier} \\
& (n,v) & \\
\text{find } s \text{ st } v = s^2 \bmod n & & \\
\text{pick } r, x = r^2 \bmod n & \xrightarrow{\qquad x \qquad} & \\
& \xleftarrow{\qquad e \qquad} & e = 0 \text{ or } 1 \\
y = s^e r \bmod n & \xrightarrow{\qquad y \qquad} & \text{check } \mathsf{gcd}(n,v) = 1, y^2 \equiv v^e x \pmod{n}
\end{array}$$

Termination and completeness are straightforward to check for this protocol. For soundness, we show that if a malicious prover P^* makes the verifier V accept with probability strictly greater than $\frac{1}{2}$, then it must be the case that $(n,v) \in L$. Clearly, we have that n and v are coprime. Now, the probability is an average over the random coins of P^*, so there must be some fixed coins making V accept with probability strictly greater than $\frac{1}{2}$. This actually means that there must be a P^* which is deterministic. By running the proof twice with P^*, with different e's, we thus obtain the same x, but some answer y_0 to $e = 0$ and some answer y_1 to $e = 1$ which satisfy $y_0^2 \equiv x \pmod{n}$ and $y_1^2 \equiv vx \pmod{n}$. So, $y_1/y_0 \bmod n$ is a square root of v, so $(n,v) \in L$.

To prove zero-knowledge, we construct a simulator S based on a malicious verifier V^* as follows: S first picks a guess $e_0 \in \{0,1\}$ for e and a random $y \in \mathbf{Z}_n^*$, then simulate the prover giving $x = y^2 v^{-e_0} \bmod n$ to V^*. If V^* gives $e \neq e_0$, this is bad luck and S restarts. Otherwise, $e = e_0$ and S can continue by giving y to V^* and obtain the final view of V^*. We have to prove that the bad luck happens with probability $\frac{1}{2}$ and that the obtained distribution is identical to the one obtained by running P and V^*.

[1] Statistical distance was defined on p. 25.

We note here that the simulator S is a *black-box* simulator. I.e., it is constructed by using V^* as a subroutine and does not depend on V^*. All the zero-knowledge protocols that we will see use a black-box simulator.

It was shown in 1986 by Goldreich, Micali, and Wigderson [33], that all $\mathcal{NP}$ languages have a computational zero-knowledge proof.

Theorem 4.5. *For every language L in $\mathcal{NP}$, there exists a computational zero-knowledge interactive proof system.*

They show it by the following GMW86 protocol for an $\mathcal{NP}$-complete language: the language of 3-colorable graphs. A graph (V,E) is specified by a vertex set V and an edge set $E \subseteq V^2$. A 3-coloring is a mapping $\varphi : V \to \{1,2,3\}$ such that for every edge $(u,v) \in E$, we have $\varphi(u) \neq \varphi(v)$. The GMW86 protocol runs as follows:

1. the prover finds a 3-coloring φ of (V,E);
2. P and V run $\#E$ times the following protocol;
   (a) the prover picks a random permutation π of $\{1,2,3\}$, some coins r_u for each $u \in V$, computes $R_u = \mathsf{commit}(\pi(\varphi(u)), r_u)$, and sends all R_u to the verifier;
   (b) the verifier picks a random $(u,v) \in E$ and sends it to the prover;
   (c) the prover sends r_u, r_v, $c_u = \pi(\varphi(u))$, and $c_v = \pi(\varphi(v))$;
   (d) the verifier checks that $R_u = \mathsf{commit}(c_u, r_u)$, $R_v = \mathsf{commit}(c_v, r_v)$, and $c_u \neq c_v$;
3. if all iteration succeeded, the verifier accepts.

$$\begin{array}{rcl}
\textbf{Prover} & & \textbf{Verifier} \\
 & (V,E) & \\
\text{find } \varphi & & \\
\hline
 & \text{repeat } \#E \text{ times} & \\
\text{pick } \pi \in S_3 & & \\
r_u \text{ for each } u \in V & & \\
c_u = \pi(\varphi(u)) & & \\
R_u = \mathsf{commit}(c_u, r_u) & \xrightarrow{R} & \\
 & \xleftarrow{u,v} & \text{pick } (u,v) \in E \\
\text{if } (u,v) \in E & \xrightarrow{c_u, c_v, r_u, r_v} & \text{check } R_u, R_v \\
 & & \text{check } c_u \neq c_v \\
\hline
\end{array}$$

The protocol is based on a commitment scheme which is computationally hiding and perfectly binding.

Finally, instead of proving membership, we would like that a prover proves his knowledge of a witness.

Definition 4.6. *Given a language $L \in \mathcal{NP}$ over an alphabet Z defined by a relation R, an* interactive proof of knowledge *for L is a pair $(\mathcal{P}, \mathcal{V})$ of interactive machines such that there exists a polynomial P, α, β such that* $0 \leq \beta < \alpha \leq 1$ *and*

- *termination: this is like for interactive proof systems*
- α-completeness*: this is like for interactive proof systems*
- β-soundness*: there exists an oracle algorithm $\mathcal{E}$ called* extractor *verifying what follows. For any $\mathcal{P}^*$ we let*
$$\varepsilon(x) = \Pr_{r_P, r_V}\left[\mathsf{Out}_{\mathcal{V}}\left(\mathcal{P}^*(r_P) \overset{x}{\leftrightarrow} \mathcal{V}(r_V)\right) = \mathsf{accept}\right]$$
If $\varepsilon(x) > \beta$ then $\mathcal{E}^{\mathcal{P}^*}(x)$ outputs w such that $R(x,w)$ holds with complexity at most $P(|x|)/(\varepsilon(x) - \beta)$.

Our typical prover starts with finding w then runs a polynomial-time algorithm. So, an equivalent notion could be to say that P has a private input with w and that P is a polynomially bounded algorithm. Indeed, if we want to prove knowledge of w, we must give w to P! We give examples of proof of knowledge in the next section.

4.3 Zero-Knowledge Construction from Σ Protocol

We consider simple protocols running in three phases: the prover sends some message a, the verifier sends some random challenge e selected from a set E, the prover sends back an answer z, and the verifier decides to accept or not. With additional properties, this defines Σ-protocols.

Definition 4.7. *Given a language $L \in \mathcal{NP}$ over an alphabet Z defined by a relation R, a* Σ-protocol *for L is a pair (P,V) of interactive machines such that*

- *V is polynomially bounded*
- 3-move*: P starts with a message a, V answers with a challenge $e \in_U E$, P terminates with a response z, V accepts (always for $x \in L$) or reject only depending on (x,a,e,z)*
- special soundness*: there exists a polynomially bounded algorithm $\mathcal{E}$ called* extractor *such that for any x, if $(x,a,z;r)$ and $(x,a,z';r')$ are two accepting views for V such that $e \neq e'$ where $e = V(x,a;r)$ and $e' = V(x,a;r')$ then $\mathcal{E}(x,a,e,z,e',z')$ yields w such that $R(x,w)$*
- special honest-verifier zero-knowledge (HVZK)*: there exists a polynomially bounded algorithm $\mathcal{S}$ called* simulator *such that for any $x \in L$ and e, the transcript (a,e,z) of the interaction $P(r_P) \overset{x}{\leftrightarrow} V(r_V)$ conditioned to e has same distribution as $\mathcal{S}(x,e;r)$.*

To fully define a Σ-protocol we thus need

- a relation R defining the language;
- a function for $a = P(x,w;r_P)$;
- a samplable domain E for e;
- a function for $z = P(x,w,e;r_P)$;
- a verification relation $V(x,a,e,z)$;
- a function $\mathcal{E}(x,a,e,z,e',z')$;
- a function $\mathcal{S}(x,e;r)$.

The properties to satisfy are:

1. R, P, V, $\mathcal{E}$, $\mathcal{S}$ and sampling are polynomially computable in $|x|$;
2. $\forall (x,w) \in R \; \forall r_P \; \forall e \in E \quad V(x,a,e,z)$,
   with a and z defined by $a = P(x,w;r_P)$ and $z = P(x,w,e;r_P)$;
3. $\forall x \; \forall e,e' \in E \; \forall a,z,z' \quad (e \neq e', V(x,a,e,z), V(x,a,e',z')) \Longrightarrow R(x, \mathcal{E}(x,a,e,z,e',z'))$;
4. $\forall (x,w) \in R \; \forall e \in E \quad \mathsf{distrib}_{r_P}(a,e,z) = \mathsf{distrib}_r(\mathcal{S}(x,e;r))$,
   with a and z defined by $a = P(x,w;r_P)$ and $z = P(x,w,e;r_P)$.

What is nice with Σ-protocols is that they are already proofs of knowledge, honest-verifier zero-knowledge, and composable in parallel. This is stated in the results below.

Theorem 4.8. *A Σ-protocol (P,V) for an $\mathcal{NP}$ language L defined by a relation R is an interactive proof of knowledge for L. The soundness probability is $\beta = \frac{1}{\#E}$, where E is the set of possible challenges in the Σ-protocol.*

Proof. Termination and 1-completeness are straightforward. It is less easy to show the soundness of the proof of knowledge. For that, we define the knowledge extractor $\mathcal{E}^{P^*}$ as follows. We denote by $\varepsilon(x)$ the probability that P^* makes V accept on the instance x and we assume that $\varepsilon(x) > \beta$. To define the extractor, we first pick some random r_P, r_V, r'_V and make the oracle P^* run twice with the same random coins r_P and interact with a simulation of V, first with $V(r_V)$, then with $V(r'_V)$. By construction, the a set by $P^*(r_P)$ is the same in both executions since r_P is the same and no message from V is used to compute a. We let e resp.e' be the challenge set by $V(r_V)$ resp. $V(r'_V)$, and z resp. z' be the response of $P^*(r_P)$. We let b resp. b' be the acceptance bit from the verification. Clearly, if $e \neq e'$ and $b = b' = 1$, we can execute the Σ-extractor $\mathcal{E}(x,a,e,z,e',z')$ and obtain a witness w for x which is given as output of the knowledge extractor. Clearly, all this is polynomially bounded. Below, we prove that $\Pr[e \neq e', b = b' = 1] \geq \varepsilon(x)(\varepsilon(x) - \beta)$. Since $\varepsilon(x) > \beta$ and β is a constant, we need $O\left(\frac{1}{\varepsilon(x)-\beta}\right)$ attempts of the above process to succeed to extract a witness. This shows the result.

Now, we analyze $\Pr[e \neq e', b = b' = 1]$. When $P^*(r_P)$ interacts with $V(r_V)$, we have $\Pr[b = 1] = \varepsilon(x)$. We denote $\varepsilon(x, r_P) = \Pr[b = 1 | r_P]$. Hence, $E(\varepsilon(x, r_P)) = \varepsilon(x)$ over a random r_P.

Since r_V and r'_V are independent, we have $\Pr[b = b' = 1 | r_P] = \varepsilon(x, r_P)^2$. So,

$$\Pr[b = b' = 1, e \neq e' | r_P] = \varepsilon(x, r_P)^2 - \Pr[b = b' = 1, e = e' | r_P]$$

We note that if $e = e'$, then P^* will give $z = z'$ so $b = b'$. Hence,

$$\Pr[b = b' = 1, e = e' | r_P] = \Pr[b = 1, e = e' | r_P]$$

Let A be the set of all e for which P^* produce a z leading to $b = 1$. We have

$$\Pr[b = 1, e = e' | r_P] = \sum_{e \in A} (\Pr[\mathsf{pick}\ e])^2 = \sum_{e \in A} \Pr[\mathsf{pick}\ e]\beta = \varepsilon(x, r_P)\beta$$

since the challenge is uniformly distributed so $\Pr[\mathsf{pick}\ e] = \beta$ for all e. So, we have

$$\Pr[b = b' = 1, e \neq e' | r_P] = \varepsilon(x, r_P)(\varepsilon(x, r_P) - \beta)$$

We consider the random variable $Z = \varepsilon(x, r_P)$ defined by a random r_P. We have $\Pr[b = b' = 1, e \neq e' | r_P] = f(Z)$ for $f(z) = z(z - \beta)$. Since $f''(z) > 0$, f is a convex function, we can apply the Jensen inequality to obtain $E(f(Z) \geq f(E(Z))$. This gives $\Pr[b = b' = 1, e \neq e'] \geq \varepsilon(x)(\varepsilon(x) - \beta)$. □

Definition 4.9. *An interactive proof system (P,V) is $*$-**honest verifier zero-knowledge** if there exists a ppt algorithm $\mathcal{S}$ such that*

$$\mathsf{View}_V\left(P(r_P) \overset{x}{\leftrightarrow} V(r_V)\right)$$

and $\mathcal{S}(x,r)$ produce $*$-identical distributions.

This is just the regular zero-knowledge property, but only guaranteed when the verifier is following the honest protocol.

