Last Revised: January 26, 2000

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 wonderful conceptual starting points 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 try to understand their basic properties.

Let us remind the reader that we know from Chapter 2 what, *syntactically*, a block cipher is: it was defined to be a map $E : \mathit{Keys} \times \{0,1\}^n \to \{0,1\}^n$ such that $E(K,\cdot)$ is a permutation for every $K \in \mathit{Keys}$. Here n is some positive number, called the blocksize. In this chapter we'll investigate the question of what a block cipher supposed to do—when is it cryptographically useful.

3.1 Key Recovery

Classically, the security of a block ciphers has been looked at with regard to key recovery. That is, analysis of a block cipher E has focused primarily on the following question: given some number q of input-output examples $(M_1, C_1)), \ldots, (M_q, C_q)$, where K is a random, unknown key and $C_i = E_K(M_i)$, how hard is it for an attacker to find K? Good security requires that it be highly unlikely to find K unless q is huge or the running time for the attacker is unreasonable. This idea can be formalized, and we now turn towards doing so.

(Be warned that we begin by looking at key recovery primarily to build up our skills and intuition. Focusing on key recovery is going to turn out to be the wrong thing to do when trying to understand block-cipher security.)

To formalize an adversary's inability to be effective at key recovery we first need to answer the question of how the plaintexts $M_1, M_2, \ldots$ might be chosen. One simple possibility is that they are uniformly chosen at random. If we want to capture an adversary's inability to recover the key given random plaintexts and their ciphertexts, we can make a definition like this. Let A be an adversary that attacks a block cipher $E : \mathit{Keys} \times \{0,1\}^n \to \{0,1\}^n$. Then we'd define the *advantage* of A in recovering the key on random plaintext/ciphertext pairs as:

$$\begin{aligned}\mathbf{Adv}_E^{\mathrm{key}}(A) \;=\; & \Pr[K \stackrel{R}{\leftarrow} \mathit{Keys};\ \textbf{for } i \in \mathbf{N} \textbf{ do } M_i \stackrel{R}{\leftarrow} \{0,1\}^n; \\ & \textbf{for } i \in \mathbf{N} \textbf{ do } C_i = E_K(M_i) : \\ & A(M_1, C_1,\ M_2, C_2,\ \cdots) = K]\,.\end{aligned}$$

We'll be using this sort of notation a lot, so you better get used to it. Inside the brackets is an experiment, a colon, an then a boolean outcome. The experiment is, in effect, a little program. It is a semicolon-separated list of instructions. The instructions are executed one after another, going left to right. So the first instruction we execute is $K \stackrel{R}{\leftarrow} \mathit{Keys}$, which means to choose a random value from the set *Keys* and assign it to the variable K. Done. Next we execute the instruction which assigns a random n-bit sting to each plaintext M_i. Next we encipher M_i using $E_K(\cdot)$, calling the answer C_i. Now the experiment is over, and it is time to gauge the outcome. This is indicated by the colon. After the colon is the Boolean event. Here the event is that the adversary A, given all the plaintexts and ciphertexts, manages to output the correct key K.

Usually we think of our adversary as probablistic. In this case, there's actually one further, "hidden" part of the experiment, which is the selection of the adversary's random coins for the adversary. These coins are given to the adversary. If we wanted to explicitly indicate this dependency, we'd have to write

$$\begin{aligned}\mathbf{Adv}_E^{\mathrm{key}}(A) \;=\; & \Pr[K \stackrel{R}{\leftarrow} \mathit{Keys};\ \textbf{for } i \in \mathbf{N} \textbf{ do } M_i \stackrel{R}{\leftarrow} \{0,1\}^n; \\ & \textbf{for } i \in \mathbf{N} \textbf{ do } C_i = E_K(M_i);\ r \leftarrow \{0,1\}^\omega : \\ & A(M_1, C_1,\ M_2, C_2,\ \cdots; r) = K]\,.\end{aligned}$$

The function $\mathbf{Adv}^{\mathrm{key}}$ associates to each adversary A and block cipher E : $\mathit{Keys} \times \{0,1\}^n \to \{0,1\}^n$ a number between zero and one. The number quantifies how successful is the adversary. A score of zero indicates that the adversary never recovers the underlying key K; this is a terrible score from the adversary's point of view. A score of one indicates that the adversary recovers the key every single time; this is a great score from the adversary's point of view. A score of 0.1, for example, indicates that the adversary manages to recover the key 10% of the time.

Now security isn't just a matter of defeating some particular adversary A; we want to defeat *any* reasonable adversary A. To do this, we should ascertain what are the computational resources that matter. In this case, the number of (M_i, C_i) pairs that A inspects is interesting, as is the running time that A spends. We can make the following definition:

$$\mathbf{Adv}_E^{\mathrm{key}}(t,q) \quad = \quad \max\{\mathbf{Adv}_E^{\mathrm{key}}(A)\}$$

where the maximum is taken over all adversaries A that run in time t and inspect at most q ciphertexts from the list. While we don't have a formal definition of "secure," the idea is that E is secure (in the sense of key recovery, with random (plaintext, ciphertext) pairs) if $\mathbf{Adv}_E^{\mathrm{key}}(t,q)$ is close to zero as long as t and q are reasonable numbers.

Problems with key recover as a notion of security. Now that we have a notion of security we can ask if it is any good. A good notion should be sufficiently strong to be useful: if $\mathbf{Adv}_E^{\mathrm{key}}(t,q)$ is small when t and q are reasonable, then it should be possible to use E to make worthwhile constructions. But even a cursory glance at common block cipher usages shows that good security in the sense of key recovery is not sufficient for security.

