G22.3033-003 Introduction to Cryptography November 5^{th}, 2001

Lecture 9

Lecturer: Yevgeniy Dodis *Scribe: Antonio Nicolosi*

Last time we defined what we mean by Secret Key Encryption scheme (SKE) and we introduced two related notions of security. Then, we showed how PRGs may be used to construct *stateful* SKE IND-secure against CPA. The problem with that solution was that the state overlaps with the secret key (actually, they are the same information), so that in case the synchronization is temporarily lost, it is not possible for the two parties involved to exchange their own states on the clear, in order to regain synchronization.

To tackle this issue, we mentioned two reasonable ways out: either having the state being just a simple counter, or having no state at all, and adding randomization to the encryption algorithm.

But how efficiently can these two constructions be achieved? To answer this question, in this lecture we introduce the concept of Pseudo Random Function Family (PRF), a really strong cryptographic primitive with several interesting properties. Afterwards, we present a very important application of PRFs, namely a new argument technique which allows us to separate the discussion of efficiency issues from the analysis of the security of the system.

1 Pseudo Random Function Family

1.1 Introduction

When we introduced Pseudo Random Generators, we saw that they are "efficient stretchers of randomness": given a truly random k-bit string, a PRG outputs a much longer (but polynomial in k) stream of bits that "looks random" with respect to any real (i.e. PPT) algorithm. Now we want to go further, and try to answer the question: can we do better? In other words, can we extract an exponential amount of pseudo random bits from a k-bit long seed?

Stated this way, this question does not really make sense, since it is impossible to even write down such a sequence efficiently. Anyway, we may want to know whether we can get an *implicit* representation of an exponential number of bits. One well-known class of exponentially long objects having "short" descriptions is that of *computable functions*, since the dimension of a mapping from $\{0,1\}^l$ to $\{0,1\}^L$ is $2^l \cdot L$.
After all, our question may be rephrased as:

> Can we use a k-bit long seed s to efficiently sample a computable function f_s from the space $\mathcal{R}(l(k), L(k))$ of all possible functions $F : \{0,1\}^{l(k)} \to \{0,1\}^{L(k)}$, in an 'almost-random' manner?

1.2 Definition

The essence of the above question can be formalized by the following definition.

Definition 1 (Pseudo Random Function Family)
A family $\mathcal{F} = \langle f_s \mid s \in \{0,1\}^k \rangle_{k \in \mathbb{N}}$ *is called a family of* $(l(k), L(k))$ *Pseudo Random Functions if:*

- $\forall k \in \mathbb{N}, \forall s \in \{0,1\}^k, \quad f_s : \{0,1\}^{l(k)} \to \{0,1\}^{L(k)}$;
- $\forall k \in \mathbb{N}, \forall s \in \{0,1\}^k, \quad f_s$ *is polynomial time computable;*
- $\mathcal{F}$ *is pseudo random:* $\forall$ PPT $\mathcal{A}$

$$\left| Pr[\mathcal{A}^{f_s}(1^k) = 1 \mid s \leftarrow_R \{0,1\}^k] - Pr[\mathcal{A}^F(1^k) = 1 \mid F \leftarrow_R \mathcal{R}(l(k), L(k))] \right| \leq \mathsf{negl}(k)$$

In other words, for a family $\mathcal{F}$ to be pseudo random, it is not required for its generic element f_s to be indistinguishable from a function F drawn at random from $\mathcal{R}(l(k), L(k))$; rather, the *behavior* of any PPT adversary $\mathcal{A}$ which is given oracle access to the function f_s must be indistinguishable from the behavior of the same adversary when given oracle access to a function F:

$$\forall \text{ PPT } \mathcal{A} \qquad \mathcal{A}^{f_s} \approx \mathcal{A}^F$$

To put this yet in another way, imagine there are two distinct worlds: in the first world the adversary $\mathcal{A}$ queries a function chosen from the family $\mathcal{F}$, while in the second world the adversary's queries are answered by a truly random function F. Here by "truly random function" we mean a black box which outputs a fresh random value on each invocation, except that it is *consistent*, i.e. if queried twice on the same value, it always returns the same output. Now we say that $\mathcal{F}$ is a good family of PRFs if, although the outputs given by f_s are clearly correlated while F's answers are completely independent, the behavior of the adversary is essentially the same, so that it is not possible to tell these two worlds apart.

Comments.
Observe that this is a very strong requirement: how is it possible that no efficient adversary can realize whether it has been interacting with a function f_s that uses only k bits of randomness or with a function F that consist of $2^{l(k)} \cdot L(k)$ random bits? A partial answer is that the adversary can only make polynomially many queries, and thus it doesn't have enough time to infer "which worlds it is in".

1.3 PRFs vs PRGs

Our initial intent was to generalize the notion of PRG: this is indeed the case, since it actually turns out that PRGs can be viewed as a particular instantiation of PRFs.

