CS 6222 Grad Crypto September 11, 2025

Lecture 6: Pseudorandom Functions

Lecturer: Jack Doerner *Scribe: Deebakkarthi Chinnasame Rani*

1 Topics Covered

- Random Functions
- Oracle Indistinguishability
- Pseudorandom Functions

2 Review and Warm-up

We have previously shown that one-time pad achieves (and is in fact *optimal* for) a perfect notion of security. However, it is very cumbersome to use in the real world. This is primarily due to the following reasons:

- Keys as long as the message
- Inability to reuse keys

In a previous class, we introduced a weaker, *computational* notion of security for encryption, and a special kind of function called a pseudorandom generator (PRG) that allows us to achieve our relaxed security notion while having short encryption keys. Recall:

Definition 1 (EAV1 Security)**.** *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *for the message space* $\mathcal{M}$ *is* secure against single-ciphertext eavesdropping *if* $\forall m_0, m_1 \in \mathcal{M}$,

$$\{\mathsf{Enc}_k(m_0) : k \leftarrow \mathsf{Gen}(1^n)\}_{n\in\mathbb{N}} \approx_c \{\mathsf{Enc}_k(m_1) : k \leftarrow \mathsf{Gen}(1^n)\}_{n\in\mathbb{N}}$$

Construction 1 ("Computational OTP" for $\ell(n)$-bit messages)**.** *For some polynomial function* ℓ*, let* $\mathcal{M} = \mathcal{C} = \{0,1\}^{\ell(n)}$*, let* $\mathcal{K} = \{0,1\}^n$*, and let* $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ *be a pseudorandom generator. The scheme is defined as follows:*

$$\mathsf{Gen} : 1^n \mapsto k : k \leftarrow \mathcal{K}$$
$$\mathsf{Enc} : k, m \mapsto m \oplus G(k)$$
$$\mathsf{Dec} : k, c \mapsto c \oplus G(k)$$

Theorem 1. *If* G *is a PRG then Construction 1 is EAV1-secure.*

The above construction works for a single message, but in the real world, we need to be able to handle multiple messages. We could imagine using G to stretch k enough that it can be used to padd *all* the messages we want to send, but if we're not sending them at the same time, then this solution requires keeping state, which is not desirable. In this lecture we will introduce another primitive which will allow us to construct multi-message *randomized* encryption without keeping state.

Suppose that could build a magical PRG with *exponential* output.[1] Specifically, imagine

$$G : \{0,1\}^n \to \{0,1\}^{\ell(n)\cdot 2^n}$$

was a PRG, and then suppose that we had an auxiliary function F that could efficiently compute any block of $\ell(n)$ bits from the output of G. For any $s \in \{0,1\}^n$,

$$F_s : x \mapsto y_{x\cdot\ell(n)} \| \ldots \| y_{(x+1)\cdot\ell(n)-1} \qquad \text{where} \qquad y_0 \| \ldots \| y_{2^n\cdot\ell(n)-1} = G(s)$$

Using the above functions, we could construct the following *randomized* encryption scheme that handles many messages without keeping state:

Construction 2 (Encryption for many $\ell(n)$-bit messages)**.** *Let $\mathcal{M} = \{0,1\}^{\ell(n)}$, $\mathcal{C} = \{0,1\}^{l(n)+n}$, and $\mathcal{K} = \{0,1\}^n$. The encryption scheme is:*

$$\begin{aligned}
&\mathsf{Gen} : 1^n \mapsto k : k \leftarrow \mathcal{K} \\
&\mathsf{Enc} : k, m \mapsto r || (m \oplus F_k(r)) : r \leftarrow \{0,1\}^n \\
&\mathsf{Dec} : k, c \mapsto c' \oplus F_k(r) : r || c' = c
\end{aligned}$$

In other words, we *randomly* choose a block of pseudorandom bits (indexed by r) from the exponentially-long output of $G(k)$, use F_k to compute that block efficiently, and then use that block of bits to mask the message m. In order to decrypt, we need both k and the index r of the block of pseudorandom bits we used during encryption. Transmitting r in the clear is perfectly fine, as long as k remains secret.

Though the above direction seems useful, the function $G : \{0,1\}^n \to \{0,1\}^{\ell(n)\cdot 2^n}$ *cannot* be a PRG. The output of G is exponentially long, and providing it to a NUPPT adversary grants that adversary exponential running time, which the adversary can use to distinguish the output from a uniform string! Nevertheless, we can hope to define and build functions that work like F (which has only polynomially-much output). We call these *pseudorandom functions (PRFs)*. In order to define them *without* passing through the nonsensical G, we must first introduce the concept of *random* functions .

