Cryptography Lecture 5
Stream and (more) Block Ciphers

David Wagner (lecturer) and Leon Barrett (scribe)

January 31, 2006

1 Stream ciphers

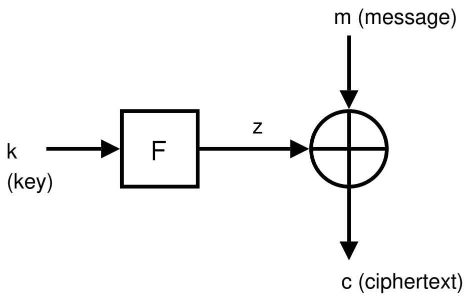


Figure 1: A stream cipher

In a stream cipher, a "key stream" is generated and then used to encrypt the message in a simple way (usually an XOR). If F is a PRF (pseudorandom function), then z is indistinguishable from a uniform random string, so z can be used just like a 1-time pad, as shown in Fig. 1. Because it works like a one time pad, the system in the figure has the flaw that it can only encrypt a single message for each key, k. To avoid the flaw, we want to extend this design as in Fig. 2, where we add a non-repeating "nonce" to lengthen the key and allow us to reuse k.

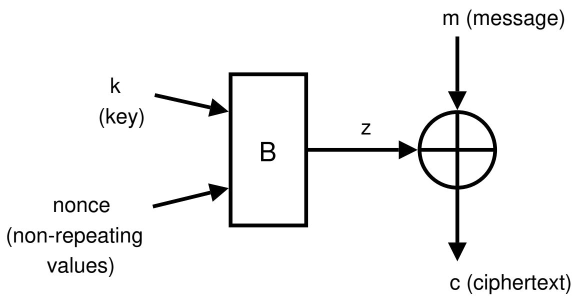


Figure 2: A stream cipher with reusable key

So, what properties do we need for security? We still want z to be indistinguishable from a uniform random bit string. Making box B a PRF (pseudorandom function) should be sufficient. Consider the case where B is a random function, R : keys $\times$ nonces $\rightarrow \{0,1\}^n$. As long as the inputs are distinct, then the outputs are uniform and independent. But, of course, storing the table needed for R requires exponential space. A PRF can produce the same sort of (pseudo) random output without the huge table. This is the model we are looking for—it ensures that any application using this cipher can choose nonces arbitrarily (as long as there are no repetitions) and still get a random z for encryption.

1.1 Expanding the message length

What if the PRF only produces a short output (e.g. 128-bit outputs, as AES does)? Then, we can only encrypt tiny messages, which means it is not a very useful cipher. There are two ways to make the encryption work on longer messages.

The first way to fix this is to expand the nonce with a counter and break the message up into small, numbered chunks. Then, we use these numbers as input to the encryption function. So, the space of nonces becomes the space of pairs of nonces $\times$ counters. Since pairs of nonces and counters are never duplicated (a different nonce for each message, and a different counter value for each part of the message), the result is still secure.

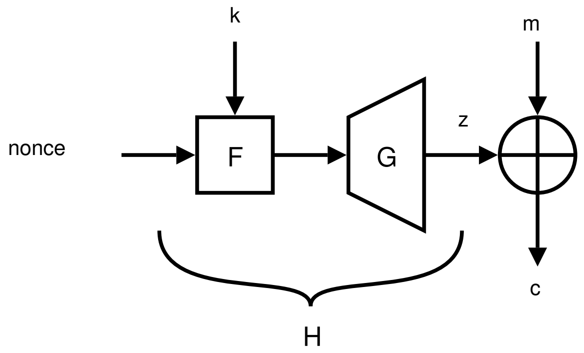


Figure 3: A PRF that can encode longer messages

The second way is to expand the output of the PRF by using a PRG, as in Fig. 3. It looks secure (z should still be random), but we want to know for sure, which means we should prove it. Suppose we have:

$$\begin{aligned} F &: \text{keys} \times \text{nonces} \rightarrow \{0,1\}^{128} \\ G &: \{0,1\}^{128} \rightarrow \{0,1\}^n \\ H(k, \text{nonce}) &= G(F(k, \text{nonce})) \end{aligned}$$

Theorem 1. *If F is a (t, q, ϵ) PRF and G is a (t, ϵ') PRG then H is a $(t - O(q), q, \epsilon + q\epsilon')$ PRF.*

Outline of proof: Consider Fig. 4. Suppose H is a black box, so the adversary specifies an input x and gets an output y. We will argue that H is indistinguishable from random function R, where k is a random key which the adversary does not know. We can show that case (1) is $(t - O(q), q, \epsilon)$ indistinguishable from case (3). First, we will show that (1) is indistinguishable from (2) (Lemma 2). Then, we will show that (2) is indistinguishable from (3).