In the case of a PRF, the adversary is given oracle access to the chosen function f_s; in the case of a PRG, since the output of a generator G is just polynomially long (say k^c), the adversary can be given the entire string $G(s)$.

Anyway, it is trivial to simulate the knowledge of the string $G(s)$ using oracle access to a function f_s: the adversary may ask which bit it wants to know (specifying its position),

and the oracle replies with the value of that bit. Since the position of one bit in a k^c long string can be determined using $c \log k$ bits, and the answer is always one bit long, PRGs may be thought as PRFs where $l(k) = c \log k$ and $L(k) = 1$.

Therefore, for $l(k) = O(\log k)$, PRFs *degenerate* to PRGs; to ensure a gain in power with respect to PRGs, we have to enforce the *non-triviality* condition: $l(k) = \omega(\log k)$.

Notice that there is no such lower bound for the value $L(k)$, since any family $\mathcal{F}$ of PRFs with $L(k) = 1$ can be easily extended to a family $\mathcal{F}'$ with $L(k) = k^c$; for each $f_s \in \mathcal{F}$, include in $\mathcal{F}'$ the function $f'_s : \{0,1\}^{l(k) - c \log k} \to \{0,1\}^{k^c}$ defined as follow:

$$f'_s(x') = \underbrace{f_s(\overbrace{0 \dots 0}^{c \log k} x') \circ f_s(\overbrace{0 \dots 1}^{c \log k} x') \circ \dots \circ f_s(\overbrace{1 \dots 1}^{c \log k} x')}_{k^c \text{ bits}}$$

2 A general construction for PRFs

Once we have defined the notion of PRF family, we look at the problem of building such a family, and of the minimal assumption for the construction to go through. Surprisingly, it turns out that the existence of PRG implies the existence of PRF, although the transformation is too elaborated to be useful in practice. This result is due to Goldreich, Goldwasser and Micali, and is therefore known as **GGM construction**.

The GGM construction presents a loose resemblance to the technique used to obtain an IND-CPA secure stateful SKE scheme from any length-doubling PRG G. Recall from the previous lecture that to this aim we denote by $G_0(x)$ and $G_1(x)$ respectively the first and the second halves of the output of $G(x)$:

$$G : \{0,1\}^k \to \{0,1\}^{2k} \qquad G_0, G_1 : \{0,1\}^k \to \{0,1\}^k$$
$$G(x) \doteq G_0(x) \circ G_1(x)$$

To encrypt the first message, we apply G to the shared key s_0 and set the new state s_1 to be $G_0(s_0)$, while masquerading the message with the pad $p_1 = G_1(s_0)$. The next time a message must be encrypted, the generator G will be applied to the new state s_1, yielding $s_2 = G_0(s_1)$ and $p_2 = G_1(s_1)$. We can think of the whole process as the construction of an *unbalanced* binary tree (sketched in figure 1), in which we always go down to the left: the problem with this approach is that to have n leaves, we have to construct a tree with height n.

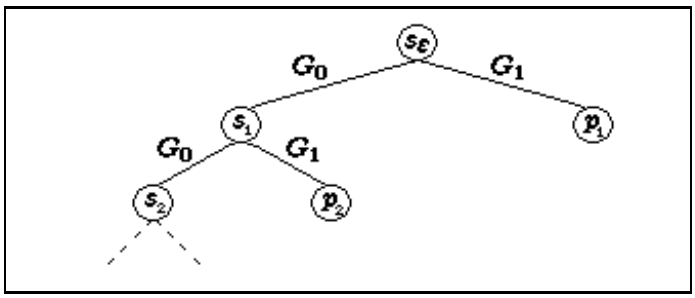

Figure 1: The unbalanced binary tree in the stateful SKE construction.

Clearly, it would be much more efficient to construct a *complete* binary tree, since this would give 2^n leaves on a tree of height n. To do so, for each fixed $s \in \{0,1\}^k$ we define $G_x(s)$ through the following recursion:

$$G_\varepsilon(s) = s$$
$$G_{0\circ\bar{x}}(s) = G_{\bar{x}}(G_0(s))$$
$$G_{1\circ\bar{x}}(s) = G_{\bar{x}}(G_1(s))$$

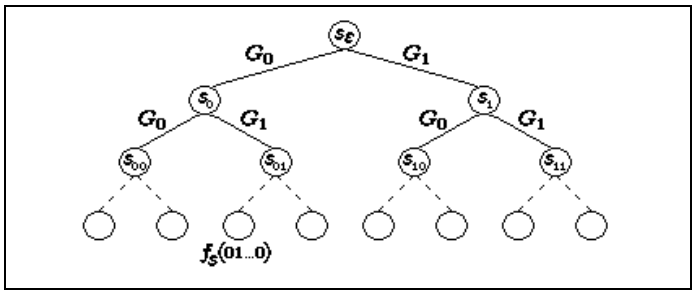

Figure 2: The complete binary tree in the GGM construction.

