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 block ciphers, and they thereby enable the security analysis of protocols based on block ciphers. But PRFs and PRPs are also a useful conceptual starting point in contexts where block ciphers 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 $F\colon \mathcal{K} \times D \times R$. 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* on strings is a map whose domain and range are the same set, and the map is a length-preserving bijection on this set. That is, a map π: $D \to D$ is a permutation if $|\pi(x)| = |x|$ for all $x \in D$ and also π is one-to-one and onto. 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 A block cipher is a family of permutations. In particular DES is a family of permutations DES: $\mathcal{K} \times D \times R$ 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 AES: $\mathcal{K} \times D \times R$ with

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

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

3.2 Random functions and permutations

Let $D, R \subseteq \{0,1\}^*$ be finite nonempty sets and let $\ell, L \geq 1$ be integers. There are two function families that we fix. One is $\mathsf{Rand}(D,R)$, the family of all functions of D to R. The other is $\mathsf{Perm}(D)$, the family of all permutations on D. For compactness of notation we let $\mathsf{Rand}(\ell,L)$, $\mathsf{Rand}(\ell)$, and $\mathsf{Perm}(\ell)$ denote $\mathsf{Rand}(D,R)$, $\mathsf{Rand}(D,D)$, and $\mathsf{Perm}(D)$, where $D = \{0,1\}^\ell$ and $R = \{0,1\}^L$.

What are these families? The family $\mathsf{Rand}(D,R)$ has domain D and range R, while the family $\mathsf{Perm}(D)$ has domain and range D. The set of instances of $\mathsf{Rand}(D,R)$ is the set of all functions mapping D to R, while the set of instances of $\mathsf{Perm}(D)$ is the set of all permutations on D. The key describing any particular instance function is simply a description of this instance function in some canonical notation. For example, order the domain D lexicographically as $X_1, X_2, \ldots$, and then let the key for a function f be the list of values $(f(X_1), f(X_2), \ldots)$. The key-space of $\mathsf{Rand}(D,R)$ is simply the set of all these keys, under the uniform distribution.

Let us illustrate in more detail some of the cases in which we are most interested. The key of a function in $\mathsf{Rand}(\ell,L)$ is simply a list of of all the output values of the function as its input ranges over $\{0,1\}^l$. Thus

$$\mathsf{Keys}(\mathsf{Rand}(\ell,L)) \;=\; \{\, (Y_1, \ldots, Y_{2^\ell}) \,:\, Y_1, \ldots, Y_{2^\ell} \in \{0,1\}^L \,\}$$

is the set of all sequences of length 2^ℓ in which each entry of a sequence is an L-bit string. For any $x \in \{0,1\}^\ell$ we interpret X as an integer in the range $\{1, \ldots, 2^\ell\}$ and set

$$\mathsf{Rand}(\ell,L)((Y_1, \ldots, Y_{2^\ell}), X) = Y_X .$$

Notice that the key space is very large; it has size 2^{L2^ℓ}. There is a key for every function of ℓ-bits to L-bits, and this is the number of such functions. The key space is equipped with the uniform distribution, so that $f \xleftarrow{\$} \mathsf{Rand}(\ell,L)$ is the operation of picking a random function of ℓ-bits to L-bits.

On the other hand, for $\mathsf{Perm}(\ell)$, the key space is

$$\begin{aligned}\mathsf{Keys}(\mathsf{Perm}(\ell)) \;&=\; \{(Y_1,\ldots,Y_{2^\ell}) :\;\; Y_1,\ldots,Y_{2^\ell} \in \{0,1\}^\ell \text{ and} \\ &\qquad Y_1,\ldots,Y_{2^\ell} \text{ are all distinct}\}\ .\end{aligned}$$

For any $X \in \{0,1\}^\ell$ we interpret X as an integer in the range $\{1,\ldots,2^\ell\}$ and set

$$\mathsf{Perm}(\ell)((Y_1,\ldots,Y_{2^\ell}),X) = Y_X\ .$$

The key space is again equipped with the uniform distribution, so that $f \xleftarrow{\$} \mathsf{Perm}(l)$ is the operation of picking a random permutation on $\{0,1\}^\ell$. In other words, all the possible permutations on $\{0,1\}^\ell$ are equally likely.

Example 3.2 We exemplify $\mathsf{Rand}(3,2)$, meaning $\ell = 3$ and $L = 2$. The domain is $\{0,1\}^3$ and the range is $\{0,1\}^2$. An example instance f of the family is illustrated below via its input-output table:

x	000	001	010	011	100	101	110	111
$f(x)$	10	11	01	11	10	00	00	10

The key corresponding to this particular function is

$$(10,11,01,11,10,00,00,10)\ .$$

The key-space of $\mathsf{Rand}(3,2)$ is the set of all such sequences, meaning the set of all 8-tuples each component of which is a two bit string. There are

$$2^{2\cdot 2^3} \;=\; 2^{16} \;=\; 65,536$$

such tuples, so this is the size of the key-space. ▌

Example 3.3 We exemplify $\mathsf{Perm}(3)$, meaning $\ell = 3$. The domain and range are both $\{0,1\}^3$. An example instance f of the family is illustrated below via its input-output table:

x	000	001	010	011	100	101	110	111
$f(x)$	010	111	101	011	110	100	000	001

The function f is a permutation because each 3-bit string occurs exactly once in the second row of the table. The key corresponding to this particular permutation is

$$(010,111,101,011,110,100,000,001)\ .$$

The key-space of $\mathsf{Perm}(3)$ is the set of all such sequences, meaning the set of all 8-tuples whose components consist of all 3-bit strings in some order. There are

$$8! \;=\; 40,320$$

such tuples, so this is the size of the key-space. ▌

We will hardly ever actually think about these families in terms of this formalism. Indeed, it is worth pausing here to see how to think about them more intuitively, because they are important objects.

We will consider settings in which you have black-box access to a function g. This means that there is a box to which you can give any value X of your choice (provided X is in the domain of g), and the box gives you back $g(X)$. But you can't "look inside" the box; your only interface to it is the one we have specified.

A random function g: $\{0,1\}^\ell \to \{0,1\}^L$ being placed in this box corresponds to the following. Each time you give the box an input, you get back a random L-bit string, with the sole constraint that if you twice give the box the same input X, it will be consistent, returning both times the same output $g(X)$. In other words, a random function of ℓ-bits to L-bits can be thought of as a box which given any input $X \in \{0,1\}^\ell$ returns a random number, except that if you give it an input you already gave it before, it returns the same thing as last time.

On the other hand, a random permutation g: $\{0,1\}^\ell \to \{0,1\}^\ell$ being placed in this box corresponds to the following. Each time you give the box an input, you get back a random ℓ-bit string, with two constraints: that if you twice give the box the same input X, it will be consistent, returning both times the same output $g(X)$, and that the box never returns the same output for two different inputs.

It is this "dynamic" view that we suggest the reader have in mind when thinking about random functions or random permutations.

The dynamic view of a random function can be thought of as implemented by the following computer program. The program 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 empty. The program processes an input $X \in \{0,1\}^\ell$ as follows:

```
If T[X] is not yet defined then
    Pick at random a string Y ∈ {0,1}^L and set T[X] ← Y
EndIf
Return T[X]
```

The answer on any point is random and independent of the answers on other points.

The dynamic view of a random *permutation* can be thought of as implemented by the following computer program. The program 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 empty, and the set R below is also empty. The program processes an input $X \in \{0,1\}^\ell$ as follows:

```
If T[X] is not yet defined then
    Pick at random a string Y ∈ {0,1}^ℓ − R and set T[X] ← Y
    R ← R ∪ {T[X]}
EndIf
Return T[X]
```

The answer on any point is random, but not independent of the answers on other points, since it is distinct from those.

Another way to think about a random function is as a large, pre-determined random table. The entries are of the form (X, Y). For each X someone has flipped coins to determine Y and put it into the table. For a random permutation, the entries have the property that if (X_1, Y_1) and (X_2, Y_2) are in the table then $Y_1 \neq Y_2$.

One must remember that the terms "random function" or "random permutation" are misleading. They 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 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 3.4 Let's do some simple probabilistic computations to understand random functions. In all of the following, the probability is taken over a random choice of f from $\mathsf{Rand}(\ell,L)$, meaning that we have executed the operation $f \xleftarrow{\$} \mathsf{Rand}(\ell,L)$.

1. Fix $X \in \{0,1\}^\ell$ and $Y \in \{0,1\}^L$. Then:

$$\Pr[f(X) = Y] = 2^{-L} .$$

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 $Y_1, Y_2 \in \{0,1\}^L$, and assume $X_1 \neq X_2$. Then

$$\Pr[f(X_1) = Y_1 \mid f(X_2) = Y_2] \;=\; 2^{-L} .$$

The above is a conditional probability, and says that even if we know the value of f on X_1, its value on a different point X_2 is equally likely to be any L-bit string.

3. Fix $X_1, X_2 \in \{0,1\}^\ell$ and $Y \in \{0,1\}^L$. Then:

$$\Pr[f(X_1) = Y \text{ and } f(X_2) = Y] = \begin{cases} 2^{-2L} & \text{if } X_1 \neq X_2 \\ 2^{-L} & \text{if } X_1 = X_2 \end{cases}$$

4. Fix $X_1, X_2 \in \{0,1\}^\ell$ and $Y \in \{0,1\}^L$. Then:

$$\Pr[f(X_1) \oplus f(X_2) = Y] = \begin{cases} 2^{-L} & \text{if } X_1 \neq X_2 \\ 0 & \text{if } X_1 = X_2 \text{ and } Y \neq 0^L \\ 1 & \text{if } X_1 = X_2 \text{ and } Y = 0^L \end{cases}$$

5. Assume L is even, say $L = 2l$, and let τ: $\{0,1\}^\ell \to \{0,1\}^l$ denote the function that on input $X \in \{0,1\}^\ell$ returns the first l bits of $f(X)$. Fix distinct $X_1, X_2 \in$

$\{0,1\}^\ell$, $Y_1 \in \{0,1\}^L$ and $Z_2 \in \{0,1\}^l$. Then:

$$\Pr[\tau(X_2) = Z_2 \mid f(X_1) = Y_1] = 2^{-l} .$$

∎

Example 3.5 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. In all of the following, the probability is taken over a random choice of π from $\mathsf{Perm}(\ell)$, meaning that we have executed the operation $\pi \xleftarrow{\$} \mathsf{Perm}(\ell)$.

