Chapter 3

Pseudorandom Functions

Pseudorandom functions (PRFs) and their cousins, pseudorandom permutations (PRPs), figure as central tools in the design of protocols, especially those for shared-key cryptography. At one level, PRFs and PRPs can be used to model blockciphers, and they thereby enable the security analysis of protocols based on blockciphers. But PRFs and PRPs are also a useful conceptual starting point in contexts where blockciphers don't quite fit the bill because of their fixed block-length. So in this chapter we will introduce PRFs and PRPs and investigate their basic properties.

3.1 Function families

A *function family* is a map $F\colon\ \mathcal{K} \times D \to R$. Here $\mathcal{K}$ is the set of keys of F and D is the domain of F and R is the range of F. The set of keys and the range are finite, and all of the sets are nonempty. The two-input function F takes a key K and an input X to return a point Y we denote by $F(K, X)$. For any key $K \in \mathcal{K}$ we define the map $F_K\colon\ D \to R$ by $F_K(X) = F(K, Y)$. We call the function F_K an *instance* of function family F. Thus F specifies a collection of maps, one for each key. That's why we call F a function *family* or *family of functions*.

Sometimes we write $\mathsf{Keys}(F)$ for $\mathcal{K}$, $\mathsf{Dom}(F)$ for D, and $\mathsf{Range}(F)$ for R. Usually $\mathcal{K} = \{0,1\}^k$ for some integer k, the *key length*. Often $D = \{0,1\}^\ell$ for some integer ℓ called the *input length*, and $R = \{0,1\}^L$ for some integers L called the *output length*. But sometimes the domain or range could be sets containing strings of varying lengths.

There is some probability distribution on the (finite) set of keys $\mathcal{K}$. Unless otherwise indicated, this distribution will be the uniform one. We denote by $K \xleftarrow{\$} \mathcal{K}$ the operation of selecting a random string from $\mathcal{K}$ and naming it K. We denote by $f \xleftarrow{\$} F$ the operation: $K \xleftarrow{\$} \mathcal{K};\ f \leftarrow F_K$. In other words, let f be the function F_K where K is a randomly chosen key. We are interested in the input-output behavior of this randomly chosen instance of the family.

A *permutation* is a bijection (i.e. a one-to-one onto map) whose domain and range are the same set. That is, a map $\pi\colon\ D \to D$ is a permutation if for every $y \in D$ there is exactly one $x \in D$ such that $\pi(x) = y$. We say that F is a family of permutations if $\mathsf{Dom}(F) = \mathsf{Range}(F)$ and each F_K is a permutation on this common set.

Example 3.1.1 A blockcipher is a family of permutations. In particular DES is a family of permutations $\mathsf{DES}\colon\ \mathcal{K} \times D \to R$ with

$$\mathcal{K} = \{0,1\}^{56} \ \text{ and } \ D = \{0,1\}^{64} \ \text{ and } \ R = \{0,1\}^{64}\ .$$

Here the key length is $k = 56$ and the input length and output length are $\ell = L = 64$. Similarly AES (when "AES" refers to "AES128") is a family of permutations $\mathsf{AES}\colon \mathcal{K} \times D \to R$ with

$$\mathcal{K} = \{0,1\}^{128} \quad \text{and} \quad D = \{0,1\}^{128} \quad \text{and} \quad R = \{0,1\}^{128} .$$

Here the key length is $k = 128$ and the input length and output length are $\ell = L = 128$. ▌

3.2 Games

We will use code-based games [1] in definitions and some proofs. We recall some background here. A game —see Fig. 3.1 for an example— has an **Initialize** procedure, procedures to respond to adversary oracle queries, and a **Finalize** procedure. A game G is executed with an adversary A as follows. First, **Initialize** executes and its outputs are the inputs to A. Then, A executes, its oracle queries being answered by the corresponding procedures of G. When A terminates, its output becomes the input to the **Finalize** procedure. The output of the latter, denoted G^A, is called the output of the game, and we let "$G^A \Rightarrow y$" denote the event that this game output takes value y. Variables not explicitly initialized or assigned are assumed to have value $\perp$, except for booleans which are assumed initialized to false. Games G_i, G_j are *identical until* bad if their code differs only in statements that follow the setting of the boolean flag bad to true. The following is the Fundamental Lemmas of game-playing:

Lemma 3.2.1 [1] Let G_i, G_j be identical until bad games, and A an adversary. Let BAD_i (resp. BAD_j) denote the event that the execution of G_i (resp. G_j) with A sets bad. Then

$$\Pr\left[G_i^A \wedge \mathsf{BAD}_i\right] = \Pr\left[G_j^A \wedge \mathsf{BAD}_j\right] \text{ and } \Pr\left[G_i^A\right] - \Pr\left[G_j^A\right] \leq \Pr\left[\mathsf{BAD}_j\right] .$$

When the **Finalize** is absent, it is understood to be the identity function.

Finalize(d)
Return d.

In this case the output G^A of the game is the same as the output of the adversary.

3.3 Random functions and permutations

A particular game that we will consider frequently is the game Rand_R described on the right hand side of Fig. 3.1. Here R is a finite set, for example $\{0,1\}^{128}$. The game provides the adversary access to an oracle **Fn** that implements a random function. This means that on any query the oracle returns a random point from R as response subject to the restriction that if twice queried on the same point, the response is the same both time. The game maintains the function in the form of a table T where $T[X]$ holds the value of the function at X. Initially, the table is everywhere undefined, meaning holds $\perp$ in every entry.

One must remember that the term "random function" is misleading. It might lead one to think that certain functions are "random" and others are not. (For example, maybe the constant function that always returns 0^L on any input is not random, but a function with many different range values is random.) This is not right. The randomness of the function refers to the way it was chosen, not to an attribute of the selected function itself. When you choose a function at random, the constant function is just as likely to appear as any other function. It makes no sense to talk of the randomness of an individual function; the term "random function" just means a function chosen at random.

Example 3.3.1 Let's do some simple probabilistic computations to understand random functions. In all of the following, we refer to Rand_R where $R = \{0,1\}^L$.

1. Fix $X \in \{0,1\}^\ell$ and $Y \in \{0,1\}^L$. Let A be

Adversary A
$Z \leftarrow \mathbf{Fn}(X)$
Return $(Y = Z)$

Then:

$$\Pr\left[\mathrm{Rand}_R^A \Rightarrow \mathsf{true}\right] = 2^{-L} .$$

Notice that the probability doesn't depend on ℓ. Nor does it depend on the values of X, Y.

2. Fix $X_1, X_2 \in \{0,1\}^\ell$ and $Y \in \{0,1\}^L$. Let A be

Adversary A
$Z_1 \leftarrow \mathbf{Fn}(X_1)$
$Z_2 \leftarrow \mathbf{Fn}(X_2)$
Return $(Y = Z_1 \wedge Y = Z_2)$

Then:

$$\Pr\left[\mathrm{Rand}_R^A \Rightarrow \mathsf{true}\right] = \begin{cases} 2^{-2L} & \text{if } X_1 \neq X_2 \\ 2^{-L} & \text{if } X_1 = X_2 \end{cases}$$

3. Fix $X_1, X_2 \in \{0,1\}^\ell$ and $Y \in \{0,1\}^L$. Let A be

Adversary A
$Z_1 \leftarrow \mathbf{Fn}(X_1)$
$Z_2 \leftarrow \mathbf{Fn}(X_2)$
Return $(Y = Z_1 \oplus Z_2)$

Then:

$$\Pr\left[\mathrm{Rand}_R^A \Rightarrow \mathsf{true}\right] = \begin{cases} 2^{-L} & \text{if } X_1 \neq X_2 \\ 0 & \text{if } X_1 = X_2 \text{ and } Y \neq 0^L \\ 1 & \text{if } X_1 = X_2 \text{ and } Y = 0^L \end{cases}$$

4. Suppose $l \leq L$ and let τ: $\{0,1\}^L \to \{0,1\}^l$ denote the function that on input $Y \in \{0,1\}^L$ returns the first l bits of Y. Fix $X_1 \in \{0,1\}^\ell$ and $Y_1 \in \{0,1\}^l$. Let A be

Adversary A
$Z_1 \leftarrow \mathbf{Fn}(X_1)$
Return $(\tau(Z_1) = Y_1)$

Then:

$$\Pr\left[\mathrm{Rand}_R^A \Rightarrow \mathsf{true}\right] = 2^{-l} . \blacksquare$$

3.3.1 Random permutations

The game Perm_D shown on the right hand side of Fig. 3.2 provides the adversary access to an oracle that implements a random permutation over the finite set D. Random permutations are somewhat harder to work with than random functions, due to the lack of independence between values on different points. Let's look at some probabilistic computations involving them.

Example 3.3.2 In all of the following we refer to game Perm_D where $D = \{0,1\}^\ell$.