But how does this construction relate to Pseudo Random Functions? To sample a function $f_s : \{0,1\}^{l(k)} \to \{0,1\}^k$ given a random seed $s \in \{0,1\}^k$, define $f_s(x) = G_x(s)$. In this way, each input x identifies a path in the complete binary tree having s as root, and the output of the function f_s is the value associated to the leaf down such path. According to the definition of $G_x(s)$, to compute the function f_s on a single input it is necessary to evaluate the generator G $l(k)$ times, which is still polynomial, although not very efficient.

It is not obvious that the above construction defines a family $\mathcal{F}$ of PRFs: we are extracting $2^{l(k)}$ different values out of one single truly random string! Still, no efficient adversary bears a (significantly) different behavior whether it interacts with this function or with a genuine random function (i.e. a function in which the $2^{l(k)}$ "leaves" are all random values). This claim is proved in the following theorem, which makes an extensive use of the hybrid argument.

Theorem 2 (Goldreich-Goldwasser-Micali)
The family $\mathcal{F} = \langle f_s \mid s \in \{0,1\}^k \rangle_{k\in\mathbb{N}}$ where $f_s(x) = G_x(s)$ (as explained above) is a family of $(l(k), k)$ Pseudo Random Functions.

Proof: To prove this theorem we want to use the hybrid argument: anyway, in this case the situation is a little different from what we have seen so far, since what we want to prove is not that a single pair of objects are computationally indistinguishable (like in $G(s) \approx R$), but rather that an infinity of related pair of objects are indistinguishable from each other:

$$\forall \text{ PPT } \mathcal{A} \qquad \mathcal{A}^{f_s} \approx \mathcal{A}^F$$

Accordingly, to apply the hybrid argument, we need to find a sequence of *oracles* $T_0 \ldots T_{\mathsf{poly}(k)}$ such that $f_s \equiv T_0, F \equiv T_{\mathsf{poly}(k)}$, and for all the intermediate oracle it holds that:

$$\forall \text{ PPT } \mathcal{A} \qquad \mathcal{A}^{T_i} \approx \mathcal{A}^{T_{i+1}}$$

Let's start defining the appropriate sequence of oracles. Observe that both f_s and F are queried from the adversary $\mathcal{A}$ about the value (in some specific point) of the function they "represent". As a consequence, we can think of those oracles as a set of $2^{l(k)}$ nodes, each containing the value of the function in one of the possible inputs.

In the case of f_s, this nodes can in turn be thought as the leaves of a complete binary tree of height $l(k)$, whose root contains the seed s: this is the tree we talked about when discussing the construction of the function f_s from the PRG $G(x) = G_0(x) \circ G_1(x)$. We can think in a similar way also about the oracle F, even if in the case the "tree structure" is not inherent to its construction: all the nodes in the tree are "empty" nodes, except for the leaves, which contain all the random values of the F.

Stated this way, it is easier to figure out a possible way to "smoothly change" the oracle f_s into the oracle F: instead of having randomness only in the root, and computing all the rest of the tree (and in particularly the leaves, using the PRG G (as in f_s); or having all the randomness in the leaves, so that nothing is to be computed pseudorandomly (as in F), we can define the intermediate oracle T_i to have an empty tree structure from level 0 to level $i-1$, all nodes at level i containing truly random values, and the rest of the tree from level $i+1$ to level $l(k)$ (where we find the leaves, i.e. the values returned by this oracle) being calculated via successive applications of the pseudo random generator G.

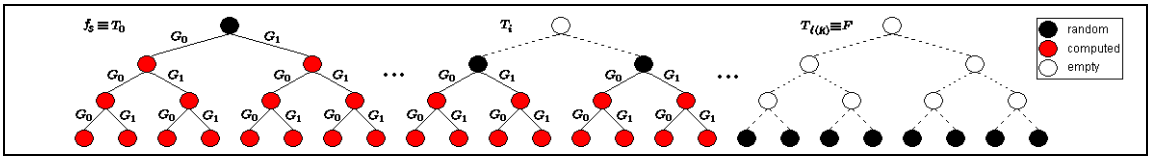

Figure 3: The sequence of oracles used in the first hybrid argument.

In this way we have constructed polynomially many intermediate oracles: in particular $f_s \equiv T_0$, since the randomness is only at level 0 (i.e. the root), and all the rest is computed through applications of G; in addition, $F = T_{l(k)}$, since all the structure above the level $l(k)$ (i.e. the last level of the tree), is empty, and the randomness is contained directly in the leaves (see figure 3). Therefore, by the hybrid argument, we can reduce the proof of the theorem to proving that:

$$\forall \text{ PPT } \mathcal{A} \qquad \mathcal{A}^{T_i} \approx \mathcal{A}^{T_{i+1}}$$

To this aim, let us fix an arbitrary adversary $\mathcal{A}$, and prove that $\mathcal{A}^{T_i} \approx \mathcal{A}^{T_{i+1}}$: having shown that, from the generality of such adversary the above statement will hold true, and thus the proof of this theorem will follow.