Theorem 4.10. *A Σ-protocol (P,V) for an $\mathcal{NP}$ language L defined by a relation R is honest verifier zero-knowledge.*

Proof. Since the honest V does not depend on a to select e, we can just run V with some dummy a_0 and random coins r_V to get e with the good distribution, then sun the Σ simulator $\mathcal{S}(x,e;r)$ on some random r to obtain a transcript (a,e,z) with the correct distribution. We can then produce $(x,a,z;r_V)$, the simulated view of V. Clearly, it has the good distribution. □

Theorem 4.11. *Given an integer t and a Σ-protocol with set of challenges E, we consider the Σ^t-protocol consisting in executing t times in parallel the Σ-protocol and having the verifier accept if and only if all executions accept. This define a new Σ-protocol in which the set of challenges is E^t.*

So, the soundness probability is seriously amplified.

Goldreich-Micali-Wigderson for graph isomorphism. One example of Σ-protocol is the Goldreich-Micali-Wigderson protocol GMW86 for graph isomorphism from 1986 [33]. It is for the language of pairs of isomorphic graphs (G_0, G_1). Clearly, a witness can just be the isomorphism φ from G_0 to G_1. The obtained protocol could hold for any notion of isomorphism, not only for graphs. We just require that φ is a bijection, that $\varphi(G_0) = G_1$, and that it must be hard to find φ given G_0 and G_1 (which is believed to be the case for graphs).

In the GMW86 protocol, the set of challenges is $E = \{0,1\}$. The prover starts by selecting a random permutation π and sending $H = \pi(G_0)$. After receiving e, he answers by $\sigma = \pi$ if $e = 0$ and $\sigma = \pi \circ \varphi^{-1}$ if $e = 1$. So, $\sigma = \pi \circ \varphi^{-e}$. Then, the verifier accepts if and only if $H = \sigma(G_e)$.

$$\begin{array}{rcl}
\textbf{Prover} & & \textbf{Verifier} \\
\varphi \text{ st } \varphi(G_0) = G_1 & (G_0, G_1) & \\
\text{pick } \pi \text{ invertible} & & \text{pick } e \in \{0,1\} \\
H = \pi(G_0) & \xrightarrow{\quad\quad H \quad\quad} & \\
 & \xleftarrow{\quad\quad e \quad\quad} & \\
\sigma = \pi \circ \varphi^{-e} & \xrightarrow{\quad\quad \sigma \quad\quad} & \sigma(G_e) \overset{?}{=} H
\end{array}$$

The extractor works based on σ_0, the answer to $e = 0$ for some H, and on σ_1, the answer to $e = 1$ for the same H. Since $H = \sigma_0(G_0)$ and $H = \sigma_1(G_1)$, we have that $\sigma_1^{-1} \circ \sigma_0$ is a valid witness for (G_0, G_1) since $\sigma_1^{-1} \circ \sigma_0(G_0) = G_1$.

The simulator works based on G_0, G_1, and e. It picks σ uniformly and sets $H = \sigma(G_e)$.

Fiat-Shamir for modular square root. Another famous example is the FS86 protocol by Fiat and Shamir [27] for the language of pairs of integers (n, v) such that $v \in \mathbf{Z}_n^*$ and there exists s (the witness) such that $s^2 v \bmod n = 1$. Again the set of challenges is $E = \{0,1\}$. The prover starts by selecting a random $r \in \mathbf{Z}_n^*$ and sending $x = r^2 \bmod n$. After receiving e, he answers by $y = r$ if $e = 0$ and $y = rs \bmod n$ if $e = 1$, i.e., $y = rs^e \bmod n$. The verifier accepts if $y^2 v^e \bmod n = x$, and $v, y \in \mathbf{Z}_n^*$.

$$\begin{array}{rcl}
\textbf{Prover} & & \textbf{Verifier} \\
s \text{ st } s^2 v \bmod n = 1 & (n, v) & \\
\text{pick } r \in \mathbf{Z}_n^* & & \text{pick } e \in \{0,1\} \\
x = r^2 \bmod n & \xrightarrow{\quad\quad x \quad\quad} & \\
 & \xleftarrow{\quad\quad e \quad\quad} & \\
y = rs^e \bmod n & \xrightarrow{\quad\quad y \quad\quad} & y^2 v^e \bmod n \overset{?}{=} x,\ v, y \overset{?}{\in} \mathbf{Z}_n^*
\end{array}$$

The extractor is based on the answer y_e to $e = 0, 1$ with the same x. It computes $y_1 / y_0 \bmod n$ which is such that

$$\left(\frac{y_1}{y_0}\right)^2 v \bmod n = 1$$

so, a valid witness. The simulator picks $y \in \mathbf{Z}_n^*$ and computes $x = y^2 v^e \bmod n$ from e.

Schnorr for discrete logarithm. Finally, another famous protocol is the Schnorr protocol from 1989 [52, 53] for the language of (G, q, g, y) tuples with the following properties:

- G is a group in which it is easy to do operations (product and inverse) and comparisons;
- g is an element of G of prime order q;
- it is easy to check if a value belongs to $\langle g \rangle$;
- $y \in \langle g \rangle$.

The relation R is defined by $R((G, q, g, y), x)$ if and only if $y = g^x$. I.e., x is the discrete logarithm of y.

The Schnorr protocol has a parameter t which must be such that $q > 2^t$. The set of challenges is $E = \{1, \ldots, 2^t\}$. The prover starts by selecting a random $k \in \mathbf{Z}_q$ and sending $r = g^k$. After receiving e, he answers by $s = ex + k \bmod q$. The verifier accepts if $ry^e = g^s$ and $y \in \langle g \rangle$.

$$\begin{array}{rcl}
\textbf{Prover} & & \textbf{Verifier} \\
x \text{ st } g^x = y & (G,q,g,y) & \\
\text{pick } k \in \mathbf{Z}_q & & \text{pick } e \in \{1,\ldots,2^t\} \\
r = g^k & \xrightarrow{\quad r \quad} & q \text{ prime} > 2^t \\
 & \xleftarrow{\quad e \quad} & g \text{ of order } q?,\ y^q \overset{?}{=} 1 \\
s = ex + k \bmod q & \xrightarrow{\quad s \quad} & ry^e \overset{?}{=} g^s
\end{array}$$

The extractor is based on the answers s and s' to e and e', for $e \neq e'$, and with the same r. Since q is prime and $1 \leq e, e' \leq 2^t < q$, $e - e'$ is invertible modulo q and we can show that $g^{\frac{s-s'}{e-e'}} = y$. So, $\frac{s-s'}{e-e'} \bmod q$ is the extracted witness. The simulator picks $s \in \mathbf{Z}_q$ and computes $r = g^s y^{-e}$.

Strengthening Σ-protocols. A malicious verifier could select his challenge e based on the first message sent by the prover. If the set of challenges is very small, this is not a problem and we can actually show that honest-verifier zero-knowledge and zero-knowledge are equivalent. When the set of challenges is large, this is no longer equivalent. In the Schnorr protocol, a malicious verifier could select $e = f(y,r)$ and his view may become unforgeable by a simulator. As we will see later, this could indeed be used to construct a signature scheme with unforgeable signatures. However if we do want to obtain a zero-knowledge protocol, we must enrich the Σ-protocol with a commitment.

One solution could be that the verifier first commits to his challenge (without revealing it). Then, after receiving the first message from the prover, he would open his commitment and let the protocol continue as before. If the commitment is binding (i.e., a malicious verifier could not change his mind), this protocol becomes fully zero-knowledge. However, we now have troubles to prove soundness as we need to extract two answer with the same message from a malicious prover who would have received a commitment of the challenge. One solution to get around this is that we use a trapdoor commitment: a commitment in which there exists a trapdoor to break the binding property. The construction runs as follows:

1. P generates a commitment trapdoor τ and its associated key h and sends h to V;
2. V selects his challenge e and commit to it with key h; the commit value is sent to P;
3. P starts the Σ-protocol and sends the message a;
4. V opens his commitment to e;
5. P answers to the challenge by z and also discloses τ.

$$\begin{array}{rcl}
\textbf{Prover} & & \textbf{Verifier} \\
w \text{ st } R(x,w) & x & \\
\text{pick } r_P & & \text{pick } e \in E \\
\text{pick } \tau & & \\
h = g^\tau \bmod p & \xrightarrow{\quad h \quad} & \\
 & \xleftarrow{\mathsf{Commit}_h(e;r)} & \text{pick } r \\
a = \mathcal{P}(x,w;r_P) & \xrightarrow{\quad a \quad} & \\
\text{verify } \mathsf{Commit}(e;r) & \xleftarrow{\quad e,r \quad} & \\
z = \mathcal{P}(x,w,e;r_P) & \xrightarrow{\quad z,\tau \quad} & V(x,a,e,z)?,\ h \overset{?}{=} g^\tau \bmod p
\end{array}$$

This protocol becomes computationally zero-knowledge and remains a proof-of-knowledge. One example for a trapdoor commitment is the following one.

Pedersen commitment 1991 [44]. We set up the commitment with some parameters (p,q,g), where p and q are prime, q divides $p-1$, and g is an element of $\mathbf{Z}_p^*$ of order q. The trapdoor is an element $\tau \in \mathbf{Z}_q$. The key is $h = g^\tau \bmod p$. To commit on X with coins $r \in \mathbf{Z}_q$, we compute $c = g^X h^r \bmod p$. This is unconditionally hiding, and computationally binding (breaking the binding property is equivalent to computing τ, i.e., solving the discrete logarithm problem for h). With τ, we can equivocate a commitment to X_0 with coins r_0 to any X. We just set $r = r_0 + \frac{X_0 - X}{\tau} \bmod q$ and we have

$$c = g^{X_0} h^{r_0} \bmod p = g^X h^r \bmod p$$

4.4 Setup Models

In the previous strengthened model, the use of an ephemeral trapdoor in the commitment looks artificial, since we never need the equivocation property of the commitment in practice. Furthermore, it is dangerous for security. We could adopt a more practical approach by having the commitment key to be set up once and for all participants, with the trapdoor held by nobody. This is in line with what we call the *common reference string (CRS)* model. In that case, there is a CRS (e.g., the commitment public key) which is set up for all participants.

To show soundness/zero-knowledge, we may need to assume that the extractor/simulator can use the trapdoor. This is fine, except that one property of zero-knowledge may be a bit trickier: *deniability*. This assumes that having run the protocol can be denied as the verifier can extract no evidence of having run it in the protocol. Normally, zero-knowledge protocols are inherently deniable since whatever the verifier extracts can be simulated. However, when the simulator needs the trapdoor, since no participant has the trapdoor in practice, what the practical verifier extracts may become non-simulatable. Clearly, this does not expose the secret of the prover but could still leak evidence of having run the protocol.

Besides the CRS, another setup model is the *Random Oracle Model (ROM)*. In this model, we have an oracle H who answers at random to any query, but consistently. I.e., making the same query several times will produce the same response. This oracle can be accessed by all participants. In the notion of zero-knowledge proof of knowledge, the extractor/simulator may further *simulate* the behavior of H. I.e., they answer at the place of H to all queries, but they must do it in a way which is indistinguishable from querying a real random oracle. Again, we may loose deniability. But otherwise, we can have more efficient protocols. In the strengthening, we commit to e by disclosing $H(e\|r)$ and open by revealing e and r.

There are other setup models. For instance, we can assume that all participants are initialized with a public/private key pair. We can assume the existence of a public directory, to which we could register public keys. We can assume the existence of secure hardware tokens. Etc.

4.5 A Building Block for Making Cryptographic Primitives

In 1986, Fiat and Shamir [27] proposed to transform (what is now called) a Σ-protocol into a proof which is non-interactive. This is the notion of a *Non-Interactive Zero-Knowledge proof (NIZK)*.

The idea is that the verifier is now simulated by a hash function. That is, the challenge e used in the Σ-protocol is computed by $e = H(x\|a)$. Namely, to prove x, the prover computes a as usual, then $e = H(x\|a)$, then the answer z. The (a,z) pair is the proof. It is verified as usual, by re-computing e.

Note that here, the verifier is choosing e adaptively based on a, which is normally not allowed. Consequently, we may loose the simulatability. Even worse: we do need to loose this property. Otherwise, a malicious prover could forge a proof by running this simulation!

The Fiat-Shamir construction is also used to create a signature scheme. Essentially, we take $e = H(\mathsf{message}\|x\|a)$ and do the same. I.e., the signature is the (a,z) pair. We will prove (Th. 5.8 in the next chapter), in the random oracle model, that this construction is secure against existential forgeries under chosen message attacks (EF-CMA), when the relation of the Σ-protocol is such that finding a witness w for x is a hard problem.

