G22.3033-003 Introduction to Cryptography November 8^{th}, 2001

Lecture 10

Lecturer: Yevgeniy Dodis *Scribe: Shabsi Walfish*

Last lecture we introduced PRFs, along with a few applications. In particular, we presented a stateful encryption (the CTR scheme) as well as a stateless encryption (the XOR scheme). In this lecture, we consider a new paradigm using families of Pseudo Random Permutations (PRPs). First we will define several kinds PRPs, and suggest some possible applications. A device known as a Feistel Network is introduced, and we then use this device to construct PRPs from PRFs (this is known as the Luby-Rackoff construction). As a result of this construction, it is easy to show that existence of PRFs is equivalent to existence of PRPs. Next, we discuss block ciphers, and present several modes of operations for block ciphers, including the Cipher Block Chaining (CBC) mode. We then prove the security of CBC and other modes, and discuss practical applications. One of such applications is secure integration of public- and secret-key encryption schemes.

1 Pseudo Random Permutation Family

1.1 Introduction

In the last lecture, we introduced the concept of a family of pseudo random functions, and showed some potential applications. Now, we would like to try and develop a new paradigm for encryption by adding some new restrictions to these pseudo random function families in order to produce families of Pseudo Random Permutations (PRP). This concept 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. We now make several definitions.

1.2 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 efficiently computable,* $\forall s$.
2. g_s *is a permutation,* $\forall s$.
3. g_s *is pseudo random:* $\forall$ PPT $\mathcal{A}$

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

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

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 $\mathcal{A}$ 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:

$$\forall \text{ PPT } \mathcal{A} \qquad \mathcal{A}^{g_s} \approx \mathcal{A}^{P}$$

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.

We now wish to provide two extensions to the previous definition that will provide us with useful properties.

Definition 2 (Efficient PRP) *An Efficient* PRP *satisfies all the properties of a family of Pseudo Random Permutations, with the following additional constraint:*

4. g_s^{-1} *is efficiently computable*

Finally, we strengthen the definition one more time.

Definition 3 (Strong PRP) *A Strong [Efficient]* PRP *satisfies all the properties of a family of [Efficient] Pseudo Random Permutations, with the following additional constraint:*

5. g_s *is pseudo random even if oracle access to* g_s^{-1} *is provided as well:*

$$\mathcal{A}^{(g_s,\ g_s^{-1})} \approx \mathcal{A}^{(P,\ P^{-1})}$$

This new definition provides us with an additional measure of security by allowing us to give $\mathcal{A}$ oracle access to g_s^{-1}. We note that it makes no sense to talk about a Strong PRP that is not also an Efficient PRP, because we cannot make use of the additional security on g_s^{-1} if we cannot compute it in PPT.

1.3 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 PK-only-secure, 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.

2 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 4 (PRP $\Rightarrow$ PRF) *Assume $\mathcal{G}$ is a* PRP *family. Then $\mathcal{G}$ is also a* PRF *family.*

Proof: The proof that this works is simple, by hybrid arguement. If we start by considering that members of $\mathcal{G}$ are indistinguishable from random permutations to the adversary $\mathcal{A}$, all we need to show is that a random permutation is indistinguishable from a random function. In other words, we must show $\mathcal{A}^P \approx \mathcal{A}^F$, 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 (the same output occurs for two different inputs). The Birthday Paradox problem (discussed in the previous lecture) tells us that the probability of such a collision is negligible as long as $\mathcal{A}$ can only make polynomially many calls to its oracle ($\mathcal{A}$ is PPT). Therefore, random permutations are indistinguishable from random functions, completing the proof. □

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

2.1 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 5 (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)*

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

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

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 PRF. To summarize, the right side is somehow "mangled" and used to encrypt the left side, and then the two halves are swapped.