1. Fix $X, Y \in \{0,1\}^\ell$. Let's A be

Adversary A
$Z \leftarrow \mathbf{Fn}(X)$
Return $(Y = Z)$

Then

$$\Pr\left[\mathrm{Perm}_D^A \Rightarrow \mathsf{true}\right] = 2^{-\ell}\ .$$

2. Fix $X_1, X_2 \in \{0,1\}^\ell$ and $Y_1, Y_2 \in \{0,1\}^L$, and assume $X_1 \neq X_2$. Let A be

Adversary A
$Z_1 \leftarrow \mathbf{Fn}(X_1)$
$Z_2 \leftarrow \mathbf{Fn}(X_2)$
Return $(Y_1 = Z_1 \wedge Y_2 = Z_2)$

Then

$$\Pr\left[\mathrm{Perm}_D^A \Rightarrow \mathsf{true}\right] = \begin{cases} \dfrac{1}{2^\ell(2^\ell - 1)} & \text{if } Y_1 \neq Y_2 \\ 0 & \text{if } Y_1 = Y_2 \end{cases}$$

3. Fix $X_1, X_2 \in \{0,1\}^\ell$ and $Y \in \{0,1\}^\ell$. Let A be

Adversary A
$Z_1 \leftarrow \mathbf{Fn}(X_1)$
$Z_2 \leftarrow \mathbf{Fn}(X_2)$
Return $(Y = Z_1 \oplus Z_2)$

Then:

$$\Pr\left[\mathrm{Perm}_D^A \Rightarrow \mathsf{true}\right] = \begin{cases} \dfrac{1}{2^\ell - 1} & \text{if } X_1 \neq X_2 \text{ and } Y \neq 0^\ell \\ 0 & \text{if } X_1 \neq X_2 \text{ and } Y = 0^\ell \\ 0 & \text{if } X_1 = X_2 \text{ and } Y \neq 0^\ell \\ 1 & \text{if } X_1 = X_2 \text{ and } Y = 0^\ell \end{cases}$$

In the case $X_1 \neq X_2$ and $Y \neq 0^\ell$ this is computed as follows:

$$\begin{aligned} & \Pr\left[\mathbf{Fn}(X_1) \oplus \mathbf{Fn}(X_2) = Y\right] \\ &= \sum_{Y_1} \Pr\left[\mathbf{Fn}(X_1) = Y_1 \wedge \mathbf{Fn}(X_2) = Y_1 \oplus Y\right] \\ &= \sum_{Y_1} \frac{1}{2^\ell - 1} \cdot \frac{1}{2^\ell} \\ &= 2^\ell \cdot \frac{1}{2^\ell - 1} \cdot \frac{1}{2^\ell} \\ &= \frac{1}{2^\ell - 1}\ . \end{aligned}$$

Above, the sum is over all $Y_1 \in \{0,1\}^\ell$. In obtaining the second equality, we used item 2 above and the assumption that $Y \neq 0^\ell$.

3.4 Pseudorandom functions

A pseudorandom function is a family of functions with the property that the input-output behavior of a random instance of the family is "computationally indistinguishable" from that of a random function. Someone who has only black-box access to a function, meaning can only feed it inputs and get outputs, has a hard time telling whether the function in question is a random instance of the family in question or a random function. The purpose of this section is to arrive at a suitable formalization of this notion. Later we will look at motivation and applications.

We fix a family of functions F: $\mathcal{K} \times D \to R$. (You may want to think $\mathcal{K} = \{0,1\}^k$, $D = \{0,1\}^\ell$ and $R = \{0,1\}^L$ for some integers $k, \ell, L \geq 1$.) Imagine that you are in a room which contains a terminal connected to a computer outside your room. You can type something into your terminal and send it out, and an answer will come back. The allowed questions you can type must be elements of the domain D, and the answers you get back will be elements of the range R. The computer outside your room implements a function $\mathbf{Fn}$: $D \to R$, so that whenever you type a value X you get back $\mathbf{Fn}(X)$. However, your only access to $\mathbf{Fn}$ is via this interface, so the only thing you can see is the input-output behavior of $\mathbf{Fn}$.

We consider two different ways in which $\mathbf{Fn}$ will be chosen, giving rise to two different "worlds." In the "real" world, $\mathbf{Fn}$ is a random instance of F, meaning is F_K for a random K. In the "random" world, $\mathbf{Fn}$ is a random function with range R.

You are not told which of the two worlds was chosen. The choice of world, and of the corresponding function $\mathbf{Fn}$, is made before you enter the room, meaning before you start typing questions. Once made, however, these choices are fixed until your "session" is over. Your job is to discover which world you are in. To do this, the only resource available to you is your link enabling you to provide values X and get back $\mathbf{Fn}(X)$. After trying some number of values of your choice, you must make a decision regarding which world you are in. The quality of pseudorandom family F can be thought of as measured by the difficulty of telling, in the above game, whether you are in the real world or in the random world.

In the formalization, the entity referred to as "you" above is an algorithm called the adversary. The adversary algorithm A may be randomized. We formalize the ability to query $\mathbf{Fn}$ as giving A an *oracle* which takes input any string $X \in D$ and returns $\mathbf{Fn}(X)$. A can only interact with the function by giving it inputs and examining the outputs for those inputs; it cannot examine the function directly in any way. Algorithm A can decide which queries to make, perhaps based on answers received to previous queries. Eventually, it outputs a bit b which is its decision as to which world it is in. Outputting the bit "1" means that A "thinks" it is in the real world; outputting the bit "0" means that A thinks it is in the random world.

The worlds are formalized via the game of Fig. 3.1. The following definition associates to any adversary a number between 0 and 1 that is called its prf-advantage, and is a measure of how well the adversary is doing at determining which world it is in. Further explanations follow the definition.

Definition 3.4.1 Let F: $\mathcal{K} \times D \to R$ be a family of functions, and let A be an algorithm that takes an oracle and returns a bit. We consider two games as described in Fig. 3.1. The *prf-advantage* of A is defined as

$$\mathbf{Adv}^{\mathrm{prf}}_F(A) = \Pr\left[\mathrm{Real}^A_F \Rightarrow 1\right] - \Pr\left[\mathrm{Rand}^A_R \Rightarrow 1\right]$$

It should be noted that the family F is public. The adversary A, and anyone else, knows the description of the family and is capable, given values K, X, of computing $F(K, X)$.

Game Real_F	Game Rand_R
procedure Initialize	**procedure Fn**(x)
$K \xleftarrow{\$} \mathsf{Keys}(F)$	If $\mathsf{T}[x] = \perp$ then
procedure Fn(x)	$\quad \mathsf{T}[x] \xleftarrow{\$} R$
Return $F_K(x)$	Return $\mathsf{T}[x]$

Figure 3.1: Games used to define PRFs.

Game Real_F picks a random instance F_K of family F and then runs adversary A with oracle $\mathbf{Fn} = F_K$. Adversary A interacts with its oracle, querying it and getting back answers, and eventually outputs a "guess" bit. The game returns the same bit. Game Rand_R implements $\mathbf{Fn}$ as a random function with range R. Again, adversary A interacts with the oracle, eventually returning a bit that is the output of the game. Each game has a certain probability of returning 1. The probability is taken over the random choices made in the game. Thus, for the first game, the probability is over the choice of K and any random choices that A might make, for A is allowed to be a randomized algorithm. In the second game, the probability is over the random choice made by the game in implementing $\mathbf{Fn}$ and any random choices that A makes. These two probabilities should be evaluated separately; the two games are completely distinct.

To see how well A does at determining which world it is in, we look at the difference in the probabilities that the two games return 1. If A is doing a good job at telling which world it is in, it would return 1 more often in the first game than in the second. So the difference is a measure of how well A is doing. We call this measure the prf-advantage of A. Think of it as the probability that A "breaks" the scheme F, with "break" interpreted in a specific, technical way based on the definition.

Different adversaries will have different advantages. There are two reasons why one adversary may achieve a greater advantage than another. One is that it is more "clever" in the questions it asks and the way it processes the replies to determine its output. The other is simply that it asks more questions, or spends more time processing the replies. Indeed, we expect that as an adversary sees more and more input-output examples of $\mathbf{Fn}$, or spends more computing time, its ability to tell which world it is in should go up.

The "security" of family F as a pseudorandom function must thus be thought of as depending on the resources allowed to the attacker. We may want to know, for any given resource limitations, what is the prf-advantage achieved by the most "clever" adversary amongst all those who are restricted to the given resource limits.

The choice of resources to consider can vary. One resource of interest is the time-complexity t of A. Another resource of interest is the number of queries q that A asks of its oracle. Another resource of interest is the total length μ of all of A's queries. When we state results, we will pay attention to such resources, showing how they influence maximal adversarial advantage.

Let us explain more about the resources we have mentioned, giving some important conventions underlying their measurement. The first resource is the time-complexity of A. To make sense of this we first need to fix a model of computation. We fix some RAM model, as discussed in Chapter 1. Think of the model used in your algorithms courses, often implicitly, so that you could measure the running time. However, we adopt the convention that the *time-complexity* of A refers not just to the running time of A, but to the maximum of the running times of the two games in the definition, plus the size of the code of A. In measuring the running time of the first game, we must count the time to choose the key K at random, and the time to compute the value $F_K(x)$ for any query x

Game Real_F	Game Perm_D
procedure Initialize $K \xleftarrow{\$} \mathsf{Keys}(F)$	**procedure Initialize** $\mathsf{UR} \leftarrow \emptyset$
procedure Fn(x) Return $F_K(x)$	**procedure Fn**(x) If $\mathsf{T}[x] = \perp$ then $\quad \mathsf{T}[x] \xleftarrow{\$} D \setminus \mathsf{UR}$; $\mathsf{UR} \leftarrow \mathsf{UR} \cup \{\mathsf{T}[x]\}$ Return $\mathsf{T}[x]$

Figure 3.2: Games used to define PRP under CPA.

made by A to its oracle. In measuring the running time of the second game, we count the execution time of **Fn** over the call made to it by A.