Lemma 2. *$G(F_k(\cdot))$ is $(t - O(q), q, \epsilon)$ indistinguishable from $G(S(\cdot))$.*

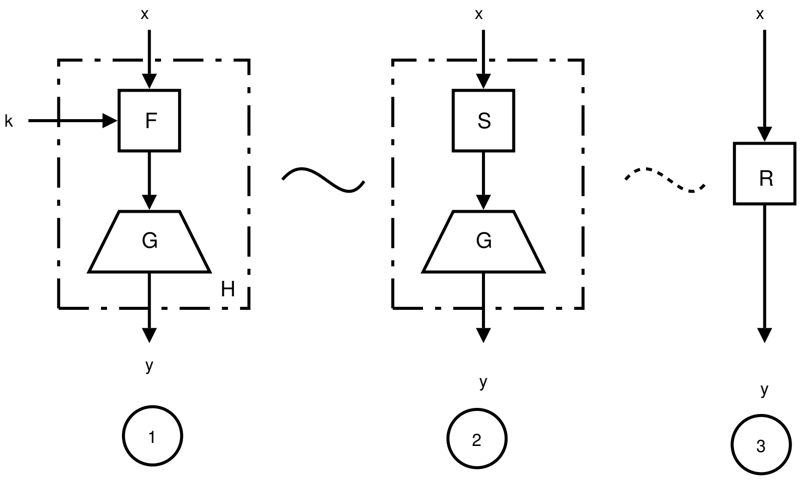


Figure 4: Showing that the expanded PRF is indistinguishable from a random function

Proof: Suppose A distinguishes them with Adv$A > \epsilon$. Let B be $B^g : A^{G(g(\cdot))}$, i.e., whenever A makes an oracle query x, send it back $G(g(x))$, then output the response of A.

In Fig. 5(a), the situation is shown graphically. A can be hooked up to either $G(F_k(\cdot))$ or $G(S(\cdot))$, and it can distinguish between them with advantage $> \epsilon$. So, we abstract out the difference between them, producing a structure as in Fig. 5(b). This is B, and it may be connected to either $F_k(\cdot)$ or $S(\cdot)$ as in Fig. 5(c).

$$\begin{aligned} \text{Adv}B &= |Pr[B^{F_k(\cdot)} = 1] - Pr[B^{S(\cdot)} = 1]| \\ &= |Pr[A^{G(F_k(\cdot))} = 1] - Pr[A^{G(S(\cdot))} = 1]| \\ &= \text{Adv}A \end{aligned}$$

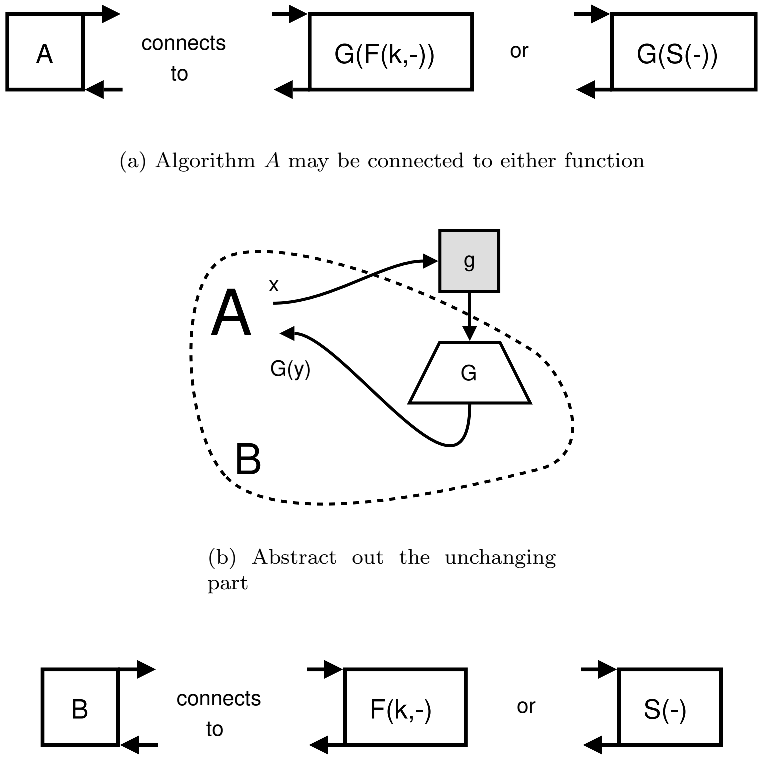
So, the advantage of B is the same that of A; but the advantage of B cannot be greater than ϵ because F is a (t, q, ϵ) PRF. Also, the running time of B is the running time of A, plus an $O(q)$ term which comes from emulating G. So, $G(F_k(\cdot))$ is $(t - O(q), q, \epsilon)$ indistinguishable from $G(S(\cdot))$.

