Last time we introduced the concept of Pseudo Random Function Family (PRF), and showed a generic construction of PRFs from PRGs, as well as a more efficient construction under the DDH assumption. This lecture we present several important applications of PRFs. First, a new argument technique which allows us to separate the discussion of efficiency issues from the analysis of the security of the system. And, second, how to apply PRFs to easily build CPA-secure symmetric cryptosystems. In particular, we will answer the 3 questions we asked in the previous lecture.

# 1   APPLICATIONS OF PRFS

Pseudo Random Functions are a very powerful cryptographic tool: their key property — 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 with PRFs. Let's look at some simple applications.

## 1.1   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 negligible probability in predicting the value $f_i(r)$ for an unseen, random $r$. Concretely, if the attacker saw up to $q$ previous $(r', f_i(r'))$ pairs, and (negligibly small) $\varepsilon$ is the security of our PRF against a distingusiher making at most $q$ PRF queries, then the maximum probability the attacker can predict the $k$-bit value $f_i(r)$ is at most $\varepsilon + \frac{q}{2^k}$, which is negligible. 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!

## 1.2   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 Adv 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 Adv's advantage remains negligible even in the original scheme, because otherwise we would have found an efficient algorithm (the adversary Adv) 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 Adv, because otherwise $f_s$ will not "look random" to Adv.

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

## 1.3   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 1** *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:

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

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

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 $c = F(\texttt{ctr}) \oplus m_b$, 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 $F(\texttt{ctr})$ is completely random and independent from the values $F(0), \ldots, F(\texttt{ctr} - 1)$ 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 $\ell(k)$ bits, and so it would take $2^{\ell(k)}$ 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 $c = F(\texttt{ctr}') \oplus m'$ which is a completely random quantity (thanks to the randomness of $F(\texttt{ctr}')$), 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. $\qquad \square$

**Remark 1** *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* $\mathcal{F}$.

## 1.4 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 $m = f_s(r) \oplus c$.

**Theorem 2** *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:

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

$$D_F(c): \quad \begin{aligned} &\textbf{set:} \quad && m \leftarrow F(r) \oplus c \\ &\textbf{output:} \quad && m \end{aligned}$$

Suppose that Eve has queried the encryption oracle a total of $q = q(k)$ times. Let $r_1 \ldots r_q$ be the random values chosen by the encryption oracle. And let $r$ be the random value used to encrypt the challenge ciphertext $\langle r, F(r) \oplus m_b \rangle$, where bit $b$ is random. It is clear the only way the attacker gets any information about the bit $b$ is if the "challenge" value $r$ belongs to the encryption oracle choices $\{r_1 \ldots r_q\}$. But since $r$ is random, this

probability is at most $q/2^\ell$, which is negligible. Therefore, in the Random Function Model the probability of success is at most $1/2 + q/2^\ell = 1/2 + \mathsf{negl}(k)$. 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. $\qquad\square$

**Remark 2** *It is worth pointing out that in th case of the* XOR *scheme Eve has an advantage slightly better than guessing even in the Random Function Model. 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, and also because the length of the ciphertext is clisghly shorter (because it has to include the value r).*

## 1.5 Encrypting Long Messages

As we will see shortly, "practical" PRFs typically have fixed input and outputr size, often roughly equal to the security parameter $k$ (say, both being 128 bits). How can we use such PRFs to encrypt arbitrarily long messages? There are several ways out.

First, we notice that the CTR and XOR scheme only require long outputs size to support long messages. And we already mentioned that we can expand the output by a factor $2^a$ by "wasting" $a$ input bits. Thus, to encrypt 1 Gigabyte $\approx 2^{30}$ file using $L = \ell = 128 = 2^7$ one can create a new PRF with input size $128 - (30 - 7) = 103$ and output size $2^{30}$. Indeed, we will later study this scheme (under a different, but equivalent definition) which will be called "CTR mode of operation".

However, we will see that there are several other popular and effective ways to go about supporting long messages from "fixed length" PRFs. Historically, however, these methods, which will be called *modes of operation* originated from a stronger cryptographic primitive called a *block cipher*, or a *pseudorandom permutation* (PRP).[1] As it turned out, some of these modes of operations, like the counter mode, indeed work even better using regular PRFs. Others, like the "CBC mode" that we study shortly actually require the stronger structure of PRPs. Therefore, we make a brief detour and study PRPs, and then come back to various modes of operations used to encrypt long messages.

## 2 PSEUDO RANDOM PERMUTATION FAMILY

The concept of *Pseudo Random Permutations* (PRPs) is motivated by the desire to create a length-preserving encryption, so that we do not have to transmit a lot of additional bits for each bit of data that we encrypt. By now we know that such an encryption cannot be

---

[1]Usually, the term "block cipher" is used to describe a "fixed length PRP".

semantically secure, but the introduction of PRPs happens to be nevertheless useful both from practical and theoretical perspectives. We now make several definitions.