The number of queries made by A captures the number of input-output examples it sees. In general, not all strings in the domain must have the same length, and hence we also measure the sum of the lengths of all queries made.

The strength of this definition lies in the fact that it does not specify anything about the kinds of strategies that can be used by a adversary; it only limits its resources. A adversary can use whatever means desired to distinguish the function as long as it stays within the specified resource bounds.

What do we mean by a "secure" PRF? Definition 3.4.1 does not have any explicit condition or statement regarding when F should be considered "secure." It only associates to any adversary A attacking F a prf-advantage function. Intuitively, F is "secure" if the value of the advantage function is "low" for all adversaries whose resources are "practical."

This is, of course, not formal. However, we wish to keep it this way because it better reflects reality. In real life, security is not some absolute or boolean attribute; security is a function of the resources invested by an attacker. All modern cryptographic systems are breakable in principle; it is just a question of how long it takes.

This is our first example of a cryptographic definition, and it is worth spending time to study and understand it. We will encounter many more as we go along. Towards this end let us summarize the main features of the definitional framework as we will see them arise later. First, there are *games*, involving an adversary. Then, there is some *advantage* function associated to an adversary which returns the probability that the adversary in question "breaks" the scheme. These two components will be present in all definitions. What varies is the games; this is where we pin down how we measure security.

3.5 Pseudorandom permutations

A family of functions F: $\mathcal{K} \times D \to D$ is a pseudorandom permutation if the input-output behavior of a random instance of the family is "computationally indistinguishable" from that of a random permutation on D.

In this setting, there are two kinds of attacks that one can consider. One, as before, is that the adversary gets an oracle for the function **Fn** being tested. However when F is a family of permutations, one can also consider the case where the adversary gets, in addition, an oracle for $\mathbf{Fn}^{-1}$. We consider these settings in turn. The first is the setting of chosen-plaintext attacks while the second is the setting of chosen-ciphertext attacks.

Game Real_F

procedure Initialize
$K \xleftarrow{\$} \mathsf{Keys}(F)$

procedure $\mathbf{Fn}(x)$
Return $F_K(x)$

procedure $\mathbf{Fn}^{-1}(x)$
Return $F_K^{-1}(x)$

Game Perm_D

procedure Initialize
$\mathsf{UR} \leftarrow \emptyset$; $\mathsf{UD} \leftarrow \emptyset$

procedure $\mathbf{Fn}(x)$
If $\mathsf{T}[x] = \perp$ then
  $\mathsf{T}[x] \xleftarrow{\$} D \setminus \mathsf{UR}$
  $\mathsf{S}[\mathsf{T}[x]] \leftarrow x$
  $\mathsf{UR} \leftarrow \mathsf{UR} \cup \{\mathsf{T}[x]\}$; $\mathsf{UD} \leftarrow \mathsf{UD} \cup \{x\}$
Return $\mathsf{T}[x]$

procedure $\mathbf{Fn}^{-1}(y)$
If $\mathsf{S}[y] = \perp$ then
  $\mathsf{S}[y] \xleftarrow{\$} D \setminus \mathsf{UD}$
  $\mathsf{T}[\mathsf{S}[y]] \leftarrow y$
  $\mathsf{UD} \leftarrow \mathsf{UD} \cup \{\mathsf{S}[y]\}$; $\mathsf{UR} \leftarrow \mathsf{UR} \cup \{y\}$
Return $\mathsf{S}[y]$

Figure 3.3: Games used to define PRP under CCA.

3.5.1 PRP under CPA

We fix a family of functions $F\colon \mathcal{K} \times D \to D$. (You may want to think $\mathcal{K} = \{0,1\}^k$ and $D = \{0,1\}^\ell$, since this is the most common case. We do not mandate that F be a family of permutations although again this is the most common case.) As before, we consider an adversary A that is placed in a room where it has oracle access to a function **Fn** chosen in one of two ways.

In the "real" world, **Fn** is a random instance of F, meaning is F_K for a random K. In the "random" world, **Fn** is a random permutation on D.

Notice that the real world is the same in the PRF setting, but the random world has changed. As before the task facing the adversary A is to determine in which world it was placed based on the input-output behavior of **Fn**.

Definition 3.5.1 Let $F\colon \mathcal{K} \times D \to D$ be a family of functions, and let A be an algorithm that takes an oracle **Fn** for a function $\mathbf{Fn}\colon D \to D$, and returns a bit. We consider two games as described in Fig. 3.2. The *prp-cpa-advantage* of A is defined as

$$\mathbf{Adv}^{\text{prp-cpa}}_F(A) = \Pr\left[\text{Real}_F^A \Rightarrow 1\right] - \Pr\left[\text{Perm}_D^A \Rightarrow 1\right]$$

The intuition is similar to that for Definition 3.4.1. The difference is that here the "ideal" object that F is being compared with is no longer a random function, but rather a random permutation.

In game Real_F, the probability is over the random choice of key K and also over the coin tosses of A if the latter happens to be randomized. The game returns the same bit that A returns. In game Perm_D, a permutation $\mathbf{Fn}\colon D \to D$ is chosen at random, and the result bit of A's computation with oracle **Fn** is returned. The probability is over the choice of **Fn** and the coins of A if any. As before, the measure of how well A did at telling the two worlds apart, which we call the prp-cpa-advantage of A, is the difference between the probabilities that the games return 1.

Conventions regarding resource measures also remain the same as before. Informally, a family F is a secure PRP under CPA if $\mathbf{Adv}^{\text{prp-cpa}}_F(A)$ is "small" for all adversaries using a "practical" amount of resources.

3.5.2 PRP under CCA

We fix a family of permutations $F\colon \mathcal{K} \times D \to D$. (You may want to think $\mathcal{K} = \{0,1\}^k$ and $D = \{0,1\}^\ell$, since this is the most common case. This time, we do mandate that F be a family of permutations.) As before, we consider an adversary A that is placed in a room, but now it has oracle access to two functions, $\mathbf{Fn}$ and its inverse $\mathbf{Fn}^{-1}$. The manner in which $\mathbf{Fn}$ is chosen is the same as in the CPA case, and once $\mathbf{Fn}$ is chosen, $\mathbf{Fn}^{-1}$ is automatically defined, so we do not have to say how it is chosen.

In the "real" world, $\mathbf{Fn}$ is a random instance of F, meaning is F_K for a random K. In the "random" world, $\mathbf{Fn}$ is a random permutation on D. In either case, $\mathbf{Fn}^{-1}$ is the inverse of $\mathbf{Fn}$. As before the task facing the adversary A is to determine in which world it was placed based on the input-output behavior of its oracles.

Definition 3.5.2 Let $F\colon \mathcal{K} \times D \to D$ be a family of permutations, and let A be an algorithm that takes an oracle $\mathbf{Fn}$ for a function $\mathbf{Fn}\colon D \to D$, and also an oracle $\mathbf{Fn}^{-1}$ for the function $\mathbf{Fn}^{-1}\colon D \to D$, and returns a bit. We consider two games as described in Fig. 3.3. The *prp-cca-advantage* of A is defined as

$$\mathbf{Adv}^{\text{prp-cca}}_F(A) = \Pr\left[\text{Real}^A_F \Rightarrow 1\right] - \Pr\left[\text{Perm}^A_D \Rightarrow 1\right]$$

The intuition is similar to that for Definition 3.4.1. The difference is that here the adversary has more power: not only can it query $\mathbf{Fn}$, but it can directly query $\mathbf{Fn}^{-1}$. Conventions regarding resource measures also remain the same as before. However, we will be interested in some additional resource parameters. Specifically, since there are now two oracles, we can count separately the number of queries, and total length of these queries, for each. As usual, informally, a family F is a secure PRP under CCA if $\mathbf{Adv}^{\text{prp-cca}}_F(A)$ is "small" for all adversaries using a "practical" amount of resources.