1. Fix $X, Y \in \{0,1\}^\ell$. Then:

$$\Pr[\pi(X) = Y] = 2^{-\ell} .$$

This is the same as if π had been selected at random from $\mathsf{Rand}(\ell,\ell)$ rather than from $\mathsf{Perm}(\ell)$. However, the similarity vanishes when more than one point is to be considered.

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

$$\Pr[\pi(X_1) = Y_1 \mid \pi(X_2) = Y_2] = \begin{cases} \dfrac{1}{2^\ell - 1} & \text{if } Y_1 \neq Y_2 \\ 0 & \text{if } Y_1 = Y_2 \end{cases}$$

The above is a conditional probability, and says that if we know the value of π on X_1, its value on a different point X_2 is equally likely to be any L-bit string other than $\pi(X_1)$. So there are $2^\ell - 1$ choices for $\pi(X_2)$, all equally likely, if $Y_1 \neq Y_2$.

It is important with regard to understanding the concepts to realize that the above probability is not $2^{-\ell}$ but slightly more, but from the practical point of view there is little difference. In practice ℓ is large, at least 64, and $2^{-\ell}$ and $1/(2^\ell - 1)$ are too close to each other for any important difference to manifest itself. As we will see when we consider birthday attacks, however, when more points are considered, the difference between functions and permutations becomes more manifest.

3. Fix $X_1, X_2 \in \{0,1\}^\ell$ and $Y \in \{0,1\}^L$. Then:

$$\Pr[\pi(X_1) = Y \text{ and } \pi(X_2) = Y] = \begin{cases} 0 & \text{if } X_1 \neq X_2 \\ 2^{-\ell} & \text{if } X_1 = X_2 \end{cases}$$

This is true because a permutation can never map distinct X_1 and X_2 to the same point.

4. Fix $X_1, X_2 \in \{0,1\}^\ell$ and $Y \in \{0,1\}^\ell$. Then:

$$\Pr[\pi(X_1) \oplus \pi(X_2) = Y] = \begin{cases} \frac{1}{2^\ell - 1} & \text{if } X_1 \neq X_2 \text{ and } Y \neq 0^\ell \\ 0 & \text{if } X_1 \neq X_2 \text{ and } Y = 0^\ell \\ 0 & \text{if } X_1 = X_2 \text{ and } Y \neq 0^\ell \\ 1 & \text{if } X_1 = X_2 \text{ and } Y = 0^\ell \end{cases}$$

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

$$\begin{aligned} &\Pr[\pi(X_1) \oplus \pi(X_2) = Y] \\ &= \sum_{Y_1} \Pr[\pi(X_2) = Y_1 \oplus Y \mid \pi(X_1) = Y_1] \cdot \Pr[\pi(X_1) = Y_1] \\ &= \sum_{Y_1} \frac{1}{2^\ell - 1} \cdot \frac{1}{2^\ell} \\ &= 2^\ell \cdot \frac{1}{2^\ell - 1} \cdot \frac{1}{2^\ell} \\ &= \frac{1}{2^\ell - 1}\,. \end{aligned}$$

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

5. Assume ℓ is even, say $\ell = 2l$, and let τ: $\{0,1\}^\ell \to \{0,1\}^l$ denote the function that on input $X \in \{0,1\}^\ell$ returns the first l bits of $\pi(X)$. (Note that although π is a permutation, τ is not.) Fix distinct $X_1, X_2 \in \{0,1\}^\ell$, $Y_1 \in \{0,1\}^L$ and $Z_2 \in \{0,1\}^l$. Let $Y[1]$ denote the first l bits of an ℓ bit string Y. Then:

$$\Pr[\tau(X_2) = Z_2 \mid \pi(X_1) = Y_1] = \begin{cases} \frac{2^l}{2^\ell - 1} & \text{if } Z_2 \neq Y_1[2] \\ \frac{2^l - 1}{2^\ell - 1} & \text{if } Z_2 = Y_1[2] \end{cases}$$

This is computed as follows. Let

$$S \;=\; \{\, Y_2 \in \{0,1\}^\ell \;:\; Y_2[1] = Z_2 \text{ and } Y_2 \neq Y_1 \,\}\,.$$

We note that $|S| = 2^l$ if $Y_1[1] \neq Z_2$ and $|S| = 2^l - 1$ if $Y_1[1] = Z_2$. Then

$$\begin{aligned} \Pr[\tau(X_2) = Z_2 \mid \pi(X_1) = Y_1] &= \sum_{Y_2 \in S} \Pr[\pi(X_2) = Y_2 \mid \pi(X_1) = Y_1] \\ &= |S| \cdot \frac{1}{2^\ell - 1}\,, \end{aligned}$$

and the claim follows from what we said about the size of S.

∎

3.3 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 definition of this notion. Later we will look at motivation and applications.

We fix a family of functions $F\colon\ \mathcal{K}\times D \to R$. (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 strings from the domain D, and the answers you get back will be strings from the range R. The computer outside your room implements a function $g\colon\ D \to R$, so that whenever you type a value X you get back $g(X)$. However, your only access to g is via this interface, so the only thing you can see is the input-output behavior of g. We consider two different ways in which g will be chosen, giving rise to two different "worlds."

World 0: The function g is drawn at random from $\mathsf{Rand}(D,R)$, namely, the function g is selected by the experiment $g \xleftarrow{\$} \mathsf{Rand}(D,R)$.

World 1: The function g is drawn at random from F, namely, the function g is selected by the experiment $g \xleftarrow{\$} F$. (This means that a key is chosen via $K \xleftarrow{\$} \mathcal{K}$ and then g is set to F_K.)

You are not told which of the two worlds was chosen. The choice of world, and of the corresponding function g, 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 $g(X)$. 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 World 0 or in World 1.

The act of trying to tell which world you are in is formalized via the notion of a *distinguisher*. This is an algorithm that is provided oracle access to a function g and tries to decide if g is random or pseudorandom (that is, whether it is in world 0 or world 1). A distinguisher 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. We write A^g to mean that distinguisher A is being given oracle access to function g. Intuitively, a family is pseudorandom if the probability that the distinguisher says 1 is roughly the same regardless of which world it is in. We capture this mathematically below. Further explanations follow the definition.

Definition 3.6 Let $F\colon \mathcal{K} \times D \to R$ be a family of functions, and let A be an algorithm that takes an oracle for a function $g\colon D \to R$, and returns a bit. We consider two experiments:

Experiment $\mathbf{Exmt}_F^{\text{prf-1}}(A)$	Experiment $\mathbf{Exmt}_F^{\text{prf-0}}(A)$
$K \xleftarrow{\$} \mathcal{K}$	$g \xleftarrow{\$} \mathsf{Rand}(D,R)$
$b \xleftarrow{\$} A^{F_K}$	$b \xleftarrow{\$} A^{g}$
Return b	Return b

The *prf-advantage* of A is defined as

$$\mathbf{Adv}_F^{\text{prf}}(A) \;=\; \Pr\left[\mathbf{Exmt}_F^{\text{prf-1}}(A) = 1\right] - \Pr\left[\mathbf{Exmt}_F^{\text{prf-0}}(A) = 1\right] .$$

∎

The algorithm A models the person we were imagining in our room, trying to determine which world he or she was in by typing queries to the function g via a computer. In the formalization, the person is an algorithm, meaning a piece of code. This algorithm may be randomized. We formalize the ability to query g as giving A an oracle which takes input any string $X \in D$ and returns $g(X)$. 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 world 1; outputting the bit "0" means that A thinks it is in world 0.

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 $F(K, X)$.

The worlds are captured by what we call *experiments.* The first experiment picks a random instance F_K of family F and then runs adversary A with oracle $g = F_K$. Adversary A interacts with its oracle, querying it and getting back answers, and eventually outputs a "guess" bit. The experiment returns the same bit. The second experiment picks a random function $g\colon D \to R$ and runs A with this as oracle, again returning A's guess bit. Each experiment has a certain probability of returning 1. The probability is taken over the random choices made in the experiment. Thus, for the first experiment, 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 experiment, the probability is over the random choice of g and any random choices that A makes. These two probabilities should be evaluated separately; the two experiments are completely different.

To see how well A does at determining which world it is in, we look at the difference in the probabilities that the two experiments 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 experiment 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 distinguishers will have different advantages. There are two reasons why one distinguisher 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 you see more and more input-output examples of g, or spend more computing time, your ability to tell which world you are in should go up. The "security" of family F must thus be thought of as depending on the resources allowed to the attacker. We may want to want to know, for any given resource limitations, what is the prf-advantage achieved by the most "clever" distinguisher 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 experiments in the definition, plus the size of the code of A. In measuring the running time of the first experiment, 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 made by A to its oracle. In measuring the running time of the second experiment, we count the time to choose the random function g in a dynamic way, meaning we count the cost of maintaining a table of values of the form $(X, g(X))$. Entries are added to the table as g makes queries. A new entry is made by picking the output value at random.

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.

There is one feature of the above parameterization about which everyone asks. Suppose that F has key-length k. Obviously, the key length is a fundamental determinant of security: larger key length will typically mean more security. Yet, the key length k does not appear explicitly in the advantage function $\mathbf{Adv}^{\mathrm{prf}}_F(t, q, \mu)$. Why is this? The advantage function is in fact a function of k, but without knowing more about F it is difficult to know what kind of function. The truth is that the key length itself does not matter: what matters is just the advantage a distinguisher

can obtain. In a well-designed block cipher, one hopes that the maximal advantage an adversary can get if it runs in time t is about $t/2^k$. But that is really an ideal; in practice we should not assume ciphers are this good.

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 distinguisher; it only limits its resources. A distinguisher 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 3.6 does not have any explicit condition or statement regarding when F should be considered "secure." It only associates to F a prf-advantage function. Intuitively, F is "secure" if the value of the advantage function is "low" for "practical" values of the input parameters. This is, of course, not formal. It is possible to formalize the notion of a secure PRF using a complexity theoretic framework; one would say that the advantage of any adversary whose resources are polynomially-bounded is negligible. This requires an extension of the model to consider a security parameter in terms of which asymptotic estimates can be made. We will discuss this in more depth later, but for now we stick to a framework where the notion of what exactly is "secure" is not something binary. One reason is that this better reflects real life. 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 *experiments*, 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. Finally, there is an advantage function associated to the cryptographic protocol itself, taking as input resource parameters and returning the maximum possible probability of "breaking" the scheme if the attacker is restricted to those resource parameters. These three components will be present in all definitions. What varies is the experiments; this is here that we pin down how we measure security.