Having shown the truth of Lemma 2, we wish to continue our proof of Theorem 1.

For Fig. 4, part (1), we just showed that the output of F is indistinguishable from uniform and random when seen through the lens of G. We can then show that the output of G must be indistinguishable from a uniform random output, through a chain of inference as in Fig. 6. We will construct a series of hybrid systems, relate them to their neighbors in the chain, and thus relate system 0 to system q. Specifically, hybrid i is a system in which the first i queries are taken from Fig. 4 (3), and the rest are then taken from Fig. 4 (2). Thus, each system differs from its neighbors in only one query. Since there are only q queries, then there are at most q hybrids, and they begin in system (3) and end in system (2).

So, to prove that systems 0 and q are equivalent, we need only show that any two adjacent hybrids are equivalent. As you can see in Fig. 7, any adjacent hybrids differ only in a single query. Now, if an algorithm A can distinguish between these systems, then it can break G with a construction as shown in Fig. 8. (The logic for this should be becoming familiar by now.) So, the two neighboring hybrids are (t, q, ϵ') indistinguishable. This means the chain costs ϵ' at each step, and there are q steps, so the total cost is $q\epsilon'$. Thus, Fig. 4 (2) and (3) are equivalent, and we have completed the proof of Theorem 1.

There is a minor problem that occurs if A repeats its queries. $G(S(x))$ is deterministic, and $R(x)$ is deterministic, but $G(S(x))$ is not necessarily equal to $R(x)$. Thus, a repeated query could detect



(a) Algorithm A may be connected to either function

(b) Abstract out the unchanging part

(c) Algorithm B may be connected to either function

Figure 5: Indistinguishability of F_k and S

the transition from one to the other and distinguish between adjacent hybrids. To avoid this, we simply assume that A never repeats its queries. (After all, the cipher breaks down anyway if a repeated key/nonce pair is used, just like in the one-time pad.)