3.5.3 Relations between the notions

If an adversary does not query $\mathbf{Fn}^{-1}$ the oracle might as well not be there, and the adversary is effectively mounting a chosen-plaintext attack. Thus we have the following:

Proposition 3.5.3 [PRP-CCA implies PRP-CPA] Let $F\colon \mathcal{K} \times D \to D$ be a family of permutations and let A be a prp-cpa adversary. Suppose that A runs in time t, asks q queries, and these queries total μ bits. Then there exists a prp-cca adversary B that runs in time t, asks q chosen-plaintext queries, these queries totaling μ bits, and asks no chosen-ciphertext queries, such that

$$\mathbf{Adv}^{\text{prp-cpa}}_F(A) \leq \quad \mathbf{Adv}^{\text{prp-cca}}_F(B) \;. \blacksquare$$

Though the technical result is easy, it is worth stepping back to explain its interpretation. The theorem says that if you have an adversary A that breaks F in the PRP-CPA sense, then you have some *other* adversary B that breaks F in the PRP-CCA sense. Furthermore, the adversary B will be just as efficient as the adversary A was. As a consequence, if you think there is *no* reasonable adversary B that breaks F in the PRP-CCA sense, then you have no choice but to believe that there is *no* reasonable adversary A that breaks F in the PRP-CPA sense. The inexistence of a reasonable adversary B that breaks F in the PRP-CCA sense means that F is PRP-CCA secure, while the inexistence of a reasonable adversary A that breaks F in the PRP-CPA sense means that F is PRP-CPA secure. So PRP-CCA security implies PRP-CPA security, and a statement like the proposition above is how, precisely, one makes such a statement.

3.6 Modeling blockciphers

One of the primary motivations for the notions of pseudorandom functions (PRFs) and pseudorandom permutations (PRPs) is to model blockciphers and thereby enable the security analysis of protocols that use blockciphers.

As discussed in the chapter on blockciphers, classically the security of DES or other blockciphers has been looked at only with regard to key recovery. That is, analysis of a blockcipher F has focused on the following question: Given some number of input-output examples

$$(X_1, F_K(X_1)), \ldots, (X_q, F_K(X_q))$$

where K is a random, unknown key, how hard is it to find K? The blockcipher is taken as "secure" if the resources required to recover the key are prohibitive. Yet, as we saw, even a cursory glance at common blockcipher usages shows that hardness of key recovery is not *sufficient* for security. We had discussed wanting a *master* security property of blockciphers under which natural usages of blockciphers could be proven secure. We suggest that this *master* property is that the blockcipher be a secure PRP, under either CPA or CCA.

We cannot prove that specific blockciphers have this property. The best we can do is assume they do, and then go on to use them. For quantitative security assessments, we would make specific conjectures about the advantage functions of various blockciphers. For example we might conjecture something like:

$$\mathbf{Adv}^{\text{prp-cpa}}_{\mathsf{DES}}(A_{t,q}) \;\leq\; c_1 \cdot \frac{t/T_{\mathsf{DES}}}{2^{55}} + c_2 \cdot \frac{q}{2^{40}}$$

for any adversary $A_{t,q}$ that runs in time at most t and asks at most q 64-bit oracle queries. Here T_{DES} is the time to do one DES computation on our fixed RAM model of computation, and c_1, c_2 are some constants depending only on this model. In other words, we are conjecturing that the best attacks are either exhaustive key search or linear cryptanalysis. We might be bolder with regard to AES and conjecture something like

$$\mathbf{Adv}^{\text{prp-cpa}}_{\mathsf{AES}}(B_{t,q}) \;\leq\; c_1 \cdot \frac{t/T_{\mathsf{AES}}}{2^{128}} + c_2 \cdot \frac{q}{2^{128}} \;.$$

for any adversary $B_{t,q}$ that runs in time at most t and asks at most q 128-bit oracle queries. We could also make similar conjectures regarding the strength of blockciphers as PRPs under CCA rather than CPA.

More interesting is the PRF security of blockciphers. Here we cannot do better than assume that

$$\mathbf{Adv}^{\text{prf}}_{\mathsf{DES}}(A_{t,q}) \;\leq\; c_1 \cdot \frac{t/T_{\mathsf{DES}}}{2^{55}} + \frac{q^2}{2^{64}}$$

$$\mathbf{Adv}^{\text{prf}}_{\mathsf{AES}}(B_{t,q}) \;\leq\; c_1 \cdot \frac{t/T_{\mathsf{AES}}}{2^{128}} + \frac{q^2}{2^{128}} \;.$$

for any adversaries $A_{t,q}, B_{t,q}$ running in time at most t and making at most q oracle queries. This is due to the birthday attack discussed later. The second term in each formula arises simply because the object under consideration is a family of permutations.

We stress that these are all conjectures. There could exist highly effective attacks that break DES or AES as a PRF without recovering the key. So far, we do not know of any such attacks, but the amount of cryptanalytic effort that has focused on this goal is small. Certainly, to assume that a blockcipher is a PRF is a much stronger assumption than that it is secure against key recovery.

Nonetheless, the motivation and arguments we have outlined in favor of the PRF assumption stay, and our view is that if a blockcipher is broken as a PRF then it should be considered insecure, and a replacement should be sought.

3.7 Example attacks

Let us illustrate the models by providing adversaries that attack different function families in these models.

Example 3.7.1 We define a family of functions F: $\{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^L$ as follows. We let $k = L\ell$ and view a k-bit key K as specifying an L row by ℓ column matrix of bits. (To be concrete, assume the first L bits of K specify the first column of the matrix, the next L bits of K specify the second column of the matrix, and so on.) The input string $X = X[1] \ldots X[\ell]$ is viewed as a sequence of bits, and the value of $F(K, x)$ is the corresponding matrix vector product. That is

$$F_K(X) \;=\; \begin{bmatrix} K[1,1] & K[1,2] & \cdots & K[1,\ell] \\ K[2,1] & K[2,2] & \cdots & K[2,\ell] \\ \vdots & & & \vdots \\ K[L,1] & K[L,2] & \cdots & K[L,\ell] \end{bmatrix} \cdot \begin{bmatrix} X[1] \\ X[2] \\ \vdots \\ X[l] \end{bmatrix} \;=\; \begin{bmatrix} Y[1] \\ Y[2] \\ \vdots \\ Y[L] \end{bmatrix}$$

where

$$\begin{aligned} Y[1] &= K[1,1] \cdot x[1] \oplus K[1,2] \cdot x[2] \oplus \ \ldots \ \oplus K[1,\ell] \cdot x[\ell] \\ Y[2] &= K[2,1] \cdot x[1] \oplus K[2,2] \cdot x[2] \oplus \ \ldots \ \oplus K[2,\ell] \cdot x[\ell] \\ \vdots &= \vdots \\ Y[L] &= K[L,1] \cdot x[1] \oplus K[L,2] \cdot x[2] \oplus \ \ldots \ \oplus K[L,\ell] \cdot x[\ell] \ . \end{aligned}$$

Here the bits in the matrix are the bits in the key, and arithmetic is modulo two. The question we ask is whether F is a "secure" PRF. We claim that the answer is no. The reason is that one can design an adversary algorithm A that achieves a high advantage (close to 1) in distinguishing between the two worlds.

We observe that for any key K we have $F_K(0^\ell) = 0^L$. This is a weakness since a random function of ℓ-bits to L-bits is very unlikely to return 0^L on input 0^ℓ, and thus this fact can be the basis of a distinguishing adversary. Let us now show how the adversary works. Remember that as per our model it is given an oracle $\mathbf{Fn}$ for $\mathbf{Fn}$: $\{0,1\}^\ell \to \{0,1\}^L$ and will output a bit. Our adversary A works as follows:

Adversary A
$Y \leftarrow \mathbf{Fn}(0^\ell)$
if $Y = 0^L$ then return 1 else return 0

This adversary queries its oracle at the point 0^ℓ, and denotes by Y the ℓ-bit string that is returned. If $y = 0^L$ it bets that $\mathbf{Fn}$ was an instance of the family F, and if $y \neq 0^L$ it bets that $\mathbf{Fn}$ was a random function. Let us now see how well this adversary does. Let $R = \{0,1\}^L$. We claim that

$$\begin{aligned} \Pr\left[\mathrm{Real}_F^A \Rightarrow 1\right] &= 1 \\ \Pr\left[\mathrm{Rand}_R^A \Rightarrow 1\right] &= 2^{-L} \ . \end{aligned}$$

Why? Look at Game Real_F as defined in Definition 3.4.1. Here $\mathbf{Fn} = F_K$ for some K. In that case it is certainly true that $\mathbf{Fn}(0^\ell) = 0^L$ so by the code we wrote for A the latter will return 1. On the other hand look at Game Rand_R as defined in Definition 3.4.1. Here $\mathbf{Fn}$ is a random function. As we saw in Example 3.3.1, the probability that $\mathbf{Fn}(0^\ell) = 0^L$ will be 2^{-L}, and hence this is the probability that A will return 1. Now as per Definition 3.4.1 we subtract to get

$$\begin{aligned}\mathbf{Adv}^{\text{prf}}_F(A) &= \Pr\left[\text{Real}^A_F \Rightarrow 1\right] - \Pr\left[\text{Rand}^A_R \Rightarrow 1\right] \\ &= 1 - 2^{-L} \ .\end{aligned}$$

Now let t be the time complexity of F. This is $O(\ell + L)$ plus the time for one computation of F, coming to $O(\ell^2 L)$. The number of queries made by A is just one, and the total length of all queries is l. Our conclusion is that there exists an extremely efficient adversary whose prf-advantage is very high (almost one). Thus, F is not a secure PRF. ▌

Example 3.7.2 . Suppose we are given a secure PRF F: $\{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^L$. We want to use F to design a PRF G: $\{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^{2L}$. The input length of G is the same as that of F but the output length of G is twice that of F. We suggest the following candidate construction: for every k-bit key K and every ℓ-bit input x

$$G_K(x) \ = \ F_K(x) \,\|\, F_K(\overline{x}) \ .$$

Here " $\|$ " denotes concatenation of strings, and $\overline{x}$ denotes the bitwise complement of the string x. We ask whether this is a "good" construction. "Good" means that under the assumption that F is a secure PRF, G should be too. However, this is not true. Regardless of the quality of F, the construct G is insecure. Let us demonstrate this.