## 2.1 Definitions

We start with the following definition of a PRP:

DEFINITION 1 [Pseudo Random Permutation Family] A family $\mathcal{G} = \langle g_s : \{0,1\}^{\ell(k)} \rightarrow \{0,1\}^{\ell(k)} \mid s \in \{0,1\}^k \rangle_{k \in \mathbb{N}}$ is called a family of Pseudo Random Permutations if the following properties hold:

1. $g_s$ is a permutation, $\forall s$.

2. $g_s$ and $g_s^{-1}$ are both efficiently computable, $\forall s$.

3. $g_s$ is pseudo random: $\forall$ PPT Adv

$$\left| Pr[\, \mathsf{Adv}^{g_s}(1^k) = 1 \mid s \leftarrow_R \{0,1\}^k \,] - Pr[\, \mathsf{Adv}^P(1^k) = 1 \mid P \leftarrow_R \mathcal{P}(\ell(k)) \,] \right| \leq \mathsf{negl}(k)$$

where $P$ represents a random *permutation* on $\ell(k)$ bits, chosen from the set of all such permutations $\mathcal{P}(\ell(k))$.

$\diamondsuit$

This definition is very similar to the definition of PRF from the previous lecture, and it would be a good idea to look back and compare. Again, when we state that $g_s$ is pseudo random, we mean that if Adv is given oracle access to $g_s$ (in particular, he does not know $s$) he cannot tell it apart from access to a "fake" oracle $P$ which represents a *truly random* permutation. Using indistinguishability notation:

$$\forall \text{ PPT Adv} \qquad \mathsf{Adv}^{g_s} \approx \mathsf{Adv}^P$$

Again, be sure to refresh your memory of PRFs from the definition in the previous lecture notes, and compare it with our definition of a PRP.

A stronger, and often required form of PRP is that of a strong PRP.

DEFINITION 2 [Strong PRP] A Strong PRP satisfies all the properties of a family of Pseudo Random Permutations, except the constraint [3.] is replaced by a stronger requirement below:

3'. $g_s$ is pseudo random even if oracle access to $g_s^{-1}$ is provided as well: for any PPT Adv,

$$\mathsf{Adv}^{(g_s,\, g_s^{-1})} \approx \mathsf{Adv}^{(P,\, P^{-1})}$$

$\diamondsuit$

This new definition provides us with an additional measure of security by allowing us to give Adv oracle access to $g_s^{-1}$.

## 2.2 Encryption from PRP's?

To partially motivate previous definitions, let us propose a naive "encryption" scheme that is length preserving: given a secret key $s$ and a message $m$,

$$c = E_s(m) = g_s(m) \qquad m = D_s(c) = g_s^{-1}(c)$$

If we let $g_s$ be taken from a family of Efficient PRPs, the scheme works, since $c$ can be decrypted it PPT. Also, since $E_s$ is a permutation, this encryption is length preserving, and we need not transmit any excess data as part of the ciphertext. However, there is a serious security flaw in the scheme: the adversary can tell if we send the same message twice, since the encryption is deterministic. Hence, the scheme is certainly not CPA-secure (even though it is obviously one-time, but this is uninteresting). Intuitively, though, the only thing that the adversary can learn from seeing several encryptions is which encryptions correspond to the same message. This is clearly the best we can hope for with length-preserving encryption: namely such encryption must look like a random permutation! However, can we build an efficient CPA-secure system from a PRP family?

As we will see, the answer is "yes". In fact, there are several ways to do it. However, we will examine these later, and start by relating PRFs and PRPs.

## 3 PRF $\Longleftrightarrow$ PRP

It turns out that any valid PRG is also a PRF, so if we wish to construct a PRF from a PRP, don't really have to do any work at all.

**Theorem 3** (PRP $\Rightarrow$ PRF) *Assume $\mathcal{G}$ is a PRP family. Then $\mathcal{G}$ is also a PRF family.*

Before turning to the proof of this statement, we need a result from Elementary Probability Theory, called the *Birthday Paradox.*

**Lemma 1 (Birthday Paradox)**
*Choosing at random $q$ values from a set of $N \gg q$ possible values, the probability of taking twice the same value is approximately $\frac{q^2}{2N}$.*

**Proof:** We have

$$\Pr[\text{``at least one repetition choosing } q \text{ times an element out of } N\text{''}] \leq$$

$$\leq \sum_{1 \leq i < j \leq q} \Pr[\text{``item } i \text{ collides with item } j\text{''}] = \binom{q}{2} \cdot \frac{1}{N} = \frac{q(q-1)}{2N} \leq \frac{q^2}{2N}$$

$\square$

Now we can prove Theorem 3.