Take for example the CTR mode of operation discussed in Chapter 2. Suppose that the block cipher had the following weakness: Given $C, F_K(C+1), F_K(C+2)$, it is possible to compute $F_K(C+3)$. Then clearly the encryption scheme is not secure, because if an adversary happens to know the first two message blocks, it can figure out the third message block from the ciphertext. (It is perfectly reasonable to assume the adversary already knows the first two message blocks. These might, for example, be public header information, or the name of some known recipient.) This means that if CTR mode encryption is to be secure, the block cipher must have the property that given $C, F_K(C+1), F_K(C+2)$, it is computationally infeasible to compute $F_K(C+3)$. Let us call this property SP1, for "security property one".

Of course, anyone who knows the key K can easily compute $F_K(C+3)$ given $C, F_K(C+1), F_K(C+2)$. And it is hard to think how one can do it without knowing the key. But there is no guarantee that someone *cannot* do this without knowing the key. That is, confidence in the security of F against key recovery does *not* imply that SP1 is true.

This phenomenon continues. As we see more usages of ciphers, we build up a longer and longer list of security properties SP1, SP2, SP3, ... that are necessary for the security of some block cipher based application.

Furthermore, even if SP1 is true, CTR mode encryption may still be weak. SP1 is not *sufficient* to guarantee the security of CTR mode encryption. Similarly with other security properties that one might naively come up with.

This long list of necessary but not sufficient properties is no way to treat security. What we need is *one single* "MASTER" property of a block cipher which, if met, *guarantees* security of *lots of* natural usages of the cipher. This property is that the block cipher is a pseudorandom permutation (or function).

Exercise 3.1.1 *Consider the block cipher E whose block size is 128 bits and which, for any $K \in$ Keys, is the identity function. This is clearly a terrible block cipher. What is $\mathbf{Adv}_E^{\mathrm{key}}(t,q)$? Interpret your answer.*

Exercise 3.1.2 *Let $E : \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ be a block cipher which, intuitively, is secure in the sense of key recover: if t, q are reasonable, $\mathbf{Adv}_E(t,q)$ is small. Modify E to a block cipher $E'_K(M)$ by setting $E'_K(M) = E_K(M)$ if $M \neq 0^{128}$, but $E_K(0^{128}) = K$. Relate the security key-recovery security of E' to the key-recovery security of E. Does E seem OK to you?*

Exercise 3.1.3 *Alfred says that $\Pr[x, x_1, \ldots, x_q \xleftarrow{R} \{0,1\}^n : x \in \{x_1, \ldots, x_q]$. is $q2^{-n}$. Is he right? (The notation, $a_1, \ldots, a_m \xleftarrow{R} A$ is shorthand for $a_1 \xleftarrow{R} A;\ a_2 \xleftarrow{R} A;\ \cdots a_m \xleftarrow{R} A$.)*

3.2 Function families

A *function family* is a map F: *Keys*$(F) \times$ *Dom*$(F) \to$ *Range*(F). Here *Keys*(F) is the set of keys of F; *Dom*(F) is the domain of F; and *Range*(F) is the range of F. The two-input function F takes a key K and input x to return a point y we denote by $F(K,x)$.

For any key $K \in$ *Keys*(F) we define the map F_K: *Dom*$(F) \to$ *Range*(F) by $F_K(x) = F(K,x)$. Thus, F specifies a collection of maps, one for each key. That's why we call F a family of functions.

Most often $Keys(F) = \{0,1\}^k$ and $Dom(F) = \{0,1\}^l$ and $Range(F) = \{0,1\}^L$ for some values of k, l, L. But there are some times where the domain or range could be sets of different kinds, containing strings of varying lengths.

There is some probability distribution on the set of keys $Keys(F)$. Most often, just the uniform distribution. So with $Keys(F) = \{0,1\}^k$, we are just drawing a random k-bit string as a key.

We denote by $K \stackrel{R}{\leftarrow} \{0,1\}^k$ the operation of selecting a random k-bit string from $\{0,1\}^k$ and naming it K. Then $f \stackrel{R}{\leftarrow} F$ denotes the operation: $K \stackrel{R}{\leftarrow} \{0,1\}^k$; $f \leftarrow F_K$. 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 function.

A *permutation* is a map whose domain and range are the same set, and the map is a bijection, ie. one-to-one and onto. That is, $f\colon D \to R$ is a permutation if $D = R$ and $x \neq y$ implies $f(x) \neq f(y)$ for all $x, y \in D$.

We say that F is a family of permutations if $Dom(F) = Range(F)$ and each F_K is a permutation on this common set.

Example 3.2.1 A block cipher is a family of permutations. For example, DES is a family of permutations with $Keys(\mathrm{DES}) = \{0,1\}^{56}$ and $Dom(\mathrm{DES}) = \{0,1\}^{64}$ and $Range(\mathrm{DES}) = \{0,1\}^{64}$. Here $k = 56$ and $l = L = 64$. Similarly for RC6.

There are two function families that we fix. One is $\mathrm{Rand}(l, L)$, the family of all functions of $\{0,1\}^l$ to $\{0,1\}^L$. The other is $\mathrm{Perm}(l)$, the family of all permutations on $\{0,1\}^l$. A random member of the first is simply a random function; a random member of the second is a random permutation.

What are these families, more precisely?