Σ-protocols can also be used to construct other cryptographic primitives. As an example, we construct a trapdoor commitment. Assuming that finding a witness for R is hard and that we have a Σ protocol for R, we take as a common reference string and instance x and as a trapdoor a witness w for this instance. So, $R(x,w)$ holds. We can commit on elements of the set of challenges E. To commit on $e \in E$, we pick some random coins r and compute $(a,e,z) = \mathcal{S}(x,e;r)$. The commit value is a and the opening value is (e,z). For opening, we just check that $V(x,a,e,z)$ holds. We can check that the commitment is perfectly hiding as the distribution of a is like in the correct interactive proof, so independent from e. We can also check that the commitment is computationally binding. Indeed, being able to open a commitment a on two values of e would lead (thanks to the Σ extractor) to a witness for x, which is assumed to be hard to find. Finally, using w we can equivocate the commitment by just running the correct interactive protocol: P produces a, the commit value. Then, if we want to open to e, we just compute the correct z by using w.

Chapter 5

Proving Security

Foundations of cryptography, as presented in the previous chapter about interaction, show that proving techniques heavily rely on the notion of modeling, simulation, interactive Turing machine rewinding, complexity reduction, etc. In a former chapter, we intuitively introduced some notions of security for encryption and signature. In the present chapter, we present more formally the modern notions of security and techniques to prove security.

5.1 The Security of Encryption

Previously, we considered the security of encryption in terms of *key recovery* and *decryption* problems. These security notions may be insufficient. For instance, a cryptosystem doing nothing (i.e., with a ciphertext y equal to the plaintext x no matter x or the secret key) makes key recovery hard but is clearly insecure. A cryptosystem only encrypting one part of the message may make the full decryption hard but would leak sensitive information and be considered as insecure. Recovering one particular bit of the plaintext may be sensitive, and still be feasible without decrypting completely.

In RSA, we can prove that recovering the least significant bit of the plaintext is equivalent to decrypting completely. So, if the decryption problem is hard, the least significant bit is called a *hard-core bit*.

More precisely, we define $\mathsf{lsb}(x)$ to be the least significant bit of x and $\mathsf{lsbdec}(y)$ to be the lsb of the decryption of y. We show below that the RSA decryption problem reduces to computing lsbdec. For that, we assume that we have a subroutine to compute lsbdec and we show that we can decrypt y given the public key (e,N).

Let us now assume that we know that the decryption x of y is in some interval $a \leq x < b$ with $a = \frac{k}{2^i}N$ and $b = \frac{k+1}{2^i}N$ for some integers k and i. Note that we can start with $k = 0$ and $i = 0$. We can see now how to update k and increment i. Indeed, we could write $\frac{2k+\beta}{2^{i+1}}N \leq x < \frac{2k+\beta+1}{2^{i+1}}N$ with $\beta = 0$ or $\beta = 1$. So, we can update k to $2k+\beta$, meaning updating either a or b to $\frac{a+b}{2}$, if we could compute the bit β. Since the inequality implies $\beta\frac{N}{2} \leq 2^i x - kN < (\beta+1)\frac{N}{2}$. So, β is such that $\beta\frac{N}{2} \leq 2^i x \bmod N < (\beta+1)\frac{N}{2}$. We deduce $\beta = \mathsf{lsb}(2^{i+1}x \bmod N)$. Finally, $\beta = \mathsf{lsbdec}(2^{(i+1)e}y \bmod N)$. We deduce the following algorithm:

```
1: $a \leftarrow 0, b \leftarrow N$
2: for $i = 0$ to $\lfloor \log_2 N \rfloor$ do
3:     if $\mathsf{lsbdec}(2^{(i+1)e}y \bmod N) = 1$ then
4:         $a \leftarrow (a+b)/2$
5:     else
6:         $b \leftarrow (a+b)/2$
7:     end if
8: end for
9: yield $\lfloor a \rfloor$
```

Chor and Goldreich have shown that computing lsbdec with errors also enables the full decryption [15]. It

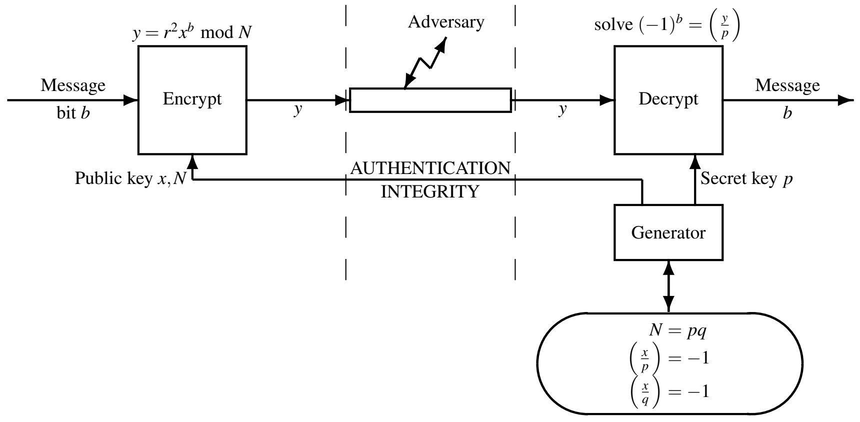


Figure 5.1: Goldwasser-Micali Cryptosystem

was even shown that each bit of the plaintext has the same property [1]. This shows that every bit of the plaintext is a hard core bit in RSA. However, this only applies to each bit of the binary expansion of x, but not every bit of information about x is a hard-core bit. Indeed, we can define a Boolean function on x which is easy to compute from $x^e \bmod N$: we can just consider the Jacobi symbol. Indeed, if we define

$$\mathsf{jac}(x) = \left(\frac{x}{N}\right) \quad , \quad \mathsf{jacdec}(y) = \left(\frac{y^d \bmod N}{N}\right)$$

then we have $\mathsf{jacdec}(y) = \mathsf{jac}(y)^d = \mathsf{jac}(y)$ since d must be odd to be invertible modulo $\varphi(N)$. So, it is easy to compute $\mathsf{jacdec}(y)$. So $\mathsf{jac}(x)$ is not a hard-core bit.

The *semantic security* aims at saying that every bit of information is hard to compute. This is formalized by a game. A key pair is generated and the public key is given to the adversary. Then, the adversary selects two messages x_0 and x_1. A bit b is flipped and x_b is encrypted into y. Then, y is given to the adversary who must give a bit b'. The adversary wins if $b = b'$. Since there is a trivial strategy to win with probability $\frac{1}{2}$ (just select b' at random), we say that the encryption is secure if $\Pr[b = b'] - \frac{1}{2}$ is negligible for every adversary.

This security notion is only feasible when the encryption is non-deterministic (otherwise, the adversary can compare y with the encryption of x_0 and x_1 and deduce b). It was proposed with the Goldwasser-Micali cryptosystem [29, 30], which only encrypts a bit.

In the Goldwasser-Micali cryptosystem [29, 30], the public key consists of a pair (x, N) where $N = pq$, the product of two large primes, and $x \in \mathbf{Z}_N^*$ which is neither a quadratic residue modulo p nor modulo q (see Fig. 5.1). To encrypt b, we select $r \in \mathbf{Z}_N^*$ and give $y = r^2x^b \bmod N$. To decrypt, we just find b such that $(-1)^b = (y/p)$. This is semantically secure. (Actually, since we encrypt a bit, semantic security is equivalent to the hardness of the decryption problem.)

The semantic security definition is a bit complicated but it was shown to be equivalent to the following one.

Definition 5.1. *A cryptosystem* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CPA-secure (indistinguishable under chosen plaintext attacks) if for every polynomially bounded adversary* $\mathcal{A}$*, if we let* $(K_p, K_s) = \mathsf{Gen}(1^\lambda; r_g)$*,* $(x_0, x_1) = \mathcal{A}(K_p; \rho)$*,* $b \in \{0, 1\}$*,* $y = \mathsf{Enc}(x_b; r)$*,* $b' = \mathcal{A}(K_p, y; \rho)$*, where* r_g, ρ, b, r *are independent uniformly distributed sequences of bits, then* $\Pr[b = b'] - \frac{1}{2}$ *is a negligible function of* λ*. It is required that* x_0 *and* x_1 *have the same length.*

This experiment is usually called a *game* between a challenger who runs Gen and Enc and selects b, and an adversary $\mathcal{A}$ who creates two plaintexts x_0 and x_1 for which he would be able to distinguish their encryptions. We say the adversary wins if $b = b'$. Intuitively, there is a Boolean function that he is supposed to be able to compute on the plaintext and he just selects two plaintexts having different outputs through this Boolean function. The IND-CPA game is illustrated on Fig. 5.2.

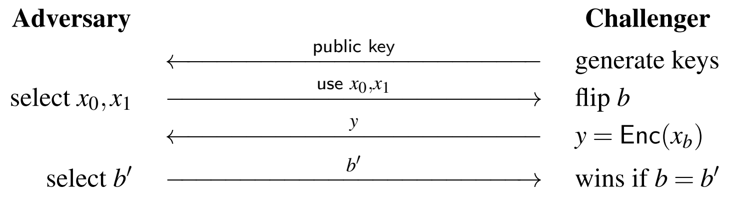


Figure 5.2: IND-CPA Game

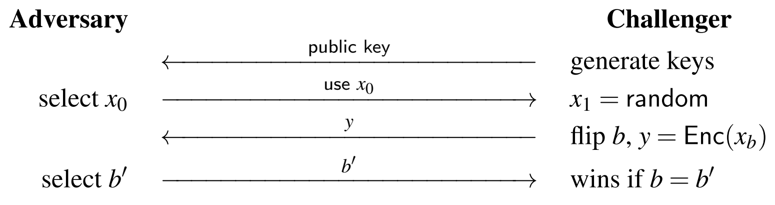


Figure 5.3: IND\$-CPA Game

For cryptosystems encrypting plaintexts of variable length, it is required that the length of x_0 and x_1 is the same, since it is impossible to perfectly hide the length of a plaintext on infinite message spaces.

In the case of the Goldwasser-Micali cryptosystem, the message space has only two elements: the message 0 and the message 1. So, the only relevant case reduces to $x_0 = 0$ and $x_1 = 1$. Therefore, IND-CPA security is equivalent in having $\Pr[b = \mathcal{A}(K_p, \mathsf{Enc}(b;r);\rho)] - \frac{1}{2}$ negligible. For the Goldwasser-Micali cryptosystem, this means $\Pr[b = \mathcal{A}(x,N,r^2x^b \bmod N;\rho)] = \frac{1}{2} + \mathsf{negl}(\lambda)$.

There is an equivalent definition proposed with a slightly different game [50] in which the adversary only proposes one plaintext x_0, and either this one is selected, or a random x_1 one (see Fig 5.3).

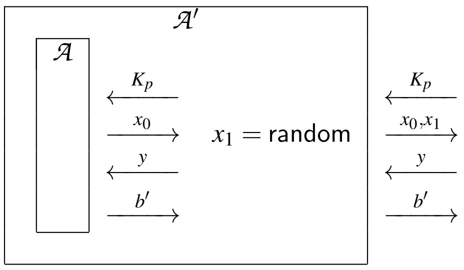
Definition 5.2. *A cryptosystem* $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ *is IND\$-CPA-secure if for every polynomially bounded adversary* $\mathcal{A}$*, if we let* $(K_p,K_s) = \mathsf{Gen}(1^\lambda;r_g)$*,* $x_0 = \mathcal{A}(K_p;\rho)$*,* x_1 *random with same length as* x_0*,* $b \in \{0,1\}$*,* $y = \mathsf{Enc}(x_b;r)$*,* $b' = \mathcal{A}(K_p,y;\rho)$*, where* r_g,ρ,x_1,b,r *are independent uniformly distributed sequences of bits, then* $\Pr[b = b'] - \frac{1}{2}$ *is a negligible function of* λ*.*

This game is often called the *real-or-random* encryption game while the previous IND-CPA game is called the *left-or-right* encryption game.

Theorem 5.3. *IND-CPA security and IND\$-CPA security are equivalent.*

Proof. To show this, we first show that IND-CPA security implies IND\$-CPA security. We consider an adversary $\mathcal{A}$ in the real-or-random game. Let us transform it into an adversary $\mathcal{A}'$ in the left-or-right game (see Fig. 5.4). To define $\mathcal{A}'(K_p;\rho')$, we first run $x_0 = \mathcal{A}(K_p;\rho)$ and select x_1 of same length of x_0. To define ρ from ρ', we just run $\mathcal{A}(K_p;\rho')$ by watching at which coins in ρ' are used by $\mathcal{A}$. The next unused coins are taken to select x_1. Finally, ρ is just ρ' without the coins used for x_1 (that is, the left over coins are let in ρ for the next part). Then, we set $(x_0,x_1) = \mathcal{A}'(K_p;\rho')$ and we define $\mathcal{A}'(K_p,y;\rho') = \mathcal{A}(K_p,y;\rho)$. Clearly, $\mathcal{A}'$ simulates well the selection of x_1. So, $\mathcal{A}$ and $\mathcal{A}'$ win with exactly the same probabilities in their respective game. Due to IND-CPA security, $\mathcal{A}'$ wins with probability $\frac{1}{2} + \mathsf{negl}$. So, $\mathcal{A}$ wins with probability $\frac{1}{2} + \mathsf{negl}$. Since this applies to any $\mathcal{A}$, we obtain IND\$-CPA security.

