CSE 5852, Modern Cryptography: Foundations Fall 2016

Lecture 10, 2016

Prof. Benjamin Fuller *Scribe: Chaoqun Yue*

1 Last class

Last class: equivalence between semantic security and indistinguishable encryption. Over the next few classes we are going to build a single message private key encryption scheme (and then a multiple message private key encryption scheme). As a reminder, we're going to conjecturing that these things exist. Essentially all of cryptography relies on the assumption that $P \neq NP$. Furthermore, it actually rests on stronger assumptions, P vs NP is a question of whether there are search problems that are hard on average. Cryptography is general requires problems that are difficult to solve on average as we'll be sampling a random instance.

2 Indistinguishable encryptions: modified one time pad

Now that we've shown the definition of a computationally secure encryption scheme, let's return to the one-time pad and see if we can overcome some of its limitations by using computational security.

One time pad:

1) Generate k

2) $c = Enc(k, m) = m \oplus k$

3) $m = Dec(k, c) = c \oplus m$

Note that the OTP also satisfies semantic security and indistinguishable encryptions. What if we could construct an object that mimic'd the behavior of the one time pad without using as much randomness?

The main idea of the OTP was to completely mask the message using an unknown key. We know here that we don't have enough key material.

2.1 How to construct the key for modified OTP:

Consider some function $G : \{0,1\}^n \to \{0,1\}^m, m > n$. Let's consider a modified one-time pad that first passes the key through this expanding function G We can write the Enc and Dec as:

$Enc(k, m) = G(k) \oplus m = c$

$Dec(k, c) = G(k) \oplus c = m$

Note that at most $1/2$ of $\{0,1\}^n$ is in range of G. For G, what do we need for the scheme to satisfy indistinguishable encryptions? Recall that indistinguishable encryption is:

$\forall PPTA, M_1, M_2, |Pr[A(C_{m_1}) = 1] - Pr[A(C_{m_2}) = 1]| < \epsilon$

Suppose A is given $c = Enc(m_1)$, we know $\exists k$ such that $G(k) \oplus c = m_1$. If m_2 is random to Enc, it may happen that no k such that $G(k) \oplus c = m_2$. In fact this happens with probability roughly 1/2. Thus, if an attacker can check if there exists some k such that $G(k) \oplus m = c$ they can distinguish messages.

So at a minimum we are hoping that an attacker can't tell the difference between strings in the range of G and strings outside the range of G. If an attacker can't tell the difference between a random string and an output of G, its just as good. This notion is called pseudorandomness. We're trying to generate a string that is not completely random but "works" the same way as a random string. No attacker can tell which way one they are working with.

What properties might we want out of G?

1. Injective (or at least large range)
2. Can't tell for any $x \in \{0,1\}^m$, if there is an inverse
3. Can't tell anything if given something random from $G(k)$ or $\{0,1\}^m$

Items 2 and 3 are types of *pseudorandomness*

3 Bit Predictor, Blum Micali 82

Consider such scenario: Attacker $\mathcal{A}$ takes a bit y_i, he can say: 1) next(then get the next bit) or 2) output a guess 1 or 0. The goal for attacker is to guess y_{i+1}. Some notes:

1. No matter what strategy $\mathcal{A}$ uses it guesses correctly on truly random string half of the time.
2. If G is expanding there is some point in the sequence where the output of G is uniquely determined by the previous bits.

We'll now formally define this experiment:

Experiment: `prg − predict`

1. Create k
2. Create $y = G(k)$
3. Give $\mathcal{A}$ bit y_i while $\mathcal{A}$ says "next"

$\mathcal{A}$ wins if $\mathcal{A}$ outputs b and $b = y_{i+1}$

Definition 1. *A function G is next bit unpredictable if:* $\Pr[\mathcal{A}$ *wins PRG-predict*$] \leq 1/2 + \epsilon$*, for negligiable ϵ.*

And G is called pseudorandom number generator (PRG) if satisfies this definition.