Since $\mathcal{A}$ is a PPT algorithm, it can do at most a polynomial number of queries to its oracle, say $t = \text{poly}(k)$. In order to prove that the behavior of $\mathcal{A}$ with oracle T_i is indistinguishable from the behavior of $\mathcal{A}$ with oracle T_{i+1}, we want to use again the hybrid argument: let's look at the correct sequence of intermediate oracle to use.

It would be tempting to consider the sequence of oracles $\overline{T_{ij}}$ in which the randomness is contained in the first j nodes at level i, and in the children (at level $i+1$) of the remaining nodes at level i (see figure 4).

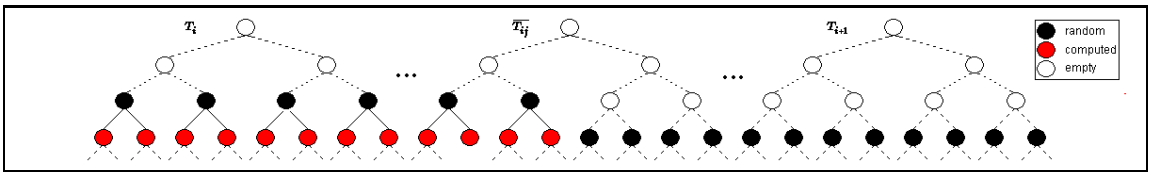

Figure 4: A first (wrong) step towards the second hybrid argument.

Clearly the two extreme of such sequence would be T_i and T_{i+1}, but how many intermediate oracles would result? At level i there are 2^i nodes, and so when i approaches $l(k)$ there would be exponentially many intermediate oracles, way too much for the hybrid argument to go through.

How can we find a way out this situation? Recall that the adversary $\mathcal{A}$ queries its oracle at most t time; intuitively, the problem with the intermediate oracles $\overline{T_{ij}}$ was that they induced a "too fine" differentiation: since $\mathcal{A}$ makes just t queries, having $O(t)$ intermediate oracles must suffice.

Accordingly, for $j \in [0..t]$, define the intermediate oracle T_{ij} as follows: to answer each of the first j queries posed by $\mathcal{A}$, the oracle T_{ij} associates a random value to the unique node at level i that lies in the path from the root to the leaf containing the value requested by the query, and then it computes the requested value via $l(k) - i$ applications of the generator G (notice that up to now this is exactly the behavior born by the oracle T_i). Beginning with the $(j+1)^{st}$ query, the oracle T_{ij} starts behaving like T_j: to respond to a request for the value associated with a certain leaf, the oracle picks a random value and puts it inside the ancestor at level $i+1$ of the leaf at hand; afterwards, it computes (as usual) the desired value through $l(k) - i - 1$ calls to the PRG G. There is a last technicality to be added to completely specify the oracle T_{ij}: it acts consistently, i.e. if, while looking at the path from the root to the leaf associated to the value requested by $\mathcal{A}$, the oracle T_{ij} finds out that an ancestor of that leaf has already been filed out (in answering a previous query), it uses that ancestor to compute the value of the leaf, without adding any new randomness to the tree.

Now this sequence is well-suited: it consists of $t+1$ intermediate oracles, and $T_i \equiv T_{i0}$, while $T_{i+1} \equiv T_{it}$. To complete this hybrid argument, it is left to prove that $\mathcal{A}^{T_{ij}} \approx \mathcal{A}^{T_{ij+1}}$. But this is of course the case, since the only difference between the two oracles is that one answered $\mathcal{A}$'s $(j+1)^{st}$ query putting a a random value z at level i (and thus filling its left child l and its right child r with $G_0(z)$ and $G_1(z)$ respectively), while the other answered the same query putting two random values R_1 and R_2 respectively in l and r. If $\mathcal{A}$ behaves differently in the two cases, it would imply that $\mathcal{A}$ is able to distinguish between $G(z) \doteq G_0(z) \circ G_1(z)$ and $R_1 \circ R_2 \equiv R$, or, in other words, $G(z) \not\approx R$, contradicting the pseudorandomness of the generator G used in the GGM construction. It follows that $\mathcal{A}^{T_{ij}} \approx \mathcal{A}^{T_{ij+1}}$, for any j, which entails (by the hybrid argument) that $\mathcal{A}^{T_i} \approx \mathcal{A}^{T_{i+1}}$. From the arbitrariness of $\mathcal{A}$, this holds true for any i, and finally (again by the hybrid argument):

$$\forall \text{ PPT } \mathcal{A} \qquad \mathcal{A}^{f_s} \approx \mathcal{A}^{F}$$

□

Remark 3 *This is by far the most intensive use of the hybrid argument we have seen: it is actually so intense that in the reduction we lose an important fraction of the advantage. This is because, from the proof of the validity of the hybrid argument (see Lecture 5), we know that if the advantage in distinguishing two intermediate distributions is ϵ, then the advantage in distinguishing between the two initial distributions can increase by a factor equal to the number of intermediate elements. Therefore, in our case, we are losing a factor of $t \cdot l(k)$ in comparison with the advantage in breaking the initial* PRG.