Then, we show that IND\$-CPA security implies IND-CPA security. For that, we consider an adversary $\mathcal{A}$ in the left-or-right game. We define an adversary $\mathcal{A}'$ in the real-or-random game as follows. We let $\mathcal{A}'(K_p;\rho') = x_{b''}$ where $\mathcal{A}(K_p;\rho) = (x_0,x_1)$ and b'' is one coin from ρ' which is just removed to define ρ. I.e., $\rho' = b''\|\rho$. Then, $\mathcal{A}'(K_p,y;\rho') = \mathcal{A}(K_p,y;\rho) \oplus b''$. We let b be the bit selected by the challenger to define $y = \mathsf{Enc}(x_{b''})$ if $b = 0$ or $y = \mathsf{Enc}(\mathsf{random})$ otherwise. (See Fig. 5.5.) Let p be the probability for $\mathcal{A}$ to win in the IND-CPA game. In our construction, when $b = 0$, we have $\Pr[b = \mathcal{A}'(K_p,y;\rho')|b = 0] = p$ since this case perfectly simulates the IND-CPA game. When $b = 1$, y gives no information about b'', so $\Pr[b = \mathcal{A}'(K_p,y;\rho')|b = 1] = \frac{1}{2}$. So, $\Pr[b = \mathcal{A}'(K_p,y;\rho')] - \frac{1}{2} = \frac{1}{2}(p - \frac{1}{2})$. Due to IND\$-CPA security, $\mathcal{A}'$



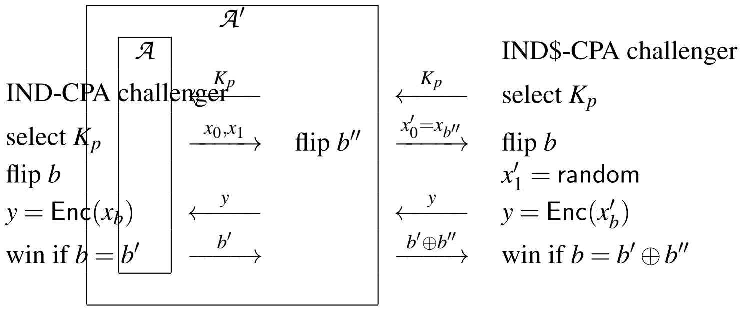


Figure 5.4: IND-CPA security implies IND$-CPA security

Figure 5.5: IND$-CPA security implies IND-CPA security

wins with probability $\frac{1}{2} + \mathsf{negl}$. So, $\mathcal{A}$ wins with probability $p = \frac{1}{2} + \mathsf{negl}$. Since this applies to any $\mathcal{A}$, we obtain IND-CPA security. □

The ElGamal cryptosystem, in a group $\langle g \rangle$, is also semantically secure if we assume that the Decisional Diffie-Hellman problem is hard in $\langle g \rangle$ and if we only encrypt messages which are elements of $\langle g \rangle$.

Theorem 5.4. *If the DDH problem is hard in the group generated by the ElGamal cryptosystem, and if the plaintext space is included in the group, then the cryptosystem is IND-CPA secure.*

We remind that the DDH problem is not always hard. For instance, the DDH problem in $\mathbf{Z}_p^*$ is easy.

We also observe that that the assumption that we only encrypt messages which are elements of $\langle g \rangle$ may be a problem because we may have to map bitstrings (arbitrary messages) into group elements in a reversible way. One possible instance is that we take a strong prime p. I.e., a large prime number p such that $q = \frac{p-1}{2}$ is also prime. Then, we consider the subgroup of $\mathbf{Z}_p^*$ of order q. Clearly, -1 is not in this subgroup since $(-1)^q \neq 1$ (because q must be odd). So, for every m, either m or $-m$ is in the subgroup. We can define $\mathsf{map}(m) = \pm m$ in the subgroup for $1 \leq m \leq q$. This mapping is invertible. So, we can encrypt integers between 1 and q by encrypting the subgroup element $\mathsf{map}(m)$, assuming that the DDH problem is hard in this subgroup.

Proof. We show IND$-CPA security. Let $\mathcal{A}$ be an adversary for the real-or-random game. We construct a distinguisher $\mathcal{A}'$ for the DDH problem as follows. In the DDH problem, $\mathcal{A}'$ receives an order q and a group generator g, some $y = g^x$ for $x \in_U \mathbf{Z}_q$, and a pair (u, v') in which $u = g^r$ for $r \in_U \mathbf{Z}_q$ and either $v' = y^r$ or v' is random in the group generated by g. Clearly, (q, g, y) simulates the generation of an ElGamal public key. Let $x_0 = \mathcal{A}(q, g, y)$. Given (u, v'), we define $v = x_0 v'$. Clearly, (u, v) simulates the ElGamal ciphertext obtained by submitting x_0 in the real-or-random game: either it is $(g^r, x_0 y^r)$ or it is $(g^r, \mathsf{random} \times y^r)$ for random in the subgroup generated by g. Let b be the guess from $\mathcal{A}$. Clearly, b is a guess for the DDH problem which is correct if and only if $\mathcal{A}$ wins. So, the distinguisher has the same advantage of $\mathcal{A}$. Since the DDH problem is hard, the wining probability of $\mathcal{A}$ is $\frac{1}{2} + \mathsf{negl}$. □

There exist stronger security notions. For instance, we may consider the *non-malleability* security [25]. Intuitively, it means that an adversary cannot replace a ciphertext y (with unknown $\mathsf{Dec}(y)$ to him) into another ciphertext $y \neq y'$ such that $\mathsf{Dec}(y)$ and $\mathsf{Dec}(y')$ are "related". This actually looks like an integrity protection for the plaintext.

One example where this notion of security is not satisfied is the traditional family of stream ciphers, where $y = x \oplus k$. Indeed, replacing y by $y' = y \oplus \delta$ leads to $\mathsf{Dec}(y)$ and $\mathsf{Dec}(y')$ to be within a difference of δ. We can call this a relation and then have the malleability property.

Let us take the Goldwasser-Micali cryptosystem as another example. Given $y = r^2 x^b \bmod N$, we can compute $s^2 x^c y \bmod N$ for a random s and a random bit c. This would be a valid encryption of $b \oplus c$ with a correct distribution. If we can decrypt this new ciphertext, then XOR the result to c, we obtain b.

The RSA and ElGamal cases are also malleable, as it will be later discussed.

There is a theorem saying that non-malleability is equivalent [4] to the *IND-CCA security* [48], where IND-CCA security is defined as follows (see Fig. 5.6).

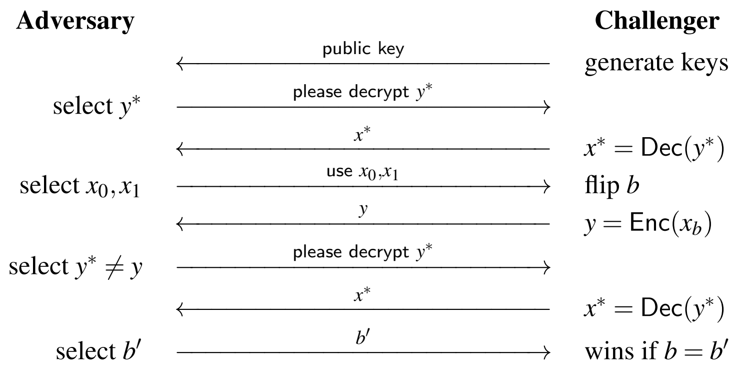


Figure 5.6: IND-CCA Game

Definition 5.5. *A cryptosystem* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is IND-CCA-secure (indistinguishable under chosen ciphertext attacks) if for every polynomially bounded adversary* $\mathcal{A}$ *playing the following game, the probability to win is* $\frac{1}{2} + \mathsf{negl}(\lambda)$.

1. *one key pair is generated and the public one* K_p *is given to* $\mathcal{A}$;
2. $\mathcal{A}$ *can make some decryption queries: he submits* y^* *and gets* $x^* = \mathsf{Dec}(y^*)$ *in return;*
3. $\mathcal{A}$ *selects two plaintexts* x_0 *and* x_1;
4. *one bit* b *is flipped and* $y = \mathsf{Enc}(x_b)$ *is computed, and given to* $\mathcal{A}$;
5. $\mathcal{A}$ *can continue to make some decryption queries: he submits* y^* *and gets* $x^* = \mathsf{Dec}(y^*)$ *in return; he is not allowed to query* $y^* = y$;
6. $\mathcal{A}$ *produces a bit* b' *and wins if* $b = b'$.

IND-CCA security historically followed another notion called *IND-CCA1 security* or *lunchtime attack* [41], where the adversary was not allowed to make decryption queries after having received the challenge y.

In general, "textbook cryptosystems" are not IND-CCA-secure because they are malleable, with some kind of homomorphic property. For instance, the ElGamal cryptosystem has the property that if x is the decryption of (u, v), then xw is the decryption of (u, vw). So, the adversary can take the challenge $y = (u, v)$, compute $y^* = (u, vw)$, make a decryption query with y^*, divide the result by w and compare with x_0 and x_1 to deduce b.

The RSA cryptosystem (which is deterministic and homomorphic, so with no chance to be IND-CCA secure or even IND-CPA secure) can be transformed into another one called *RSA-OAEP* [6] which is proven to be IND-CCA secure based on some *random oracle*.

5.2 The Random Oracle Model

In the *random oracle model (ROM)*, all participants in the game can query an oracle H, but do not see the queries of others. The oracle is responding randomly (so the name), but consistently. That is, the answer to a fresh query will be random, but forthcoming identical queries will produce the same response. So, a random oracle models a deterministic function which is selected at random before the game starts. Formally, the response is a "long enough" bitstring. In most of applications, we assume it is a string of pre-determined length. The trick in the random oracle model is that reductions can simulate the random oracle (so that it looks like a real random oracle) but with some hidden but useful information.

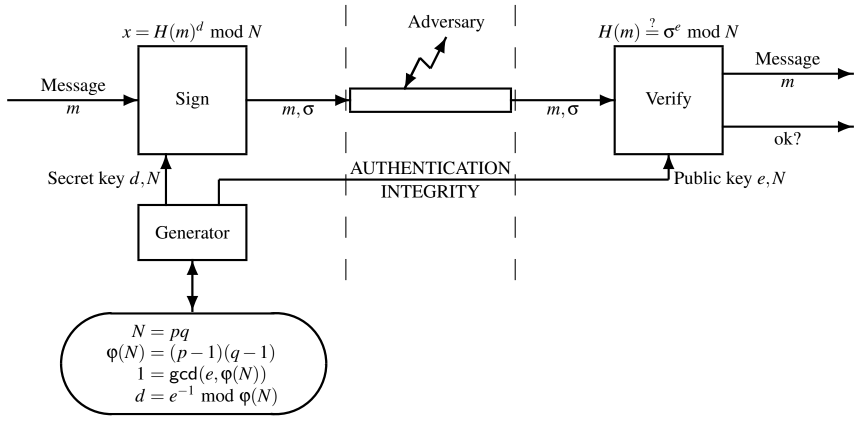


Figure 5.7: Full Domain Hash (FDH) Signature

Signatures with Full-Domain Hash (FDH). The FDH signature scheme [7] is based on RSA (see Fig. 5.7): we consider that the random oracle H returns random $\mathbf{Z}_N^*$ elements, given the RSA modulus N. The signature of a message m is the RSA signature of $H(m)$. The verification algorithm then follows.

Theorem 5.6. *If the RSA decryption problem is hard, then FDH is EF-CMA-secure. I.e., it resists to existential forgeries under chosen message attacks.*

Proof. We give here the proof by Coron [23]. We consider an adversary $\mathcal{A}$ playing the EF-CMA game. I.e., he is given oracle access to H and the challenger also makes hash queries. He receives some public key (e,N). He can make signing oracle queries: he chooses one message m and gets its signature σ by an oracle call. Then, he produces one pair (m,σ) and wins if σ is a valid signature for m and m was not queried to the signing oracle.

By changing $\mathcal{A}$ a bit, we reduce without loss of generality to cases where either the attack aborts or the final output is always valid, m was not queried to the signing oracle, and m was queried to the hash oracle.

Then, we construct an algorithm $\mathcal{B}$ to solve the RSA decryption problem: $\mathcal{B}$ receives a public key (e,N) and a ciphertext y and must decrypt it. For that, $\mathcal{B}$ simulates $\mathcal{A}$ receiving (e,N) as a public key, then playing with a simulation for the hashing oracle and the signing oracle.