In the last few classes we've mentioned that the attacker always receives the security parameter as a string 1^n. We're not only telling the attacker the security parameter but we are giving it a string with the same length. The reason for this is when we define polynomial time we mean a polynomial in the length of their inputs. So if we want to allow the attacker to run in time polynomial in the security parameter we need to provide an input with the same length as the security parameter.

We'll now turn to construct these objects, as a reminder it is impossible to construct these things information-theoretically. The existence of a pseudorandom generator implies that $P \neq NP$. This would be a major result in computer science. (Like the biggest result.)

3.1 Discrete logarithm Problem

Recall our two groups: $\mathbb{Z}_p = \{0, ..., p-1\}$ with addition and $\mathbb{Z}_p^* = \{0, ..., p-1\}$ with multiplication, both $\mod p$.

An element $g \in \mathbb{Z}_p^*$ is called a generator if $g \mod p, g^2 \mod p, ..., g^{p-1} \mod p$ are all unique and distinct, or formally: $f(x) = g^x \mod p$ is a bijection.

Theorem: For any p at least one generator exists

Theorem: It is possible to efficiently find p (of a particular length) and the corresponding g

Above we denoted the exponentiation operation as $f(x) = g^x$. Since this function is a bijection there is a well defined inverse f^{-1}. This function is known as the discrete logarithm problem when working $\mod p$.[1] We will write $f^{-1}(y) = \log_g y$ to denote the unique x, $1 \leq x \leq p-1$ such that $g^x \equiv y \mod p$.

We know how to find p, g efficiently, furthermore there is an efficient way to compute g^x (in the lengths of g, x). For example, we can use a square and multiply algorithm whose time is proportional in the length of p and x. People have been trying to compute f^{-1} for a long time.

Assumption 2. *[DH76] Given random $p, g, g^x \mod p$, for any PPT$\mathcal{A}$, $Pr[A(p, g, g^x \mod p) = x] < \epsilon$ for some negligible $\epsilon(n)$ or*

$$\Pr[\mathcal{A}(1^n, p, g, g^x \mod p) = x] \leq \epsilon(n).$$

When this problem is used as a cryptographic assumption in practice primes of $|p| = 2048$ or 4096 bits are used. Also there exists a polynomial time algorithm to solve this problem on a quantum computer [Sho94]. So if quantum computers are practical this problem should not be basis of cryptography.

3.2 Use Discrete logarithm to construct PRG

How can we build a pseudorandom generator based on this assumption? We need a random prime p, generator g of $\mathbb{Z}_p^*$ and $x \in \mathbb{Z}_p^*$. Create $p, g, y_1 = g^x \mod p, y_2 = g^{y1} \mod p, ..., y_m = g^{y_{m-1}}$

[1]We can also define the discrete logarithm problem over other groups as we'll see soon.

mod p. Note that in this order it is possible to predict the next group element from the previous element. However, it should be difficult to go backwards in the sequence. So we'll try outputting the elements in reverse order and having $G(x) = y_m, ..., y_1 = g^{y_{m-1}}, g^{y_{m-2}}, ..., g^x$. Because to compute next element from previous need to compute Discrete log.

However, the Discrete logarithm problem doesn't say that each bit of y is hard to predict, only that each element is hard to predict. So we need to generate the unpredictable bits based on G. The original proposal for how to do this was to use whether the exponent was greater than $p/2$:

$$B(x) = \begin{cases} 0 & x < p/2 \\ 1 & \text{otherwise} \end{cases}$$

Lemma 3. *If discrete log is hard then for any PPT$\mathcal{A}$, $Pr[A(p, g, g^x \mod p) = B(x)] \leq 1/2 + \epsilon$ for some negligible $\epsilon(n)$.*

We'll show this next time. Furthermore this fix is sufficient to show that security of the PRG. This lemma is sufficient to build a pseudorandom generator.

Theorem 4. *The function G, $B(x_n), B(x_{n-1}), ..., B(x_1)$ is a pseudorandom generator under Definition 1 assuming the discrete log problem is hard.*

We'll show this theorem in class as well.

References

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[Sho94] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.