3 Random Functions

Consider a function $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ from n bit strings to $\ell(n)$ bit strings, for some polynomial ℓ. We can represent this function as a table.

[1] Further down the page, we will discover that this is a nonsensical idea.

x	$y = f(x)$
0	$f(0)$
1	$f(1)$
$\vdots$	$\vdots$
$2^n - 1$	$f(2^n - 1)$

The number of elements in this table is 2^n. Each entry in the $y = f(x)$ column has a length of $\ell(n)$ bits. Therefore, the size of this table is $\ell(n) \cdot 2^n$ bits. We don't need to store the values of x since they are implicit in the ordering of the elements.

Example 1. *Let us set $n = 3$ and $\ell(n) = n$ for simplicity. The three-bit increment function $f : x \mapsto x + 1 \bmod 8$ is represented by the following table:*

x	$y = f(x)$
000	001
001	010
010	011
011	100
100	101
101	110
110	111
111	000

We can represent this function as a bitstring of length $\ell(n) \cdot 2^n = 3 \cdot 2^3 = 24$. Thus, $f = 001010011100101110111000$. This is powerful, as we can represent any function as a bitstring of its outputs concatenated together.

Let $\mathcal{F}_{n,\ell(n)}$ be the set of *all* functions from $\{0,1\}^n \to \{0,1\}^{\ell(n)}$. The cardinality of $\mathcal{F}_{n,\ell(n)}$ is the same as the number of possible tables, i.e. $2^{\ell(n) \cdot 2^n}$.

Definition 2. *If we sample f from the above-mentioned set $\mathcal{F}_{n,\ell(n)}$, then f is called a* random function.[2]

To help us understand how a random function behaves, we can construct a stateful, randomized algorithm[3] that behaves identically:

[2]This should not be confused with a *randomized algorithm* that takes a sequence of random coins as auxiliary input. We might sometimes informally refer to randomized algorithms as a "random functions", but in cryptography, formally speaking, a random function is a (deterministic) function selected randomly from some set or distribution. It doesn't make much sense to describe a single function as "random" without reference to the sampling process, just it doesn't make much sense to describe a single number as not random without reference to the process by which it was selected.

[3]Unlike the random functions described previously, this algorithm doesn't require exponential space because it is lazily defined, but the statefulness of this algorithm means that it cannot be shared by multiple participants beforehand.

Construction 3.

on input $x \in \{0,1\}^n$:
  if the record (x, y) exists in memory for any y :
    output y
  else :
    $y \leftarrow \{0,1\}^{\ell(n)}$
    store (x, y) in memory
    output y

Because of the description of a random function is exponentially large, there is no practical way to use one that accepts a polynomial number of bits as input.[4] We seek to define a primitive that has behavior *similar* to that of a random function, and also has a polynomial-size description. In some sense, what we want is analogous to a PRG, which allows us to represent long "random looking" (i.e. pseudorandom) strings compactly.

However, the framework that we used to express the idea that pseudorandom strings are indistinguishable from random strings will not work directly for pseudorandom functions and random functions. In that framework, we provided the adversary with complete descriptions of values sampled from the pseudorandom or random ensembles. It is trivial to distinguish the description of a pseudorandom function (which we insist should be at most polynomially large) from the description of a truly random one (which has size $2^{\ell(n)\cdot 2^n}$).

Instead of giving a distinguisher D the descriptions of the functions, we must make D distinguish the functions *in use*. In layman's terms, suppose we have two black boxes. One contains a random function, and the other contains a pseudorandom function. D should not be able to identify which is which. We formalize this idea using *Oracle Turing machines*.

4 Oracle Indistinguishability

Definition 3 (Oracles, Oracle Machines, Oracle Queries)**.** *A NUPPT* oracle machine M *has* oracle access *to f if M is augmented with a special tape on which it can write* oracle queries *and receive answers. When M writes on the special tape, the oracle is activated, and the Turing machine pauses while the oracle computes. When the oracle writes an answer on the special tape, M resumes. This process is assumed to be instantaneous.*

Note 1 (Oracles and Runtime)**.** *For the purposes of this course, we will assume that oracles only output strings of length at most polynomial in the length of the queries that produce them, and that querying an oracle doesn't add to the runtime of M.*

Note 2 (Notation for Oracles)**.** *M^f denotes a Turing machine M with oracle access to the function f. If f accepts multiple arguments, we use $(\cdot)$ to denote the arguments that M supplies in its queries; any arguments with fixed values are unknown to M except insofar as the outputs of f reveal them.*

[4] Random functions with sufficiently small input spaces are an exception.