Remark 6 *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:

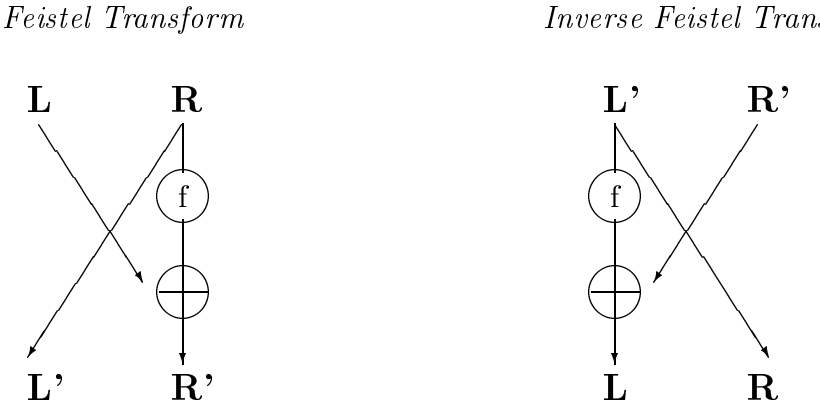

Figure 1: The Feistel Transform and Inverse Feistel Transform

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) 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 7 (Feistel Network) *Given $f_1 \ldots f_n$, the corresponding Feistel Network, $H_{(f_1, f_2, \ldots, f_n)}$, is given by:*

$$H_{(f_1, f_2, \ldots, f_n)} = H_{f_1}(H_{f_2}(\ldots(H_{f_n}(L, R))\ldots))$$

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

$$H^{-1}_{(f_1, f_2, \ldots, f_n)} = H^{-1}_{f_n}(\ldots(H^{-1}_{f_2}(H^{-1}_{f_1}(L', R'))\ldots)$$

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 8 (PRF Feistel Network) *Let $\mathcal{F}$ be a* PRF *family whose functions map k bits to k bits (i.e. $l(k) = L(k) = k$). Let*

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

We denote by $H^n_{\mathcal{F}}$ a random member of $\mathcal{H}^n_{\mathcal{F}}$, 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 $\mathcal{F}$.*

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.

2.2 The Luby-Rackoff Construction

Theorem 9 (PRF $\Rightarrow$ (Efficient) PRP) *Let $\mathcal{F}$ be a* PRF *family whose functions map k bits to k bits (i.e. $l(k) = L(k) = k$). Then $\mathcal{G} = \mathcal{H}^3_{\mathcal{F}}$ 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 $\mathcal{A}$ 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 4, 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 $\mathcal{A}$ 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 $\mathcal{A}$ that a right-half collision happened. Thus, to show that $\mathcal{A}$ does not see the difference between World 1 and World 2, we must argue that the probability of collision is negligible. Assume $\mathcal{A}$ 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 10 (PRF⇒ Strong PRP) *Let $\mathcal{F}$ be a* PRF *family whose functions map k bits to k bits (i.e. $l(k) = L(k) = k$). Then $\mathcal{G} = \mathcal{H}^4_{\mathcal{F}}$ is an efficient* strong PRP *family. Namely,* 4 *rounds of the Feistel Network yield an efficient strong* PRP.

3 Using PRP's for Secret-Key Encryption

3.1 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 (PK-only) secure.

Next, we examine two obvious schemes resulting when we treat our PRP family as a PRF family (which is justified by Theorem 4).

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.

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

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 4 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 11 *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):*

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

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

3.2 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* ℓ PRP's (say, $\ell = 64$ bits), just long enough to avoid brute force attacks. Then to encrypt a long message m, m is split in to *blocks* $m_1 \dots m_n$, each of *block length* ℓ, 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 $m_1 \dots m_n$. 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 $m_1 \dots m_n$ as "separate messages". Since C-CTR is CPA-secure, we know that the resulting scheme is CPA-secure as well: $c_i = g_s(\mathsf{cnt} + i - 1) \oplus m_i$. At the end, update $\mathsf{ctr} = \mathsf{ctr} + n$. Here and everywhere, the addition is modulo 2^ℓ. 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 $(r, c_1, \dots, c_n)$, 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 $(r, r+1, \dots, r+n-1)$ 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 4), 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 $(c_0, c_1 \dots c_n)$. 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.

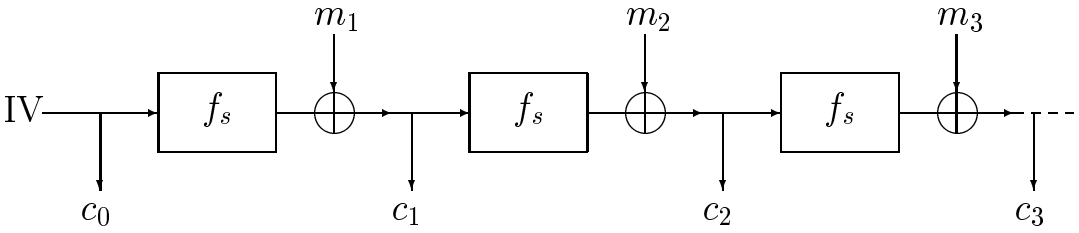

Figure 3: Cipher Feedback Mode

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 $(c_0, c_1 \dots c_n)$. 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.