3 PRFs IN PRACTICE

The GGM construction is very interesting: it features an original application of the hybrid argument and it demonstrates the use of complete binary trees to build complex primitives out of known, more basic ones. Anyway, the structure of the reduction is such that the construction loses both in efficiency and in security with respect to the underling "building block", namely the pseudo random generator G.

For these reasons, PRFs to be used in practice cannot be obtained in this way: a more concrete, number-theoretic construction is needed. Below we present one of the best-to-date practical (and yet provable!) construction, due to Naor-Reingold, which is based on the DDH assumption.

The construction works in the group $G = QR_p$ of quadratic residues modulo p, where p is a strong prime (i.e. it is of the form $p = 2q + 1$, for some prime q.) In this setting, the DDH assumption can be stated as:

$$\langle g, g^a, g^b, g^{ab} \rangle \approx \langle g, g^a, g^b, g^c \rangle$$

where g is a random generator of G and a, b, c are chosen uniformly at random in $\mathbb{Z}_q$.

For a given choice of the (public) parameters p, q, g, the Naor-Reingold pseudo random function family is $\mathcal{NR} = \langle NR_{p,g,a_0,a_1,\ldots,a_k} \mid a_0, a_1, \ldots, a_k \leftarrow_R \mathbb{Z}_q \rangle_{k \in \mathbb{N}}$, where each function $NR_{p,g,a_0,a_1,\ldots,a_k} : \{0,1\}^k \to G$ is defined as follows:

$$NR_{p,g,a_0,a_1,\ldots,a_k}(x_1, \ldots, x_k) = (g^{a_0})^{\prod_{i \in S(x)} a_i} \qquad \text{where } S(x) = \{i \geq 1 \mid x_i = 1\}$$

In other words, the input $x = x_1, \ldots, x_k$ is considered bit by bit and, for each x_i equal to 1, the corresponding value a_i is included in the modular exponentiation. The advantage of such definition is that the value of the function on a particular point can be computed with $O(k)$ multiplications: we can compute the exponent $\alpha = a_0 \prod_{i \in S(x)} a_i \bmod q$ with at most k multiplications, and then compute $g^\alpha \bmod p$ with at most $2k$ multiplications (using the "Square & Multiply" algorithm.)

The NR construction for PRFs can be though as a particular instantiation of the complete binary tree technique seen in the GGM construction, where the PRG used is $G_{p,q,g,g^a}(g^b) \doteq G'_0(g^b) \circ G'_1(g^b) = \langle g^b, g^{ab} \rangle$. However, in some sense, the solution proposed by Naor Reingold also generalizes the previous construction, since a different exponent a_i is considered at each level of the tree, while in the standard GGM the same PRG is used at all levels. This is showed in figure 5.

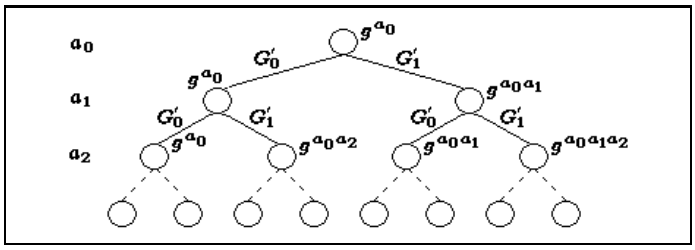

Figure 5: The complete binary tree in the NR construction.

The proof of security of this PRF family bears a close resemblance to the proof of the previous theorem; consequently, we state the result without explicitly including the supportive argument.

Theorem 4 (Naor Reingold)
Under the DDH *assumption, the family* $\mathcal{NR} = \langle NR_{p,g,a_0,a_1,\ldots,a_k} \mid a_0, a_1, \ldots, a_k \leftarrow_R \mathbb{Z}_q \rangle_{k \in \mathbb{N}}$ *is a* PRF *family.*

4 Applications

Pseudo Random Functions are a very powerful cryptographic tool: their key property (namely, no efficient algorithm substantially changes its behavior whether it interacts with a pseudo random function or a truly random one) is so strong that we can do a lot of things out of them. Let's look at some application.