3.4 Pseudorandom permutations

Recall that a block cipher F is a family of permutations: each instance F_K of the family is a permutation. With the intent of modeling block ciphers we introduce the notion of a pseudorandom permutation. We proceed exactly as above, but replace $\mathsf{Rand}(D,R)$ with $\mathsf{Perm}(D)$.

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 g being tested. However when g is a permutation one can also consider the case where the adversary gets, in addition, an oracle for g^{-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.

3.4.1 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 g chosen in one of two ways.

World 0: The function g is drawn at random from $\mathsf{Perm}(D)$, namely, we choose g according to $g \xleftarrow{\$} \mathsf{Perm}(D)$.

World 1: The function g is drawn at random from F, namely $g \xleftarrow{\$} F$. (This means that a key is chosen via $K \xleftarrow{\$} \mathcal{K}$ and then g is set to F_K.)

Notice that World 1 is the same in the PRF setting, but World 0 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 g.

Definition 3.7 Let $F\colon \mathcal{K} \times D \to D$ be a family of functions, and let A be an algorithm that takes an oracle for a function $g\colon D \to D$, and returns a bit. We consider two experiments:

Experiment $\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A)$	Experiment $\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A)$
$K \xleftarrow{\$} \mathcal{K}$	$g \xleftarrow{\$} \mathsf{Perm}(D)$
$b \xleftarrow{\$} A^{F_K}$	$b \xleftarrow{\$} A^{g}$
Return b	Return b

The *prp-cpa-advantage* of A is defined as

$$\mathbf{Adv}_F^{\text{prp-cpa}}(A) \;=\; \Pr\left[\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A) = 1\right] - \Pr\left[\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A) = 1\right] .$$

∎

The intuition is similar to that for Definition 3.6. The difference is that here the "ideal" object that F is being compared with is no longer the family of random functions, but rather the family of random permutations.

Experiment $\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A)$ is actually identical to $\mathbf{Exmt}_F^{\text{prf-1}}(A)$. 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 experiment returns the same bit that A returns. In Experiment $\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A)$, a permutation $g\colon \{0,1\}^\ell \to \{0,1\}^\ell$ is chosen at random, and the result bit of A's computation with oracle g is returned. The probability is over the choice of g 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 experiments return 1.

Conventions regarding resource measures also remain the same as before. Informally, a family F is a secure PRP under CPA if $\mathbf{Adv}^{\text{prp-cpa}}_F(t, q, \mu)$ is "small" for "practical" values of the resource parameters.

3.4.2 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, g and its inverse g^{-1}. The manner in which g is chosen is the same as in the CPA case, and once g is chosen, g^{-1} is automatically defined, so we do not have to say how it is chosen.

World 0: The function g is drawn at random from $\mathsf{Perm}(D)$, namely via $g \xleftarrow{\$} \mathsf{Perm}(D)$. (So g is just a random permutation on D.)

World 1: The function g is drawn at random from F, namely $g \xleftarrow{\$} F$. (This means that a key is chosen via $K \xleftarrow{\$} \mathsf{Keys}(F)$ and then g is set to F_K.)

In World 1 we let $g^{-1} = F_K^{-1}$ be the inverse of the chosen instance, while in World 0 it is the inverse of the chosen random permutation. 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 3.8 Let $F\colon\ \mathcal{K} \times D \to D$ be a family of permutations, and let A be an algorithm that takes an oracle for a function $g\colon\ D \to D$, and also an oracle for the function $g^{-1}\colon\ D \to D$, and returns a bit. We consider two experiments:

Experiment $\mathbf{Exmt}^{\text{prp-cca-1}}_F(A)$	Experiment $\mathbf{Exmt}^{\text{prp-cca-0}}_F(A)$
$K \xleftarrow{\$} \mathcal{K}$	$g \xleftarrow{\$} \mathsf{Perm}(D)$
$b \xleftarrow{\$} A^{F_K, F_K^{-1}}$	$b \xleftarrow{\$} A^{g, g^{-1}}$
Return b	Return b

The *prp-cca-advantage* of A is defined as

$$\mathbf{Adv}^{\text{prp-cca}}_F(A) \;=\; \Pr\left[\mathbf{Exmt}^{\text{prp-cca-1}}_F(A) = 1\right] - \Pr\left[\mathbf{Exmt}^{\text{prp-cca-0}}_F(A) = 1\right] .$$

∎

The intuition is similar to that for Definition 3.6. The difference is that here the adversary has more power: not only can it query g, but it can directly query g^{-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.

3.4.3 Relations between the notions

If an adversary does not query g^{-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 3.9 [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 attacking) adversary. Suppose that A runs in time t, asks q queries, and these queries total μ bits. Then there exists a (PRP-CCA attacking) adversary B that runs in time t, asks q chosen-plaintext queries, these queries totaling μ bits, and asks no chosen-ciphertext queries, where

$$\mathbf{Adv}^{\text{prp-cca}}_F(B) \;\geq\; \mathbf{Adv}^{\text{prp-cpa}}_F(A)$$

∎

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 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.

3.5 Usage of PRFs and PRPs

We discuss some motivation for these notions of security.

3.5.1 The shared-random-function model

In symmetric (i.e., shared-key) cryptography, Alice and Bob share a key K that the adversary doesn't know. They want to use this key to achieve various things—in particular, to encrypt and authenticate the data they send to each other. A key is (or ought to be) a short string. Suppose however that we allow the parties a very long shared string—one that takes the form of a random function f of ℓ bits to L bits, for some pre-specified ℓ, L. This is called the *shared-random-function model*.

The shared-random-function model cannot really be realized in practice because the description of a random function is just too big to even store. It is a conceptual model. To work in this model, we give the parties oracle access to f. They may write down $x \in \{0,1\}^\ell$ and in one step be returned $f(x)$.

It turns out that the shared-random-function model is a very convenient one in which to think about cryptography, formulate schemes, and analyze them. In particular, we will see many examples where we design schemes in the shared random function model and prove them secure. This is true for a variety of problems, but most importantly for encryption and message authentication. The proof of security here is absolute: we do not make any restrictions on the computational power of the adversary, but are able to simply provide an upper bound on the success probability of the adversary.

As an example, consider the CTR mode of operation discussed in Section 2.4.3. Consider the version where the initial vector is a counter. Consider replacing every invocation of E_K with an invocation of the random function f (we assume here that $\ell = L$). In that case, the mode of operation turns into the one-time-pad cryptosystem. The shared random key is just the random function f. As we have discussed, this is well known to meet a strong and well-defined notion of security. So, in the shared-random-function model, CTR mode is is "good". Well, it would be, if we had yet defined what "good" means!

But now what? We have schemes which are secure but a priori can't be efficiently realized, since they rely on random functions. That's where pseudorandom function or permutation families come in. A PRF family is a family F of functions indexed by small keys (e.g., 56 or 128 bits). However, it has the property that if K is shared between Alice and Bob, and we use F_K in place of a random function f in some scheme designed in the shared-random-function model, the resulting scheme is still secure as long as the adversary is restricted in resource usage.

In other words, instances of PRFs can be used in place of random functions in shared-key schemes. The definition of a PRF is crafted to make this possible for as wide a range of applications as possible. An instance of a pseudorandom function is specified by a short key K, and the parties need only store this key. Then, they use this function in place of the random function in the scheme. And things should work out, in the sense that if the scheme was secure when a random function was used, it should still be secure.

This is a very rough idea. Technically, it is not always true: this is the intuition. Pseudorandom functions don't always work. That is, you can't substitute them for random functions in any usage of the latter and expect things to work out. But if used right, it works out in a large number of cases. How do we identify these cases? We have to resort to the formal definition of a pseudorandom function family and prove the security of our construct based on it. We will see how to do this later.

In this context we stress one important point. The security of a PRF relies on the key K being *secret*. The adversary is not given K and cannot directly compute the function. (Of course it might gain some information about values of F_K on various points via the usage of F_K by the legitimate parties, but that will be OK.) In other words, you can substitute *shared, secret* random functions by PRFs, but not public ones.

Pseudorandom functions are an intriguing notion and a powerful tool that en-

able the following design paradigm. When you want to design a scheme for encryption, authentication, or some other purpose, design it in the shared-random-function model. Then simply substitute the random function with a pseudorandom one, and your scheme should still be secure.

3.5.2 Modeling block ciphers

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

As discussed in Section 2.6, classically the security of DES or other block ciphers has been looked at only with regard to key recovery. That is, analysis of a block cipher 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 block cipher is taken as "secure" if the resources required to recover the key are prohibitive. Yet, as we saw, even a cursory glance at common block cipher usages shows that hardness of key recovery is not *sufficient* for security. We had discussed wanting a *master* security property of block ciphers under which natural usages of block ciphers could be proven secure. We suggest that this *master* property is that the block cipher be a secure PRP, under either CPA or CCA.

We cannot prove that specific block ciphers 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 block ciphers. For example we might conjecture something like:

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

if A is an adversary 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. 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

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

if B is an adversary 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 block ciphers as PRPs under CCA rather than CPA.

More interesting is $\mathbf{Adv}^{\text{prf}}_{\text{DES}}(t, q)$. Here we cannot do better than assume that

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

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

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 block cipher 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 block cipher is broken as a PRF then it should be considered insecure, and a replacement should be sought.