We want to specify an adversary attacking G. Since an instance of G maps ℓ bits to $2L$ bits, the adversary D will get an oracle for a function $\mathbf{Fn}$ that maps ℓ bits to $2L$ bits. In the random world, $\mathbf{Fn}$ will be chosen as a random function of ℓ bits to $2L$ bits, while in the real world, $\mathbf{Fn}$ will be set to G_K where K is a random k-bit key. The adversary must determine in which world it is placed. Our adversary works as follows:

Adversary A
$y_1 \leftarrow \mathbf{Fn}(1^\ell)$
$y_2 \leftarrow \mathbf{Fn}(0^\ell)$
Parse y_1 as $y_1 = y_{1,1} \,\|\, y_{1,2}$ with $|y_{1,1}| = |y_{1,2}| = L$
Parse y_2 as $y_2 = y_{2,1} \,\|\, y_{2,2}$ with $|y_{2,1}| = |y_{2,2}| = L$
if $y_{1,1} = y_{2,2}$ then return 1 else return 0

This adversary queries its oracle at the point 1^ℓ to get back y_1 and then queries its oracle at the point 0^ℓ to get back y_2. Notice that 1^ℓ is the bitwise complement of 0^ℓ. The adversary checks whether the first half of y_1 equals the second half of y_2, and if so bets that it is in the real world. Let us now see how well this adversary does. Let $R = \{0,1\}^{2L}$. We claim that

$$\begin{aligned}\Pr\left[\text{Real}^A_G \Rightarrow 1\right] &= 1 \\ \Pr\left[\text{Rand}^A_R \Rightarrow 1\right] &= 2^{-L} \ .\end{aligned}$$

Why? Look at Game Real_G as defined in Definition 3.4.1. Here $g = G_K$ for some K. In that case we have

$$\begin{aligned}G_K(1^\ell) &= F_K(1^\ell) \,\|\, F_K(0^\ell) \\ G_K(0^\ell) &= F_K(0^\ell) \,\|\, F_K(1^\ell)\end{aligned}$$

by definition of the family G. Notice that the first half of $G_K(1^\ell)$ is the same as the second half of $G_K(0^\ell)$. So A will return 1. On the other hand look at Game Rand_R as defined in Definition 3.4.1. Here **Fn** is a random function. So the values $\mathbf{Fn}(1^\ell)$ and $\mathbf{Fn}(0^\ell)$ are both random and independent $2L$ bit strings. What is the probability that the first half of the first string equals the second half of the second string? It is exactly the probability that two randomly chosen L-bit strings are equal, and this is 2^{-L}. So this is the probability that A will return 1. Now as per Definition 3.4.1 we subtract to get

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_G(A) &= \Pr\left[\mathrm{Real}^A_G \Rightarrow 1\right] - \Pr\left[\mathrm{Rand}^A_R \Rightarrow 1\right] \\ &= 1 - 2^{-L}\ .\end{aligned}$$

Now let t be the time complexity of A. This is $O(\ell + L)$ plus the time for two computations of G, coming to $O(\ell + L)$ plus the time for four computations of F. The number of queries made by D is two, and the total length of all queries is 2ℓ. Thus we have exhibited an efficient adversary with a very high prf-advantage, showing that G is not a secure PRF. ▌

3.8 Security against key recovery

We have mentioned several times that security against key recovery is not sufficient as a notion of security for a blockcipher. However it is certainly necessary: if key recovery is easy, the blockcipher should be declared insecure. We have indicated that we want to adopt as notion of security for a blockcipher the notion of a PRF or a PRP. If this is to be viable, it should be the case that any function family that is insecure under key recovery is also insecure as a PRF or PRP. In this section we verify this simple fact. Doing so will enable us to exercise the method of reductions.

We begin by formalizing security against key recovery. We consider an adversary that, based on input-output examples of an instance F_K of family F, tries to find K. Its advantage is defined as the probability that it succeeds in finding K. The probability is over the random choice of K, and any random choices of the adversary itself.

We give the adversary oracle access to F_K so that it can obtain input-output examples of its choice. We do not constrain the adversary with regard to the method it uses. This leads to the following definition.

Definition 3.8.1 Let $F\colon \mathcal{K} \times D \to R$ be a family of functions, and let B be an algorithm that takes an oracle **Fn** for a function $\mathbf{Fn}\colon D \to R$ and outputs a string. We consider the game as described in Fig. 3.4. The *kr-advantage* of B is defined as

$$\mathbf{Adv}^{\mathrm{kr}}_F(B) = \Pr\left[\mathrm{KR}^B_F \Rightarrow 1\right]$$

This definition has been made general enough to capture all types of key-recovery attacks. Any of the classical attacks such as exhaustive key search, differential cryptanalysis or linear cryptanalysis correspond to different, specific choices of adversary B. They fall in this framework because all have the goal of finding the key K based on some number of input-output examples of an instance F_K of the cipher. To illustrate let us see what are the implications of the classical key-recovery attacks on DES for the value of the key-recovery advantage function of DES. Assuming the exhaustive key-search attack is always successful based on testing two input-output examples leads to the fact that there exists an adversary B such that $\mathbf{Adv}^{\mathrm{kr}}_{\mathsf{DES}}(B) = 1$ and B makes two oracle queries and

Game KR_F

procedure Initialize
$K \xleftarrow{\$} \mathsf{Keys}(F)$

procedure Fn(x)
return $F_K(x)$

procedure Finalize(K')
return $(K = K')$

Figure 3.4: Game used to define KR.

has running time about 2^{55} times the time T_{DES} for one computation of DES. On the other hand, linear cryptanalysis implies that there exists an adversary B such that $\mathbf{Adv}^{\mathrm{kr}}_{\mathsf{DES}}(B) \geq 1/2$ and B makes 2^{44} oracle queries and has running time about 2^{44} times the time T_{DES} for one computation of DES.

For a more concrete example, let us look at the key-recovery advantage of the family of Example 3.7.1.

Example 3.8.2 Let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be the family of functions from Example 3.7.1. We saw that its prf-advantage was very high. Let us now compute its kr-advantage. The following adversary B recovers the key. We let e_j be the l-bit binary string having a 1 in position j and zeros everywhere else. We assume that the manner in which the key K defines the matrix is that the first L bits of K form the first column of the matrix, the next L bits of K form the second column of the matrix, and so on.

Adversary B
  $K' \leftarrow \varepsilon$ // ε is the empty string
  for $j = 1, \ldots, l$ do
    $y_j \leftarrow \mathbf{Fn}(e_j)$
    $K' \leftarrow K' \,\|\, y_j$
  return K'

The adversary B invokes its oracle to compute the output of the function on input e_j. The result, y_j, is exactly the j-th column of the matrix associated to the key K. The matrix entries are concatenated to yield K', which is returned as the key. Since the adversary always finds the key we have

$$\mathbf{Adv}^{\mathrm{kr}}_F(B) \;=\; 1\,.$$

The time-complexity of this adversary is $t = O(l^2 L)$ since it makes $q = l$ calls to its oracle and each computation of **Fn** takes $O(lL)$ time. The parameters here should still be considered small: l is 64 or 128, which is small for the number of queries. So F is insecure against key-recovery. ▌

Note that the F of the above example is less secure as a PRF than against key-recovery: its advantage function as a PRF had a value close to 1 for parameter values much smaller than those above. This leads into our next claim, which says that for any given parameter values, the kr-advantage of a family cannot be significantly more than its prf or prp-cpa advantage.

Proposition 3.8.3 Let $F\colon \mathcal{K} \times D \to R$ be a family of functions, and let B be a key-recovery adversary against F. Assume B's running time is at most t and it makes at most $q < |D|$ oracle queries. Then there exists a PRF adversary A against F such that A has running time at most t plus the time for one computation of F, makes at most $q+1$ oracle queries, and

$$\mathbf{Adv}^{\mathrm{kr}}_F(B) \;\leq\; \mathbf{Adv}^{\mathrm{prf}}_F(A) + \frac{1}{|R|}\;. \tag{3.1}$$

Furthermore if $D = R$ then there also exists a PRP CPA adversary A against F such that A has running time at most t plus the time for one computation of F, makes at most $q+1$ oracle queries, and

$$\mathbf{Adv}^{\mathrm{kr}}_F(B) \;\leq\; \mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_F(A) + \frac{1}{|D|-q}\;. \blacksquare \tag{3.2}$$

The Proposition implies that if a family of functions is a secure PRF or PRP then it is also secure against all key-recovery attacks. In particular, if a blockcipher is modeled as a PRP or PRF, we are implicitly assuming it to be secure against key-recovery attacks.