$\mathcal{B}$ simulates H as follows: he answers consistently to repeating queries. For a fresh query m, $\mathcal{B}$ picks $r \in_U \mathbf{Z}_N^*$ and flips a biased coin b such that $\Pr[b = 1] = p$ for some magic parameter p to be later explained. Then, $\mathcal{B}$ answers as if $H(m) = y^b r^e \bmod N$. It is clear that $H(m)$ is perfectly distributed, even if b is fixed. So, this is a valid simulation of H. More importantly, $\mathcal{B}$ keeps a record of y and b such that $H(m) = y^b r^e \bmod N$ as they will play a role.

$\mathcal{B}$ simulates the signing oracle as follows: to sign a message m, he queries m to H and takes y and e such that $H(m) = y^b r^e \bmod N$. Then, if $b = 1$, $\mathcal{B}$ aborts. Otherwise, we have $H(m) = r^e \bmod N$, and the signature of m is clearly r. We can thus simulate without the signing key, unless we abort. Clearly, this simulation is also perfect, except when aborting.

Since the simulations are perfect, $\mathcal{A}$ behaves with the same probability as in the EF-CMA game and either aborts or produces a forgery (m,σ).

Finally, when the simulation of $\mathcal{A}$ terminates on (m,σ), $\mathcal{B}$ takes y and e such that $H(m) = y^b r^e \bmod N$. Then, if $b = 0$, $\mathcal{B}$ aborts. Otherwise, we have $H(m) = yr^e \bmod N$. Since σ is a valid signature, we also have $H(m) = \sigma^e \bmod N$. So, $y = (\sigma/r)^e \bmod N$. Hence, the decryption of y is $\sigma/r \bmod N$.

The probability that $\mathcal{B}$ succeeds is the probability that all hashing queries by the challenger used $b = 0$, that the hashing query related to the forged signature used $b = 1$, and that $\mathcal{A}$ succeeds. By assumption, the message in the forgery was not queried to the signing oracle. So, this happens $p(1-p)^{q_S}$ times the success probability of $\mathcal{A}$, where q_S is the number of signing queries. By taking $p = \frac{1}{q_S+1}$, we have

$$p(1-p)^{q_S} \geq \frac{e^{-1}}{q_S+1}$$

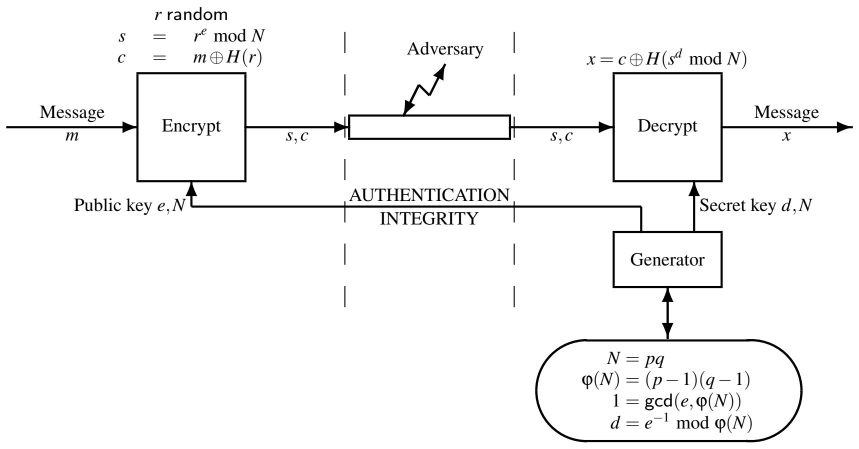


Figure 5.8: Hybrid RSA Encryption in the Random Oracle Model (ROM)

Since the RSA decryption problem is hard, we deduce that $\Pr[\mathcal{A} \text{ succeeds}]/(q_S + 1)$ is negligible. Since q_S is polynomially bounded, this means that $\Pr[\mathcal{A} \text{ succeeds}]$ is negligible as well. So, $\mathcal{A}$ cannot win in the EF-CMA game except with negligible probability. □

Hybrid RSA encryption in ROM. We consider a cryptosystem based on RSA, in which the encryption of m is a pair (s, c) such that s is the RSA encryption of some random r, and $c = m \oplus H(r)$ (where messages have a fixed length and $H(r)$ is assumed to be as long as the message). (See Fig. 5.8.)

Theorem 5.7. *If the RSA decryption problem is hard, then the above cryptosystem is IND-CPA secure.*

Proof. Let $\mathcal{A}$ be an adversary playing the IND-CPA game and wining with probability $\frac{1}{2} + \varepsilon$. We want to show that ε is negligible. Following the rules of the game, $\mathcal{A}$ receives a public key (e, N), makes some hash queries to H, selects m_0 and m_1, gets a ciphertext (s, c) which encrypts m_b, and makes a guess for b.

We let E be the event that $\mathcal{A}$ makes a query r to H that is such that $r^e \bmod N = s$. Note that by running $\mathcal{A}$, we can always check if E occurs once $\mathcal{A}$ terminates. Clearly, if E occurs, the decryption of (s, c) must be $c \oplus H(r)$. So, we can construct another adversary who always answer by b' such that $m_{b'} = c \oplus H(r)$ when E occurs. Without loss of generality, we assume that this is what $\mathcal{A}$ does.

We note that if E holds, $\mathcal{A}$ always win. If E does not occur, we have that $r = s^d \bmod N$ is not queried to H by $\mathcal{A}$ and $c = m_b \oplus H(r)$ with $H(r)$ random. Note that $H(r)$ is uniformly distributed and only used to compute c. So, c is statistically independent from b. Therefore, the view of $\mathcal{A}$ is independent from b and the probability that $\mathcal{A}$ guesses b is exactly $\frac{1}{2}$. We deduce that

$$\frac{1}{2} + \varepsilon = \Pr[\mathcal{A} \text{ wins}|E]\Pr[E] + \Pr[\mathcal{A} \text{ wins}|\neg E](1 - \Pr[E]) = \frac{1}{2} + \frac{1}{2}\Pr[E]$$

So, $\varepsilon = \frac{1}{2}\Pr[E]$.

We construct an algorithm $\mathcal{B}$ to solve the RSA decryption problem. This algorithm receives an instance (e, N, y). Then, he picks $r_0 \in \mathbf{Z}_N^*$ and runs $\mathcal{A}$ playing with a simulation of H and the challenger.

To simulate H receiving a query r, if $(r/r_0)^e \bmod N = y$, the simulation stops and $\mathcal{B}$ answers $r/r_0 \bmod N$. Otherwise, the simulation of H is natural.

To simulate the challenger receiving m_0 and m_1 by $\mathcal{A}$, $\mathcal{B}$ picks c of same length and answers by (s, c) where $s = yr_0^e \bmod N$.

Clearly, this perfectly simulates $\mathcal{A}$ playing the IND-CPA game in the case that E does not occur. When E occurs, $\mathcal{B}$ wins. Now, since the RSA decryption problem is hard, $\Pr[E]$ must be negligible. So, ε is negligible as well. □

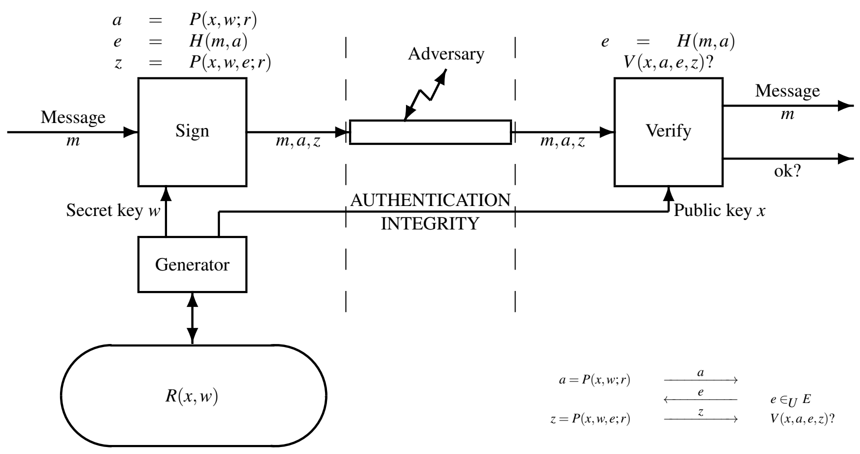


Figure 5.9: Fiat-Shamir Signature from a Σ Protocol

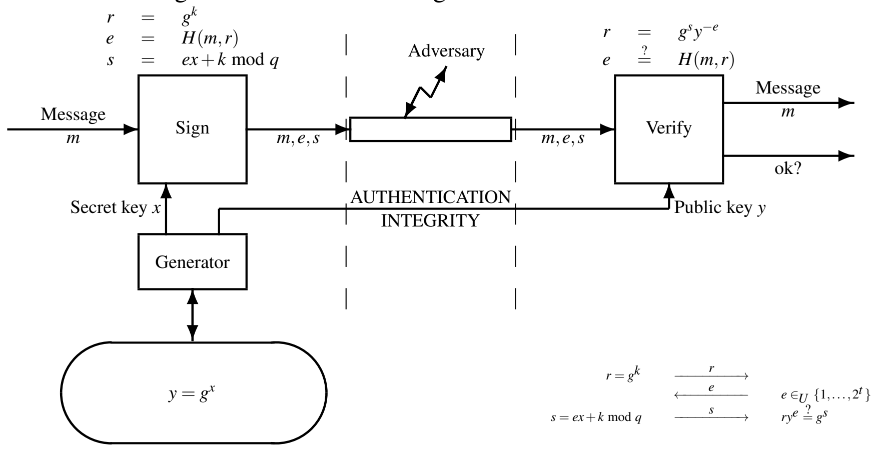


Figure 5.10: Schnorr Signature

Fiat-Shamir signatures [27]. A Σ-protocol $(R,P,V,\mathcal{E},\mathcal{S})$ in which the set of challenges E is large enough can be transformed into a signature scheme in the random oracle model. Concretely, we are given a pair (x,w) such that $R(x,w)$ holds, x is a public key and w is the secret key. To sign a message m, we simulate the prover P who sends $a = P(x,w)$, receives $e = H(m,a)$, and sends z (see Fig. 5.9. The signature is (a,z). To verify the signature, we check that $V(x,a,H(m,a),z)$ holds.

Theorem 5.8. *If the problem of finding a witness for x is hard and if $1/\#E$ is negligible, then the above signature scheme is EF-CMA-secure.*

This construction can be applied to some parallel repetitions of the Fiat-Shamir Σ-protocol. (Indeed, the Fiat-Shamir Σ-protocol has only 2 possible challenges, so we need some parallel repetitions to make $1/\#E$ negligible.) This is based on the problem of finding square roots in $\mathbf{Z}_n^*$. It can also be applied to the Schnorr Σ-protocol to obtain the Schnorr signature scheme (see Fig. 5.10). The only change to make to obtain the Schnorr protocol is to use (e,s) as a signature for m instead of (r,s). This is valid since we can compute (m,e,s) from (m,r,s) and vice versa. It is more interesting to use (m,e,s) because the hash e is typically much shorter than the group element r. It is based on the discrete logarithm problem.

The idea of the proof of this last theorem is that we transform an EF-CMA adversary into a witness finding algorithm as follows:

- First, we transform it into an EF adversary making no chosen message queries. To do this, we simulate the original adversary. We also keep track of the history of queries to H and their responses.

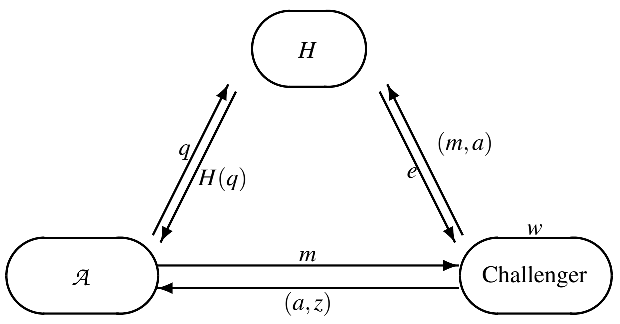


Figure 5.11: EF-CMA Game with a Random Oracle

Whenever it makes a signing query m, the simulator can select a random $e \in E$ and run $\mathcal{S}(x,e) = (a,e,z)$, then insert $((m,a),e)$ in the table collecting the queries to H, as if we had queried $H(m,a)$ and obtained e. The simulator can then answer (a,z) to the signing query. Additionally, any query $H(m,a)$ should be intercepted and the answer replaced by e.

Technically, we must check that there is no prior (m,a) query to H which would conflict with this simulation.