3.6 Example Attacks

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

Example 3.10 We define a family of functions $F\colon \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^L$ 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 g: $\{0,1\}^\ell \to \{0,1\}^L$ and will output a bit. Our adversary D works as follows:

Adversary D^g
  Let $Y \leftarrow g(0^\ell)$
  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 g was an instance of the family F, and if $y \neq 0^L$ it bets that g was a random function. Let us now see how well this adversary does. We claim that

$$\begin{aligned} \Pr\left[\mathbf{Exmt}_F^{\text{prf-1}}(D) = 1\right] &= 1 \\ \Pr\left[\mathbf{Exmt}_F^{\text{prf-0}}(D) = 1\right] &= 2^{-L} \ . \end{aligned}$$

Why? Look at Experiment $\mathbf{Exmt}_F^{\text{prf-1}}(D)$ as defined in Definition 3.6. Here $g = F_K$ for some K. In that case it is certainly true that $g(0^\ell) = 0^L$ so by the code we wrote for D the latter will return 1. On the other hand look at Experiment $\mathbf{Exmt}_F^{\text{prf-0}}(D)$ as defined in Definition 3.6. Here g is a random function. As we saw in Example 3.4, the probability that $g(0^\ell) = 0^L$ will be 2^{-L}, and hence this is the probability that D will return 1. Now as per Definition 3.6 we subtract to get

$$\begin{aligned} \mathbf{Adv}_F^{\text{prf}}(D) &= \Pr\left[\mathbf{Exmt}_F^{\text{prf-1}}(D) = 1\right] - \Pr\left[\mathbf{Exmt}_F^{\text{prf-0}}(D) = 1\right] \\ &= 1 - 2^{-L} \ . \end{aligned}$$

Now let t be the time complexity of D. 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 D is just one, and the total length of all queries is l. Thus we have

$$\begin{aligned} \mathbf{Adv}_F^{\text{prf}}(t, 1, \ell) &= \max_A \{\, \mathbf{Adv}_F^{\text{prf}}(A) \,\} \\ &\geq \mathbf{Adv}_F^{\text{prf}}(D) \\ &= 1 - 2^{-L} \ . \end{aligned}$$

The first inequality is true because the adversary D is one member of the set of adversaries A over which the maximum is taken, and hence the maximum advantage is at least that attained by D. Our conclusion is that the advantage function of F as a PRF is very high even for very low values of its resource parameter inputs, meaning F is very insecure as a PRF. ▌

Example 3.11 . 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 g that maps ℓ bits to $2L$ bits. In World 0, g will be chosen as a random function of ℓ bits to $2L$ bits, while in World 1, g 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 D^g
  Let $y_1 \leftarrow g(1^\ell)$
  Let $y_2 \leftarrow g(0^\ell)$
  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 World 1. Let us now see how well this adversary does. We claim that

$$\begin{aligned} \Pr\left[\mathbf{Exmt}_G^{\text{prf-1}}(D) = 1\right] &= 1 \\ \Pr\left[\mathbf{Exmt}_G^{\text{prf-0}}(D) = 1\right] &= 2^{-L} \;. \end{aligned}$$

Why? Look at Experiment $\mathbf{Exmt}_G^{\text{prf-1}}(D)$ as defined in Definition 3.6. 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 D will return 1. On the other hand look at Experiment $\mathbf{Exmt}_G^{\text{prf-0}}(D)$ as defined in Definition 3.6. Here g is a random function. So the values $g(1^\ell)$ and $g(0^\ell)$ 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 D will return 1. Now as per Definition 3.6 we subtract to get

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{G}(D) &= \Pr\left[\mathbf{Exmt}^{\text{prf-1}}_{G}(D)=1\right] - \Pr\left[\mathbf{Exmt}^{\text{prf-0}}_{G}(D)=1\right] \\ &= 1-2^{-L}\ .\end{aligned}$$

Now let t be the time complexity of D. 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

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{G}(t,2,2\ell) &= \max_{A}\{\,\mathbf{Adv}^{\mathrm{prf}}_{G}(A)\,\} \\ &\geq \mathbf{Adv}^{\mathrm{prf}}_{G}(D) \\ &= 1-2^{-L}\ .\end{aligned}$$

Our conclusion is that the advantage function of G as a PRF is very high even for very low values of its resource parameter inputs, meaning G is very insecure as a PRF. ▮

3.7 Security against key recovery

We have mentioned several times that security against key recovery is not sufficient as a notion of security for a block cipher. However it is certainly necessary: if key recovery is easy, the block cipher should be declared insecure. We have indicated that we want to adopt as notion of security for a block cipher 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 3.12 Let $F\colon \mathcal{K}\times D\to R$ be a family of functions, and let B be an algorithm that takes an oracle for a function $g\colon D\to R$ and outputs a string. We consider the experiment:

Experiment $\mathbf{Exmt}^{\mathrm{kr}}_{F}(B)$
  $K \xleftarrow{\$} \mathsf{Keys}(F)$
  $K' \leftarrow B^{F_K}$
  If $K=K'$ then return 1 else return 0

The *kr-advantage* of B is defined as

$$\mathbf{Adv}^{\mathrm{kr}}_{F}(B) \;=\; \Pr\left[\mathbf{Exmt}^{\mathrm{kr}}_{F}(B) = 1\right] .$$

∎

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 search attack is always successful based on testing two examples leads to the fact that

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathrm{DES}}(t, 2, 2 \cdot 64) \;=\; 1$$

for t being about 2^{55} times the time T_{DES} for one computation of DES. On the other hand, linear cryptanalysis implies that

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathrm{DES}}(t, 2^{43}, 2^{43} \cdot 64) \;=\; 1$$

for t being about $2^{43} \cdot T_{\mathrm{DES}}$. This gives us a couple of data points on the curve $\mathbf{Adv}^{\mathrm{kr}}_{\mathrm{DES}}(t, q, ql)$. For a more concrete example, let us look at the key-recovery advantage of the family of Example 3.10.

Example 3.13 Let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be the family of functions from Example 3.10. 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^{F_K}
  Let K' be the empty string
  For $j = 1, \ldots, l$ do
    $y_j \leftarrow F_K(e_j)$
    $K' \leftarrow K' \,\|\, y_j$
  EndFor
  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 F_K takes $O(lL)$ time. Thus

$$\mathbf{Adv}^{\mathrm{kr}}_F(t, l, l^2) \;=\; 1 \;.$$

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 however that F 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. ▌

Now we claim that if a block cipher is a secure PRF or PRP then it is also secure against all key-recovery attacks. Put another way, the advantage of F with respect to key recovery cannot be much larger than its advantage as a PRF.

Proposition 3.14 Let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions. Then for any t, q with $q < 2^l$ we have

$$\mathbf{Adv}^{\mathrm{kr}}_F(t, q, ql) \;\;\leq\;\; \mathbf{Adv}^{\mathrm{prf}}_F(t', q+1, (q+1)l) + \frac{1}{2^L} \;, \tag{3.1}$$

and furthermore, if $L = l$, then also

$$\mathbf{Adv}^{\mathrm{kr}}_F(t, q, ql) \;\;\leq\;\; \mathbf{Adv}^{\text{prp-cpa}}_F(t', q+1, (q+1)l) + \frac{1}{2^L - q} \;, \tag{3.2}$$

where we set t' to be t plus the time for one computation of F. ▌

The proof introduces the central idea of *reductions*. We will show a transformation $B \mapsto A_B$ of any kr-adversary B into a prf-adversary A_B such that

$$\mathbf{Adv}^{\mathrm{kr}}_F(B) \;\leq\; \mathbf{Adv}^{\mathrm{prf}}_F(A_B) + \frac{1}{2^L}$$

and also, if the resources used by B are t, q, ql, then those used by A_B are $t', q+1, (q+1)l$. We claim that barring manipulation, this proves the first equation of the claim. Indeed, by taking maximums on both sides, we will be able to get the equation in question, as we will see later.

The problem that adversary A_B is trying to solve is to determine whether its given oracle g is a random instance of F or a random function of l bits to L-bits. The idea behind a reduction is that A_B 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 F_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_B runs B, it produces some key K'. A_B can test K' by checking whether $F(K', x)$ agrees with $g(x)$ for some value x. If so, it bets that g was an instance of F, and if not it bets that g was random.

If B does make oracle queries, we must ask how A_B 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_B 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 $F_K(x)$. That is not what A_B gives it. Instead, it returns $g(x)$, where g is A_B's oracle. When A_B is in World 1, $g(x) = F_K(x)$, 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_B is in World 0, g 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 World 0, we have no idea what properties this output will have. But it is some k-bit string, and A_B 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_B will make sure the latter is true via its choice of x. Let us now proceed to the actual proof.

Proof of Proposition 3.14: We prove the first equation and then briefly indicate how to alter the proof to prove the second equation.

We will show that given any adversary B whose resources are restricted to t, q, ql we can construct an adversary A_B, using resources $t', q+1, (q+1)l$, such that

$$\mathbf{Adv}^{\mathrm{kr}}_F(B) \;\leq\; \mathbf{Adv}^{\mathrm{prf}}_F(A_B) + \frac{1}{2^L}\ . \tag{3.3}$$

If this is true then we can establish Equation (3.1) as follows:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{kr}}_F(t,q,ql) &= \max_B \{\, \mathbf{Adv}^{\mathrm{kr}}_F(B) \,\} \\
&\leq \max_B \{\, \mathbf{Adv}^{\mathrm{prf}}_F(A_B) + 2^{-L} \,\} \\
&\leq \max_A \{\, \mathbf{Adv}^{\mathrm{prf}}_F(A) + 2^{-L} \,\} \\
&= \mathbf{Adv}^{\mathrm{prf}}_F(t', q+1, (q+1)l) + 2^{-L}\ .
\end{aligned}$$

The maximum, in the case of B, is taken over all adversaries whose resources are t, q, ql. In the second line, we apply Equation (3.3). In the third line, we maximize over all A whose resources are $t', q+1, (q+1)l$. The inequality on the third line is true because this set includes all adversaries of the form A_B. The last line is simply

by definition. So it remains to show how to design A_B so that Equation (3.3) holds. (This is the core of the argument, namely what is called the "reduction.")