4.1 Identify Friend or Foe

The problem of identifying friend or foe (IFF) consists in deciding, in a dynamic setting, whether you are facing an enemy or an ally. Consider an air battle between two parties A and B, in which all the planes belonging to the same air force share a secret value, and let i and j be respectively the secret associated with parties A and B. Before shooting a potential target, a warplane of party A challenge the target with a random r: if the target reply with $f_i(r)$, then it is identified as a friend and it is not destroyed. In this scenario, the adversary will not be able to reply correctly, since even after seeing many pairs $(r', f_i(r'))$, he still has no clue of the value $f_i(r)$ for an unseen, random r. Moreover, since this setting is dynamic, the adversary has no time to mount a "man-in-the-middle" attack, forwarding the challenge to another warplane: indeed, the adversary itself cannot distinguish its friends from its enemies!

4.2 The Random Function Model

The most important application of Pseudo Random Functions is that they enable a higher level technique to argue about security. Given a cryptographic scheme which uses PRFs, to prove its security against an adversary, we consider the chances the adversary $\mathcal{A}$ has to break the system in the *Random Function Model*, where *every pseudo random function* f_s *is replaced with a truly random function* F. What we gain from this transformation is that

it is usually simpler to deal with security in the Random Function Model, since one can make use of Information-Theoretic considerations to show that the adversary's advantage (whatever it is defined to be) is negligible. Then, we can argue that $\mathcal{A}$'s advantage remains negligible even in the original scheme, because otherwise we would have found an efficient algorithm (the adversary $\mathcal{A}$) whose behavior is significantly different whether it interacts with a PRF or a truly random one, contradicting the definition of PRF.

In brief, to argue about a cryptographic scheme that uses PRFs, we proceed as follows:

1. we discuss its *efficiency* in the PRF world;
2. we prove the *security* (information-theoretically) in the Random Function Model;
3. we conclude that the original system is also secure when using PRF, since otherwise we would contradict the pseudo randomness of the PRF family.

The only thing to care of is that in going from point 1 to point 2, it is allowed to substitute random functions only in place of pseudo random function f_s whose seed s is never shown to the adversary $\mathcal{A}$, because otherwise f_s will not "look random" to $\mathcal{A}$.

This is a powerful technique, that allows us to quickly establish the security of complex constructions: we discuss two examples in the following subsections.

4.3 Construction of an IND-CPA-secure SKE with counter as state

Now that we have developed the new tool of PRFs, it is a simple matter to define a stateful SKE scheme IND-secure against chosen plaintext attack which uses a simple counter to maintain the state. This scheme is known as CTR *scheme*.

1. On input 1^k, the *key-generation algorithm* G chooses a family $\mathcal{F}$ of PRFs, and sets the public key PK to be the description of that particular family (for example, in the case of the NR construction, this consists of the choice of the strong prime $p = 2q + 1$ along with a generator g of QR_p.) Afterwards, G takes a random seed $s \in \{0,1\}^k$ and sets $SK = s$. Finally, G outputs (PK, SK, M_k), where $M_k = \{0,1\}^{L(k)}$. The counter for both the encryption and decryption algorithm is initially set to 0.
2. To encrypt a message $m \in M_k$ when the counter is `ctr`, the *encryption algorithm* E_{PK} outputs the ciphertext $c = f_s(\texttt{ctr}) \oplus m$, and increment its counter `ctr`.
3. In order to decrypt a ciphertext c, having counter `ctr`, the decryption algorithm D compute $m = f_s(\texttt{ctr}) \oplus c$, and increment the value of `ctr`.

In attacking this cryptosystem, the adversary Eve knows the PRF family $\mathcal{F}$ being used, the security parameter k and the current value of the counter (since she can easily keep track of all the ciphertexts Alice has sent to Bob.) Nevertheless, all that she knows can be expressed as the knowledge of polynomially many pairs of the form $(r', f_s(r') \oplus m')$ and this does not help her to gain any advantage in guessing m from its encryption $f_s(\texttt{ctr}) \oplus m$.

It is possible to give a formal proof of the above claim using a standard reductionist argument; however, we prefer to use the higher level technique introduced in previous section, both to demonstrate the use of such technique and because the argument becomes simpler.

Theorem 5 *The* CTR *scheme defined above is an* IND-CPA*-secure* SKE.

Proof: The CTR scheme is clearly an SKE since both the encryption and the decryption algorithm can be efficiently computed (notice that f_s is poly-time computable by the definition of PRF family), and the correctness property is trivially fulfilled.