The key of a function in $\mathrm{Rand}(l, L)$ is simply the full description of the function. That is,

$$Keys(\mathrm{Rand}(l,L)) \quad = \quad \{\, (y_1, \ldots, y_{2^l}) \;:\; y_1, \ldots, y_{2^l} \in \{0,1\}^L \,\}$$

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

$$\mathrm{Rand}(l,L)((y_1, \ldots, y_{2^l}), x) = y_x \ .$$

Notice that the key space is very large; it has size 2^{L2^l}. Naturally, since 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 $f \stackrel{R}{\leftarrow} \text{Rand}(l, L)$ is the operation of picking a random function of l-bits to L-bits.

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

$$\begin{aligned} Keys(\text{Perm}(l)) &= \{(y_1, \ldots, y_{2^l}) : \; y_1, \ldots, y_{2^l} \in \{0,1\}^l \text{ and} \\ & \qquad y_1, \ldots, y_{2^l} \text{ are all distinct}\} \,. \end{aligned}$$

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

$$\text{Perm}(l)((y_1, \ldots, y_{2^l}), x) = y_x \,.$$

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

We will hardly ever actually think about these two 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.

Random functions and permutations A random function of l-bits to L-bits is a function chosen randomly from $\text{Rand}(l, L)$. That is, all functions are equally likely to be chosen. The function is identified with its table, which gives the output value for any associated input value. This table is exactly what we called the key above.

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 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.

Different things may be placed in the box, but one such thing is a random function. This means that someone picked g at random from $\text{Rand}(l, L)$, and put g in the box. Now, you query the box, with various values of inputs, all from $\{0,1\}^l$. What do you see?

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 $g(x)$. That is exactly what is a random function.

In other words, a random function of l-bits to L-bits can be thought of as a box which given any input $x \in \{0,1\}^l$ 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. It is this "dynamic" view that we suggest the reader have in mind in thinking about random functions.

The dynamic view can be thought of as following program. The program maintains a table of entries of the form (x, y).

Input: $x \in \{0,1\}^l$
If an entry (x, y) exists in table return y and exit
Else flip coins to determine a string $y \in \{0,1\}^L$ and return it
Also put (x, y) into the table.

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

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.

We are more used to the idea of picking points at random. Here we are picking a *function* at random.

Let's do some simple probabilistic computations to understand random functions. Fix $x \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\Pr\left[f \stackrel{R}{\leftarrow} \mathrm{Rand}(l, L) \ : \ f(x) = y \right] = 2^{-L} \ .$$

Notice it doesn't depend on l. Also it doesn't depend on the values of x, y.

Now fix $x_1, x_2 \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\Pr\left[f \stackrel{R}{\leftarrow} \mathrm{Rand}(l, L) \ : \ f(x_1) = f(x_2) = y \right] = \begin{cases} 2^{-2L} & \text{if } x_1 \neq x_2 \\ 2^{-L} & \text{if } x_1 = x_2 \end{cases}$$

This illustrates *independence*. Finally fix $x_1, x_2 \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\Pr\left[f \stackrel{R}{\leftarrow} \mathrm{Rand}(l, L) \ : \ f(x_1) \oplus f(x_2) = y \right] = \begin{cases} 2^{-L} & \text{if } x_1 \neq x_2 \\ 0 & \text{if } x_1 = x_2 \text{ and } y \neq 0^L \\ 1 & \text{if } x_1 = x_2 \text{ and } y = 0^L \end{cases}$$

Similar things hold for the sum of more than two things.

3.3 Pseudorandom functions

Let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions. Now, consider two worlds, in each of which you are given a box containing a function g, as follows:

World 0: You are given a box for a function g drawn at random from $\mathrm{Rand}(l, L)$, namely $g \stackrel{R}{\leftarrow} \mathrm{Rand}(l, L)$. (So g is just a random function of l bits to L bits.)

World 1: You are given a box for a function g drawn at random from F, namely $g \stackrel{R}{\leftarrow} F$. (If F is keyed via keys of length K, this means someone chose a key K at random, let $g = F_K$, and put g in the box. But they did NOT give you K!)

Now someone puts you in one of these worlds and asks you which one you are in. To find out, you are allowed to query the function g on points of your choice. Based on its input-output behavior, you must eventually make some decision as to which world you are in. Note that the only way you are allowed to access the box is by giving it an input x and obtaining output $g(x)$; you can't otherwise "look inside" the box.

The quality of pseudo-random family F can be thought of as measured by the difficulty of telling, in the above game, whether a function belongs in World 0 or World 1.

Intuitively, the game just models some way of "using" the function g in an application like an encryption scheme. If it is not possible to distinguish the input-output behavior of a random member of F from a truly random function, the application should behave in roughly the same way whether it uses a function from F or a random function. Later we will see exactly how this works out; for now let us continue to develop the notion. But right away one warning should be issued. Pseudorandom functions can't be substituted for random functions in *all* usages of random functions. To make sure it is OK in a particular application, you have to make sure that it falls within the realm of applications for which the formal definition below can be applied.

Example 3.3.1 Suppose F is a family of functions which happens to have the property that $F_K(0^l) = 0^L$ for all keys $K \in \{0,1\}^k$. Then F is *not* a pseudorandom function family. This is because we could easily tell whether we were in world 0 or world 1. Query the given function g at 0^l. If we get back 0^L we predict we are in world 1, else in world 0. The probability of being wrong is the probability that $g(0^l) = 0^L$ if g is random, and this is at most 2^{-L}.