- Then, we run the adversary making no chosen message query and simulate the random oracle H. We show that the message m from the final forgery (a,z) must be queried together with a to $H(m,a)$ (otherwise, there is a negligible probability that the forgery is correct). Then, we run again with the same coins and same simulation for H, until this query (m,a) is responded differently. The magic in this trick, called the *forking lemma* [46], is that we are likely to obtain two simulations producing the same m and a with two different $H(m,a)$. Then, we can call the extractor $\mathcal{E}$ to create a witness w.

For this, we will first show two lemmas.

Lemma 5.9. *Given a relation R s.t. it is hard to find witnesses and a Σ-protocol for its language s.t. $1/\#E = \mathsf{negl}$, we consider the signature scheme obtained by the Fiat-Shamir construction using a random oracle.*

There is a compiler which can transform an adversary $\mathcal{A}$ playing the EF-CMA game into another adversary $\mathcal{A}'$ making no chosen message queries such that the complexity of $\mathcal{A}'$ is the one by $\mathcal{A}$ multiplied by some polynomial and

$$\Pr[\mathcal{A}' \text{ wins}] = \Pr[\mathcal{A} \text{ wins}] - \mathsf{negl}$$

Proof. The EF-CMA game is depicted by the interaction between the adversary $\mathcal{A}$, a challenger, and a random oracle H (see Fig. 5.11). The challenger selects x and w and sends x to $\mathcal{A}$, then answers to any signature query m by computing some signature with the help of the random oracle: the challenger picks r, computes $a = P(x,w;r)$, queries $m\|a$ to H, gets e, computes $z = P(x,w,e;r)$, and answers (a,e,z) to $\mathcal{A}$.

We denote by (m,a,z) be the final forgery produced by $\mathcal{A}$. We first define an equivalent adversary $\mathcal{A}_1$ as follows:

- $\mathcal{A}_1$ simulates $\mathcal{A}$ until $\mathcal{A}$ yields its final forgery.
- If m was queried to the challenger ($\mathcal{A}_1$ can see it), $\mathcal{A}_1$ aborts. So, there is no oracle query from challenger of form (m,a').
- If (m,a) was not queried to H by $\mathcal{A}$, query it to get $e = H(m,a)$.
- If $V(x,a,e,z)$ returns false, abort.
- Yield the forgery (m,a,z).

We obtain: a new EF-CMA adversary $\mathcal{A}_1$ with similar complexity and same success probability, who either aborts or yields a valid forgery (m,a,z), and who always queries $(m\|a)$ to H.

We let $\varepsilon = \Pr[\mathcal{A}\ \mathsf{wins}] = \Pr[\mathcal{A}_1\ \mathsf{wins}]$.

We now define another adversary $\mathcal{A}_2$ as follows:

- $\mathcal{A}_2$ simulates $\mathcal{A}_1$ and makes a list of all queries to the random oracle. The queries (m,a) from the challenger can be deduced from m and (a,z).
- If the challenger does a query which was done before (let ε' be the probability this happens), $\mathcal{A}_2$ aborts.

We obtain a new EF-CMA adversary $\mathcal{A}_2$ with similar complexity and success probability $\varepsilon - \varepsilon'$ such that the challenger makes queries which are fresh.

Since the total number of queries to H must be polynomially bounded, we have $\varepsilon' \leq \mathsf{poly} \times \max_a p_a$ where $p_a = \Pr[\mathcal{P}(x,w;r) = a]$. If we prove that p_a is negligible for all a, we deduce that ε' is also negligible.

Let us now prove that for all a, p_a is negligible. The algorithm running $\mathcal{S}(x,e;r)$ and $\mathcal{S}(x,e';r')$ with random e,e',r,r' yields a twice with probability p_a^2. When it is the case, it can then run $\mathcal{E}$ to extract w. This works with probability $p_a^2(1-1/\#E)$. Since we assumed that finding a witness was hard, this must be negligible. So, p_a is negligible.

We finally define $\mathcal{A}'$ as follows:

- $\mathcal{A}'$ simulates $\mathcal{A}_2$ until a query m to the challenger is made.
- Upon the query m, $\mathcal{A}'$ picks r, e, computes $(a,e,z) = \mathcal{S}(x,e;r)$, and returns (a,z). Since a has the correct distribution, (m,a) must look like a fresh query to H. Since e was selected at random, it looks like a correct response from H. So, $\mathcal{A}'$ just takes note that (m,a) is supposed to hash onto e.
- If $\mathcal{A}_2$ makes a (m,a) query to H, $\mathcal{A}'$ intercepts it and answer e. Since this cannot be the query corresponding to the final forgery, this does not affect the correctness of the final forgery.

Clearly, this simulation of the challenger queries to H are made with the correct distribution. So, it does not affect the probability of success. We thus obtain an EF-CMA adversary $\mathcal{A}'$ making no chosen message query, with similar complexity and success probability $\varepsilon - \varepsilon'$. Since ε' is negligible, we obtain the result.

□

The Forking Lemma was first proposed by Pointcheval and Stern in 1996 [46]. We give here a generalized version of it.

Lemma 5.10 (Forking Lemma). *We consider a finite tree and a mapping* dist *which maps any leaf* λ *of the tree to one of its ancestors* $\mathsf{dist}(\lambda)$. *We call it a* distinguished ancestor. *We assume we are given a distribution which defines a random leaf* X. *We let* $\mathsf{visit}(\nu)$ *be the event that the descent from the root to* X *goes through* ν, *i.e. that* ν *is an ancestor of* X. *We let* $\mathsf{succ}(\lambda)$ *be true if and only if* $\mathsf{dist}(\lambda) \neq \lambda$. *When it occurs we say that* λ *is* successful. *We let* $p = \Pr[\mathsf{succ}(X)]$, $\bar{d} = E(\mathsf{depth}(X))$, *and* $f(\nu) = \Pr[\mathsf{succ}(X) \text{ and } \mathsf{dist}(X) = \nu | \mathsf{visit}(\nu)]$.

For any real number $\theta > 0$, *we have*

$$\Pr\left[f(\mathsf{dist}(X)) > (1-\theta)\frac{p}{\bar{d}}\middle|\mathsf{succ}(X)\right] \geq \theta.$$

So, if a random descent going to X is successful, another random descent starting from the distinguished ancestor of X is likely to be successful (with probability at least $(1-\theta)\frac{p}{\bar{d}}$) with the very same distinguished ancestor. We can even estimate the probability that the two consecutive descents are successful with the same distinguished ancestor:

$$\begin{aligned} E(f(\mathsf{dist}(X))) &= \int_0^1 \Pr[f(\mathsf{dist}(X)) \geq t, \mathsf{succ}(X)]\, dt \\ &= p\int_0^1 \Pr[f(\mathsf{dist}(X)) \geq t | \mathsf{succ}(X)]\, dt \end{aligned}$$

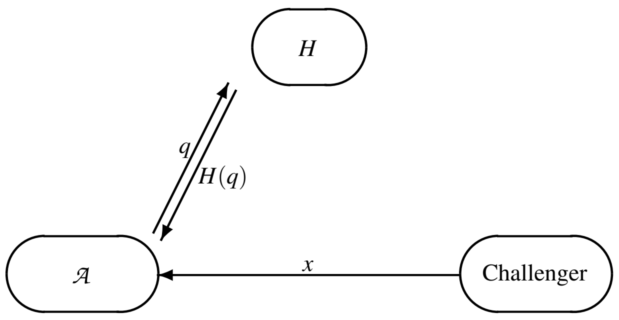


Figure 5.12: EF-CMA Game with a Random Oracle and no Chosen Message

$$\begin{aligned}
&\geq p\int_0^1 \max\left(0, 1 - t\frac{\bar{d}}{p}\right)\, dt \\
&= \frac{p^2}{2\bar{d}}
\end{aligned}$$

So, if p is not negligible and $\bar{d}$ is polynomial, then $E(f(\mathsf{dist}(X)))$ is not negligible. This will be used later to prove the theorem.

Proof. We have $\Pr[\mathsf{dist}(X) = \nu | \mathsf{succ}(X)] = f(\nu)\frac{\Pr[\mathsf{visit}(\nu)]}{p}$. We let Bad be the set of ν's such that $f(\nu) \leq (1-\theta)\frac{p}{\bar{d}}$. We have

$$\begin{aligned}
\Pr[\mathsf{dist}(X) \in \mathsf{Bad} | \mathsf{succ}(X)] &= \sum_{\nu \in \mathsf{Bad}} f(\nu)\frac{\Pr[\mathsf{visit}(\nu)]}{p} \\
&\leq (1-\theta)\frac{\sum_\nu \Pr[\mathsf{visit}(\nu)]}{\bar{d}} \\
&\leq 1-\theta
\end{aligned}$$

so, $\Pr[\mathsf{dist}(X) \notin \mathsf{Bad} | \mathsf{succ}(X)] \geq \theta$. □

We can now fully prove the Fiat-Shamir signature security.

Proof (of Th. 5.8). By first applying Lemma 5.9, we reduce to the case where the adversary makes no chosen message queries. So, we are in the situation of Fig. 5.12.

We define an algorithm $\mathcal{B}(x)$ as follows (see Fig. 5.13):

- $\mathcal{B}$ simulates $\mathcal{A}$ with initial x and simulates H to $\mathcal{A}$.
- If $\mathcal{A}$ does not output any (m,a,z), $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ runs $\mathcal{A}$ again with same random coins. The answers from H use the same random answers until (m,a) is queried to H. Then, they use fresh coins.
- If $\mathcal{A}$ does not output any (m,a,z'), $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ gets two forgeries (a,z) and (a,z') with same a so he can get the corresponding e and e' then extract $w = \mathcal{E}(x,a,e,z,e',z')$.

We build the tree of the $\mathcal{A}$ executions depending on the random answers from H (each node ν corresponds to a query by $\mathcal{A}$, each leaf λ corresponds to a termination).

A random descent in the tree corresponds to a complete execution of $\mathcal{A}$ interacting with H. This descent ends up to a random leaf X. This defines a distribution on leaves. We say that X is successful and write $\mathsf{succ}(X)$ is the leaf corresponds to an execution yielding a valid forgery (m,a,z). By construction, (m,a) must have been queried. The query to (m,a) corresponds to a distinguished ancestor $\mathsf{dist}(X)$ of X. If X is not successful, we just define $\mathsf{dist}(X) = X$. So, the second execution of $\mathcal{A}$ corresponds to a second descent starting from $\mathsf{dist}(X)$. Let Y be the leaf obtained in this second descent. If Y is successful and

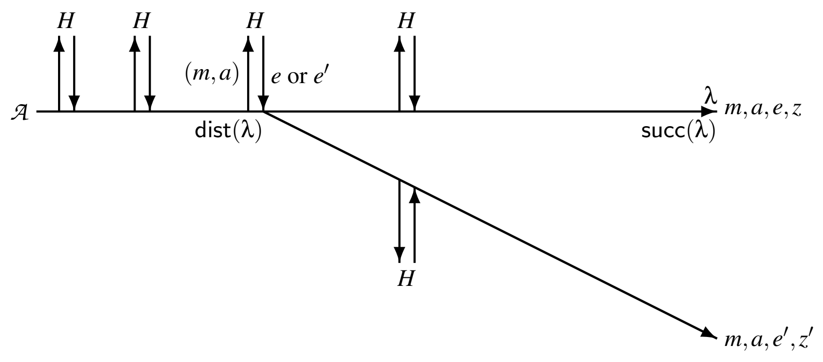


Figure 5.13: The Witness Extractor

$\mathsf{dist}(X) = \mathsf{dist}(Y)$, then we have two forgeries (m,a,z) and (m,a,z') with the same (m,a), corresponding to some e and e'. If $e \neq e'$, the extractor finds a witness and $\mathcal{B}$ succeeds.

Since $\Pr[e = e'] = \mathsf{negl}$, the success probability of $\mathcal{B}$ is greater than $E(f(\mathsf{dist}(X))) - \mathsf{negl}$. Since extracting a witness is assumed to be hard, $E(f(\mathsf{dist}(X)))$ must be negligible. Thanks to the Forking Lemma, we deduce that $\mathsf{Succ}(X)$ is negligible as well. So, $\mathcal{A}$ has a negligible probability of success. □

Controversy about the random oracle model. This model has been controversial, because random oracles are never used in practice. They are replaced by a practical hash function. However, we can construct schemes which are secure in the random oracle model but insecure whenever the random oracle is replaced by *any* hash function. We give as an example a construction proposed by Canetti, Goldreich, and Halevi in 1998 [14].

We use a construction similar as FDH. To sign a message m, we first interpret m as the code of an algorithm implementing a function h_m (we must define a programming language and add safeguards so that the execution of these algorithms always terminate in due time). Then, we pick some r, query $H(r)$, and compute $h_m(r)$. If $H(r) = h_m(r)$, the signature is set to the RSA secret exponent d. Otherwise, the signature is $H(m)^d \bmod N$. Clearly, in the random oracle model, there is nearly no chance that $H(r)$ becomes accidentally equal to $h_m(r)$, so the security proof works like for FDH. When we replace H by a concrete hash function h, we could consider the code m implementing it (i.e., m such that $h_m = h$), and we suddenly obtain $h(r) = h_m(r)$ whatever the selection of r. So, any signing query will obtain the RSA secret exponent which is enough to make forgeries. So, this is EF-CMA insecure.