**Proof:** The proof uses the hybrid arguement. If we start by considering that members of $\mathcal{G}$ are indistinguishable from random permutations to the adversary Adv, all we need to show is that a random permutation is indistinguishable from a random function. In

other words, we must show $\mathsf{Adv}^P \approx \mathsf{Adv}^F$, where $P$ is a random permutation oracle and $F$ is a random function oracle. However, the only way the adversary can tell something is a random function and not a random permutation is if a collision occurs: namely, $\mathsf{Adv}$ gets the same output for two different inputs. But the Birthday Paradox discussed above tells us that the probability of such a collision is negligible as long as $\mathsf{Adv}$ can only make polynomially many calls to its oracle (which is true since $\mathsf{Adv}$ is $\mathsf{PPT}$). Therefore, random permutations are indistinguishable from random functions, completing the proof. $\qquad \square$

Next, we need to show the more difficult construction, $\mathsf{PRPs}$ from $\mathsf{PRFs}$. In order to do this, we will take a brief detour to define something known as a Feistel Network.

## 3.1 Feistel Networks

The Feistel Network is a way of constructing an efficient, invertible permutation which can be neatly structured in "rounds". Feistel Networks generally consist of multiple rounds of a Feistel transformation, which we now define.

DEFINITION 3 [Feistel Transform] Let $f : \{0,1\}^k \to \{0,1\}^k$ be any efficient function. The *Feistel Transform* of $f$ is a permutation $H_f : \{0,1\}^{2k} \to \{0,1\}^{2k}$, given by: (below $L$ and $R$ are of length $k$ each)

$$H_f(L, R) = (R, \ f(R) \oplus L)$$

We sometimes write $H_f(L, R) = (L', R')$ with $L' = R$ and $R' = f(R) \oplus L$ as a shorthand.
$\diamondsuit$

In the following, we will always use $f$ which is a (pseudo)random function. In this case, the permutation $H_f$ essentially takes two inputs of $k$ bits, and outputs the right input in the clear (but on the left hand side of the output) and then uses the given function $f$ of the right input to "encrypt" the left input (the result of the encryption is on the right side of the output). Indeed, $(R, f(R) \oplus L)$ could be viewed as encryption of $L$ when $R$ is chosen at random and $f$ is a shared $\mathsf{PRF}$. To summarize, the right side is somehow "mangled" and used to encrypt the left side, and then the two halves are swapped.

**Remark 3** *The Feistel Transform is indeed a permutation (even though $f$ need not be!), as can be seen by choosing a fixed $R$ first. It can be seen that the output $L' = R$ is the identity permutation of the right input $R$, and the output $R' = f(R) \oplus L$ for a fixed $R$ has become a permutation of the input $L$. Thus there is a unique output pair $(L', R')$ for each input pair $(L, R)$. More specifically, its inverse is given by:*

$$H_f^{-1}(L', R') = (f(L') \oplus R', \ L')$$

*This computation is strikingly similar to the original Feistel Transform, and in fact it is a "mirror image" of the Feistel Transform.*

Here are graphical representations for the definition of the Feistel transform and its inverse:

These picture suggest the possibility of "stacking" one Feistel Transform on top of another. In fact, the Feistel transform was originally designed to be efficient (computationally)
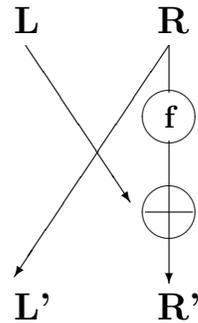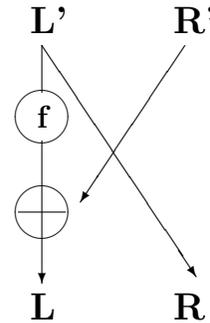
Figure 1: The Feistel Transform and Inverse Feistel Transform

when used in "multiple rounds", and this is where the power of the Feistel transform lies. Repeated applications of the Feistel transform constitute what is known as a Feistel Network. To visualize a Feistel Network, and the corresponding Inverse Feistel Network, one need only stack copies of the above drawings so that the outputs of one stage become the inputs to the next. There is one very important point that must be made: even though one can use the same $f$ from round to round, we will see that we should use *different* (pseudo)random $f$'s in different rounds. Of course, in this case to invert the Feistel Network, it is important to reverse the ordering of $f$'s as well.

DEFINITION 4 [Feistel Network] Given $f_1 \ldots f_n$, the corresponding Feistel Network, $H_{(f_1, f_2, \ldots, f_n)}$, is given by:

$$H_{(f_1, f_2, \ldots, f_n)} = H_{f_n}(H_{f_{n-1}}(\ldots (H_{f_1}(L, R)) \ldots))$$

The operation performed by the Feistel Network is invertible, and the inverse may be obtained by the following (Inverse Feistel Network):

$$H_{(f_1, f_2, \ldots, f_n)}^{-1} = H_{f_1}^{-1}(\ldots (H_{f_{n-1}}^{-1}(H_{f_n}^{-1}(L', R'))\ldots)$$

$\Diamond$

These Feistel Networks are used heavily in real world cryptographic schemes (perhaps most notably, the *Data Encryption Standard* (DES)). It is difficult to prove anything about these Feistel Networks in their most general form, however, and the construction of many cryptographic schemes that use Feistel Networks is considered a "black art", since no one can really directly prove that it works. However, we *can* prove some *formal* results, by assuming that the functions $f_i$ used in the Feistel Network are independently drawn from a family $\mathcal{F}$ of PRFs! We now define a notation for such PRF Feistel Networks.