Following the steps suggested in the previous subsection, we argue about the security of this scheme in the Random Function Model. Since the adversary does not know the seed s used in the pseudo random function f_s, it is legitimate to replace this function with a truly random function F within the Encryption and the Decryption algorithms:

$$\begin{array}{rll} E_F(m): & \textbf{set:} & c \leftarrow F(\mathtt{ctr}) \oplus m, \quad \mathtt{ctr} \leftarrow \mathtt{ctr} + 1 \\ & \textbf{output:} & c \\ D_F(c): & \textbf{set:} & m \leftarrow F(\mathtt{ctr}) \oplus c, \quad \mathtt{ctr} \leftarrow \mathtt{ctr} + 1 \\ & \textbf{output:} & m \end{array}$$

In the chosen plaintext attack, Eve, after having queried her encryption oracle a number of times, chooses a pair of messages m_0 and m_1 whose encryptions she believes to be able to distinguish from each other. Once challenged with the encryption $c = F(\mathtt{ctr}) \oplus m_b$, for a random $b \in \{0, 1\}$, she can make polynomially more queries to the oracle, and then she has to decide which message was encrypted. How likely is she to succeed? We claim that her probability of success is exactly $1/2$.

To see why, notice that we are working in the Random Function Model, and thus the value $F(\mathtt{ctr})$ is completely random and independent from the values $F(0), \ldots, F(\mathtt{ctr} - 1)$ computed so far, and possibly known to Eve. Moreover, it is independent from all values used in subsequent queries made by Eve to her oracle, since the counter consists of $l(k)$ bits, and so it would take $2^{l(k)}$ queries for the counter to overflow and take again the same value. From the non-triviality condition for PRF families, this is more than polynomial, and thus Eve does not have enough time to wait until this happens. It follows that each time she ask the oracle to encrypt a message, she will get back a ciphertext $c = F(\mathtt{ctr}') \oplus m'$ which is a completely random quantity (thanks to the randomness of $F(\mathtt{ctr}')$), and thus she cannot learn anything from it. Therefore, the best she can do to guess the bit b is flipping a coin, and hence the CTR scheme is IND-CPA-secure in the Random Function Model.

From the pseudorandomness of the family $\mathcal{F}$ we can now conclude that the original CTR scheme is IND-CPA-secure, since otherwise Eve would have a different behavior whether accessing the oracle f_s or the random oracle F, contradicting the assumption that $\mathcal{F}$ is a family if PRFs. □

Remark 6 *Notice that the advantage of Eve in the Random Function Model is exactly* $1/2$*, since although she has access to an oracle, she cannot learn anything from its responses, so that she cannot do anything better then guessing a bit at random. When we go from the Random Function Model back to the "real world", Eve gains at most the same negligible advantage possible in distinguish between the "two worlds", which is known to be at most negligible from the pseudorandomness of the family* $\mathcal{F}$*.*

4.4 Construction of a stateless IND-CPA-secure SKE

Once we have seen the CTR scheme, it is easy to modify it so that no state is required neither for encryption nor for decryption. Indeed, in the CTR SKE each ciphertext has the form $F(\texttt{ctr}) \oplus m$, and the legitimate recipient (Bob) is able to decrypt it because it maintains his state in sync with Alice's state. What if the synchronization is lost? The two parties can simply exchange their own states, since this would be of no help for the attacker.

But therefore, why do we need synchronization? Alice and Bob can simply exchange their states each time, or, even better, they can avoid keeping state at all, by having the encryption algorithm choose a new, fresh random value to be used as the "state" for the encryption at hand, and send it along with the "real" ciphertext to Bob. This leads to the definition of the XOR scheme, included below.

1. On input 1^k, the *key-generation algorithm* G chooses a family $\mathcal{F}$ of PRFs, and sets the public key PK to be the description of that particular family (for example, in the case of the NR construction, this consists of the choice of the strong prime $p = 2q + 1$ along with a generator g of QR_p.) Afterwards, G takes a random seed $s \in \{0,1\}^k$ and sets $SK = s$. Finally, G outputs (PK, SK, M_k), where $M_k = \{0,1\}^{L(k)}$.

2. To encrypt a message $m \in M_k$, the *encryption algorithm* E_{PK} chooses a random value $r \in \{0,1\}^k$, compute $c = f_s(r) \oplus m$ and outputs the ciphertext $c' = \langle r, c \rangle$.

3. In order to decrypt a ciphertext $c' = \langle r, c \rangle$, the decryption algorithm D computes the "pad" $f_s(r)$ and then the message $m = f_s(r) \oplus c$.

In mounting a chosen plaintext attack against this cryptosystem, the adversary Eve knows the PRF family $\mathcal{F}$ being used and the security parameter k, and she has access to an encryption oracle. To show that her probability of success will anyway be at most negligible, again we can use Information-Theoretic arguments in the Random Function Model; before turning to the proof of security, we need a result from Elementary Probability Theory, the *Birthday Paradox*, which can be exploited by Eve in the form of the *Birthday Attack*.

Lemma 7 (Birthday Paradox)
Choosing at random q values from a set of $N >> q$ possible values, the probability of taking twice the same value is approximately $\frac{q^2}{2N}$.