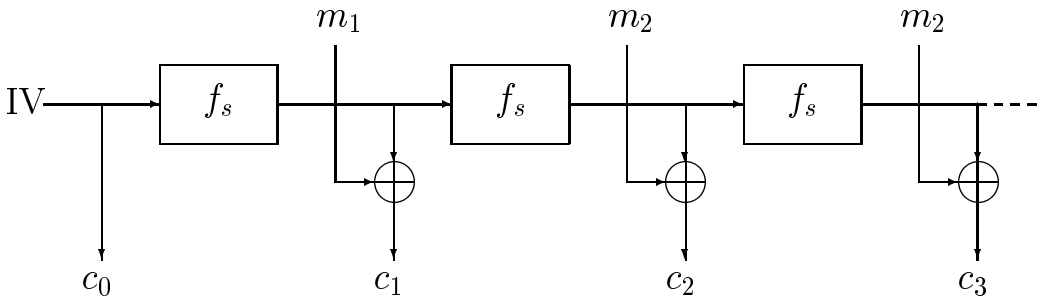

Figure 4: Output Feedback Mode

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 return (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 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.

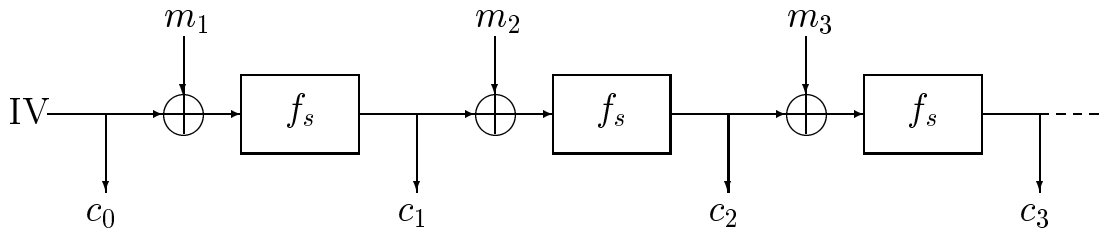

Figure 5: Cipher Block Chaining Mode

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.

3.3 Security of CBC, CFB and OFB Modes

The CPA-security of the last three modes we discussed — CFB, OFB, CBC — is shown by very similar technique. In fact, one "universal proof" suffices. We give a proof sketch here.

First, as usual, the block cipher (i.e., our PRP) g_s is replaced by a truly random function F (this is justified by Theorem 4 by first replacing it by a random permutation, and then replacing the latter by a random function). Now, take any adversary $\mathcal{A}$. Rather than choosing the whole random function F right away for the experiment with $\mathcal{A}$, we select it "as needed" as we move along. Namely, we keep the table of $(input, output)$ pairs that were used so far. When a new input x to F arrives, we use the table if the input was used before, and otherwise add $(x, random)$ to the table, and use $random$ as the output of F. It is easy to see from the description of the CFB, OFB and CBC modes, that as long as we never perform the table look-up, i.e. no input is repeated twice during the run, the advantage of $\mathcal{A}$ is 0. Namely, all $\mathcal{A}$ sees is a bunch of random strings, independent from anything else.

Thus, it suffices to show that the probability an "input collision" happens is negligible. Let B_j denote the ("bad") event that the input collision happened within first j evaluations of F. We want to show that $\Pr(B_T) = \mathsf{negl}(k)$, where T is the total number of calls to F (roughly $T = nt$, where t is the number of encryption queries made by $\mathcal{A}$ and n is the number of blocks per message). We will show by induction on j that

$$\Pr(B_j) \le \frac{(j-1)T}{2^\ell}$$

Indeed, $\Pr(B_1) = 0$. Now, in order for B_j to happen, either there should be a collision among the first $(j-1)$ calls to F (i.e., B_{j-1} should happen), or the j-th input to F collided with one of the previous $(j-1)$ inputs. Formally,