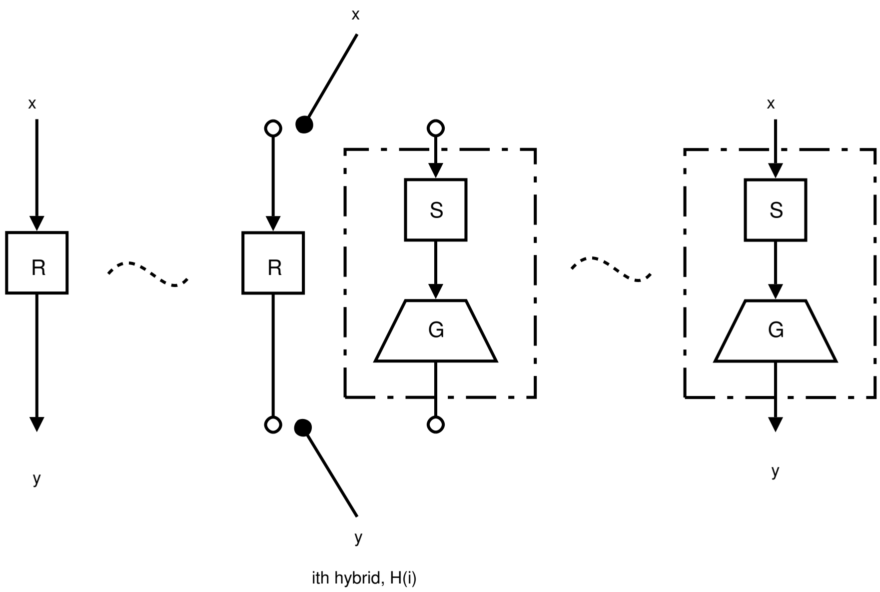


Figure 6: Show indistinguishability through a sequence of q hybrids

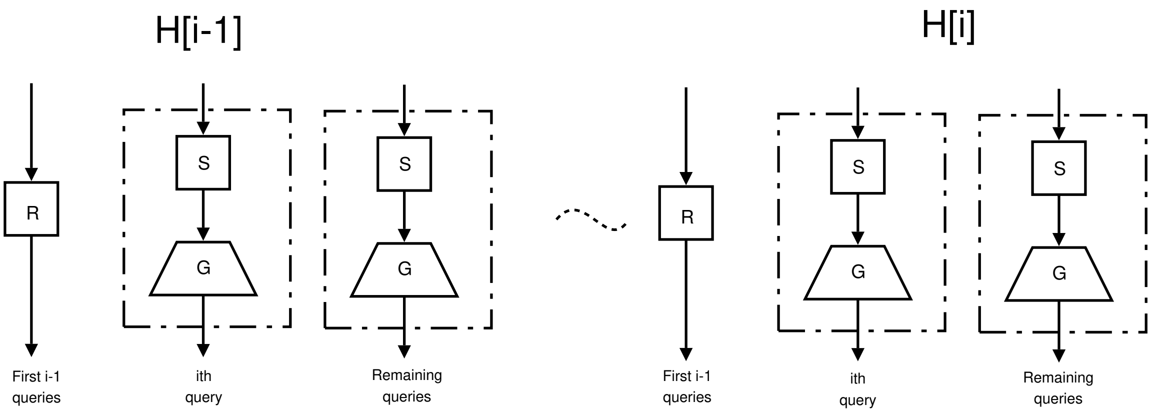


Figure 7: Show indistinguishability through a sequence of q hybrids

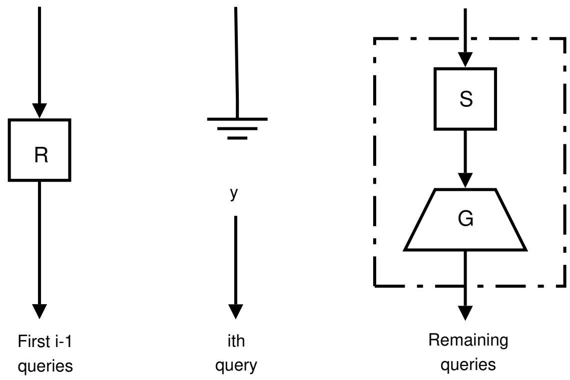


Figure 8: Constructing B from A

2 Block Ciphers

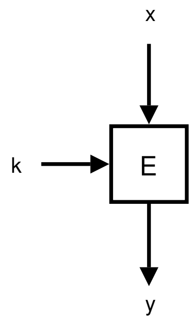


Figure 9: A block cipher

A block cipher is shown in Fig. 9. It is a function such as

$$E : k \times \{0,1\}^{128} \to \{0,1\}g^{128}$$

128 bits are chosen in this example because the common, secure block cipher AES uses 128 bits. However, any E_k is 1-to-1, so, for example, $E_k(17) \neq E_k(18)$. (This 1-1 property is needed for decryption, so each possible message has a different ciphertext.) But, for a random function, duplicate outputs are possible, so E does not behave like a random function. Instead, it behaves like a random permutation (Fig. 10):

$$\pi \xleftarrow{\$} \mathrm{Perm}(\{0,1\}^{128} \to \{0,1\}^{128})$$

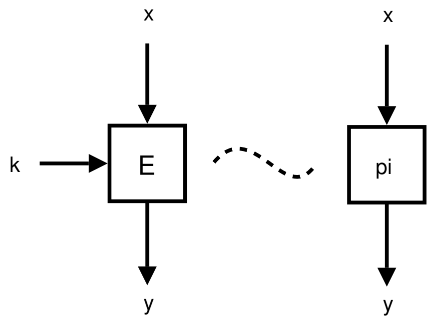


Figure 10: A PRP compared to a PRF

Definition 1. *E is a (t, q, ϵ) pseudorandom permuatation (PRP) if Adv $A \leq \epsilon$ $\forall A$ where*

$$Adv\ A = \left| Pr[k \xleftarrow{\$} keys;\ A^{E_k(\cdot)} = 1] - Pr[\pi \xleftarrow{\$} Perms;\ A^{\pi} = 1] \right|$$

(We think that AES is a $(t, q, t/2^{128})$ PRP. There is not yet any known way to break it, other than a brute-force attack.)

So, last class, saying that block ciphers are indistinguishable from PRFs was a bit of a lie. . . but not much. Actually, it looks like PRPs are a lot like PRFs (Fig. 11). We could certainly attack this pair by

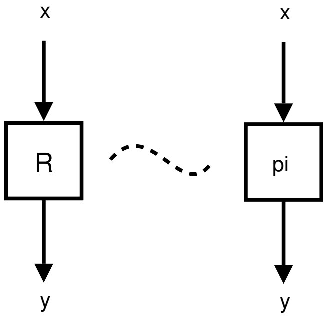


Figure 11: A PRP compared to a random function

looking for repetitions in q queries. That attack is: if $\exists_{i \neq j} y_i = y_j$ (i.e. there is a repetition), then output 1; else output 0. The result of this (like the birthday problem, discussed in § 2.1) is

$$\text{Adv} = \frac{\binom{q}{2}}{2^n} \approx \frac{q^2}{2^{n+1}}$$

This is nice and low, but can we trust this? Can we prove that it is the only source of advantage? Indeed we can, and so we shall.