The act of trying to tell which world you are in is formalized via the notion of a *distinguisher*. This is an algorithm which is provided oracle access to a function g and tries to decide if g is random or pseudorandom. (Ie. 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. Intuitively, a family is pseudorandom if the probability that the distinguisher says 0 is roughly the same regardless of which world it is in. We capture this mathematically by via the two "success" probabilities defined below.

Definition 3.3.2 Let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions, and let A be an algorithm that takes an oracle for a function g: $\{0,1\}^l \to \{0,1\}^L$, and outputs a bit. Then the "advantage" of A (in the prf-sense) is

$$\mathbf{Adv}_F^{\mathrm{prf}}(A) \quad = \quad \Pr\left[\, g \stackrel{R}{\leftarrow} F \;:\; A^g = 1 \,\right] - \Pr\left[\, g \stackrel{R}{\leftarrow} \mathrm{Rand}(l, L) \;:\; A^g = 1 \,\right] .$$

Finally for any t, q we let

$$\mathbf{Adv}_F^{\mathrm{prf}}(t,q) \quad = \quad \max_A \{\, \mathbf{Adv}_F^{\mathrm{prf}}(A) \,\}$$

where the maximum is over all A running in time t and making at most q oracle queries. ▌

Here t is on some fixed RAM model and includes the size of the code.

If $\mathbf{Adv}_F^{\mathrm{prf}}(A)$ is small, it means that a is outputting 1 just about as often in world 0 as in world 1, meaning it is not doing a good job of telling which world it is in. If this quantity is large (meaning close to one) then the distinguisher is doing well, meaning our family F is not pseudorandom.

For the family to be pseudorandom, the advantage of a distinguisher must be small, no matter what strategy the distinguisher tries. However, we expect that the advantage grows as the distinguisher invests more effort (more time, more queries) in the process. We can't expect it to be impossible for *any* distinguisher to tell the two worlds apart, but then neither is this what we need. Conforming to a general theme in cryptographic definitions, we only care what happens when the adversary (here the distinguisher) is limited in the amount of resources it can bring to the problem. This is because in reality, adversaries are limited; for example, they can't do 2^{80} steps of computation. To capture this we define the insecurity function $\mathbf{Adv}_F^{\mathrm{prf}}(\cdot,\cdot)$ as above. This is a function associated to any function family F. Once we fix F, this function becomes fixed. We may not always know its value at all points, but the value exists.

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 time and number of queries. A distinguisher can use whatever means desired to distinguish the function as long as it stays within the specified resource bounds. Nonetheless, it should fail, except possibly with probability $\mathbf{Adv}_F^{\mathrm{prf}}(t,q)$ where t is its running time and q the number of its queries.

There is one feature of the above parameterization about which everyone asks. Recall that k is the key-length for F. 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 insecurity function $\mathbf{Adv}_F^{\mathrm{prf}}(t,q)$. Why is this? $\mathbf{Adv}_F^{\mathrm{prf}}(t,q)$ 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, $\mathbf{Adv}_F^{\mathrm{prf}}(t,q)$ should be about $t/2^k$. But that is really an ideal; in practice we should not assume ciphers are this good.

3.4 PRF-Security vs. Key-Recovery Security

What I did here was to show that key recovery doesn't imply PRF-security, but that PRF-security implied key-recovery security. Here is the theorem, which was actually done, initially, with concrete parameters ($n = 128$ etc, $q = 10^6$, $\mathrm{Time}_E = 1$ msec.)

Theorem 3.4.1 Let $E : \mathit{Keys} \times \{0,1\}^n \to \{0,1\}^n$ be a function family. Let t and q be numbers such that $q < 2^n$. Suppose that there exists an adversary A who, in time t, inspecting q ciphertexts, achieves key-recovery advantage of $\epsilon = \mathbf{Adv}_E^{\mathrm{key}}(A)$. Then there exists an adversary A' that runs in time t', asks q' oracle queries, and achieves prf-advantage ϵ', where

$$
\begin{aligned}
t' &= t + \tilde{O}(nq) + \mathrm{Time}_E \\
q' &= q + 1 \\
\epsilon' &= \epsilon - 2^{-n}
\end{aligned}
$$

Then change notations, just for practice.

Corollary 3.4.2 Let $E : \mathit{Keys} \times \{0,1\}^n \to \{0,1\}^n$ be a function family. Let q and t be numbers such that $1 \le q < 2^n$. Then

$$\mathbf{Adv}_E^{\mathrm{prf}}(t,q) \le \mathbf{Adv}_E^{\mathrm{key}}(t',q') + 2^{-n}$$

where $t' = t - \tilde{O}(nq) - \mathrm{Time}_E$ and $q' = q - 1$.

3.5 An Apparently Stronger Notion

Next I related the PRF definition to a related one in which the adversary has a reference oracle for enciphering. This should build skills, and the proof is a gentle introduction to hybrid arguments.