$$\Pr(B_j) \le \Pr(B_{j-1}) + \Pr(B_j \mid \overline{B_{j-1}})$$

Notice, by inductive assumption the first probability is at most $(j-2)T/2^\ell$. On the other hand, from the facts that (1) $\overline{B_{j-1}}$ means that the previous $(j-1)$-st input was distinct from all the earlier inputs; (2) F is the random function, which combined with (1) gives that $(j-1)$-st output is truly random; and (3) the description of OFB/CFB/CBC modes together with (1) and (2) implies that the j-th input is totally random, we conclude that the probability that $random$ j-th input to F collides with the previous $(j-1)$ inputs is at most $\frac{j-1}{2^\ell} \le \frac{T}{2^\ell}$. Combining these,

$$\Pr(B_j) \le \frac{(j-2)T}{2^\ell} + \frac{T}{2^\ell} = \frac{(j-1)T}{2^\ell}$$

But now we see that $\Pr(B_T) \le T^2/2^\ell$, which is indeed negligible since T is polynomial. Namely, w.h.p. no input collisions happen, and all the adversary simply sees is a bunch of truly random strings, giving him no way to predict the challenge b.

Remark 12 *Notice, how more subtle the proof becomes for these modes. In some sense, it is surprising that much simpler C-CTR and R-CTR modes are not used more often: unlike OFB,CFB and CBC modes, they are easily parallelizable for both encryption and decryption; unlike the CBC mode they work with a* PRF *and not only a* PRP*; finally, they give comparable (or even better in case of C-CTR) security guarantees.*

4 Secret-Key Encryption in Practice

In practice, specific efficient block and stream ciphers are used. These practical schemes usually follow some high level guidelines we saw in our formal study. However, the design of block and stream ciphers is usually viewed as "black art". We name a few such ciphers used in practice, and refer you to outside sources to learn more about those (i.e., see "Applied Cryptography" by Bruce Schneier).

- Stream ciphers: RC4, SEAL, WAKE, A5, Hughes XPD/KPD, and many-many others.
- Block Ciphers: DES, AES (Rijndael), Lucifer, FEAL, IDEA, RC2, RC6, Blowfish, Serpent, and many-many-many-many others.

We notice that block ciphers are used much more commonly. Most of them (with a notable exception of AES) use Feistel Network as part of their design, so Feistel transform is very useful. The current standard is AES — advanced encryption standard, which recently replaced DES — old data encryption standard. Also, using any stream cipher, or a block cipher with a good mode of operation, the size of the ciphertext is (almost) equal to the size of the plaintext, which is very efficient.

5 Public-Key Encryption in Practice

In general, secret-key (or symmetric) encryption is very efficient (even the theoretical ones we constructed using Feistel network, etc.). On the contrary, public-key (or asymmetric) encryption is much less efficient, especially for large messages (aside from efficiency issues, the size of the ciphertext is usually much larger than the plaintext as well). On the other hand, a major drawback to symmetric key schemes is that they require that you share a secret with the intended recipient of your communication. On the contrary, public key encryption does *not* require you to share any secret knowledge with the recipient.

So can we have a relatively efficient public-key encryption, especially for long messages? The answer is "yes". And the technique is to combine the "slow" ordinary public-key encryption E' with the fast secret-key encryption Enc. This folklore technique is known as integration of public- and secret-key schemes. If auxiliary E' has public key PK and secret key SK, we define new public-key encryption E, which has the same public/secret keys, as follows:

$$E_{PK}(m) = (E'_{PK}(s), \mathsf{Enc}_s(m)), \qquad \text{where } s \leftarrow_R \{0,1\}^k$$
$$D_{SK}(c_1, c_2): \quad \text{get } s = D'_{SK}(c_1); \quad \text{output } m = \mathsf{Dec}_s(c_2)$$

That is, we encrypt a short, randomly chosen secret using s a public key scheme, and then transmit a stream or block cipher encryption of the message using the random secret. The recipient first decrypts the random secret, then uses that to decrypt the message. Provided that E'_{PK} and Enc are CPA-secure, the new scheme is also CPA-secure (in fact, PK-only, i.e. one-message, security suffices for the symmetric scheme Enc). The proof is left as an exercise.