As per Definition 3.6, adversary A_B will be provided an oracle for a function g: $\{0,1\}^l \to \{0,1\}^L$, 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_B^g
  $i \leftarrow 0$
  Run adversary B, replying to its oracle queries as follows
  When B makes an oracle query x do
    $i \leftarrow i+1$; $x_i \leftarrow x$
    $y_i \leftarrow g(x_i)$
    Return y_i to B as the answer
  Until B stops and outputs a key K'
  Let x be an l bit string not in the set $\{x_1, \ldots, x_q\}$
  $y \leftarrow g(x)$
  If $F(K', x) = y$ then return 1 else return 0

As indicated in the discussion preceding the proof, A_B is running B and itself providing answers to B's oracle queries via the oracle g. When B has run to completion it returns some k-bit string K', which A_B tests by checking whether $F(K'x)$ agrees with $g(x)$. 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 $q < 2^l$ in the statement of the Proposition. Now we claim that

$$\Pr\left[\mathbf{Exmt}_F^{\text{prf-1}}(A_B) = 1\right] \;\geq\; \mathbf{Adv}_F^{\text{kr}}(B) \tag{3.4}$$

$$\Pr\left[\mathbf{Exmt}_F^{\text{prf-0}}(A_B) = 1\right] \;=\; 2^{-L}\,. \tag{3.5}$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, as per Definition 3.6, we get

$$\begin{aligned}\mathbf{Adv}_F^{\text{prf}}(A_B) &= \Pr\left[\mathbf{Exmt}_F^{\text{prf-1}}(A_B) = 1\right] - \Pr\left[\mathbf{Exmt}_F^{\text{prf-0}}(A_B) = 1\right] \\ &\geq \mathbf{Adv}_F^{\text{kr}}(B) - 2^{-L}\,.\end{aligned}$$

Re-arranging terms gives us Equation (3.3). It remains to justify Equations (3.4) and (3.5).

Equation (3.4) is true because in $\mathbf{Exmt}_F^{\text{prf-1}}(A_B)$ the oracle g is F_K for some K, which is the oracle that B expects, and thus B functions as it does in $\mathbf{Exmt}_F^{\text{kr}}(B)$. If B is successful, meaning the key K' it outputs equals K, then certainly A_B returns 1. (It is possible that A_B 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

$\Pr\left[\mathbf{Exmt}_F^{\text{prf-1}}(A_B) = 1\right]$ is greater than or equal to $\mathbf{Adv}_F^{\text{kr}}(B)$ rather than merely equal to it.) Equation (3.5) is true because in $\mathbf{Exmt}_F^{\text{prf-0}}(A_B)$ the function g is random, and since x was never queried by B, the value $g(x)$ is unpredictable to B. Imagine that $g(x)$ is chosen only when x is queried to g. At that point, K', and thus $F(K', x)$, is already defined. So $g(x)$ has a 2^{-L} 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 $g(x)$.

For the proof of Equation (3.2) we seek a reduction $B \mapsto A_B$ with the property that

$$\mathbf{Adv}_F^{\text{kr}}(B) \;\leq\; \mathbf{Adv}_F^{\text{prp-cpa}}(A_B) + \frac{1}{2^L - q}\,. \tag{3.6}$$

The reduction is identical to the one given above, meaning the adversary A_B is the same. For the analysis we see that

$$\begin{aligned}
\Pr\left[\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A_B) = 1\right] &= \mathbf{Adv}_F^{\text{kr}}(B) \\
\Pr\left[\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A_B) = 1\right] &\leq \frac{1}{2^L - q}\,.
\end{aligned}$$

Subtracting yields

$$\begin{aligned}
\mathbf{Adv}_F^{\text{prp-cpa}}(A_B) &= \Pr\left[\mathbf{Exmt}_F^{\text{prp-cpa-1}}(A_B) = 1\right] - \Pr\left[\mathbf{Exmt}_F^{\text{prp-cpa-0}}(A_B) = 1\right] \\
&\geq \mathbf{Adv}_F^{\text{kr}}(B) - \frac{1}{2^L - q}
\end{aligned}$$

and re-arranging terms gives us Equation (3.6). The first equation above is true for the same reason as before. The second equation is true because in World 0 the map g is now a random permutation of l-bits to l-bits. So $g(x)$ assumes any random value except the values $y_1, \ldots, y_q$, meaning there are $2^L - q$ things it could be. (Remember $L = l$ 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 3.15 Define the block cipher E: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ 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 all values of t, q, however high,

$$\mathbf{Adv}_E^{\text{kr}}(t, q, ql) \;=\; 2^{-k}\,,$$

and on the other hand

$$\mathbf{Adv}_E^{\text{prp-cpa}}(t, 1, l) \;\geq\; 1 - 2^{-l}$$

for $t = O(l)$. 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 g: $\{0,1\}^l \to \{0,1\}^l$, and very little time, you can tell almost with certainty whether g 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 $g(0^l) = 0^l$, and by returning 1 if and only if this is true, attain a prp-advantage of $1 - 2^{-l}$. ▌

3.8 The birthday attack

Suppose E: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ is a family of permutations, meaning a block cipher. If we are given an oracle g: $\{0,1\}^l \to \{0,1\}^l$ 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 g were a permutation, the values $y_1, y_2, \ldots, y_q$ must be distinct. If g 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 distinguisher, as we will argue below. Roughly, it takes $q = \sqrt{2^l}$ 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 Appendix A, and then come back to the following.

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

Proposition 3.16 Let E: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a family of permutations. Suppose q satisfies $2 \le q \le 2^{(l+1)/2}$. Then

$$\mathbf{Adv}^{\mathrm{prf}}_E(t, q, ql) \quad \ge \quad 0.3 \cdot \frac{q(q-1)}{2^l} \ ,$$

where t is the time for q computations of E, plus $O(ql)$. ▌

Proof of Proposition 3.16: The birthday attack is implemented by an adversary D who, given an oracle g: $\{0,1\}^l \to \{0,1\}^l$, works like this:

Adversary D^g
For $i = 1, \ldots, q$ do
    Let x_i be the i-th l-bit string in lexicographic order
    $y_i \leftarrow g(x_i)$

End For
If $y_1, \ldots, y_q$ are all distinct then return 1, else return 0

We claim that

$$\mathbf{Adv}^{\mathrm{prf}}_E(D) \;\geq\; 0.3 \cdot \frac{q(q-1)}{2^l} \;,$$

from which the Proposition follows. Let us now justify this lower bound. Letting $N = 2^l$, we claim that

$$\Pr\left[\mathbf{Exmt}^{\text{prf-1}}_E(D) = 1\right] \;=\; 1 \tag{3.7}$$

$$\Pr\left[\mathbf{Exmt}^{\text{prf-0}}_E(D) = 1\right] \;=\; 1 - C(N,q)\;. \tag{3.8}$$

Here $C(N,q)$, as defined in Appendix A, 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

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prf}}_E(D) &= \Pr\left[\mathbf{Exmt}^{\text{prf-1}}_E(D) = 1\right] - \Pr\left[\mathbf{Exmt}^{\text{prf-0}}_E(D) = 1\right] \\
&= 1 - [1 - C(N,q)] \\
&= C(N,q) \\
&\geq 0.3 \cdot \frac{q(q-1)}{2^l}\;.
\end{aligned}$$

The last line is by Proposition A.1. It remains to justify Equations (3.7) and (3.8).

Equation (3.7) is clear because in World 1, $g = E_K$, and since E is a family of permutations, g is a permutation, and thus $y_1, \ldots, y_q$ are all distinct. Now, suppose D is in World 0, so that g is a random function of l bits to l bits. What is the probability that $y_1, \ldots, y_q$ are all distinct? Since g is a random function and $x_1, \ldots, x_q$ are distinct, $y_1, \ldots, y_q$ are random, independently distributed values in $\{0,1\}^l$. Thus we are looking at the birthday problem. We are throwing q balls into $N = 2^l$ 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 (3.8). ▌

3.9 The PRP/PRF switching lemma

When we come to analyses of block cipher based constructions, we will find a curious dichotomy: PRPs are what most naturally model block ciphers, but analyses are often considerably simpler and more natural assuming the block cipher is a PRF. To bridge the gap, we relate the prp-security of a block cipher 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 3.17 [PRP/PRF Switching Lemma] Let $E\colon \mathcal{K}\times\{0,1\}^n \to \{0,1\}^n$ be a function family. Let A be an adversary that asks at most q oracle queries. Then

$$\left|\Pr[\rho \xleftarrow{\$} \mathsf{Rand}(n) :\ A^{\rho}\Rightarrow 1] - \Pr[\pi \xleftarrow{\$} \mathsf{Perm}(n) :\ A^{\pi}\Rightarrow 1]\right| \ \leq\ \frac{q(q-1)}{2^{n+1}}\ . \quad (3.9)$$

As a consequence, we have that

$$\left|\mathbf{Adv}^{\mathrm{prf}}_{E}(A) - \mathbf{Adv}^{\mathrm{prp}}_{E}(A)\right| \ \leq\ \frac{q(q-1)}{2^{n+1}}\ . \quad (3.10)$$

∎

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 setup 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 (3.9), as Equation (3.10) follows from that. We need to establish that

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

where, for notational simplicity, we omit explicitly indicating that $\rho \xleftarrow{\$} \mathsf{Rand}(n)$ and $\pi \xleftarrow{\$} \mathsf{Perm}(n)$; the variable name will be enough to let you keep the experiments straight. 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

$$\Pr[A^{\rho}\Rightarrow 1] - \Pr[A^{\pi}\Rightarrow 1] \ \leq\ \frac{q(q-1)}{2^{n+1}}\ . \quad (3.11)$$

Since A is trying to distinguish a function $\rho \xleftarrow{\$} \mathsf{Rand}(n)$ from a function $\pi \xleftarrow{\$} \mathsf{Perm}(n)$, 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 being a π-oracle or a ρ-oracle, so asking invalid queries is pointless for A.

Initialization:
01 $bad \leftarrow \mathsf{false}$; **for** $X \in \{0,1\}^n$ **do** $\pi(X) \leftarrow \mathtt{undef}$

When A asks query X:
10 $Y \xleftarrow{\$} \{0,1\}^n$
11 **if** $Y \in \mathrm{Range}(\pi)$ **then** $bad \leftarrow \mathsf{true}$, $Y \xleftarrow{\$} \overline{\mathrm{Range}}(\pi)$
12 $\pi(X) \leftarrow Y$
13 **return** Y

Figure 3.1: Games used in the proof of the Switching Lemma. Game P is the pseudocode exactly as written. Game R is the same except we *omit* the highlighted statement at line 11. To play either game, start off by executing the initialization step, line 01. Then, whenever the adversary makes a query X, that query is answered by performing the pseudocode at lines 10–13, with or without the highlighted statement, as indicated.

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 whether A is speaking to a $\pi \xleftarrow{\$} \mathsf{Perm}(n)$ oracle or it is speaking to a $\rho \xleftarrow{\$} \mathsf{Rand}(n)$ oracle. 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.