Definition 3.5.1 Let F: $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a family of functions, and let A be an algorithm that takes two oracles, f, g: $\{0,1\}^l \to \{0,1\}^L$, and outputs a bit. Then the advantage of A (in the prf2-sense) is

$$\begin{aligned}\mathbf{Adv}_F^{\mathrm{prf2}}(A) \;\; = \;\; & \Pr\left[\, f, g \stackrel{R}{\leftarrow} F \;:\; A^{f,g} = 1 \,\right] - \\ & \Pr\left[\, f \stackrel{R}{\leftarrow} F;\; g \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) \;:\; A^{f,g} = 1 \,\right] .\end{aligned}$$

Finally for any t, q we let

$$\mathbf{Adv}_F^{\mathrm{prf2}}(t,q) \;\; = \;\; \max_A \{\, \mathbf{Adv}_F^{\mathrm{prf2}}(A) \,\}$$

where the maximum is over all A running in time t and making at most q oracle queries. ▮

Proposition 3.5.2 Let F : *Keys* $\times \{0,1\}^n \to \{0,1\}^n$ be a function family and let A_1 be an adversary. Suppose that A_1 runs in time t_1, asks at most q_1 oracle queries, and achieves advantage (in the prf-sense) of $\epsilon_1 = \mathbf{Adv}_F^{\mathrm{prf}}(A_1)$. Then there exists an adversary A_2 which runs in time t_2, asks at most q_2 total oracle queries, and achieves advantage (in the prf2-sense) of $\epsilon_2 = \mathbf{Adv}_F^{\mathrm{prf2}}(A_2)$, where

$$\begin{aligned}&t' = t + O(nq)\\&q' = q\\&\epsilon' = \epsilon\end{aligned}$$

Proposition 3.5.3 Let F : *Keys* $\times \{0,1\}^n \to \{0,1\}^n$ be a function family and let A_2 be an adversary. Suppose that A_2 runs in time t_2, asks at most q_2 total oracle queries, and achieves advantage (in the prf-sense) of $\epsilon_2 = \mathbf{Adv}_F^{\mathrm{prf2}}(A_2)$. Then there exists an adversary A_1 which runs in time t_1, asks at most q_1 oracle queries, and achieves advantage (in the prf-sense) of $\epsilon_1 = \mathbf{Adv}_F^{\mathrm{prf}}(A_1)$, where

$$\begin{aligned}&t' = t + O(nq)\\&q' = q\\&\epsilon' = \epsilon/2\end{aligned}$$

3.6 Pseudorandom Permutations

Recall that a block cipher F is a family of *permutations*; each individual function F_K in the family is a permutation. So it is somewhat unfair to ask a block-cipher to behave like a family of random functions; it is fairer to ask that it behave like a family of random permutations. This leads to the notion of a pseudorandom permutation, in which we proceed exactly as above, but replace $\mathrm{Rand}(l, L)$ with $\mathrm{Perm}(l)$.

Let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a family of functions. (Not necessarily permutations!) Now, consider two worlds, in each of which you are given a box containing a function g, as follows:

World 0: You are given a box for a function g drawn at random from $\mathrm{Perm}(l)$, namely $g \stackrel{R}{\leftarrow} \mathrm{Perm}(l)$. (So g is just a random permutation of l bits to l bits.)

World 1: You are given a box for a function g drawn at random from F, namely $g \stackrel{R}{\leftarrow} F$.

Definition 3.6.1 Let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a family of functions, and let D be an algorithm that takes an oracle for a function g: $\{0,1\}^l \to \{0,1\}^l$, and outputs a bit. Let

$$\begin{aligned} \mathbf{Succ}_F^{\mathrm{prp}}(D) &= \Pr\left[g \stackrel{R}{\leftarrow} F \ :\ D^g = 1 \right] \\ \mathbf{Succ}_{\mathrm{Perm}(l)}^{\mathrm{prp}}(D) &= \Pr\left[g \stackrel{R}{\leftarrow} \mathrm{Perm}(l) \ :\ D^g = 1 \right] . \end{aligned}$$

We call the difference between them the "advantage" of D:

$$\mathbf{Adv}_F^{\mathrm{prp}}(D) = \mathbf{Succ}_F^{\mathrm{prp}}(D) - \mathbf{Succ}_{\mathrm{Perm}(l)}^{\mathrm{prp}}(D) .$$

Finally for any t, q we let

$$\mathbf{Adv}_F^{\mathrm{prp}}(t, q) = \max_D \{ \mathbf{Adv}_F^{\mathrm{prp}}(D) \}$$

where the maximum is over all D running in time t and making at most q oracle queries. ▮

This $\mathbf{Adv}_F^{\mathrm{prp}}(\cdot,\cdot)$ is an insecurity function associated to any function family F. Once we fix F, this function becomes fixed.

3.7 An example

Intuitively, a if F is a PRF then, when K is random but unknown, the input-output behavior of F_K must resemble that of a random function. It may be worth seeing this illustrated in a very simply instance. Take a very simple property of a random function, that we discussed above in Section 3.2. Namely fix $x \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then

$$\Pr\left[\, f \stackrel{R}{\leftarrow} \mathrm{Rand}(l,L) \;:\; f(x) = y \,\right] = \frac{1}{2^L} \,.$$

Now, consider the quantity

$$\Pr\left[\, f \stackrel{R}{\leftarrow} F \;:\; f(x) = y \,\right] \,.$$

If F is a good PRF, this value should be close to the value in the random case, namely 2^{-L}. How close? This is determined by the insecurity function of F, once we have decided how much time, and how many queries, this property corresponds to. The latter needs some explaining.

Let us first give an answer and then justify it. Let $\epsilon = \mathbf{Adv}_F^{\mathrm{prf}}(t,1)$ where t is to be specified later. We claim that

$$\Pr\left[\, f \stackrel{R}{\leftarrow} F \;:\; f(x) = y \,\right] \leq \frac{1}{2^L} + \epsilon \,. \tag{3.1}$$

In fact it is also true that $\Pr\left[\, f \stackrel{R}{\leftarrow} F \;:\; f(x) = y \,\right] \geq 2^{-L} - \epsilon$, but since this example is for illustrative purposes only, we illustrate only the upper bound.

Let us now proceed to formally prove that Equation (3.1) is correct. How do we set up the proof? There are not too many choices; like all these proofs in cryptography it is a reductionist argument.

