CS 290G (Fall 2014) – Introduction to Cryptography Nov 13th, 2014

Lecture 9: PRFs, multi-message security, and block ciphers

Instructor: Rachel Lin *Scribe: John Retterer-Moore*

1 Previously on Cryptography

Last time, we looked at single-message security, being able to send a single message securely. We came up with a construction which was a computational version of a one-time-pad, where we used a PRG to come up with another m bit string to encrypt our message with. This works for single-message security, but it doesn't work if we want to send multiple messages. Since that method is deterministic and stateless, if we send the same message multiple times, it will always encrypt to the same thing, which poses a problem. Today we'll formally define multi-message secure encryption, formally define PRFs, and explain how PRFs will let us construct a multi-message secure encryption system.

2 Multi-message security

We formally define multi-message security like this:

$\forall$ polynomials $q(n)$, $\forall$ pairs of messages M_{0n}, M_{1n}, where M_{0n} and M_{1n} both consist of $q(n)$ m-bit messages, an encryption system is multi-message secure if

$$[k \leftarrow U_n \parallel \text{Enc}(k, M_{0n}[1])...\text{Enc}(k, M_{0n}[q(n)])] \approx [k \leftarrow U_n \parallel \text{Enc}(k, M_{1n}[1])...\text{Enc}(k, M_{1n}[q(n)])]$$

Essentially, this means that if we encrypt polynomially many messages using a randomly chosen key, the resulting distribution of cipher texts will be indistinguishable from the distribution resulting from encrypting any other choice of the same number of messages using a randomly chosen key. Our single-message-secure encryption system fails this test, since encrypting the same message twice will always give a distinct distribution from encrypting two different messages. So to achieve multi-message security, we'll need to introduce a new concept - the PRF.

3 Random functions

The basic goal of a PRF is to appear like a random function to computationally-bounded distinguishers. But to understand what that means, first we need to define what a random function is. There are two views of a random function:

1. We can think of any function $f : \{0,1\}^m \to \{0,1\}^l$, as a table that contains the l-bit output for every possible m-bit input (this will have size $2^m \cdot l$ since there are 2^m inputs and we store l bits for each one). Then a random function will just be defined by choosing $2^m \cdot l$ random bits to randomly choose a truth table, and using that truth table to compute the results for all inputs. This works, but is inefficient because we need to generate such a large table at the start.

2. A first attempt at a more efficient random function is to just pick l random bits and output them whenever we get any input to our function. However, the problem here is that this may not be a well-defined function: if we give the same input multiple times, there's a very high chance that we'll have a different output each time. So the solution is to maintain a table T that contains the output we've seen so far. Whenever we get an input, we check if there's an output for it in our table and use it if so; if not, we generate l random bits, output them, and store this input-output pair in our table.

4 PRFs

Now that we have a definition for a random function and some working implementations of it, we can define what being a PRF means.

A function $F : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^l$, where m is the length of the input, l is the length of the output, and k is the length of a key, is a **PRF** if it cannot be distinguished from a random function that outputs l bits on m bit inputs by a computationally bounded distinguisher.

How do we formally define 'distinguish'? We consider two experiments. In experiment 0, a key is randomly chosen, then we have a black box that given an input, uses the PRF with that key to give the output, and a distinguisher D makes polynomially many queries to that black box then outputs a bit D_0. In experiment 1, we have a black box that given an input, calls a random function on that input, and a distinguisher D makes polynomially many queries to that black box then outputs a bit D_1. We say that a function is a PRF if $\forall$ non-uniform PPT distinguishers D, $\exists$ negligible ϵ such that $Pr[D_0 = 1] - Pr[D_1 = 1] < \epsilon$.

5 Using PRFs for multi-message security

Now that we have a definition of PRF, let's see how it helps us achieve multi-message security. Given a PRF F, we define our encryption system like this:

Gen(1^n): $K \leftarrow \{0,1\}^k$
Enc(k, m): $c = (z, r)$, where $r \leftarrow U_m$, $z = m \oplus F(k, r)$
Dec(k, c): $c = (z, r)$, so $m = z \oplus F(k, r)$

Our encryption system works by taking a message and xor-ing the message with our PRF applied to a random input. We include the random input in our transmission to the receiver, so that they can also compute $F(k, r)$ to decrypt our message. If our PRF F is indistinguishable from a random function, xor-ing our message with F applied to a random input should behave like xor-ing our message with a random value, so we should behave like the earlier one-time-secure construction. However, we'll clearly avoid the problem with our earlier construction, since if we send the same message multiple times, we'll pick a different value for r and so we won't send the same cipher text.

6 Proof of Correctness

To prove the above system correct, we need to show that for any distinguisher D, for any polynomial $q(n)$, any two sets of $q(n)$ messages will lead to indistinguishable distributions on their cipher texts. So pick some $q(n)$ and take the following two experiments (we leave out the n from notation for the remainder since q is understood to mean $q(n)$).