5.3 The Game Proof Methodology

There is a common technique to prove security based on game reduction. It was formalized by Shoup in 2004 [56]. Indeed, most of the security results can be formalized in terms of an adversary running a game (defined by rules), with a final winning condition. We assume that the game and the winning condition can be efficiently computed by a simulator. The proof technique consists of building up a sequence of games and their associated adversaries in such a way that the initial game is the one to be proven, the final one is trivial to analyze, and we can show that every step makes the winning probabilities similar, except with some negligible gap. There are several tools for making these different steps.

First of all, we can consider an *indistinguishability step*. We start with a game Γ with an adversary $\mathcal{A}$, in which there is somewhere the selection of some random variable X based on some fresh coins which are not used any longer. We build a new game Γ' with the same adversary $\mathcal{A}$, but the selection of X is replaced by the selection of some Y such that X and Y have indistinguishable distributions. Assuming that X or Y come from outside the game, the simulation of the entire game with an outcome set to the winning condition becomes a distinguisher between X and Y. So, the winning probability must be very close for both games.

Second, we can consider *bridging steps* where a game Γ and an arbitrary adversary $\mathcal{A}$ are replaced by a game Γ' and an adversary $C(\mathcal{A})$ such that the simulation of $\Gamma(\mathcal{A})$ and $\Gamma'(C(\mathcal{A}))$ are exactly the same. For instance, we can put in $C(\mathcal{A})$ a simulator for the random oracle.

Finally, we can use the *difference Lemma*. In a game Γ, we consider a "failure event F", for some event F which can be efficiently checked and such that the game becomes somehow simpler when F does not occur. We define a new game Γ' in which $\neg F$ is an extra condition for winning. If $\neg F$ occurs, the game Γ' works exactly like in Γ. The gap between the winning probability is bounded by $\Pr[F]$. Indeed, the probability to win in Γ is

$$\Pr_{\Gamma}[\mathsf{win}] = \Pr_{\Gamma}[\mathsf{win}, F] + \Pr_{\Gamma}[\mathsf{win}, \neg F]$$

The first probability is bounded by $\Pr[F]$. The second one is equal to $\Pr_{\Gamma'}[\mathsf{win}]$, the winning probability in Γ'.

We can now consider a variant of the ElGamal cryptosystem which encrypts strings of m bits (and not group elements). To encrypt M, one has to pick some random r and random n and compute $(g^r, M \oplus h_n(y^r), n)$ where h is a family of universal hash functions (see Fig. 5.14).[1] The idea is that y^r when written as a bitstring, which has a terrible distribution but some decent min-entropy $H_\infty(y^r)$, can be replaced by some $h_n(y^r)$ with n random to have a better distribution.[2] This is called the *leftover hash Lemma*.

Lemma 5.11 (Leftover Hash Lemma, Impagliazzo-Levin-Luby 1989 [35]). *Given a random variable X, if $m \leq H_\infty(X) - 2\log\frac{1}{\varepsilon}$ (where H_∞ denotes the min-entropy), if h is a family of functions from the support of X to $\{0,1\}^m$ such that $\Pr[h_N(x) = h_N(x')] = 2^{-m}$ for all $x \neq x'$, where N is uniformly distributed, then $(h_N(X), N)$ and (U, N) are ε-indistinguishable, where U is uniformly distributed in $\{0,1\}^m$.*

Proof. We let P_0 be the distribution of $(h_N(X), N)$ and P_1 be the distribution of (U, N). We denote by $\mathcal{N}$ the support of N. We compute the Euclidean distance between P_0 and P_1:

$$\begin{aligned}
\|P_1 - P_0\|_2^2 &= \sum_{k,n} \left(\Pr_{X,N}[h_n(X) = k, N = n] - \frac{1}{2^m \#\mathcal{N}} \right)^2 \\
&= \sum_{k,n} \left(\Pr_{X,N}[h_n(X) = k, N = n]^2 - 2\frac{\Pr_{X,N}[h_n(X) = k, N = n]}{2^m \#\mathcal{N}} + \frac{1}{2^{2m}(\#\mathcal{N})^2} \right) \\
&= \sum_{k,n} \Pr_{X,N}[h_n(X) = k, N = n]^2 - \frac{1}{2^m \#\mathcal{N}} \\
&= \sum_{k,n} \Pr_{X,N}[h_n(X) = h_n(X') = k, N = N' = n] - \frac{1}{2^m \#\mathcal{N}} \\
&= \frac{1}{(\#\mathcal{N})^2} \sum_{k,n} \Pr_{X,X'}[h_n(X) = h_n(X') = k] - \frac{1}{2^m \#\mathcal{N}} \\
&= \frac{1}{\#\mathcal{N}} \sum_{x,x'} \Pr[X = x, X' = x', h_N(x) = h_N(x')] - \frac{1}{2^m \#\mathcal{N}} \\
&= \frac{1 - 2^{-m}}{\#\mathcal{N}} \sum_x \Pr[X = x]^2
\end{aligned}$$

which we obtain by splitting $x = x'$ and $x \neq x'$. So,

$$\|P_1 - P_0\|_2^2 \leq \frac{1 - 2^{-m}}{\#\mathcal{N}} \max_x \Pr[x] \leq \frac{1 - 2^{-m}}{\#\mathcal{N}} 2^{-H_\infty(X)} \leq \frac{1}{2^m \#\mathcal{N}} \varepsilon^2$$

We then use $d(\mathsf{distr}(X), \mathsf{uniform}) \leq \|\mathsf{distr}(X) - \mathsf{uniform}\|_2 \sqrt{\#\mathsf{domain}}$ to obtain $d(P_0, P_1) \leq \varepsilon$. $\square$

By using this lemma and some bridging steps, we can prove that this variant of the ElGamal cryptosystem is IND-CPA secure if the DDH problem is hard.

[1] Recall that this means $\Pr[h_N(x) = h_N(x')] = 2^{-m}$ for all $x \neq x'$, where N is uniformly distributed in the key space and 2^m is the range size of h.

[2] The min-entropy of a random variable X is defined by $H_\infty(X) = -\log_2 \max_x \Pr[X = x]$.

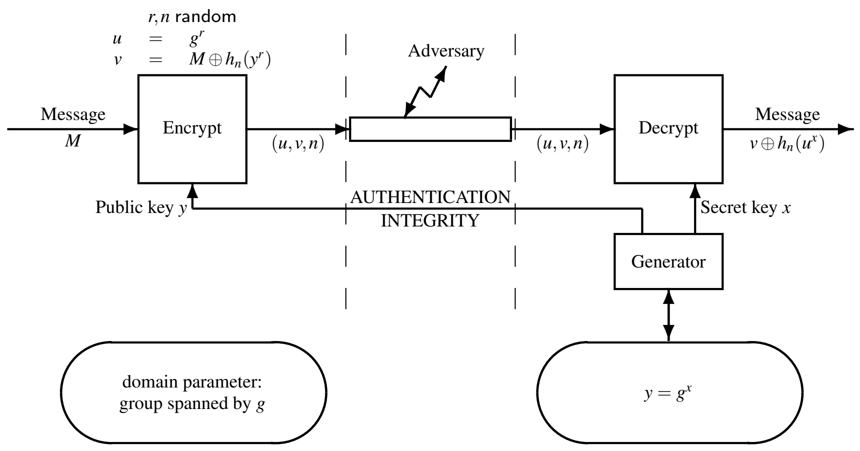


Figure 5.14: ElGamal Cryptosystem Variant

Theorem 5.12. *Assuming that the DDH assumption in the group spanned by g is hard and that h_n is a family of functions from this group to $\{0,1\}^m$ such that for all $x \neq x'$ and a uniformly distributed N, $\Pr[h_N(x) = h_N(x')] = 2^{-m}$. The ElGamal Cryptosystem Variant of Fig. 5.14 is IND-CPA secure.*

Proof. We let Γ_0^b denote the IND-CPA game using bit b. We want to show that Γ_0^0 and Γ_0^1 return 0 with probabilities with negligible difference. We define a small variant Γ_1^b of Γ_0^b. The games Γ_0^b and Γ_1^b are defined like this:

game Γ_0^b/Γ_1^b:

```
1: run key generation and get $y$
2: pick random coins $\rho$ and set view $= (y;\rho)$
3: run $\mathcal{A}(\text{view}) = (m_0, m_1)$
4: pick $r \in_U \mathbf{Z}_q$, $n \in_N \mathcal{N}$, $u = g^r$, and
       ($\Gamma_0$) $v = m_b \oplus h_n(y^r)$  |  ($\Gamma_1$) $s \in_U \mathbf{Z}_q$, and $v = m_b \oplus h_n(y^s)$
5: set view $= (y, u, v, n; \rho)$
6: run $\mathcal{A}(\text{view}) = b'$
7: return $b'$
```

We want to show that for any b, the probabilities that $\Gamma_0^b(\mathcal{A}) = 0$ and $\Gamma_1^b(\mathcal{A}) = 0$ have a negligible difference. For that, we bridge Γ_β^b to some Γ'^b_β game defined as follows:

game Γ'^b_0/Γ'^b_1:

```
1: run key generation and get $y$
2: pick random coins $\rho$ and set view $= (y;\rho)$
3: run $\mathcal{A}(\text{view}) = (m_0, m_1)$
4: pick $r \in_U \mathbf{Z}_q$, $n \in_N \mathcal{N}$, $u = g^r$, and
       ($\Gamma'_0$) $s = r$  |  ($\Gamma'_1$) $s \in_U \mathbf{Z}_q$
   and $v = m_b \oplus h_n(y^s)$
5: set view $= (y, u, v, n; \rho)$
6: run $\mathcal{A}(\text{view}) = b'$
7: return $b'$
```

Clearly, Γ_0^b and Γ'^b_0 are doing the same thing, written in a different way, as well as Γ_1^b and Γ'^b_1. So, $\Pr[\Gamma_\beta^b(\mathcal{A}) = 0] = \Pr[\Gamma'^b_\beta(\mathcal{A}) = 0]$. Next, we bridge Γ'^b_β to Γ''^b_β by changing the adversary. Given $\mathcal{A}$, we define $\mathcal{B}$ as follows:

$\mathcal{B}(g, y, u, v_0, b; \rho)$:

```
1: run $\mathcal{A}(y;\rho) = (m_0, m_1)$
2: pick $n \in_U \mathcal{N}$, $v = m_b \oplus h_n(v_0)$
3: run $\mathcal{A}(y,u,v,n;\rho) = b'$
4: return $b'' = b'$
```

The games are as follows:

game $\Gamma''^b_0/\Gamma''^b_1$:

```
1: run key generation and get $y$
2: pick $r \in_U \mathbf{Z}_q$, $u = g^r$
3: pick
       $(\Gamma'_0)$ | $(\Gamma'_1)$
       $s = r$       | $s \in_U \mathbf{Z}_q$
   and $v_0 = y^s$
4: pick random coins $\rho$
5: run $\mathcal{B}(y,u,v_0,b;\rho) = b''$
6: return $b''$
```

Clearly, $\Gamma'^b_\beta(\mathcal{A})$ and $\Gamma''^b_\beta(\mathcal{B})$ are doing the same thing. It is just a matter of moving some game operations in the adversary. So, $\Pr[\Gamma'^b_\beta(\mathcal{A}) = 0] = \Pr[\Gamma''^b_\beta(\mathcal{B}) = 0]$. Next, we realize that $\mathcal{B}$ trying to distinguish Γ''^b_0 from Γ''^b_1 is a distinguisher playing the DDH game. So, $\Pr[\Gamma''^b_0(\mathcal{B}) = 0] - \Pr[\Gamma''^b_1(\mathcal{B}) = 0]$ is negligible. Hence, we deduce that $\Pr[\Gamma^b_0(\mathcal{A}) = 0] - \Pr[\Gamma^b_1(\mathcal{A}) = 0]$ is negligible.