Imagine the claim were false, so that $\Pr\left[\, f \stackrel{R}{\leftarrow} F \;:\; f(x) = y \,\right]$ is much larger than $\frac{1}{2^L}$. But we know, from the above that if f were random, the probability would be 2^{-L}. This means we have an event whose probability is different depending on which world we are in. But that means we have a means of telling which world we are in: just see whether the event happens, and, if so, bet on world 1 (ie. f is pseudorandom) because the event is more likely here; if not, bet on world 0 because the event is less likely here. Formally, define a distinguisher D as follows:

Distinguisher D^f
  Let $w \leftarrow f(x)$
  If $w = y$ then return 1 else return 0

Now we see that

$$\begin{aligned}
\mathbf{Succ}_F^{\mathrm{prf}}(D) &= \Pr\left[f \xleftarrow{R} F \;:\; f(x) = y \right] \\
\mathbf{Succ}_R^{\mathrm{prf}}(D) &= \Pr\left[f \xleftarrow{R} \mathrm{Rand}(l,L) \;:\; f(x) = y \right] .
\end{aligned}$$

We know the second quantity is 2^{-L}, so

$$\begin{aligned}
\mathbf{Adv}_F^{\mathrm{prf}}(D) &= \Pr\left[f \xleftarrow{R} F \;:\; f(x) = y \right] - \Pr\left[f \xleftarrow{R} \mathrm{Rand}(l,L) \;:\; f(x) = y \right] \\
&= \Pr\left[f \xleftarrow{R} F \;:\; f(x) = y \right] - \frac{1}{2^L} .
\end{aligned}$$

Or, rewriting,

$$\Pr\left[f \xleftarrow{R} F \;:\; f(x) = y \right] \;\leq\; \mathbf{Adv}_F^{\mathrm{prf}}(D) + \frac{1}{2^L} .$$

However, D makes only one oracle query. So we also know that

$$\mathbf{Adv}_F^{\mathrm{prf}}(D) \;\leq\; \mathbf{Adv}_F^{\mathrm{prf}}(t,1)$$

as long as t is larger than the running time of D. The running time of D is very small. It suffices to set $t = O(l + L)$. Now combining the last two inequalities above we get Equation (3.1).

It is worth adding a word of caution. What is happening here is not going to *always* be true. Namely, not every event true for a random function happens with probability only slightly different for a pseudo-random function. Only certain kinds of events. Which kinds? The ones, intuitively, which can be "measured" by a distinguisher. In other words, ones for which a proof like the above works.

3.8 Conjectures about block ciphers

We might conjecture that existing block ciphers meet such notions to some degree. We can't prove these conjectures, however.

For example We might conjecture something like:

$$\mathbf{Adv}_{\mathrm{DES}}^{\mathrm{prp}}(t,q) \;=\; c_1 \cdot \frac{t/T_{\mathrm{DES}}}{2^{55}} + \frac{q}{2^{40}}$$

Here T_{DES} is the time to do one DES computation on our fixed model of computation.

In other words, we are conjecturing that the best attacks are either exhaustive key search or linear cryptanalysis. We might want to give ourselves a bit more slack, ie. be more conservative.

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

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{DES}}(t,q) \quad = \quad c_1 \cdot \frac{t/T_{\mathrm{DES}}}{2^{55}} + \frac{q^2}{2^{64}} \ .$$

This is due to the birthday attack. It is possible because a block cipher is a family of permutations, and is a general phenomenon, studied in the next subsection.

3.9 The birthday attack

Suppose you are given an oracle for a function g, where g is either drawn randomly from a family of permutations, or g is a random function. How can you tell which? Query the oracle at successive points $x_1, x_2, \ldots$, and get back values $y_1, y_2, \ldots$. You know that if g were a permutation, the values $y_1, y_2, \ldots$ 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. The reason is the birthday paradox. If you are not familiar with this, you may want to look at the Appendix of these notes for a refresher, and then come back to the following, which invokes results from the Appendix.

Proposition 3.9.1 Let P: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be any family of permutations. Then there is a distinguisher D, making q queries ($2 \le q \le 2^{(l+1)/2}$) and using $t = O(ql)$ time, such that

$$\mathbf{Adv}^{\mathrm{prf}}_{P}(D) \quad \ge \quad 0.158 \cdot \frac{q^2}{2^l} \ .$$

As a consequence

$$\mathbf{Adv}^{\mathrm{prf}}_{P}(t,q) \quad \ge \quad 0.158 \cdot \frac{q^2}{2^l} \ .$$

Proof: The birthday attack works like this:

Distinguisher D^g
For $i = 1, \ldots, q$ do

Let x_i be the i-th l-bit string in lexicographic order
Let $y_i = g(x_i)$
End For
If $y_1, \ldots, y_q$ are all distinct output 1, else output 0

Clearly $\mathbf{Succ}_P^{\mathrm{prp}}(D) = 1$, because if g is a permutation then $y_1, \ldots, y_q$ are all distinct.
What if g is a random function? What is the chance that $y_1, \ldots, y_q$ are all distinct? This is the birthday problem. We are throwing q balls into $N = 2^l$ buckets and asking what is the chance of at least one collision. So we can use the Fact in the Appendix to say that

$$\begin{aligned}\mathbf{Succ}_{\mathrm{Rand}(l,L)}^{\mathrm{prf}}(D) &= 1 - C(N,q) \\ &\leq 1 - 0.158 \cdot \frac{q^2}{2^l} .\end{aligned}$$