DEFINITION 5 [PRF Feistel Network] Let $\mathcal{F}$ be a PRF family whose functions map $k$ bits to $k$ bits (i.e. $\ell(k) = L(k) = k$). Let

$$\mathcal{H}_{\mathcal{F}}^n = \{H_{(f_1, f_2, \ldots, f_n)} \mid f_1, f_2, \ldots, f_n \in \mathcal{F}\}$$

We denote by $H_{\mathcal{F}}^n$ a random member of $\mathcal{H}_{\mathcal{F}}^n$, i.e. $H_{(f_1, f_2, \ldots, f_n)}$, where $f_1, f_2, \ldots, f_n$ are random *independently chosen* members of $\mathcal{F}$. That is, each round of the Feistel Transform uses a different, randomly chosen member of $\mathcal{F}$. $\diamondsuit$

Now that we have these definitions in hand, we can proceed to produce PRPs from PRFs by something known as the Luby-Rackoff Construction, which makes use of these PRF Feistel Networks.

## 3.2  The Luby-Rackoff Construction

**Theorem 4** (PRF $\Rightarrow$ **(Efficient)** PRP) *Let* $\mathcal{F}$ *be a* PRF *family whose functions map* $k$ *bits to* $k$ *bits (i.e.* $\ell(k) = L(k) = k$). *Then* $\mathcal{G} = \mathcal{H}_{\mathcal{F}}^3$ *is an efficient* PRP *family. Namely,* 3 *rounds of the Feistel Network yield an efficient* PRP.

**Proof:** The proof is by hybrid arguement over a sequence of worlds, informally shown in the table below. The goal is to start with a World 0 in which we have only a PRF, and through a sequence of intermediate worlds that are each indistinguishable from the previous world, construct a World 5 that is just the desire random permutation on $2k$ bits.

| World | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Input | $(L, R)$ | $(L, R)$ | $(L, R)$ | $(L, R)$ | $(L, R)$ | $(L, R)$ |
| Inside | $f_1, f_2, f_3 \in \mathcal{F}$ | $f_1, f_2, f_3$ are truly random | $H_{f_1}(L, R)$ $\wr\wr$ $(R, \tilde{?})$ | $H_{f_1}(L, R)$ $\downarrow$ $H_{f_2}(R, \tilde{?})$ $\wr\wr$ $(\tilde{?}, \$)$ | $H_{f_1}(L, R)$ $\downarrow$ $H_{f_2}(R, \tilde{?})$ $\downarrow$ $H_{f_3}(\tilde{?}, \$)$ $\wr\wr$ | $P$ – random permutation |
| Output | $H_{f_1, f_2, f_3}(L, R)$ | $H_{f_1, f_2, f_3}(L, R)$ | $H_{f_2, f_3}(R, \tilde{?})$ | $H_{f_3}(\tilde{?}, \$)$ | $(\$, \$)$ | $P(L, R)$ |

Figure 2: Illustration of Luby-Rackoff Proof

The following *informal*, but informative, notation is used in the table. Do not worry if it is confusing for now. $(L, R)$ is used to represent a pair of inputs that are selected by the adversary Adv as the input to the oracle that he has. Clearly, many such pairs will be selected during the course of execution (wlog, we assume all the oracle calls are made with distinct inputs $(L, R)$). We use ? to represent some string is in under direct or indirect control of the adversary (in particular, the adversary might know part or all of it). We use $\tilde{?}$ to denote a value ? which is always distinct from one oracle call to another. Namely, the adversary has partial knowledge or control over selecting this value, but w.h.p. the adversary cannot cause a collision among different such values. Finally, we use $\$$ to denote a truly random $k$-bit string. Namely, from one oracle call to the other, this string is selected randomly and independently from previous calls.

As the first step, we look at the "real" World 0, where the oracle is $H_{f_1, f_2, f_3}$, where $f_1, f_2, f_3$ are PRF's. Recall from the previous lecture the discussion of "The Random Function Model". Using this model, we know that World 0 is indistinguishable from World 1,

where $f_1, f_2, f_3$ are truly random functions (technically, we have to use three hybrid arguments eliminating one $f_i$ every time, but this is simple). Jumping ahead, World 4 will be a truly random *function* from $2k$ to $2k$ bits. By Theorem 3, we knows that a truly random *function* is indistinguishable from a truly random *permutation*, i.e. World 4 is indistinguishable from World 5.