Now we're going to imagine answering A's queries by running one of two games. Instead of thinking of A interacting with a random permutation oracle $\pi \xleftarrow{\$} \mathsf{Perm}(n)$ we're going to think of A interacting with the *game*, call it game P, specified in Figure 3.1. Instead of thinking of A interacting with a random function oracle $\rho \xleftarrow{\$} \mathsf{Rand}(n)$ we're going to think of A interacting with game R, also specified in Figure 3.1. Read the caption of the figure to see how the two games are defined.

Let's look at Games P and R. In both games, we start off performing the initialization step, setting a flag *bad* to false and setting a variable π to be $\mathtt{undef}$ at every n-bit string. As the game run, we will "fill in" values of $\pi(X)$ with n-bit strings. At any point point in time, we let $\mathrm{Range}(\pi)$ be the set of all n-bit strings Y such that $\pi(X) = Y$ for some X. We let $\mathrm{Domain}(\pi)$ be the set of all n-bit strings X such that $\pi(X) \neq \mathtt{undef}$. We let $\overline{\mathrm{Range}}(\pi)$ be all the n-bit strings that are *not* in $\mathrm{Range}(\pi)$, and we let $\overline{\mathrm{Domain}}(\pi)$ be all the n-bit strings that are *not* in $\mathrm{Domain}(\pi)$. We will use this Domain/Range/$\overline{\text{Domain}}$/$\overline{\text{Range}}$ notation from now on.

As Games P and R run, $\mathrm{Domain}(\pi)$ and $\mathrm{Range}(\pi)$ grow, getting more and more values silently put there, while $\overline{\mathrm{Domain}}(\pi)$ and $\overline{\mathrm{Range}}(\pi)$ will shrink, having values successively removed. Initially, $|\mathrm{Domain}(\pi)| = |\mathrm{Range}(\pi)| = 0$ and $|\overline{\mathrm{Domain}}(\pi)| =$

$|\overline{\text{Range}}(\pi)| = 2^n$.

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.

Completing our description of the games, 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. Line 11, next, is the most interesting step. If the point Y that we just chose is already in the range of π then we set a flag *bad*. In such a case, if we are playing game P, then we now make a *fresh* choice of Y, this time from the co-range of π. If we are playing game R then we stick with our original choice of Y. Either way, we set $\pi(X)$ to Y, effectively growing the domain of π and (usually) its range, and we return Y.

Now let's think about what A sees as it plays Games R. Whatever query X is asked, we just return a random n-bit string Y. So game R perfectly simulates a random function $\rho \xleftarrow{\$} \mathsf{Rand}(n)$. Remember that the adversary isn't allowed to repeat a query, so what the adversary would get if it had a $\rho \xleftarrow{\$} \mathsf{Rand}(n)$ oracle is a random n-bit string in response to each query—just what we are giving it. We say that A is provided exactly the same *view* if we give it a random function $\rho \xleftarrow{\$} \mathsf{Rand}(n)$ or if it is interacting with Game R. Since the environment A finds itself in is the same in these two cases, the probability that A outputs 1 must be the same in these two cases, too:

$$\Pr[A^{\rho} \Rightarrow 1] \quad = \quad \Pr[A^{\text{Game R}} \Rightarrow 1] \tag{3.12}$$

Now if we're in game P then what the adversary gets in response to each query X is a random point Y that has not already been returned to A. Seeing this requires a bit of thought. It's important that we started off, in line 10, by choosing a random point Y from a set, $\{0,1\}^n$, that is at least as big as $\overline{\text{Range}}(\pi)$. So if our sample point is already in $\overline{\text{Range}}(\pi)$ then we've chosen a random point in $\overline{\text{Range}}(\pi)$; and if our sample point is not already in $\overline{\text{Range}}(\pi)$ then we go ahead and choose a new random point in $\overline{\text{Range}}(\pi)$. So either way, we end up choosing a random point in $\overline{\text{Range}}(\pi)$ and, overall, we are choosing a random point in $\overline{\text{Range}}(\pi)$. Now the behavior of a random permutation oracle is to give a random new answer to each query, and that is exactly the behavior that Game P exhibits, and so A's distribution on views is the same if it is given $\pi \xleftarrow{\$} \mathsf{Perm}(n)$ or if it interacts with Game P. Since A's view is the same in the two cases, the probability that A outputs 1 must be the same in these two cases and we have that

$$\Pr[A^{\pi} \Rightarrow 1] \quad = \quad \Pr[A^{\text{Game P}} \Rightarrow 1]\,. \tag{3.13}$$

Now we are trying to bound $\Pr[A^{\rho} \Rightarrow 1] - \Pr[A^{\pi} \Rightarrow 1]$ and at this point we have that

$$\Pr[A^{\rho} \Rightarrow 1] - \Pr[A^{\pi} \Rightarrow 1] \quad = \quad \Pr[A^{\text{Game R}} \Rightarrow 1] - \Pr[A^{\text{Game P}} \Rightarrow 1]\,. \tag{3.14}$$

We next claim that

$$\Pr[A^{\text{Game R}} \Rightarrow 1] - \Pr[A^{\text{Game P}} \Rightarrow 1] \;\leq\; \Pr[A^{\text{Game R}} \text{ sets } \mathit{bad}] \,. \qquad (3.15)$$

To see Equation (3.15), let's think about all the random choices that happen when adversary A plays Games R or P. The adversary A may make make random choices of its own; and the Games, R or P make random choices, too. You can imagine a huge string of random coin tosses, C, that has all the random coins that might be needed—both coins for A and coins for Games P and R. (Insofar as Game P needs to sample in a set Range(π) that will sometimes have size that is not a power of two, you can imagine that some subsequences of possible bits in C are excluded. This is not an important detail.) There is a finite set $\mathcal{C}$ naming all the possible coin flips that might be needed by adversary A and Games R and P. Each sequence of coin tosses $C \in \mathcal{C}$ will result in a particular behavior of A as it plays Game P and a particular behavior of A as it plays Game R.

For a bit $b \in \{0,1\}$, lets think of all of those coin tosses $C \in \mathcal{C}$ that cause A to output b if game R is played. Call this set $\mathcal{C}^b_{\text{R}}$. Think of all of those coin tosses $C \in \mathcal{C}$ that cause A to output b if game P is played. Call this set $\mathcal{C}^b_{\text{P}}$. Finally, think of all those coin tosses $C \in \mathcal{C}$ that cause A to set the flag *bad* to true in Game R or Game P. Call this set $\mathcal{C}^*$. Note that a C causes *bad* to be set to true in Game R if and only if C causes *bad* to be set to true in game P.

Now $\Pr[A^{\text{Game R}} \Rightarrow 1] = |\mathcal{C}^1_{\text{R}}|/|\mathcal{C}|$ and $\Pr[A^{\text{Game P}} \Rightarrow 1] = |\mathcal{C}^1_{\text{P}}|/|\mathcal{C}|$ and $\Pr[A^{\text{Game R}} \Rightarrow 1] - \Pr[A^{\text{Game P}} \Rightarrow 1] = |\mathcal{C}^1_{\text{R}} \cap \mathcal{C}^0_{\text{P}}|/|\mathcal{C}|$. In other words, the only way for coin tosses $C \in \mathcal{C}$ to contribute to A's advantage is for the coin tosses to result in a 1-output in Game R and a 0-output in Game P. Any such sequence of coin tosses $C \in \mathcal{C}^1_{\text{R}} - \mathcal{C}^0_{\text{P}}$ must result in *bad* getting to true: $\mathcal{C}^1_{\text{R}} - \mathcal{C}^0_{\text{P}} \subseteq \mathcal{C}^*$. This is because coin tosses C which do not set *bad* result in the same sequence of responses in Games P and R, the same sequence of internal choices by A, and so the same output. We thus have that

$$\begin{aligned} \Pr[A^{\text{Game R}} \Rightarrow 1] - \Pr[A^{\text{Game P}} \Rightarrow 1] &\leq |\mathcal{C}^*|/|\mathcal{C}| && (3.16) \\ &= \Pr[A^{\text{Game R}} \text{ sets } \mathit{bad}] && (3.17) \end{aligned}$$

as required.

To bound $\Pr[A^{\text{Game R}} \text{ sets } \mathit{bad}]$ is simple. Line 11 is executed q times. The first time it is executed Range(π) contains 0 points; the second time it is executed Range(π) contains 1 point; the third time it is executed Range(π) 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 Range(π). By the sum bound, the probability that a Y will ever be in Range(π) 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 (3.11). To go on and show that $\mathbf{Adv}^{\text{prf}}_E(A) - \mathbf{Adv}^{prp}_E(A) \leq q(q-1)/2^{n+1}$ note that

$$\mathbf{Adv}^{\text{prf}}_E(A) - \mathbf{Adv}^{\text{prp}}_E(A) \;\leq\; \Pr[A^{E_K} \Rightarrow 1] - \Pr[A^{\rho} \Rightarrow 1] - (\Pr[A^{E_K} \Rightarrow 1] - \Pr[A^{\pi} \Rightarrow 1])$$

$$\begin{aligned} &\leq \Pr[A^{\rho} \Rightarrow 1] - \Pr[A^{\pi} \Rightarrow 1] \\ &\leq q(q-1)/2^{n+1} \end{aligned}$$

where it is understood that $K \xleftarrow{\$} \mathcal{K}$. 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.

3.10 Unix one-way function

Move off into a separate chapter

The framework for the Unix password-hashing scheme is this. We fix some function h: $\{0,1\}^k \to \{0,1\}^L$, which we call the *password hashing function.* A user U chooses a k-bit password K, and the system stores in the password file the value $y = h(K)$ together with the user's name U. When the user logs in he or she is prompted for a user name U and a password K. The system uses the user U to retrieve y, and then the system computes $h(K)$ and declares the user to be authentic if and only if this value equals y. The idea of this system—instead of storing (U, K) itself—is that a party who obtains (U, y) still can not gain trivial entry into the system: they must still find a K such that $h(K) = y$.

Assume the attacker gets access to the password file and hence to y. The attacker's task is thus to find K given y. (The attacker knows the function h, since this is public code. However we assume the attacker does not have any further powers, such as the use of Trojan horses.) Security in this model would require that it be computationally infeasible to recover K from y. Thus h must be chosen to make this true.

