Introduction to Modern Cryptography

Mihir Bellare[1] **Phillip Rogaway**[2]

November 3, 2003

Chapter 3: Edited for CIS 331 (cut short)

[1] Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093, USA. mihir@cs.ucsd.edu, http://www-cse.ucsd.edu/users/mihir

[2] Department of Computer Science, Kemper Hall of Engineering, University of California at Davis, Davis, CA 95616, USA; and Department of Computer Science, Faculty of Science, Chiang Mai University, Chiang Mai, 50200 Thailand. rogaway@cs.ucdavis.edu, http://www.cs.ucdavis.edu/∼rogaway

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.