To summarize, the heart of the proof is to show that World 1 is indistinguishable from World 4: namely, $H_{(f_1,f_2,f_3)}$, where $f_1, f_2, f_3$ are random functions, is indistinguishable from a truly random function from $2k$ to $2k$ bits. Here is the intuition. We want to say that as the input $(L, R)$ travels through 3 layers of the Feistel network, the following changes happen:

$$(L, R) \xrightarrow{H_{f_1}} (L_1, R_1) = (R, \tilde{?}) \xrightarrow{H_{f_2}} (L_2, R_2) = (\tilde{?}, \$) \xrightarrow{H_{f_3}} (L_3, R_3) = (\$, \$)$$

Thus, the first layer only achieves that the right half $R_1$ never repeats from one oracle call to the other, but Adv may know a lot about both the left and the right parts. The second layer uses this to produce a string whose right half $R_2$ always looks random and independent from everything else, while the left half $L_2 = R_1$ may still be very predictable. Finally, the third layer uses the the randomness and unpredictability of $R_2$ to make both $L_3 = R_2$ and $R_3$ look totally random, as desired.

We now formalize the above intuition. World 2 is the same as World 1 with the following exception. If any two right halves $R_1$ happens to collide during the oracle calls, we stop the execution and tell Adv that a right-half collision happened. Thus, to show that Adv does not see the difference between World 1 and World 2, we must argue that the probability of collision is negligible. Assume Adv makes $t$ calls to the its oracle. Take any two such calls with inputs $(L, R) \neq (L', R')$. If $R = R'$, we must have $L \neq L'$, and hence

$$R_1 = L \oplus f_1(R) = L \oplus f_1(R') \neq L' \oplus f_1(R') = R_1'$$

i.e. the right half collision cannot happen then. On the other hand, assume $R \neq R'$. In order for $R_1 = R_1'$, we must have $L \oplus L' = f_1(R) \oplus f_1(R')$. Irrespective of the value $Z_\ell = L \oplus L'$, since $f_1$ is a truly random function and $R \neq R'$, $Z_r = f_1(R) \oplus f_1(R')$ is a truly random string, so the probability that $Z_\ell = Z_r$ is $1/2^k$. Hence, in both cases, the probability that $R_1 = R_1'$ is at most $1/2^k$. Since there are $t^2$ pairs of inputs, the probability that *any two of them* will yield a collision in the right half is at most $t^2/2^k$, which is negligible (since $t$ is polynomial). To summarize, World 1 and World 2 are indeed indistinguishable.

We now go one more level down. World 3 is the same as level 2, except the value $R_2$, instead of being equal to $f_2(R_1) \oplus L_1$, is always chosen random. However, this *does not change the experiment at all*: namely, World 2 and World 3 are *the same*. Indeed, since World 2 only runs when no two values $R_1$ are the same, all $t$ inputs to $f_2$ are distinct. And since $f_2$ is a truly random function, all the values $f_2(R_1)$ are random and independent from each other. But, then irrespective of which values of $L_1$ are used, all $R_2 = L_1 \oplus f_2(R_1)$ are random and independent as well.

Next, we go to World 4. It is the same as world 3 up to the last round of the Feistel. There, it sets $L_3 = R_2$, as it should, but selects a truly random $R_3$ instead of expected $f_3(R_2) \oplus L_2$. First, since $R_2$ was completely random in World 3, $L_3 = R_2$ is completely

random as well. Next, by birthday paradox analysis, the probability that any two values $R_2$ (which are randomly selected) collide, is at most $t^2/2^k$, which is negligible. Thus, with overwhelming probability all the values $R_2$ are distinct. But then, since $f_3$ is a random function, by similar argument to the previous paragraph we get that all the values $f_3(R_2)$ are random and independent, and thus, so are $R_3 = f_2(R_2) \oplus L_2$. To summarize, with all but negligible probability World 3 and World 4 are the same as well.

Finally, notice that World 4 is a truly random function provided no right half collision happens on the first layer (in which case we stop the experiment). But the latter has negligible chance as we know, so World 4 is indeed indistinguishable from a random function. This completes the proof. □

We conclude our discussion with another useful theorem, which will not be proven here.

**Theorem 5** (PRF$\Rightarrow$ **Strong** PRP) *Let $\mathcal{F}$ be a* PRF *family whose functions map $k$ bits to $k$ bits (i.e. $\ell(k) = L(k) = k$). Then $\mathcal{G} = \mathcal{H}_{\mathcal{F}}^4$ is an efficient* strong PRP *family. Namely, 4 rounds of the Feistel Network yield an efficient strong* PRP.

# 4  USING PRP'S FOR SECRET-KEY ENCRYPTION

## 4.1  Several Schemes Using PRP's

In these section we propose several schemes (called ciphers) using PRP's. In all of them, let $\mathcal{G} = \{g_s\}$ be an efficient PRP family operating on the appropriate input length. The seed $s$ will always be part of the shared secret key.