Before proceeding to a formal proof let us discuss the underlying ideas. The problem that adversary A is trying to solve is to determine whether its given oracle $\mathbf{Fn}$ is a random instance of F or a random function of D to R. A will run B as a subroutine and use B's output to solve its own problem.

B is an algorithm that expects to be in a world where it gets an oracle $\mathbf{Fn}$ for some random key $K \in \mathcal{K}$, and it tries to find K via queries to its oracle. For simplicity, first assume that B makes no oracle queries. Now, when A runs B, it produces some key K'. A can test K' by checking whether $F(K', x)$ agrees with $\mathbf{Fn}(x)$ for some value x. If so, it bets that $\mathbf{Fn}$ was an instance of F, and if not it bets that $\mathbf{Fn}$ was random.

If B does make oracle queries, we must ask how A can run B at all. The oracle that B wants is not available. However, B is a piece of code, communicating with its oracle via a prescribed interface. If you start running B, at some point it will output an oracle query, say by writing this to some prescribed memory location, and stop. It awaits an answer, to be provided in another prescribed memory location. When that appears, it continues its execution. When it is done making oracle queries, it will return its output. Now when A runs B, it will itself supply the answers to B's oracle queries. When B stops, having made some query, A will fill in the reply in the prescribed memory location, and let B continue its execution. B does not know the difference between this "simulated" oracle and the real oracle except in so far as it can glean this from the values returned.

The value that B expects in reply to query x is $F_K(x)$ where K is a random key from $\mathcal{K}$. However, A returns to it as the answer to query x the value $\mathbf{Fn}(x)$, where $\mathbf{Fn}$ is A's oracle. When A is in the real world, $\mathbf{Fn}(x)$ is an instance of F and so B is functioning as it would in its usual environment, and will return the key K with a probability equal to its kr-advantage. However when A is in the random world, $\mathbf{Fn}$ is a random function, and B is getting back values that bear little relation to the ones it is expecting. That does not matter. B is a piece of code that will run to completion and produce some output. When we are in the random world, we have no idea what properties this output will have. But it is some key in $\mathcal{K}$, and A will test it as indicated above. It will fail the test with high probability as long as the test point x was not one that B queried, and A will make sure the latter is true via its choice of x. Let us now proceed to the actual proof.

Proof of Proposition 3.8.3: We prove the first equation and then briefly indicate how to alter the proof to prove the second equation.

As per Definition 3.4.1, adversary A will be provided an oracle $\mathbf{Fn}$ for a function $\mathbf{Fn}$: $D \to R$, and will try to determine in which World it is. To do so, it will run adversary B as a subroutine. We provide the description followed by an explanation and analysis.

Adversary A
$i \leftarrow 0$
Run adversary B, replying to its oracle queries as follows
When B makes an oracle query x do
$i \leftarrow i + 1$; $x_i \leftarrow x$
$y_i \leftarrow \mathbf{Fn}(x_i)$
Return y_i to B as the answer
Until B stops and outputs a key K'
Let x be some point in $D - \{x_1, \ldots, x_q\}$
$y \leftarrow \mathbf{Fn}(x)$
if $F(K', x) = y$ then return 1 else return 0

As indicated in the discussion preceding the proof, A is running B and itself providing answers to B's oracle queries via the oracle $\mathbf{Fn}$. When B has run to completion it returns some $K' \in \mathcal{K}$, which A tests by checking whether $F(K', x)$ agrees with $\mathbf{Fn}(x)$. Here x is a value different from any that B queried, and it is to ensure that such a value can be found that we require $q < |D|$ in the statement of the Proposition. Now we claim that

$$\Pr\left[\mathrm{Real}_F^A \Rightarrow 1\right] \;\geq\; \mathbf{Adv}_F^{\mathrm{kr}}(B) \tag{3.3}$$

$$\Pr\left[\mathrm{Rand}_R^A \Rightarrow 1\right] \;=\; \frac{1}{|R|}\,. \tag{3.4}$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, as per Definition 3.4.1, we get

$$\begin{aligned}\mathbf{Adv}_F^{\mathrm{prf}}(A) &= \Pr\left[\mathrm{Real}_F^A \Rightarrow 1\right] - \Pr\left[\mathrm{Rand}_R^A \Rightarrow 1\right] \\ &\geq \mathbf{Adv}_F^{\mathrm{kr}}(B) - \frac{1}{|R|}\end{aligned}$$

as desired. It remains to justify Equations (3.3) and (3.4).

Equation (3.3) is true because in Real_F the oracle $\mathbf{Fn}$ is a random instance of F, which is the oracle that B expects, and thus B functions as it does in KR_F^B. If B is successful, meaning the key K' it outputs equals K, then certainly A returns 1. (It is possible that A might return 1 even though B was not successful. This would happen if $K' \neq K$ but $F(K', x) = F(K, x)$. It is for this reason that Equation (3.3) is in inequality rather than an equality.) Equation (3.4) is true because in Rand_R the function $\mathbf{Fn}$ is random, and since x was never queried by B, the value $\mathbf{Fn}(x)$ is unpredictable to B. Imagine that $\mathbf{Fn}(x)$ is chosen only when x is queried to $\mathbf{Fn}$. At that point, K', and thus $F(K', x)$, is already defined. So $\mathbf{Fn}(x)$ has a $1/|R|$ chance of hitting this fixed point. Note this is true regardless of how hard B tries to make $F(K', x)$ be the same as $\mathbf{Fn}(x)$.

For the proof of Equation (3.2), the adversary A is the same. For the analysis we see that

$$\begin{aligned}\Pr\left[\mathrm{Real}_F^A \Rightarrow 1\right] &\geq \mathbf{Adv}_F^{\mathrm{kr}}(B) \\ \Pr\left[\mathrm{Rand}_R^A \Rightarrow 1\right] &\leq \frac{1}{|D| - q}\,.\end{aligned}$$

Subtracting yields Equation (3.2). The first equation above is true for the same reason as before. The second equation is true because in World 0 the map **Fn** is now a random permutation of D to D. So $\mathbf{Fn}(x)$ assumes, with equal probability, any value in D except $y_1, \ldots, y_q$, meaning there are at least $|D| - q$ things it could be. (Remember $R = D$ in this case.) ▌

The following example illustrates that the converse of the above claim is far from true. The kr-advantage of a family can be significantly smaller than its prf or prp-cpa advantage, meaning that a family might be very secure against key recovery yet very insecure as a prf or prp, and thus not useful for protocol design.

Example 3.8.4 Define the blockcipher $E\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$ by $E_K(x) = x$ for all k-bit keys K and all ℓ-bit inputs x. We claim that it is very secure against key-recovery but very insecure as a PRP under CPA. More precisely, we claim that for any adversary B,

$$\mathbf{Adv}^{\mathrm{kr}}_E(B) \;=\; 2^{-k}\;,$$

regardless of the running time and number of queries made by B. On the other hand there is an adversary A, making only one oracle query and having a very small running time, such that

$$\mathbf{Adv}^{\text{prp-cpa}}_E(A) \;\geq\; 1 - 2^{-\ell}\;.$$

In other words, given an oracle for E_K, you may make as many queries as you want, and spend as much time as you like, before outputting your guess as to the value of K, yet your chance of getting it right is only 2^{-k}. On the other hand, using only a single query to a given oracle $\mathbf{Fn}\colon \{0,1\}^\ell \to \{0,1\}^\ell$, and very little time, you can tell almost with certainty whether **Fn** is an instance of E or is a random function of ℓ bits to ℓ bits. Why are these claims true? Since E_K does not depend on K, an adversary with oracle E_K gets no information about K by querying it, and hence its guess as to the value of K can be correct only with probability 2^{-k}. On the other hand, an adversary can test whether $\mathbf{Fn}(0^\ell) = 0^\ell$, and by returning 1 if and only if this is true, attain a prp-advantage of $1 - 2^{-\ell}$. ▌

3.9 The birthday attack

Suppose $E\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$ is a family of permutations, meaning a blockcipher. If we are given an oracle $\mathbf{Fn}\colon \{0,1\}^\ell \to \{0,1\}^\ell$ which is either an instance of E or a random function, there is a simple test to determine which of these it is. Query the oracle at distinct points $x_1, x_2, \ldots, x_q$, and get back values $y_1, y_2, \ldots, y_q$. You know that if **Fn** were a permutation, the values $y_1, y_2, \ldots, y_q$ must be distinct. If **Fn** was a random function, they may or may not be distinct. So, if they are distinct, bet on a permutation.

Surprisingly, this is pretty good adversary, as we will argue below. Roughly, it takes $q = \sqrt{2^\ell}$ queries to get an advantage that is quite close to 1. The reason is the birthday paradox. If you are not familiar with this, you may want to look at the appendix on the birthday problem and then come back to the following.

This tells us that an instance of a blockcipher can be distinguished from a random function based on seeing a number of input-output examples which is approximately $2^{\ell/2}$. This has important consequences for the security of blockcipher based protocols.