Theorem 3. *π is $(\infty, q, \frac{q^2}{2^{n+1}})$ indistinguishable from R.*

Note: This proof contains a subtle error. It used to be presented exactly this way in textbooks (in fact, Dr. Wagner described being very astonished when he read that it was false), but fortunately we can correct that error. For a way to avoid these problems in the future, see § 3.

Proof: We take the following as a known fact:

$$0.3\frac{q^2}{2^n} \leq \Pr[\text{collision}] \leq 0.5\frac{q^2}{2^n}$$

These bounds come from our knowledge about combinatoric problems.

Without loss of generality, assume that any adversary A makes no repeated requests (it could simply cache its results). Let c be the event of a collision (when two different queries give the same response).

$$\Pr[A^R = 1] \quad = \quad \Pr[A^R = 1|\bar{c}]\Pr[\bar{c}] + \Pr[A^R = 1|c]Pr[c]$$

Upper-bound some probabilities by 1 to get

$$\Pr[A^R = 1] \leq \Pr[A^R = 1|\bar{c}] + Pr[c]$$

Then, when we require distinct values, the random function gives the same output as a permutation. *

$$\begin{aligned} \Pr[A^R = 1] &\leq \Pr[A^\pi = 1] + 0.5\frac{q^2}{2^n} \\ \Pr[A^R = 1] - \Pr[A^\pi = 1] &\leq 0.5\frac{q^2}{2^n} \end{aligned}$$

Thus, we have derived an upper bound on the advantage, and we can declare the two indistinguishable. (We can also lower-bound this quantity in a similar way.)

The subtle problem is that the statement marked with a * is not true if A does not make exactly q queries. For example, consider a 1-bit PRF. The solution is to assume, without loss of generality, that A makes exactly q queries.

2.1 The Birthday Problem

The heart of the so-called "Birthday Problem" is the surprising fact that in a group of 23 randomly-selected people, it is about 50% likely that two of them will have their birthdays on the same day of the year. This number goes up quickly and hits 90% around 40 people. This surprisingly high probability is due to the combinatorics of the problem. I like to think about it this way:

Each person has a chance to match with each person before them. So, the second person only has one other to match with; a match is unlikely. However, the third person has 2 chances (out of 365) to match. We can treat probabilities this low as being approximately additive, so by the 23rd person, we have 253 chances out of 365 to match birthdays. This approximation is only good for low numbers of people, but it provides intuition for why this probability gets so high so fast. For low numbers of people, then, the probability grows quadratically.

The exact probability can be worked out without too much difficulty, but there is no real need to do that here, as an in-depth explanation may be found at the following URL:
http://mathworld.wolfram.com/BirthdayProblem.html

3 Game-Based Proofs

Some people are arguing for "game-based proofs" to avoid making these subtle mistakes. In such proofs, the functions are spelled out very precisely in pseudocode. Then, they may be analyzed clearly and easily.

For example, in this problem, consider the pseudocode in 1.

Algorithm 1 Game P

init:
- bad $\leftarrow$ false
- **for** $x \in \{0,1\}^n$ **do**
  - $\alpha(x) \leftarrow$ undef
- **end for**

when A asks query x:
- $y \xleftarrow{\$} \{0,1\}^n$
- **if** $y \in \text{Range}(\alpha)$ **then**
  - bad $\leftarrow$ true
  - $y \xleftarrow{\$} \overline{\text{Range}(\alpha)}$ *
- **end if**
- $\alpha(x) \leftarrow y$
- **return** y

Game R is identical except that it removes the line marked *. So, Game P is random permuatation π, and Game R is random function R. If the bad flag is false, then the two are indistinguishable because they executed the same statements, so the outputs of both are the same. So, for any adversary A,

$$\Pr[A^{\text{Game } P} = 1 | \overline{\text{bad}}] = \Pr[A^{\text{Game } R} = 1 | \overline{\text{bad}}]$$

That is, *no* algorithm cannot distinguish between them when bad is false! So, we can bound the advantage by Pr[bad]. The equivalence is then obvious by inspection of the games.

This example gives a feeling for the idea of game-based proofs, but the key is that the details are taken care of in the preparation of the game language and description. However, once these are carefully created, the proof is very clear and is free of many of the hidden errors one might make otherwise.