**Electronic Code Book (ECB) Cipher.**  This is the naive suggestion we started from:

$$c = E_s(m) = g_s(m); \qquad m = D_s(c) = g_s^{-1}(m)$$

We noted that ECB cipher is length-preserving and very efficient. However, it is obviously not CPA-secure, since the encryption of a message is completely deterministic (e.g. the adversary can tell if you send the same message twice). On the positive side, it is clearly one-message secure.

Next, we examine two obvious schemes resulting when we treat our PRP family as a PRF family (which is justified by Theorem 3).

**Counter (CNT) Cipher.**  This is a stateful deterministic encryption. Players keep the counter value cnt which they increment after every encryption/decryption.

$$c = E_s(m) = g_s(\mathsf{cnt}) \oplus m; \qquad m = D_s(c) = g_s(\mathsf{cnt}) \oplus c$$

The CPA-security of this scheme was shown when we talked about PRF's.

**XOR (aka R-CNT) Cipher.** This is a stateless probabilistic encryption.

$$(r, c) \leftarrow E_s(m) = (r, g_s(r) \oplus m); \qquad m = D_s(r, c) = g_s(r) \oplus c$$

where $r$ is chosen at random per each encryption. The CPA-security of this scheme was shown when we talked about PRF's.

Next, we give several ciphers which really use the inversion power of PRP's. We start with what we call "nonce" ciphers, but first make a useful digression. The term *nonce* means something that should be unique (but possibly known to the adversary). For example, the counter and XOR schemes above effectively used a nonce to encrypt $m$ via $c = g_s(R) \oplus m$. Indeed, to analyze the security of the above schemes, we only cared that all the $R$'s are *distinct*. For the counter scheme we ensured it explicitly by using the state cnt, while for the XOR scheme we selected $R$ at random from a large enough domain, and argued the uniqueness with very probability (using the birthday paradox analysis). Similarly, the nonce ciphers below use nonce $R$ as follows ($\circ$ denotes a clearly marked concatenation):

$$c \leftarrow E_s(m) = g_s(m \circ R); \qquad D_s(c): \text{ get } m \circ R = g_s^{-1}(c), \text{ output } m$$

Assuming all the nonces $R$ are unique, the CPA-security of this "scheme" is easy. Depending whether we generate nonces using counters+state or at random, we get:

**C-Nonce Cipher.** This is a stateful deterministic encryption. Players keep the counter value cnt which they increment after every encryption/decryption.

$$c \leftarrow E_s(m) = g_s(m \circ \text{cnt}); \qquad D_s(c): \text{ get } m \circ \text{cnt} = g_s^{-1}(c), \text{ output } m$$

**R-Nonce Cipher.** This is a stateless probabilistic encryption.

$$c \leftarrow E_s(m) = g_s(m \circ r); \qquad D_s(c): \text{ get } m \circ r = g_s^{-1}(c), \text{ output } m$$

where $r$ is chosen at random per each encryption.

**Random IV (R-IV) Cipher.** It operates as follows:

$$(r, c) \leftarrow E_s(m) = (r, g_s(r \oplus m)); \qquad m = D_s(r, c) = g_s^{-1}(c) \oplus r$$

where $r$ is chosen at random per each encryption. $r$ is sometime s called the "Initialization Vector" (IV). Notice the similarity with the R-CNT cipher: R-CNT sets $c = g_s(r) \oplus m$, while R-IV sets $c = g_s(r \oplus m)$. Not surprisingly, the proof for CPA-security of R-IV is also very similar to R-CNT, with a slight extra trick.

**Proof Sketch:** Rather than arguing that all the $r$'s are distinct w.h.p., as we did for R-CNT, we argue that all $(r \oplus m)$'s are distinct w.h.p. Indeed, irrespective of how the adversary chooses the message $m$ to be encrypted, $r$ is chosen at random, so $(r \oplus m)$ is random and independent from everything else. Hence, by birthday paradox analysis w.h.p. all $(r \oplus m)$'s are distinct, so our PRP (modeled as a truly random *function*, not permutation, for the analysis; see Theorem 3 for justification) operates on different inputs, so all the values $g_s(r \oplus m)$ are random and independent from each other. Thus, in particular, w.h.p. the challenge ciphertext is a random string independent from the bit $b$ used to select $m_0$ or $m_1$ to be encrypted. □

**Remark 4** *Notice, R-IV scheme does* not *work with counters instead of random strings. The corresponding insecure scheme C-IV would be the following (where* cnt *is incremented after each encryption/decryption):*

$$c \leftarrow E_s(m) = g_s(\text{cnt} \oplus m); \qquad\qquad m = D_s(c) = g_s^{-1}(c) \oplus \text{cnt}$$