Next, we change Γ^b_1 into

game Γ^b_1/Γ^b_2:

```
1: run key generation and get $y$
2: pick random coins $\rho$ and set view $= (y;\rho)$
3: run $\mathcal{A}(\text{view}) = (M_0, M_1)$
4: pick $r \in_U \mathbf{Z}_q$, $u = g^r$, and
       $(\Gamma_1)$                                              | $(\Gamma_2)$
       $s \in_U \mathbf{Z}_q$, $X = y^s$, $n \in_U \mathcal{N}$, | $U \in_U \{0,1\}^m$, $n \in_U \mathcal{N}$,
       pair $= (h_n(X), n)$                                      | pair $= (U, n)$
5: $v = M_b \oplus \text{pair}_1$,
6: set view $= (y,u,v,\text{pair}_2;\rho)$
7: run $\mathcal{A}(\text{view}) = b'$
8: return $b'$
```

Thanks to the leftover hash lemma, we have

$$\Pr[\Gamma^b_1(\mathcal{A}) = 0] - \Pr[\Gamma^b_2(\mathcal{A}) = 0] \leq \varepsilon$$

Finally, we bridge Γ^b_2 to Γ^b_3:

game Γ^b_2/Γ^b_3:

```
1: run key generation and get $y$
2: pick random coins $\rho$ and set view $= (y;\rho)$
3: run $\mathcal{A}(\text{view}) = (M_0, M_1)$
4: pick $r \in_U \mathbf{Z}_q$, $u = g^r$, $n$, and
       $(\Gamma_2)$                                  | $(\Gamma_3)$
       $U \in_U \{0,1\}^m$, $v = M_b \oplus U$       | $v \in_U \{0,1\}^m$
5: set view $= (y,u,v,n;\rho)$
6: run $\mathcal{A}(\text{view}) = b'$
7: return $b'$
```

which produce v with exactly the same distribution. We can now notice that b is never used. So,

$$\Pr[\Gamma^0_3(\mathcal{A}) = 0] = \Pr[\Gamma^1_3(\mathcal{A}) = 0]$$

Piling everything together, we have that $\Pr[\Gamma^0_0(\mathcal{A}) = 0] - \Pr[\Gamma^1_0(\mathcal{A}) = 0]$ is negligible. Hence, we have IND-CPA security. □

Bibliography

[1] W. Alexi, B. Chor, O. Goldreich, C. Schnorr. RSA and Rabin Functions: Certain Parts are as Hard as the Whole. *SIAM Journal on Computing*, vol. 17, pp. 194–209, 1988. 5.1

[2] T. Baignères, S. Vaudenay. The Complexity of Distinguishing Distributions (Invited Talk). In *Information Theoretic Security (ICITS'08)*, Calgary, Canada, Lecture Notes in Computer Science 5155, pp. 210–222, Springer-Verlag, 2008. 3.4, 3.5

[3] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, J.-K. Tsay. Efficient Padding Oracle Attacks on Cryptographic Hardware. In *Advances in Cryptology CRYPTO'12*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 7417, pp. 608–625, Springer-Verlag, 2012. 2.1

[4] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. In *Advances in Cryptology CRYPTO'98*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 1462, pp. 26–45, Springer-Verlag, 1998. 5.1

[5] M. Bellare, R. Impagliazzo, M. Naor. Does Parallel Repetition Lower the Error in Computationally Sound Protocols? In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, Miami Beach, Florida, U.S.A., pp. 374–383, IEEE, 1997. 4.1

[6] M. Bellare, P. Rogaway. How to Encrypt with RSA. In *Advances in Cryptology EUROCRYPT'94*, Perugia, Italy, Lecture Notes in Computer Science 950, pp. 92–111, Springer-Verlag, 1995. 5.1

[7] M. Bellare, P. Rogaway. The Exact Security of Digital Signatures: How to Sign with RSA and Rabin. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lecture Notes in Computer Science 1070, pp. 399–416, Springer-Verlag, 1996. 5.2

[8] E. Biham, A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, vol. 4, pp. 3–72, 1991. 3.2

[9] E. Biham, A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. In *Advances in Cryptology CRYPTO'92*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 740, pp. 487–496, Springer-Verlag, 1993. 3.2

[10] E. Biham, A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993. 3.2

[11] D. Bleichenbacher. Generating ElGamal Signatures Without Knowing the Secret Key. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lecture Notes in Computer Science 1070, pp. 10–18, Springer-Verlag, 1996. 2.4

[12] D. Bleichenbacher. Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1. In *Advances in Cryptology CRYPTO'98*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 1462, pp. 1–12, Springer-Verlag, 1998. 2.1

[13] D. Boneh, R. A. DeMillo, R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in Cryptology EUROCRYPT'97*, Konstanz, Germany, Lecture Notes in Computer Science 1233, pp. 37–51, Springer-Verlag, 1997. 2.1

[14] R. Canetti, O. Goldreich, S. Halevi. The Random Oracle Methodology, Revisited. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, Dallas, Texas, U.S.A., pp. 209–218, ACM Press, 1998. 5.2

[15] B. Chor, O. Goldreich. RSA/Rabin Least Significant Bits are $\frac{1}{2} + \frac{1}{\mathsf{poly}(\log n)}$ Secure. In *Advances in Cryptology CRYPTO'84*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 196, pp. 303–313, Springer-Verlag, 1985. 5.1

[16] D. Coppersmith. The Data Encryption Standard (DES) and its Strength against Attacks. *IBM Journal of Research and Development*, vol. 38, pp. 243–250, 1994. 3.2

[17] J.-S. Coron, D. Naccache, J. Stern. On the Security of RSA Padding. In *Advances in Cryptology CRYPTO'99*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 1666, pp. 1–18, Springer-Verlag, 1999. 2.1

[18] J.-S. Coron, D. Naccache, M. Tibouchi, R.-P. Winmann. Practical Cryptanalysis of ISO/IEC 9796-2 and EMV Signatures. In *Advances in Cryptology CRYPTO'09*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 5677, pp. 428–444, Springer-Verlag, 2009. 2.1

[19] D. Coppersmith. Finding a Small Root of a Univariate Modular Equation. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lecture Notes in Computer Science 1070, pp. 155–165, Springer-Verlag, 1996. 2.1

[20] D. Coppersmith. Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lecture Notes in Computer Science 1070, pp. 178–189, Springer-Verlag, 1996. 2.1

[21] D. Coppersmith, M. K. Franklin, J. Patarin, M. K. Reiter. Low-Exponent RSA with Related Messages. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lecture Notes in Computer Science 1070, pp. 1–9, Springer-Verlag, 1996. 2.1

[22] S. A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, Atlanta, Georgia, U.S.A., pp. 151–158, ACM Press, 1971. 4.1

[23] J.S. Coron. On the Exact Security of Full Domain Hash. In *Advances in Cryptology CRYPTO'00*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 1880, pp. 229–235, Springer-Verlag, 2000. 5.2

[24] W. Diffie, M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, vol. IT-22, pp. 644–654, 1976. 2.3

[25] D. Dolev, C. Dwork, M. Naor. Non-Malleable Cryptography. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, New Orleans, Louisiana, U.S.A., pp. 542–552, ACM Press, 1991. 5.1

[26] T. ElGamal. A Public-key Cryptosystem and a Signature Scheme based on Discrete Logarithms. *IEEE Transactions on Information Theory*, vol. IT-31, pp. 469–472, 1985. 2.4, 2.4

[27] A. Fiat, A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology CRYPTO'86*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 263, pp. 186–194, Springer-Verlag, 1987. 4.3, 4.5, 5.2

[28] H. Gilbert, G. Chassé. A Statistical Attack of the FEAL-8 Cryptosystem. In *Advances in Cryptology CRYPTO'90*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 537, pp. 22–33, Springer-Verlag, 1991. 3.3

[29] S. Goldwasser, S. Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, San Fransisco, California, U.S.A., pp. 365–377, ACM Press, 1982. 5.1

[30] S. Goldwasser, S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, vol. 28, pp. 270–299, 1984. 5.1

[31] S. Goldwasser, S. Micali, C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, Providence, Rhode Island, U.S.A., pp. 291–304, ACM Press, 1985. 4.1

[32] S. Goldwasser, S. Micali, C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, vol. 18, pp. 186–208, 1989. 4.2

[33] S. Goldwasser, S. Micali, A. Wigderson. Proofs that Yield Nothing but their Validity and a Methodology of Cryptographic Protocol Design. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, Toronto, Canada, pp. 174–187, IEEE, 1986. 4.2, 4.3

[34] J. Håstad. Solving Simultaneous Modular Equations of low Degree. *SIAM Journal on Computing*, vol. 17, pp. 376–404, 1988. 2.1

[35] R. Impagliazzo, L.A. Levin, M. Luby. Pseudo-Random Generation from One-Way Functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, Seattle, Washington, U.S.A., pp. 12–24, ACM Press, 1989. 5.11

[36] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology CRYPTO'96*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 1109, pp. 104–113, Springer-Verlag, 1996. 2.1

[37] P. Kocher, J. Jaffe, B. Jun. Differential Power Analysis. In *Advances in Cryptology CRYPTO'99*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 1666, pp. 388–397, Springer-Verlag, 1999. 2.1

[38] M. Luby, C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal on Computing*, vol. 17, pp. 373–386, 1988. 3.10

[39] M. Matsui. Linear Cryptanalysis Methods for DES Cipher. In *Advances in Cryptology EUROCRYPT'93*, Lofthus, Norway, Lecture Notes in Computer Science 765, pp. 386–397, Springer-Verlag, 1994. 3.3

[40] M. Matsui. The First Experimental Cryptanalysis of the Data Encryption Standard. In *Advances in Cryptology CRYPTO'94*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 839, pp. 1–11, Springer-Verlag, 1994. 3.3

[41] M. Naor, M. Yung. Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, Baltimore, Maryland, U.S.A., pp. 427–437, ACM Press, 1990. 5.1

[42] P. C. van Oorschot, M. J. Wiener. On Diffie-Hellman Key Agreement with Short Exponents. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lecture Notes in Computer Science 1070, pp. 332–343, Springer-Verlag, 1996. 2.3

[43] J. Patarin. The "Coefficients H" Technique. In *Selected Areas in Cryptography'08*, Sackville, New Brunswick, Canada, Lecture Notes in Computer Science 5381, pp. 328–345, Springer-Verlag, 2008. 3.5

[44] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology CRYPTO'91*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 576, pp. 129–140, Springer-Verlag, 1992. 4.3

[45] S. Pohlig, M. Hellman. An Improved Algorithm for Computing Logarithms over GF(q) and its Cryptographic Significance. *IEEE Transactions on Information Theory*, vol. IT-24, pp. 106–110, 1978. 2.3

[46] D. Pointcheval, J. Stern. Security Proofs for Signature Schemes. In *Advances in Cryptology EUROCRYPT'96*, Zaragoza, Spain, Lecture Notes in Computer Science 1070, pp. 387–398, Springer-Verlag, 1996. 2.4, 5.2, 5.2

[47] M.O. Rabin. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Technical report MIT, Laboratory for Computer Science, TR-212, 1979. 2.2

[48] C. Rackoff, D.R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *Advances in Cryptology CRYPTO'91*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 576, pp. 433–444, Springer-Verlag, 1992. 5.1

[49] R.L. Rivest, A. Shamir and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystem. *Communications of the ACM*, vol. 21, pp. 120–126, 1978. 2.1

[50] P. Rogaway. Nonce-Based Symmetric Encryption. In *Fast Software Encryption'04*, Delhi, India, Lecture Notes in Computer Science 3017, pp. 348–359, Springer-Verlag, 2004. 5.1

[51] I.N. Sanov. On the Probability of Large Deviations of Random Variables. *Matematicheskii Sbornik*, vol. 42, pp. 11–44, 1957. 3.4

[52] C. P. Schnorr. Efficient Identification and Signature for Smart Cards. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 435, pp. 235–251, Springer-Verlag, 1990. 4.3

[53] C. P. Schnorr. Efficient Identification and Signature for Smart Cards. *Journal of Cryptology*, vol. 4, pp. 161–174, 1991. 4.3

[54] A. Shamir. IP=PSPACE. In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, Baltimore, Maryland, U.S.A., pp. 11–15, ACM Press, 1990. 4.1

[55] D. Shanks. Class Number, a Theory of Factorization and Genera. In *Symposium in Pure Mathematics*, Providence, R.I., pp. 415-440, AMS, 1971. 2.3

[56] V. Shoup. Sequences of Games: a Tool for Taming Complexity in Security Proofs. Eprint 2004/332. IACR 2004.
`http://eprint.iacr.org/2004/332` 5.3

[57] A. Tardy-Corfdir, H. Gilbert. A Known Plaintext Attack of FEAL-4 and FEAL-6. In *Advances in Cryptology CRYPTO'91*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 576, pp. 172–181, Springer-Verlag, 1992. 3.3

[58] S. Vaudenay. Decorrelation: a Theory for Block Cipher Security. *Journal of Cryptology*, vol. 16, pp. 249–286, 2003. 3.7, 3.5, 3.8, 3.9, 3.13, 3.14

[59] M. J. Wiener. Cryptanalysis of Short RSA Secret Exponents. In *Advances in Cryptology EUROCRYPT'89*, Houthalen, Belgium, Lecture Notes in Computer Science 434, pp. 372, Springer-Verlag, 1990. 2.1