Proof: First, recall that for small value of x, it holds that $e^{-x} \simeq (1 - x)$. Then, we consider the negation of the event we are interested in, i.e. "no repetitions choosing q times an element out of N." This can be said as "the first element is arbitrary," "the second element differs from the first," "the third element differs from the first and the second," and so on, until we get to "the q^{th} element differs from the first, the second, ..., and the $(q-1)^{st}$." Since each element is drawn from the set of N possible values independently,

this probability can easily be evaluated as:

$$\begin{aligned} &Pr[\text{“no repetitions choosing } q \text{ times an element out of } N\text{”}] = \\ &\quad = 1 \cdot \left(1 - \frac{1}{N}\right) \cdot \left(1 - \frac{2}{N}\right) \cdot \ldots \cdot \left(1 - \frac{q-1}{N}\right) \simeq \\ &\quad \simeq 1 \cdot e^{-\frac{1}{N}} \cdot e^{-\frac{2}{N}} \cdot \ldots \cdot e^{-\frac{q-1}{N}} = e^{-\frac{1}{N}\sum_{i=1}^{q-1} i} = e^{-\frac{q(q-1)}{2N}} \simeq 1 - \frac{q(q-1)}{2N} \end{aligned}$$

Going back to the event we started from, we finally get:

$$\begin{aligned} &Pr[\text{“at least one repetition choosing } q \text{ times an element out of } N\text{”}] = \\ &\quad = 1 - Pr[\text{“no repetitions choosing } q \text{ times an element out of } N\text{”}] \simeq \\ &\quad \simeq 1 - \left(1 - \frac{q(q-1)}{2N}\right) = \frac{q(q-1)}{2N} \simeq \frac{q^2}{2N} \end{aligned}$$

□

We are now ready to prove the security of the XOR scheme.

Theorem 8 *The* XOR *scheme defined above is an* IND-CPA*-secure* SKE.

Proof: The XOR scheme is clearly an SKE since both the encryption and the decryption algorithm can be efficiently computed; in addition, since the value used to generate the pad is sent to the intended recipient of the message, the correctness property is also satisfied.

More interesting is the discussion of security: again, we consider the resilience of the XOR scheme from a chosen plaintext attack, in the Random Function Model. Since the adversary does not know the seed s used in the pseudo random function f_s, it is legitimate to replace this function with a truly random function F within the Encryption and the Decryption algorithms, obtaining:

$$\begin{array}{lll} E_F(m): & \textbf{set:} & r \leftarrow_R \{0,1\}^{l(k)}, \quad c \leftarrow F(r) \oplus m \\ & \textbf{output:} & c' = \langle r, c \rangle \\ D_F(c): & \textbf{set:} & m \leftarrow F(r) \oplus c \\ & \textbf{output:} & m \end{array}$$

Suppose that Eve has queried the oracle a number of time, has chosen the pair of messages m_0 and m_1, and now she has been challenged on the ciphertext $c' = \langle r, F(r) \oplus m_b \rangle$, for a random $b \in \{0,1\}$. Since r was chosen at random, and F is a genuine random function, $F(r) \oplus m_b$ is also random (in other words, the pad $F(r)$ completely hide the message m_b.) Therefore, the only way for Eve to gain information about m_b is to repeatedly query the oracle on m_0 and m_1 in the hope that the randomness used by the oracle in one of its responses is the same used in the encryption of m_b, so that the response Eve gets back equals the ciphertext c'. What is the probability that this strategy (know as *Birthday Attack*) succeed?

To evaluate this probability, assume that during her attack (which must be polynomial time), Eve queries $q(k)$ times the oracle on the same message. Clearly, $q(k) = \mathsf{poly}(k)$ and

from the Birthday Paradox Lemma, we get that the probability that the same randomness is used twice is $O\left(\frac{q(k)^2}{2^{L(k)}}\right) = \mathsf{negl}(k)$.

Summarizing, using the Birthday Attack, Eve can improve her probability of guessing b only from $1/2$ to $1/2 + \frac{q(k)^2}{2^{L(k)}}$, and hence the XOR scheme is IND-CPA-secure in the Random Function Model; from the pseudorandomness of the family $\mathcal{F}$ used in actual XOR construction, we can finally conclude that the original XOR scheme is IND-CPA-secure. □

Remark 9

It is worth pointing out that in this case, because of the Birthday Attack, even in the Random Function Model Eve has an advantage slightly better than guessing in breaking the CPA*-security of the* XOR *scheme. Although this make no difference in an asymptotic sense (since her advantage is anyway negligible), in practice this can be an issue, since it may leads to the necessity of using bigger values for the security parameter k, thus worsening, in the long run, the efficiency performance of the system.*

This consideration shows that there is no simple answer to the third question we asked about SKE *schemes, namely whether it is better to have stateful or randomized encryption schemes. Both the two approaches have their* pro's *and* con's*: the* CTR *scheme is a little bit inconvenient due to the necessity of maintaining a state, while the* XOR *scheme, albeit stateless, may look unsatisfactory in terms of exact security, because of its weakness against the Birthday Attack.*