*Looking at the analysis above, try to see what goes wrong!*

## 4.2   Block Ciphers and Modes of Operation

In the above schemes we assumed that we can choose a PRP whose input length is roughly the same as the length of the message. Hence, the longer is the message, the longer should PRP's input be. In practice, this is quite inconvenient. Instead, people usually design *fixed length* $\ell$ PRP's (say, $\ell = 128$ bits), just long enough to avoid brute force attacks. Then to encrypt a long message $m$, $m$ is split in to *blocks* $m_1 \ldots m_n$, each of *block length* $\ell$, and the PRP is somehow used to encrypt these $n$ blocks. For this reason, efficient PRP's with fixed (but large enough) block length are called *block ciphers.* The block cipher itself is assumed (or modeled) to be an efficient PRP. However, the main question is how to really use this block cipher to "combine together" the $n$ message blocks $m_1 \ldots m_n$. Each specific such method is called *mode of operation.* As we will see, there are many modes of operation for block cipher — some secure and some not. The main goal of a "good" mode, aside from being secure, is to minimize the number of extra blocks output. Usually, at most one extra block/value is considered "efficient", but there are many other important criteria as well.

We now go through the previous "single" block schemes, and see how they yield a corresponding mode of operation.

**Electronic Code Book (ECB) Mode.**   Simply apply the ECB cipher block by block: $c_i = g_s(m_i)$, for all $i$. Decryption is obvious. This mode is extremely efficient, length-preserving, but totally insecure.

**Counter (C-CTR) Mode.**   Simply apply the C-CTR cipher viewing $m_1 \ldots m_n$ as "separate messages". Since C-CTR is CPA-secure, we know that the resulting scheme is CPA-secure as well: $c_i = g_s(\text{cnt} + i - 1) \oplus m_i$. At the end, update $\text{ctr} = \text{ctr} + n$. Here and everywhere, the addition is modulo $2^\ell$. Notice, the encryption is length-preserving, but keeps state as the penalty. Decryption is obvious. Notice, this works (even better) with PRF's, and not only PRP's. Also, the scheme is parallelizable both for encryption and for decryption.

The next three modes are motivated by the XOR (or R-CTR) cipher. Recall, it returns $(r, g_s(r) \oplus m)$. A straightforward way to iterate this cipher is to choose a brand new $r_i$ for every block $m_i$. However, this requires to send all the $r_i$'s, which doubles the length of the ciphertext.

**Random Counter (R-CTR) Mode.**   This modes "combines" the idea of C-CTR and R-CTR modes. As a result, it avoids state, but yields only one "extra block". Encryption picks a random $r$ for every message, and sets $c_i = g_s(r+i-1) \oplus m_i$. It outputs $(r, c_1, \ldots, c_n)$,

Decryption is obvious: $m_i = g_s(r + i - 1) \oplus c_i$. A similar analysis to R-CTR cipher shows that with high probability, all $n$-tuples $(r, r+1, \ldots, r+n-1)$ do not overlap, so no collision occurs. Notice, this works (even better) with PRF's, and not only PRP's. Also, the scheme is parallelizable both for encryption and for decryption.

**Cipher Feedback (CFB) Mode.** This mode uses a different idea to get "fresh" randomness. After selecting an initial IV $r$, it sets $c_1 = g_s(r) \oplus m_1$, and then uses $c_1$ itself as the next $r$! Namely, $c_2 = g_s(c_1) \oplus m_2$. Intuitively, since $g_s$ looks like a random function (by Theorem 3), $c_1$ indeed "looks random" and independent from the original $r$, so we can use it to encrypt $m_2$. And so on. To summarize, set $c_0 = r$ and $c_i = g_s(c_{i-1}) \oplus m_i$ and output $(c_0, c_1 \ldots c_n)$. To decrypt, recover $m_i = g_s(c_{i-1}) \oplus c_i$. Notice, this works (even better) with PRF's, and not only PRP's. Also, the scheme is parallelizable for decryption, but not for encryption.
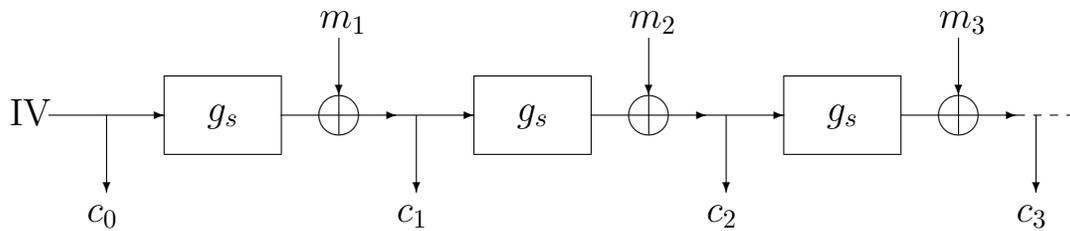