Proposition 3.9.1 Let $E\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a family of permutations. Suppose q satisfies $2 \leq q \leq 2^{(\ell+1)/2}$. Then there is an adversary A, making q oracle queries and having running time about that to do q computations of E, such that

$$\mathbf{Adv}^{\mathrm{prf}}_E(A) \;\;\geq\;\; 0.3 \cdot \frac{q(q-1)}{2^\ell}\;. \;\blacksquare \tag{3.5}$$

Proof of Proposition 3.9.1: Adversary A is given an oracle $\mathbf{Fn}$: $\{0,1\}^\ell \to \{0,1\}^\ell$ and works like this:

Adversary A
for $i = 1, \ldots, q$ do
    Let x_i be the i-th ℓ-bit string in lexicographic order
    $y_i \leftarrow \mathbf{Fn}(x_i)$
if $y_1, \ldots, y_q$ are all distinct then return 1, else return 0

Let us now justify Equation (3.5). Letting $N = 2^\ell$, we claim that

$$\Pr\left[\mathrm{Real}_E^A \Rightarrow 1\right] = 1 \tag{3.6}$$

$$\Pr\left[\mathrm{Rand}_E^A \Rightarrow 1\right] = 1 - C(N,q)\ . \tag{3.7}$$

Here $C(N,q)$, as defined in the appendix on the birthday problem, is the probability that some bin gets two or more balls in the experiment of randomly throwing q balls into N bins. We will justify these claims shortly, but first let us use them to conclude. Subtracting, we get

$$\begin{aligned}
\mathbf{Adv}_E^{\mathrm{prf}}(A) &= \Pr\left[\mathrm{Real}_E^A \Rightarrow 1\right] - \Pr\left[\mathrm{Rand}_E^A \Rightarrow 1\right] \\
&= 1 - [1 - C(N,q)] \\
&= C(N,q) \\
&\geq 0.3 \cdot \frac{q(q-1)}{2^\ell}\ .
\end{aligned}$$

The last line is by Theorem A.1 in the appendix on the birthday problem. It remains to justify Equations (3.6) and (3.7).

Equation (3.6) is clear because in the real world, $\mathbf{Fn} = E_K$ for some key K, and since E is a family of permutations, $\mathbf{Fn}$ is a permutation, and thus $y_1, \ldots, y_q$ are all distinct. Now, suppose A is in the random world, so that $\mathbf{Fn}$ is a random function of ℓ bits to ℓ bits. What is the probability that $y_1, \ldots, y_q$ are all distinct? Since $\mathbf{Fn}$ is a random function and $x_1, \ldots, x_q$ are distinct, $y_1, \ldots, y_q$ are random, independently distributed values in $\{0,1\}^\ell$. Thus we are looking at the birthday problem. We are throwing q balls into $N = 2^\ell$ bins and asking what is the probability of there being no collisions, meaning no bin contains two or more balls. This is $1 - C(N,q)$, justifying Equation (3.7). ▮

3.10 The PRP/PRF switching lemma

When we analyse blockcipher-based constructions, we find a curious dichotomy: PRPs are what most naturally model blockciphers, but analyses are often considerably simpler and more natural assuming the blockcipher is a PRF. To bridge the gap, we relate the prp-security of a blockcipher to its prf-security. The following says, roughly, these two measures are always close—they don't differ by more than the amount given by the birthday attack. Thus a particular family of permutations E may have prf-advantage that exceeds its prp-advantage, but not by more than $0.5\, q^2/2^n$.

Lemma 3.10.1 [PRP/PRF Switching Lemma] Let $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a function family. Let $R = \{0,1\}^n$. Let A be an adversary that asks at most q oracle queries. Then

$$\left|\Pr\left[\mathrm{Rand}_R^A \Rightarrow 1\right] - \Pr\left[\mathrm{Perm}_R^A \Rightarrow 1\right]\right| \;\leq\; \frac{q(q-1)}{2^{n+1}}\,. \tag{3.8}$$

As a consequence, we have that

$$\left|\mathbf{Adv}_E^{\mathrm{prf}}(A) - \mathbf{Adv}_E^{\mathrm{prp}}(A)\right| \;\leq\; \frac{q(q-1)}{2^{n+1}}\,. \blacksquare \tag{3.9}$$

The proof introduces a technique that we shall use repeatedly: a *game-playing argument*. We are trying to compare what happens when an adversary A interacts with one kind of object—a random permutation oracle—to what happens when the adversary interacts with a different kind of object—a random function oracle. So we set up each of these two interactions as a kind of game, writing out the game in pseudocode. The two games are written in a way that highlights when they have differing behaviors. In particular, any time that the behavior in the two games differ, we set a flag bad. The probability that the flag bad gets set in one of the two games is then used to bound the difference between the probability that the adversary outputs 1 in one game and the the probability that the adversary outputs 1 in the other game.

Proof: Let's begin with Equation (3.8), as Equation (3.9) follows from that. We need to establish that

$$-\frac{q(q-1)}{2^{n+1}} \;\leq\; \Pr\left[\mathrm{Rand}_R^A \Rightarrow 1\right] - \Pr\left[\mathrm{Perm}_R^A \Rightarrow 1\right] \;\leq\; \frac{q(q-1)}{2^{n+1}}$$

Let's show the right-hand inequality, since the left-hand inequality works in exactly the same way. So we are trying to establish that

$$\Pr[A^{\rho} \Rightarrow 1] - \Pr[A^{\pi} \Rightarrow 1] \;\leq\; \frac{q(q-1)}{2^{n+1}}\,. \tag{3.10}$$

We can assume that A never asks an oracle query that is not an n-bit string. You can assume that such an *invalid* oracle query would generate an error message. The same error message would be generated on any invalid query, regardless of A's oracle, so asking invalid queries is pointless for A.

We can also assume that A never *repeats* an oracle query: if it asks a question X it won't later ask the same question X. It's not interesting for A to repeat a question, because it's going to get the same answer as before, independent of the type of oracle to which A is speaking to. More precisely, with a little bit of bookkeeping the adversary can remember what was its answer to each oracle query it already asked, and it doesn't have to repeat an oracle query because the adversary can just as well look up the prior answer.

Let's look at Games G_0 and G_1 of Fig. 3.5. Notice that the adversary never sees the flag bad. The flag bad will play a central part in our analysis, but it is not something that the adversary A can get hold of. It's only for our bookkeeping.

Suppose that the adversary asks a query X. By our assumptions about A, the string X is an n-bit string that the adversary has not yet asked about. In line 10, we choose a random n-bit string Y. Lines 11,12, next, are the most interesting. If the point Y that we just chose is already in the range of the function we are defining then we set a flag bad. In such a case, if we are playing game G_0, then we now make a *fresh* choice of Y, this time from the co-range of the function. If we are playing game G_1 then we stick with our original choice of Y. Either way, we return Y, effectively growing the domain of our function.

procedure Initialize // $\boxed{G_0}$, G_1
$\mathsf{UR} \leftarrow \emptyset$

procedure Fn(x)
10 $Y \xleftarrow{\$} R$
11 if $Y \in \mathsf{UR}$ then
12 $\quad \mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{Y \xleftarrow{\$} R \setminus \mathsf{UR}}$
13 $\mathsf{UR} \leftarrow \mathsf{UR} \cup \{Y\}$
14 return Y

Figure 3.5: Games used in the proof of the Switching Lemma. Game G_0 includes the boxed code while game G_1 does not.

Now let's think about what A sees as it plays Game G_1. Whatever query X is asked, we just return a random n-bit string Y. So game G_1 perfectly simulates a random function. Remember that the adversary isn't allowed to repeat a query, so what the adversary would get if it had a random function oracle is a random n-bit string in response to each query—just what we are giving it. Hence

$$\Pr[\mathrm{Rand}_R^A \Rightarrow 1] \quad = \quad \Pr[G_1 \Rightarrow 1] \tag{3.11}$$

Now if we're in game G_0 then what the adversary gets in response to each query X is a random point Y that has not already been returned to A. Thus

$$\Pr[\mathrm{Perm}_R^A \Rightarrow 1] \quad = \quad \Pr[G_0^A \Rightarrow 1] \; . \tag{3.12}$$

But game G_0, G_1 are identical until bad and hence the Fundamental Lemma of game playing implies that

$$\Pr[G_0^A \Rightarrow 1] - \Pr[G_1^A \Rightarrow 1] \quad \leq \quad \Pr[G_1^A \text{ sets } \mathsf{bad}] \; . \tag{3.13}$$

To bound $\Pr[G_1^A \text{ sets } \mathsf{bad}]$ is simple. Line 11 is executed q times. The first time it is executed UR contains 0 points; the second time it is executed UR contains 1 point; the third time it is executed $\mathrm{Range}(\pi)$ contains at most 2 points; and so forth. Each time line 11 is executed we have just selected a random value Y that is independent of the contents of UR. By the sum bound, the probability that a Y will ever be in UR at line 11 is therefore at most $0/2^n + 1/2^n + 2/2^n + \cdots + (q-1)/2^n = (1 + 2 + \cdots + (q-1))/2^n = q(q-1)/2^{n+1}$. This completes the proof of Equation (3.10). To go on and show that $\mathbf{Adv}_E^{\mathrm{prf}}(A) - \mathbf{Adv}_E^{\mathrm{prp}}(A) \leq q(q-1)/2^{n+1}$ note that

$$\begin{aligned}
\mathbf{Adv}_E^{\mathrm{prf}}(A) - \mathbf{Adv}_E^{\mathrm{prp}}(A) &= \Pr\left[\mathrm{Real}_F^A \Rightarrow 1\right] - \Pr\left[\mathrm{Rand}_R^A \Rightarrow 1\right] - \left(\Pr\left[\mathrm{Real}_F^A \Rightarrow 1\right] - \Pr\left[\mathrm{Perm}_R^A \Rightarrow 1\right]\right) \\
&= \Pr\left[\mathrm{Perm}_R^A \Rightarrow 1\right] - \Pr\left[\mathrm{Rand}_R^A \Rightarrow 1\right] \\
&\leq q(q-1)/2^{n+1}
\end{aligned}$$

This completes the proof. ▮

The PRP/PRF switching lemma is one of the central tools for understanding block-cipher based protocols, and the game-playing method will be one of our central techniques for doing proofs.