A simple example choice of h is $h(K) = \mathrm{DES}_K(0^{64})$. (The actual choice made by Unix is somewhat more complex, involving something called a "salt," which customizes the function h to each user U. It also involves iterating the block cipher a number of times. However this does not change the heart of the analysis, so let us stick with the fiction we have described.) In this example, $k = 56$ and $L = 64$.

Obviously, the security of this scheme depends on the security of DES. If we want to prove anything meaningful about the security of the simplified password scheme, we must make some assumption about DES. We have suggested above that the appropriate assumption to make about a block cipher like DES is that it is a secure PRP. So we make this assumption and now ask what we can prove about the security of the simplified password scheme.

However to answer this effectively, we first need to decide exactly what security property we would like to target. At first, one would think the security question boils down to asking how hard it would be to recover K given $y = h(K)$. But although recovering K given y would certainly break the scheme, so would recovering any K' such that $h(K') = y$, even if $K' \neq K$. Accordingly, this is the task whose difficulty

we consider. Technically, it corresponds to asking that h be a *one-way*, meaning it is computationally infeasible to recover the pre-image of a range point.

We provide a formalization below that is more specific. Function $h\colon \{0,1\}^k \to \{0,1\}^L$ is one-way if it is hard, given y, to compute a point x' such that $h(x') = y$, when y was chosen by drawing x at random from $\{0,1\}^k$ and setting $y = h(x)$.

Definition 3.18 Let $h\colon \{0,1\}^k \to \{0,1\}^L$ be a function, and let I be an algorithm that on input an L-bit string returns a k-bit string. We consider the experiment:

Experiment $\mathbf{Exmt}_h^{\mathrm{owf}}(I)$
  $K \xleftarrow{\$} \{0,1\}^k$; $y \leftarrow h(K)$
  $x \leftarrow I(y)$
  If $h(x) = y$ then return 1 else return 0

The *owf-advantage* of I is defined as

$$\mathbf{Adv}_h^{\mathrm{owf}}(I) \;=\; \Pr\left[\mathbf{Exmt}_h^{\mathrm{owf}}(I) = 1\right] .$$

For any t the *owf-advantage* of I is defined via

$$\mathbf{Adv}_h^{\mathrm{owf}}(t) \;=\; \max_I \{\, \mathbf{Adv}_h^{\mathrm{owf}}(I) \,\}$$

where the maximum is over all I having time-complexity t. ▌

As usual, a one-way function is understood to be one for which $\mathbf{Adv}_h^{\mathrm{owf}}(t)$ is "small" for practical values of t. We want to show that if h is defined via $h(K) = F_K(0^l)$ for a secure PRF $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ then h is one-way.

We remark that one must look carefully at the models to know how to interpret the impact of such a result on the actual password scheme. Showing that h is a one-way function amounts to saying that the password scheme is secure if passwords are randomly chosen k-bit keys where k is the block length of the block cipher. In real life, passwords are often not random, and in that case this result does not apply. It also does not take into consideration other issues about password usage, such as the possibility of compromise of the channel over which the user conveys the password to the server. However, the result here is still useful and serves to illustrate an application of PRFs.

Theorem 3.19 Let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions, and define $h\colon \{0,1\}^k \to \{0,1\}^L$ via $h(K) = F(K, 0^l)$ for all $K \in \{0,1\}^k$. Then we have

$$\mathbf{Adv}_h^{\mathrm{owf}}(t) \;\le\; \frac{1}{1 - 2^{k-L}} \cdot \mathbf{Adv}_F^{\mathrm{prf}}(t', 1, l) \,, \tag{3.18}$$

under the assumption that $k \le L - 1$. Here t' is t plus the time for one computation of F. ▌

As per the theorem, $\mathbf{Adv}^{\mathrm{owf}}_{h}(t)$ can only be marginally more than $\mathbf{Adv}^{\mathrm{prf}}_{F}(t', 1, l)$. Specifically, $\mathbf{Adv}^{\mathrm{owf}}_{h}(t)$ can be at most twice $\mathbf{Adv}^{\mathrm{prf}}_{F}(t', 1, l)$, because $k \leq L - 1$ implies

$$\frac{1}{1 - 2^{k-L}} \;\leq\; 2 \;.$$

So if F is secure, meaning $\mathbf{Adv}^{\mathrm{prf}}_{F}(t', 1, l)$, is low, $\mathbf{Adv}^{\mathrm{owf}}_{h}(t)$ is also low, and hence h is secure. It is thus a proof of security, showing that h is one-way if F is a secure PRF.

Notice that security of a family F against key-recovery does not imply that the associated function h defined in the theorem is one-way, exactly due to the fact that one-wayness declares the adversary successful even it if recovers a key K' different from the K under which its challenge $y = F(K, 0^l)$ was computed. However, security as a PRF is strong enough to rule out even recovery of this different key.

Proof of Theorem 3.19: We associate to any adversary I attempting to invert h an adversary D_I attacking F such that

$$\mathbf{Adv}^{\mathrm{owf}}_{h}(I) \;\leq\; \frac{1}{1 - 2^{k-L}} \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(D_I) \;. \tag{3.19}$$

Furthermore, D_I makes only one oracle query, this of length l bits, and has time-complexity t' which is the time-complexity of I plus the time for one computation of F. Taking maximums in the usual way yields Equation (3.18), so it remains to provide D_I such that Equation (3.19) is true. This adversary takes an oracle for a function $g\colon\ \{0,1\}^l \to \{0,1\}^L$ and works as follows:

Adversary D^g_I
  $y \leftarrow g(0^l)$
  $x \leftarrow I(y)$
  If $F(x, 0^l) = y$ then return 1 else return 0

The adversary queries its oracle g at 0^l to get back a value it calls y, and then applies the inverting algorithm I to y to get back a value x. If I successfully inverted h at y our adversary bets that g is an instance of F, and otherwise it bets that g is an instance of $\mathsf{Rand}(l,L)$. To compute the advantage of this adversary it is convenient to set

$$\epsilon \;=\; \mathbf{Adv}^{\mathrm{owf}}_{h}(I) \;.$$

Now we claim that

$$\Pr\left[\mathbf{Exmt}^{\text{prf-1}}_{F}(D_I) = 1\right] \;=\; \epsilon \tag{3.20}$$

$$\Pr\left[\mathbf{Exmt}^{\text{prf-0}}_{F}(D_I) = 1\right] \;\leq\; \frac{2^k}{2^L} \cdot \epsilon \;. \tag{3.21}$$

We will justify these claims shortly, but first let us use them to conclude. Subtract-

ing, we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_F(D_I) &= \Pr\left[\mathbf{Exmt}^{\text{prf-1}}_F(D_I)=1\right] - \Pr\left[\mathbf{Exmt}^{\text{prf-0}}_F(D_I)=1\right] \\ &\geq \epsilon - \frac{2^k}{2^L}\cdot\epsilon \\ &= \left(1-2^{k-L}\right)\cdot\epsilon\ .\end{aligned}$$

Now, we divide both sides by $1-2^{k-L}$ to get

$$\epsilon \ \leq\ \frac{1}{1-2^{k-L}}\cdot\mathbf{Adv}^{\mathrm{prf}}_F(D_I)\ ,$$

which is exactly Equation (3.19). However, there is a subtle point here that should be noted. This step is only correct if the quantity $1-2^{k-L}$ by which we are dividing is non-zero (otherwise we can't divide by it) and in fact positive (if it was negative, we would have to reverse the inequality). The fact that $1-2^{k-L}$ is positive is true by our assumption that $k \leq L-1$. This is the *only* place we make use of this assumption, but it is crucial. It remains to justify Equations (3.20) and (3.21).

We claim that Experiment $\mathbf{Exmt}^{\text{prf-1}}_F(D_I)$ ends up faithfully mimicking Experiment $\mathbf{Exmt}^{\mathrm{owf}}_h(I)$. Indeed, Experiment $\mathbf{Exmt}^{\text{prf-1}}_F(D_I)$ begins by selecting a random k-bit key K, so that $y = F(K,0^l)$. By definition of h this means that $y = h(K)$, so y is distributed the same way in the two experiments. Then, both experiments run I and return 1 if and only if I is successful, so the probability that they return 1 is the same. This justifies Equation (3.19).

Now suppose D_I is in World 0, meaning $g\colon\ \{0,1\}^l \to \{0,1\}^L$ is a random function. We want to upper bound the probability that $\mathbf{Exmt}^{\text{prf-0}}_F(D_I)$ returns 1. Since g is random, y will be uniformly distributed over $\{0,1\}^L$. Thus we want to upper bound

$$\delta \ \stackrel{\text{def}}{=}\ \Pr\left[\, y \xleftarrow{\$} \{0,1\}^L\ ;\ x \leftarrow I(y)\ :\ F(x,0^l)=y\,\right]\ . \tag{3.22}$$

The notation here means that we first pick y at random from $\{0,1\}^L$, then set x to $I(y)$, and then ask what is the probability that $F(x,0^l)$ equals y. Since the algorithm I might be randomized, the probability is not only over the choice of y, but also over the random coins tossed by I itself.

For simplicity we first prove Equation (3.21) in the case where I is deterministic, so that the probability in the computation of δ is only over the choice of y. In this case it is convenient to define the sets

$$\begin{aligned}X &= \{\, x\in\{0,1\}^k\ :\ h(I(h(x))) = h(x)\,\} \\ Y &= \{\, y\in\{0,1\}^L\ :\ h(I(y)) = y\,\}\ .\end{aligned}$$

We show the sequence of steps via which Equation (3.21) can be obtained, and then justify them:

$$\delta\ =\ \frac{|Y|}{2^L}\ \leq\ \frac{|X|}{2^L}\ =\ \frac{2^k\cdot\epsilon}{2^L}\ .$$

The fact that $\delta = |Y|/2^L$ follows from Equation (3.22) and the definition of Y. The last equality uses the analogous fact that $\epsilon = |X|/2^k$, and this can be justified by looking at Experiment $\mathbf{Exmt}_h^{\mathrm{owf}}(I)$ and the definition of set X above. The main claim used above is that $|Y| \leq |X|$. To see why this is true, let

$$h(X) \;=\; \{\, h(x) \;:\; x \in X \,\} \;=\; \{\, y \in \{0,1\}^L \;:\; \exists x \in X \text{ such that } h(x) = y \,\}\ .$$

This is called the *image* of X under h. Then observe two things, from which $|Y| \leq |X|$ follows:

$$|h(X)| \leq |X| \qquad \text{and} \qquad h(X) = Y\ .$$

The first of these is true simply because h is a function. (One x value yields exactly one y value under h. Some of these y values might be the same as x ranges over X, but certainly you can't get more y values than you have x values.) The second, that $h(X) = Y$, can be justified by looking at the definitions of the sets X and Y and observing two things: If $x \in X$ then $h(x) \in Y$ and if $y \in Y$ then there is some $x \in X$ such that $h(x) = y$.