Figure 3: Cipher Feedback Mode

**Output-Feedback (OFB) Mode.** This mode also uses the block cipher itself to refresh randomness. Specifically, it picks a random IV $r$, and sets $r_0 = c_0 = r$. then, for each $i$, is sets "one-time pad" $r_i = g_s(r_{i-1})$ and computes $c_i = r_i \oplus m_i = g_s(r_{i-1}) \oplus m_i$ (the latter could also be viewed as the XOR cipher using $r_{i-1}$ as the IV). It outputs $(c_0, c_1 \ldots c_n)$. To decrypt, compute the $r_i$'s from $r_0 = c_0$ (via $r_i = g_s(r_{i-1})$) and output $m_i = c_i \oplus r_i = c_i \oplus g_s(r_{i-1})$. In other words, the initial IV $r$ defines a sequence of "independently looking" one-time pads: $r_1 = g_s(r)$, $r_2 = g_s(g_s(r))$, and so on: each $r_i$ is used to encrypt $m_i$. Notice, this works (even better) with PRF's, and not only PRP's. Also, the scheme is not parallelizable for either encryption, or decryption.

**Nonce Modes.** Those could be defined, both for the counter C-Nonce and the random R-Nonce versions, but are never used. The reason is that the overall size of encryption is by a constant fraction (rather than by one block length) larger than the length of the message, since a nonce should be used for each block. Notice, btw, *all* nonces have to be distinct.

**Cipher Block Chaining (CBC) Mode.** Finally, we define the last mode originating from the R-IV cipher. It is the only one so far that uses the inversion power of block ciphers. Recall, R-IV returns $(r, c)$, where $c = g_s(r \oplus m)$. A straightforward way to iterate this scheme is to use a different $r_i$ for each $m_i$ (check why one $r$ does not suffice!). Again, this would double the length of the encryption. Similarly to the CFB mode above, the
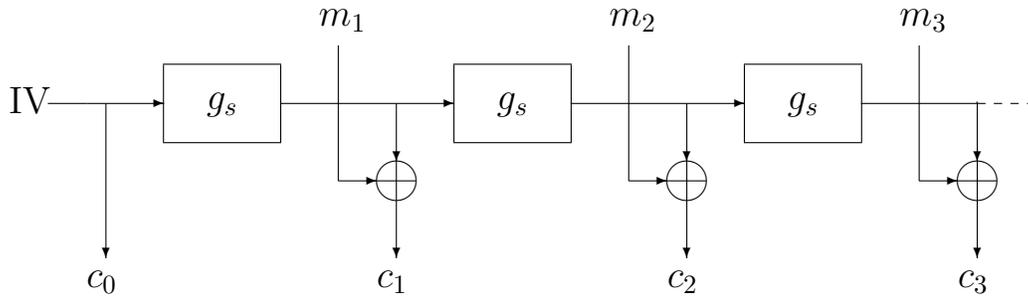
Figure 4: Output Feedback Mode

CBC mode uses the *previous cipher blocks* themselves as the new $r_i$'s, but uses those in R-IV rather than R-CTR! Specifically, as before it picks a random IV $c_0 = r$, and sets $c_1 = g_s(c_0 \oplus m_1)$. Then, it simply uses $c_1$ as the next IV: $c_2 = g_s(c_1 \oplus m_i)$. Then it uses $c_2$ as the next IV, and so on. To summarize, set $c_0 = r$, $c_i = g_s(c_{i-1} \oplus m_i)$ and output $(c_0, c_1 \ldots c_n)$. To decrypt, recover $m_i = g_s^{-1}(c_i) \oplus c_{i-1}$. Notice, the scheme is parallelizable for decryption, but not for encryption.
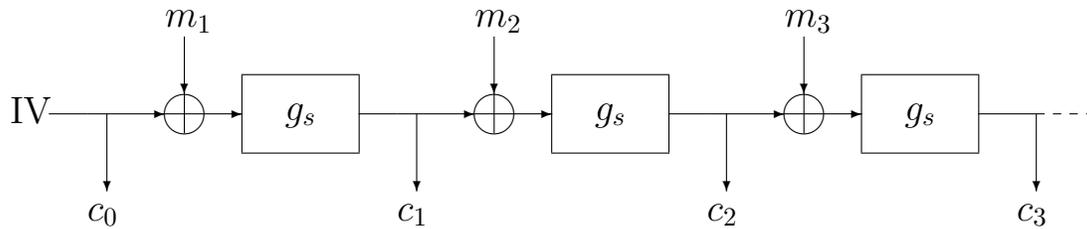


Figure 5: Cipher Block Chaining Mode

To summarize, C-CTR, R-CTR, CFB, OFB, CBC modes are all CPA-secure. For mysterious reason, CBC seems to be the most pervasively used mode in practice, despite being one of the lesser simple and secure of the above.

**Remark 5** *One can also define stateful variants of CFB, OFB and CBC modes, denoted C-CFB, C-OFB and C-CBC, where the initialization vector IV is set to be a counter instead of being chosen fresh for each encryption. It turns out that the modes C-CFB and C-OFB are still CPA-secure, which C-CBC is not! Think why this is the case (also read the next section).*