Experiment 0:
1. $k \leftarrow \{0,1\}^k$
2. $r_1 \leftarrow U_m$, ... $r_q \leftarrow U_m$
3. $z_1 = M_0[1] \oplus F(k, r_1)$, ... $z_q = M_0[q] \oplus F(k, r_q)$

Experiment 5:
1. $k \leftarrow \{0,1\}^k$
2. $r_1 \leftarrow U_m$, ... $r_q \leftarrow U_m$
3. $z_1 = M_1[1] \oplus F(k, r_1)$, ... $z_q = M_1[q] \oplus F(k, r_q)$

Clearly these two experiments are just the process of encrypting M_0 and M_1, respectively, so if we can show that they lead to identical distributions, we are done. Now take these two experiments:

Experiment 1:
1. $k \leftarrow \{0,1\}^k$
2. $r_1 \leftarrow U_m$, ... $r_q \leftarrow U_m$
3. $z_1 = M_0[1] \oplus G(r_1)$, ... $z_q = M_0[q] \oplus G(r_q)$

Experiment 4:
1. $k \leftarrow \{0,1\}^k$
2. $r_1 \leftarrow U_m$, ... $r_q \leftarrow U_m$
3. $z_1 = M_1[1] \oplus G(r_1)$, ... $z_q = M_1[q] \oplus G(r_q)$

Here we replace $F(k,r)$ with a truly random function G applied to r. Because F is a pseudorandom function, replacing it with a random function can't be noticed by any poly-time distinguisher and so Experiments 0 and 1 must have the same distribution and Experiments 4 and 5 must have the same distribution (up to poly-time distinguishing). Finally, we connect the experiments:

Experiment 2:
1. $k \leftarrow \{0,1\}^k$
2. $r_1 \leftarrow U_m$, ... $r_q \leftarrow U_m$
3. $z_1 = M_0[1] \oplus r_1$, ... $z_q = M_0[q] \oplus r_q$

Experiment 3:
1. $k \leftarrow \{0,1\}^k$
2. $r_1 \leftarrow U_m$, ... $r_q \leftarrow U_m$
3. $z_1 = M_1[1] \oplus r_1$, ... $z_q = M_1[q] \oplus r_q$

In Experiment 1, we took random values $r_1...r_q$, applied a random function to them, and xor'ed those outputs $y_1...y_q$ with our messages. In Experiment 2, we instead directly sample $y_1...y_q$. If $r_1...r_q$ were disjoint, this is clearly the same. If $r_1...r_q$ had some repeats, this will differ, since applying a random function to two identical inputs gives an identical output, but sampling two random values most likely does not. However, the probability of a repeat is negligible, since the numerator of the probability grows polynomially with the size of q, a polynomial, while the denominator grows exponentially with the length of the message. Therefore, Experiments 1 and 2 have only a negligible chance of being distinguished, and similarly for Experiments 3 and 4.

But Experiments 2 and 3 both involve taking some message and xor-ing it with a totally

random value. This will produce the same distribution of cipher texts, since every cipher text has an equal chance of being produced. Thus, Experiments 2 and 3 are indistinguishable, and by transitivity, Experiments 0 and 5 are indistinguishable and our system is multi-message secure. Next time we'll see how to actually create a PRF, which will then show that multi-message secure encryption exists.

7 Block Ciphers

A block cipher is a function $E : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ such that E and E^{-1} are efficiently computable, and for each $K \in \{0,1\}^k$, $E_K : \{0,1\}^l \to \{0,1\}^l$ is a permutation. One basic example is $E_K(x) = K \oplus x$.

A common attack that block ciphers have to resist is a key recovery attack, where from some message/cipher pairs, the attacker tries to find the key used to encrypt the messages. In a chosen message attack, the attacker can choose messages and see their encryption, and can even choose adaptively which messages to see based on earlier results and tries to find the key. In a known message attack, the attacker doesn't choose the messages (for example in real life, you may only be able to intercept the messages some third party happens to send).

The most basic brute force approach is exhaustive key search, where the attacker searches through all possible keys to find one that fits the message/cipher pairs they have so far. This is prohibitively expensive, and also may not find the actual key being used but just one that is consistent with the pairs seen so far. However, it has some advantages: it is highly parallelizable, since every key can be examined in parallel, which can speed it up on machines with a high number of cores, and it only requires one message-cipher pair be leaked for it to work. Also, when the key space is smaller than the message space, it's likely that only 1 key will exist which is consistent with the message-cipher pairs, so it's unlikely to run into the problem of finding a consistent but incorrect key. Exhaustive key search is thus an inherent limit, requiring block ciphers to use a long enough key to make brute force impractical.

DES was a widely-used block cipher from the 1970s to the 1990s, though it was phased out and replaced with AES because of its small key size making it vulnerable to increasing computing power. Much like the Caesar cipher, all of the current existing standards for block ciphers are assumed secure solely because no good attack on them has been found yet, not because they have been proven valid in any rigorous way.