That completes the proof for the case where I is deterministic. Let us now briefly indicate why Equation (3.21) remains true when I is a randomized algorithm.

In this case, when I is run on input y, it tosses coins to get a random string R, and bases its computation on both y and R, returning a value x that is a function of both of y and R. Thus, there are many different possible x values that it might return on input y. We have no idea exactly how I uses R or how it performs its computation, but we can still assess the probabilities we need to assess. For any $y \in \{0,1\}^L$ and any $x \in \{0,1\}^k$ we let

$$P_y(x) \;=\; \Pr\left[\, R \xleftarrow{\$} \{0,1\}^r \;:\; I(y;R) = x \,\right]\ .$$

In other words, having fixed x, y, we ask what is the probability that I, on input y, would output x. The probability is over the coin toss sequence R of I, and this has been made explicit. We are letting r be the number of coins that I tosses and letting $I(y;R)$ denote the output of I on input y and coins R. Note that this output is a single x value. (Towards understanding this it may be helpful to note that the case of I being deterministic corresponds to the following: for every y there is a unique x such that $P_y(x) = 1$, and for all other values of x we have $P_y(x) = 0$.)

Now for any $y \in \{0,1\}^L$ we let

$$\begin{aligned} h^{-1}(y) &= \{\, x \in \{0,1\}^k \;:\; h(x) = y \,\} \\ Y^* &= \{\, y \in \{0,1\}^L \;:\; h^{-1}(y) \neq \emptyset \,\}\ . \end{aligned}$$

Thus $h^{-1}(y)$ is the set of all pre-images of y under h, while Y^* is the image of $\{0,1\}^k$ under h, meaning the set of all range points that possess some pre-image under h. Notice that for any $y \in Y^*$ we have $|h^{-1}(y)| \geq 1$. Thus for any $y \in Y^*$ we have

$$\frac{1}{2^L} \;\leq\; \frac{|h^{-1}(y)|}{2^L} \;=\; \frac{2^k}{2^L} \cdot \frac{|h^{-1}(y)|}{2^k}\ . \tag{3.23}$$

We show the sequence of steps via which Equation (3.21) can be obtained, and then justify them:

$$\begin{aligned}
\delta &= \sum_{y\in\{0,1\}^L} \left(\sum_{x\in h^{-1}(y)} P_y(x) \right) \cdot \frac{1}{2^L} \\
&= \sum_{y\in Y^*} \left(\sum_{x\in h^{-1}(y)} P_y(x) \right) \cdot \frac{1}{2^L} \\
&\le \sum_{y\in Y^*} \left(\sum_{x\in h^{-1}(y)} P_y(x) \right) \cdot \frac{2^k}{2^L} \cdot \frac{|h^{-1}(y)|}{2^k} \\
&= \frac{2^k}{2^L} \cdot \sum_{y\in Y^*} \left(\sum_{x\in h^{-1}(y)} P_y(x) \right) \cdot \frac{|h^{-1}(y)|}{2^k} \\
&= \frac{2^k}{2^L} \cdot \sum_{y\in\{0,1\}^L} \left(\sum_{x\in h^{-1}(y)} P_y(x) \right) \cdot \frac{|h^{-1}(y)|}{2^k} \\
&= \frac{2^k}{2^L} \cdot \epsilon \ .
\end{aligned}$$

The equation for δ used in the first line comes about by looking at the probability that I succeeds for a given value of y, and then summing this over all y-values, weighted by the probability 2^{-L} of that y value being chosen. We then restrict the sum to values $y \in Y^*$ based on the fact that the terms corresponding to values $y \notin Y^*$ in the previous sum are just zero. Once this is done we can apply Equation (3.23) to obtain the inequality. We then factor $2^k/2^L$ out of the sum. We extend the sum to cover values $y \notin Y^*$ based again on the fact that the corresponding new terms are simply zero. In the last sum, we are summing the probability that I succeeds for a given value of y, weighted by the probability that y would be produced under the experiment of choosing x at random and setting $y = h(x)$, namely as in Experiment $\mathbf{Exmt}_h^{\mathrm{owf}}(I)$, and thus recover ϵ. ▌

3.11 Pseudorandom generators

Uncomment-out, revise, and move to a different chapter

3.12 Historical notes

The basic notion of pseudorandom functions is due to Goldreich, Goldwasser and Micali [17]. In particular these authors introduced the important notion of distinguishers. The notion of a pseudorandom permutation is due to Luby and Rackoff

[25]. These works are 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. The approach used here, motivated by the desire to model block ciphers, is called "concrete security," and originates with [2]. Definitions 3.6 and 3.7 are from [2], as are Propositions 3.16 and 3.17. The material of Section 3.10 is a concrete security adaptation of results from [26].

3.13 Problems

Problem 3.1 Let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a secure PRP. Consider the PRP $E'\colon \{0,1\}^k \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ defined by

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

where $|x| = |x'| = n$. Show that E' is not a secure PRP.

Problem 3.2 Consider the following block cipher $E : \{0,1\}^3 \times \{0,1\}^2 \to \{0,1\}^2$:

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 3.3 Let $D, R \subseteq \{0,1\}^*$ with D finite. Let $f : D \to R$. Consider the following definition for the success of an adversary I in breaking f as a one-way function:

$$\mathbf{Adv}^{\mathrm{owf}'}_f(I) \quad = \quad \Pr[X \xleftarrow{\$} D :\ I(f(X)) = X]$$

Is this a good definition for the security of a one-way function? Why or why not.

Problem 3.4 Suppose you are given a PRF $F\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Design a PRF $G\colon \{0,1\}^{2k} \times \{0,1\}^n \to \{0,1\}^{2n}$ which is secure as long as F is secure. Analyze the security of G in terms of the security of F.

Problem 3.5 Present a secure construction for the problem of Example 3.11. 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 3.6 Design a block cipher $E : \{0,1\}^k \times \{0,1\}^{128} \to \{0,1\}^{128}$ 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 3.7 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

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

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

$$\mathbf{Adv}^{\mathrm{prf}}_F(t,2,2l) \ge \frac{1}{2} \cdot \left(1 + \frac{1}{2^L}\right)$$

for $t = q \cdot T_F + O(\mu)$, where T_F denotes the time to perform one computation of F. Explain in a sentence why this shows that if F is inner-product preserving then F is not a secure PRF.

Problem 3.8 Let E: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher. The *two-fold cascade* of E is the block cipher $E^{(2)}$: $\{0,1\}^{2k} \times \{0,1\}^l \to \{0,1\}^l$ 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 $x \in \{0,1\}^l$. (Here " $\|$ " stands for concatenation of strings.) Prove that

$$\mathbf{Adv}^{\text{prp-cpa}}_{E^{(2)}}(t,q,lq) \;\le\; \mathbf{Adv}^{\text{prp-cpa}}_{E}(t,q,lq)$$

for all t, q. Explain in a sentence why this shows that if E is a secure PRP then so is $E^{(2)}$.

Problem 3.9 Give a construction to show that $F : \{0,1\}^{2n} \times \{0,1\}^n \to \{0,1\}^n$ can be a good PRF (secure in the sense of $\mathbf{Adv}^{\mathrm{prf}}_F$) and yet the function $f(X) = F_X(0)$ is not a secure one-way function.

Problem 3.10 Let $D, R \subseteq \{0,1\}^*$ with D finite. Let $f : D \to R$ be a function. Suppose there is a probabilistic adversary I that, in time t, obtains advantage $\epsilon = \mathbf{Adv}^{\mathrm{owf}}_f(I)$. Show that there is a deterministic adversary I' with essentially the same running time as I such that $\epsilon = \mathbf{Adv}^{\mathrm{owf}}_f(I')$.

Problem 3.11 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

$$\mathbf{Adv}(A) = \Pr[\pi \leftarrow \mathsf{Perm}(n) : A^{\pi(\cdot),\pi(\cdot)} = 1] - \Pr[\pi, \pi' \leftarrow \mathsf{Perm}(n) : A^{\pi(\cdot),\pi'(\cdot)} = 1].$$

Prove a good upper bound for $\mathbf{Adv}(A)$, say $\mathbf{Adv}(A) \leq q^2/2^n$.

Problem 3.12 Let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^\ell$ be a family of functions and $r \geq 1$ an integer. The *r-round Feistel cipher associated to F* is the family of permutations $F^{(r)}$: $\{0,1\}^{rk} \times \{0,1\}^{2\ell} \to \{0,1\}^{2\ell}$ 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 $L_r \,\|\, R_r$

1.1 Prove that

$$\mathbf{Adv}^{\mathrm{prf}}_{F^{(2)}}(t, 2, 4\ell) \geq 1 - 2^{-\ell} ,$$

where $t = O(\ell)$ plus the time for two computations of F.

1.2 Prove that

$$\mathbf{Adv}^{\text{prp-cca}}_{F^{(3)}}(t, 2, 4\ell, 1, 2\ell) \geq 1 - 3 \cdot 2^{-\ell} ,$$

where $t = O(\ell)$ plus the time for three computations of F.

Problem 3.13 Let E: $\mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ 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 a function ρ, where ρ is determined by randomly sampling from $\mathsf{Rand}(n)$. 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
>
> $$\Pr[\pi \xleftarrow{\$} \mathsf{Perm}(n) : \ A^\pi \Rightarrow 1] = \Pr[\rho \xleftarrow{\$} \mathsf{Rand}(n) : \ A^\rho \Rightarrow 1 \mid \overline{\mathsf{C}}].$$

Show that Michael and Peter have it all wrong: prove that $\Pr[\pi \xleftarrow{\$} \mathsf{Perm}(n) : \ A^\pi \Rightarrow 1]$ is not necessarily the same as $\Pr[\rho \xleftarrow{\$} \mathsf{Rand}(n) : \ A^\rho \Rightarrow 1 \mid \overline{\mathsf{C}}]$. 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.