Example 2. *Consider a three argument function $f(a, b, c)$, and suppose we wish to fix the values of a and b, but leave the value of c free so that* only *the value of c can be supplied by M. We denote this by $M^{f(a,b,\cdot)}$. M does not learn a or b except insofar as they are revealed by the output of f.*

Definition 4 (Oracle Indistinguishability)**.** *Let $\{O_n\}_{n\in\mathbb{N}}$ and $\{O'_n\}_{n\in\mathbb{N}}$ be ensembles of distributions over functions of the form $f : \{0,1\}^{\ell_1(n)} \to \{0,1\}^{\ell_2(n)}$, for two polynomials ℓ_1, ℓ_2. We say that $\{O_n\}_{n\in\mathbb{N}}$ and $\{O'_n\}_{n\in\mathbb{N}}$ are* computationally oracle-indistinguishable *if $\forall$ NUPPT oracle machines D there exists some negligible function ε such that $\forall n \in \mathbb{N}$*

$$|\Pr[D^{f(\cdot)}(1^n) = 1 : f \leftarrow O_n] - \Pr[D^{f(\cdot)}(1^n) = 1 : f \leftarrow O'_n]| < \varepsilon(n)$$

Note 3 (Notation for Oracle Indistinguishability)**.** *We will sometimes indicate that two ensembles of distributions of functions are computationally oracle-indistinguishable by writing $\{O_n\}_{n\in\mathbb{N}} \approx_c \{O'_n\}_{n\in\mathbb{N}}$. It should be clear from context that oracle indistinguishability is meant, rather than computational indistinguishability of function descriptions.*

5 Pseudorandom Functions

Definition 5 (Pseudorandom Function)**.** *For some polynomial ℓ let $\mathcal{F}_{n,\ell(n)}$ be the set of all functions $f : \{0,1\}^n \to \{0,1\}^{\ell(n)}$, and let $\{F_k : \{0,1\}^{|k|} \to \{0,1\}^{\ell(|k|)}\}_{k\in\{0,1\}^*}$ be a family of functions. $\{F_k\}_{k\in\{0,1\}^*}$ is* pseudorandom function (PRF) family *if*

1. *There exists some PPT algorithm F that computes $F_k(x)$ given k and x*
2. *$\{F_k : k \leftarrow \{0,1\}^n\}_{n\in\mathbb{N}} \approx_c \{f : f \leftarrow \mathcal{F}_{n,\ell(n)}\}_{n\in\mathbb{N}}$*

In other words no NUPPT algorithm should be able to distinguish between a function sampled from the family F_k and a random function.

Note 4 (Family Size, Description Size)**.** *Notice that $|\{F_k\}_{k\in\{0,1\}^n}| = 2^n$, whereas $|\mathcal{F}_{n,\ell(n)}| = 2^{l(n)\cdot 2^n}$. We can complactly describe F_k using F and the n-bit key k.*

Definition 6 (Pseudorandom Permutation)**.** *For some polynomial ℓ, a function family $\{F_k : \{0,1\}^{|k|} \to \{0,1\}^{\ell(|k|)}\}_{k\in\{0,1\}^*}$ is a* pseudorandom permutation (PRP) *family if*

1. *It is a pseudorandom function family*
2. *It is a permutation; i.e $\forall k \in \{0,1\}^n$, F_k is bijective and F_k^{-1} can be computed in polynomial time.*

Note 5 (Permutations and Functions)**.** *Asymptotically, it is not possible to distinguish between a random permutation and a random function [BR04, Lemma 1].*

The proofs of the following theorems are left as exercises or for future lectures.

Theorem 2. *$\exists$ PRF $\Rightarrow$ $\exists$ PRG*

Suppose F is a PRF. Our PRG is $G : s \mapsto F_s(0)||F_s(1)||F_s(2)||\ldots$

Theorem 3. *If there exists a PRF family* $\{F_k : \{0,1\}^{|k|} \to \{0,1\}^{|k|}\}_{k\in\{0,1\}^*}$ *then for any polynomial* ℓ *there exists a PRF family* $\{F'_k : \{0,1\}^{|k|} \to \{0,1\}^{\ell(|k|)}\}_{k\in\{0,1\}^*}$.

By Theorem 2 we can use F to construct a PRG $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$. Our polynomial-stretch PRF is $F'_k : x \mapsto G(F_k(x))$

Theorem 4. $\exists$ *PRG* $\Rightarrow \exists$ *PRF*

This famous theorem was proved by Goldreich, Goldwasser, and Micali [GGM84].

References

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Paper 2004/331, 2004.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 276–288, New York, NY, USA, 1984. Springer-Verlag New York, Inc.