Hence

$$\begin{aligned}\mathbf{Adv}_P^{\mathrm{prf}}(D) &= \mathbf{Succ}_F^{\mathrm{prf}}(D) - \mathbf{Succ}_{\mathrm{Rand}(l,L)}^{\mathrm{prf}}(D) \\ &\geq 1 - \left(1 - 0.158 \cdot \frac{q^2}{2^l}\right) \\ &= 0.158 \cdot \frac{q^2}{2^l} .\end{aligned}$$

Which was the claim. To obtain the second part, just take the maximum.
▮

3.10 PRFs versus PRPs

When we come to analyses of block cipher based constructions, we will find a curious dichotomy. Analyses are considerably simpler and more natural assuming the block cipher is a PRF. Yet, PRPs are what block ciphers more naturally approximate. To bridge the gap, we relate the two insecurity functions. The following says, roughly, that the birthday attack is the best possible one. A particular family of permutations P may have insecurity in the PRF sense that is greater than that in the PRP sense, but only by an amount of $q^2/2^{l+1}$, the collision probability term in the birthday attack.

Proposition 3.10.1 Suppose P: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ is a family of permutations. Then

$$\mathbf{Adv}_P^{\mathrm{prf}}(t,q) \quad \leq \quad \frac{q^2}{2^{l+1}} + \mathbf{Adv}_P^{\mathrm{prp}}(t,q) \; .$$

The proof of this is left as an exercise.

3.11 Historical Notes

The basic notion of pseudorandom functions is due to Goldreich, Goldwasser and Micali [GGM]. In particular these authors introduced the important notion of distinguishers. Their setting is however not the one used here. Theirs was "asymptotic". They worked in the standard complexity-theoretic framework 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. That approach is convenient theoretically, but not so useful in application to practice.

The motivation of [GGM] was quite different. They were not approaching this from a block cipher point of view, as we are; rather, they wanted to build pseudorandom functions out of number-theoretic or other primitives.

The notion of finite pseudorandom function families, and the quantization of security in terms of $\mathbf{Adv}_F^{\mathrm{prf}}(\cdot,\cdot)$ was introduced by [BKR]. While the basic technical idea is the same (namely that of distinguishers), the motivation, and hence the formalization, are different. The idea is now to model block ciphers via PRF families. Accordingly, input and output lengths are fixed. The provided family consists of maps of l bits to L bits (eg. $l = L = 64$ for DES) and there is no directly provided function taking more bits of input, as is true for DES. Furthermore, in the quantitative treatment of security, not only is time quantified, but queries are separately quantified because these resources have different costs; this is a distinction not made in the complexity theoretic works.

When finite PRFs are instantiated with block ciphers, the approach yields *efficient* PRFs. However, we pay a price for the efficiency, namely the finiteness. It is harder to design schemes with finite PRFs than with infinite ones. Nonetheless, to get efficient schemes, we do start with finite PRFs. We will see later how to get provably secure but efficient private key encryption and message authentication schemes from them.

One should note that underlying the "asymptotic" or complexity theoretic results are concrete constructions, and their security can be analyzed exactly by delving into the proofs. However in most cases this has not been done.

The construction in Section **??** is due to [GGM].

Luby and Rackoff [LR] introduced a notion of pseudorandom permutations. It is however a little different from ours. They take a family of permutations and measure its distance from a family of random functions. In contrast we take a family of any kind (whether permutations or functions) and measure its distance from a family of random permutations. In the asymptotic setting used in [LR], there is no difference, but in the concrete security setting we use here, it matters a lot.

For more information on the kinds of approaches used here, see [Be].

3.12 Appendix: The Birthday Problem

The setting is that we have q balls. View them as numbered, $1, \ldots, q$. We also have N bins, where $N \geq q$. We throw the balls at random into the bins, one by one, beginning with ball 1. At random means that each ball is equally likely to land in any of the N bins, and the probabilities for all the balls are independent. A collision is said to occur if some bin ends up containing at least two balls. We are interested in $C(N, q)$, the probability of a collision.

The birthday paradox is the case where $N = 365$. We are asking what is the chance that, in a group of q people, there are two people with the same birthday, assuming birthdays are randomly and independently distributed over the days of the year. It turns out that when q hits $\sqrt{365}$ the chance of a birthday collision is already quite high, around $1/2$.

This fact can seem surprising when first heard. The reason it is true is that the collision probability $C(N, q)$ grows roughly proportional to q^2/N. This is the fact to remember. The following gives a more exact rendering, providing both upper and lower bounds on this probability.

Fact 3.12.1 Let $C(N, q)$ denote the probability of at least one collision when we throw $q \geq 1$ balls at random into $N \geq q$ buckets. Then

$$C(N, q) \quad \leq \quad \frac{q^2}{2N} .$$

Also

$$C(N, q) \quad \geq \quad 1 - e^{-q(q-1)/2N} ,$$

and for $2 \le q \le \sqrt{2N}$

$$C(N,q) \quad \ge \quad 0.158 \cdot \frac{q^2}{N} \; .$$

In the proof we will find the following inequalities useful to make estimates.