3.11 Historical notes

The concept of pseudorandom functions is due to Goldreich, Goldwasser and Micali [3], while that of pseudorandom permutation is due to Luby and Rackoff [4]. These works are however in the complexity-theoretic or "asymptotic" setting, where one considers an infinite sequence of families rather than just one family, and defines security by saying that polynomial-time adversaries have "negligible" advantage. In contrast our approach is motivated by the desire to model blockciphers and is called the "concrete security" approach. It originates with [2]. Definitions 3.4.1 and 3.5.1 are from [2], as are Propositions 3.9.1 and 3.10.1.

3.12 Problems

Problem 1 Let E: $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a secure PRP. Consider the family of permutations E': $\{0,1\}^k \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ defined by for all $x, x' \in \{0,1\}^n$ by

$$E'_K(x \,\|\, x') = E_K(x) \,\|\, E_K(x \oplus x') \;.$$

Show that E' is not a secure PRP. ▌

Problem 2 Consider the following blockcipher $E : \{0,1\}^3 \times \{0,1\}^2 \to \{0,1\}^2$:

key	0	1	2	3
0	0	1	2	3
1	3	0	1	2
2	2	3	0	1
3	1	2	3	0
4	0	3	2	1
5	1	0	3	2
6	2	1	0	3
7	3	2	1	0

(The eight possible keys are the eight rows, and each row shows where the points to which 0, 1, 2, and 3 map.) Compute the maximal prp-advantage an adversary can get (a) with one query, (b) with four queries, and (c) with two queries. ▌

Problem 3 Present a secure construction for the problem of Example 3.7.2. That is, given a PRF F: $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, construct a PRF G: $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^{2n}$ which is a secure PRF as long as F is secure. ▌

Problem 4 Design a blockcipher $E : \{0,1\}^k \times \{0,1\}^{128} \to \{0,1\}^{128}$ that is secure (up to a large number of queries) against non-adaptive adversaries, but is completely insecure (even for two queries) against an adaptive adversary. (A non-adaptive adversary readies all her questions $M_1, \ldots, M_q$, in advance, getting back $E_K(M_1), ..., E_K(M_q)$. An adaptive adversary is the sort we have dealt with throughout: each query may depend on prior answers.) ▌

Problem 5 Let $a[i]$ denote the i-th bit of a binary string i, where $1 \le i \le |a|$. The *inner product* of n-bit binary strings a, b is

$$\langle\, a, b \,\rangle \;=\; a[1]b[1] \oplus a[2]b[2] \oplus \cdots \oplus a[n]b[n] \;.$$

Game G	Game H
procedure Initialize	**procedure Initialize**
$K \xleftarrow{\$} \mathsf{Keys}(F)$	$K_1 \xleftarrow{\$} \mathsf{Keys}(F)$; $K_2 \xleftarrow{\$} \mathsf{Keys}(F)$
procedure $f(x)$	**procedure** $f(x)$
Return $F_K(x)$	Return $F_{K_1}(x)$
procedure $g(x)$	**procedure** $g(x)$
Return $F_K(x)$	Return $F_{K_2}(x)$

Figure 3.6: Game used to in Problem 7.

A family of functions $F\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^L$ is said to be *inner-product preserving* if for every $K \in \{0,1\}^k$ and every distinct $x_1, x_2 \in \{0,1\}^\ell - \{0^\ell\}$ we have

$$\langle\, F(K,x_1), F(K,x_2)\,\rangle \;=\; \langle\, x_1, x_2\,\rangle\;.$$

Prove that if F is inner-product preserving then there exists an adversary A, making at most two oracle queries and having running time $2 \cdot T_F + O(\ell)$, where T_F denotes the time to perform one computation of F, such that

$$\mathbf{Adv}^{\mathrm{prf}}_F(A) \geq \frac{1}{2} \cdot \left(1 + \frac{1}{2^L}\right)\;.$$

Explain in a sentence why this shows that if F is inner-product preserving then F is not a secure PRF. ▌

Problem 6 Let $E\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a blockcipher. The *two-fold cascade* of E is the blockcipher $E^{(2)}\colon \{0,1\}^{2k} \times \{0,1\}^\ell \to \{0,1\}^\ell$ defined by

$$E^{(2)}_{K_1 \,\|\, K_2}(x) = E_{K_1}(E_{K_2}(x))$$

for all $K_1, K_2 \in \{0,1\}^k$ and all $x \in \{0,1\}^\ell$. Prove that if E is a secure PRP then so is $E^{(2)}$. ▌

Problem 7 Let A be a adversary that makes at most q total queries to its two oracles, f and g, where $f, g : \{0,1\}^n \to \{0,1\}^n$. Assume that A never asks the same query X to both of its oracles. Define

$$\mathbf{Adv}(A) = \Pr[G^A = 1] - \Pr[H^A = 1]$$

where games G, H are defined in Fig. 3.6. Prove a good upper bound for $\mathbf{Adv}(A)$, say $\mathbf{Adv}(A) \leq q^2/2^n$. ▌

Problem 8 Let $F\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a family of functions and $r \geq 1$ an integer. The *r-round Feistel cipher associated to F* is the family of permutations $F^{(r)}\colon \{0,1\}^{rk} \times \{0,1\}^{2\ell} \to \{0,1\}^{2\ell}$ defined as follows for any $K_1, \ldots, K_r \in \{0,1\}^k$ and input $x \in \{0,1\}^{2\ell}$:

Function $F^{(r)}(K_1 \,\|\, \cdots \,\|\, K_r, x)$
  Parse x as $L_0 \,\|\, R_0$ with $|L_0| = |R_0| = \ell$
  For $i = 1, \ldots, r$ do

$L_i \leftarrow R_{i-1}$; $R_i \leftarrow F(K_i, R_{i-1}) \oplus L_{i-1}$
EndFor
Return $L_r \,\|\, R_r$

(a) Prove that there exists an adversary A, making at most two oracle queries and having running time about that to do two computations of F, such that

$$\mathbf{Adv}^{\mathrm{prf}}_{F^{(2)}}(A) \geq 1 - 2^{-\ell} \,.$$

(b) Prove that there exists an adversary A, making at most two queries to its first oracle and one to its second oracle, and having running time about that to do three computations of F or F^{-1}, such that

$$\mathbf{Adv}^{\text{prp-cca}}_{F^{(3)}}(A) \geq 1 - 3 \cdot 2^{-\ell} \,. \blacksquare$$

Problem 9 Let $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a function family and let A be an adversary that asks at most q queries. In trying to construct a proof that $|\mathbf{Adv}^{\mathrm{prp}}_E(A) - \mathbf{Adv}^{\mathrm{prf}}_E(A)| \leq q^2/2^{n+1}$, Michael and Peter put forward an argument a fragment of which is as follows:

> Consider an adversary A that asks at most q oracle queries to an oracle **Fn** for a function from R to R, where $R = \{0,1\}^n$. Let C (for "collision") be the event that A asks some two distinct queries X and X' and the oracle returns the same answer. Then clearly
>
> $$\Pr[\mathrm{Perm}^A_R \Rightarrow 1] = \Pr[\mathrm{Rand}^A_R \Rightarrow 1 \mid \overline{\mathsf{C}}].$$

Show that Michael and Peter have it all wrong: prove that the quantities above are not necessarily equal. Do this by selecting a number n and constructing an adversary A for which the left and right sides of the equation above are unequal. $\blacksquare$

Bibliography

[1] M. Bellare and P. Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. *Advances in Cryptology – EUROCRYPT '06*, Lecture Notes in Computer Science Vol. , ed., Springer-Verlag, 2006

[2] M. Bellare, J. Kilian and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences* , Vol. 61, No. 3, Dec 2000, pp. 362–399.

[3] O. Goldreich, S. Goldwasser and S. Micali. How to construct random functions. *Journal of the ACM,* Vol. 33, No. 4, 1986, pp. 210–217.

[4] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput,* Vol. 17, No. 2, April 1988.