Fact 3.12.2 For any real number $x \in [0,1]$—

$$\left(1 - \frac{1}{e}\right) \cdot x \; \le \; 1 - e^{-x} \; \le \; x \; .$$

Proof of of Fact 3.12.1: Let C_i be the event that the i-th ball collides with one of the previous ones. Then $\mathbf{P}\left[C_i\right]$ is at most $(i-1)/N$, since when the i-th ball is thrown in, there are at most $i-1$ different occupied slots and the i-th ball is equally likely to land in any of them. Now

$$\begin{aligned} C(N,q) &= \mathbf{P}\left[C_1 \vee C_2 \vee \cdots \vee C_q\right] \\ &\le \mathbf{P}\left[C_1\right] + \mathbf{P}\left[C_2\right] + \cdots + \mathbf{P}\left[C_q\right] \\ &\le \frac{0}{N} + \frac{1}{N} + \cdots + \frac{q-1}{N} \\ &= \frac{q(q-1)}{2N} \\ &\le \frac{1}{2} \cdot \frac{q^2}{N} \; . \end{aligned}$$

This proves the upper bound.
For the lower bound we let D_i be the event that there is no collision after having throw in the i-th ball. If there is no collision after throwing in i balls then they must all be occupying different slots, so the probability of no collision upon throwing in the $(i+1)$-st ball is exactly $(N-i)/N$. That is,

$$\Pr\left[D_i \mid D_{i+1}\right] \; = \; \frac{N-i}{N} \; = \; 1 - \frac{i}{N} \; .$$

Also note $\mathbf{P}\left[D_1\right] = 1$. The probability of no collision at the end of the game can now be computed via

$$\begin{aligned} 1 - C(N,q) &= \mathbf{P}\left[D_q\right] \\ &= \Pr\left[D_{q-1} \mid D_q\right] \cdot \mathbf{P}\left[D_{q-1}\right] \\ &\vdots \quad \vdots \end{aligned}$$

$$= \prod_{i=1}^{q-1} \Pr\left[D_i \mid D_{i+1}\right]$$

$$= \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right) .$$

Note that $i/N \leq 1$. So we can use the inequality $1 - x \leq e^{-x}$ for each term of the above expression. This means the above is not more than

$$\prod_{i=1}^{q-1} e^{-i/N} \;=\; e^{-1/N-2/N-\cdots-(q-1)/N} \;=\; e^{-q(q-1)/2N} .$$

Putting all this together we get

$$C(N,q) \;\geq\; 1 - e^{-q(q-1)/2N} ,$$

which is the second inequality in Fact 3.12.1. To get the last one, we need to make some more estimates. We know $q(q-1)/2N \leq 1$ because $q \leq \sqrt{2N}$, so we can use $1 - e^{-x} \geq (1 - e^{-1})x$ to get

$$C(N,q) \;\geq\; \left(1 - \frac{1}{e}\right) \cdot \frac{q(q-1)}{2N} .$$

We also know that $q \geq 2$ and thus $q - 1 \geq q/2$. Plugging this in we get

$$C(N,q) \;\geq\; \left(1 - \frac{1}{e}\right) \frac{1}{4} \cdot \frac{q^2}{N} .$$

A computation of the constant here completes the proof. ▮

3.13 Problems

Exercise 3.13.1 *Analyze the exhaustive key search attack in the same vein that we analyzed the birthday attack. Namely, state and prove a lower bound on* $\mathbf{Adv}_F^{\mathrm{prf}}(t,q)$ *obtained by considering the distinguisher that runs an exhaustive key search attack.*

Exercise 3.13.2 *Prove Proposition 3.10.1.*

Exercise 3.13.3 *Let* $E : \mathit{Keys} \times \{0,1\}^n \to \{0,1\}^n$ *be a function family. Let* A *be an adversary. Recall that*

$$\mathbf{Adv}_E^{\mathrm{prf}}(A) \;=\; \Pr[K \stackrel{R}{\leftarrow} \mathit{Keys} :\; A^{E_K}(\cdot) = 1] - \Pr[\rho \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) :\; A^{\rho(\cdot)} = 1] .$$

Let

$$\begin{aligned}\mathbf{Adv}_E^{\mathrm{prf2}}(A) \;&=\; 2\Pr[b \leftarrow \{0,1\};\ \mathbf{if}\ b=1\ \mathbf{then}\ K \stackrel{R}{\leftarrow} \mathit{Keys},\ f \leftarrow E_K(\cdot) \\ &\qquad \mathbf{else}\ f \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) :\ A^{f(\cdot)} = b] - 1\ .\end{aligned}$$

Show that these two measures of advantage exactly coincide: $\mathbf{Adv}_E^{\mathrm{prf}}(A) = \mathbf{Adv}_E^{\mathrm{prf2}}(A)$.

Remark: This is a much stronger sort of equivalence than we need to regard two security measures as equivalent: it would have been enough that for every adversary A there exists an adversary A' of similar resources to A such that $\mathbf{Adv}_E^{\mathrm{prf}}(A) \approx \mathbf{Adv}_E^{\mathrm{prf2}}(A')$*; and for every adversary A' there exists an adversary A of similar resources to A' such that* $\mathbf{Adv}_E^{\mathrm{prf}}(A) \approx \mathbf{Adv}_E^{\mathrm{prf2}}(A')$.

Bibliography

[Be] MIHIR BELLARE. Practice-oriented provable security. Available via `http://www-cse.ucsd.edu/users/mihir/crypto-papers.html`.

[BKR] M. BELLARE, J. KILIAN AND P. ROGAWAY. The security of cipher block chaining. CRYPTO '94. Springer-Verlag, 1994.

[GGM] O. GOLDREICH, S. GOLDWASSER AND S. MICALI. How to construct random functions. *Journal of the ACM*, Vol. 33, No. 4, 1986, pp. 210–217.

[LR] M. LUBY AND C. RACKOFF How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput*, Vol. 17, No. 2, April 1988.