Purdue CS555: Cryptography
Lecture 4 Scribe Notes

Instructor: Hanshen Xiao
Teaching Assistant: Justin He

Fall 2025

Recap from Previous Lecture

- Pseudorandom Generator (PRG)
- One bit to (poly) many bits expansion
- Equivalence to Next-bit Unpredictability (NBU)
- Hybrid Argument

Topics Covered in This Lecture

1. Preliminaries of Computational Complexity
2. One-way Functions
3. Construction of One-way Function and PRG
4. Average and Worst Case Hardness
5. Applications of PRG

1 Preliminaries of Computational Complexity

Computational complexity is the study of resources required to solve computational problems, focusing on time and space requirements, problem classification based on inherent difficulty, and distinguishing tractable from intractable problems.

1.1 Models of Computation

1.1.1 Turing Machine

The canonical formal model of algorithms consists of:

- An infinite tape divided into cells (memory)
- A head that reads/writes one symbol at a time and moves left or right
- A transition function on finite states Q and tape alphabets Γ:

  $$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

  The function takes the current state and tape symbol, and outputs the next state, the symbol to write, and the direction to move the tape head.
- Computation proceeds step by step until reaching an accept or reject state (halting)

1.1.2 Random-Access Machine (RAM) Model

A small-space Turing machine with fixed tape size n and the additional ability to move the pointer to any position of the work tape in 1 time step. This model is equivalent to Turing machines under polynomial overhead.

1.1.3 Boolean Circuits

A Boolean circuit is a directed acyclic graph of logic gates (AND, OR, NOT) computing a Boolean function on inputs. Complexity is measured by:

- Number of gates in the circuit
- Depth: the length of the longest path (parallel time)

Boolean circuits represent a non-uniform model. A polynomial-time Turing machine corresponds to a family of polynomial-size circuits.

1.2 Complexity Classes

1.2.1 Problem Types

- **Decision Problem:** Is a given number q prime? Does a graph have a Hamiltonian cycle? Hamiltonian cycle is a path that touches all points only once.
- **Search Problem:** Find a satisfying assignment to a Boolean formula
- **Optimization Problem:** Find the shortest path in a graph
- **Counting Problem:** Count the number of perfect matchings in a bipartite graph

1.2.2 Complexity Classes for Decision Problems

Definition 1 (**P**). *The set of languages L such that there exists a Turing machine M that runs in polynomial time such that $x \in L \iff M(x) = 1$.*
Example: $L = \{(G = (V, E), v_1 \in V, v_2 \in V) : \text{there is a path from } v_1 \text{ to } v_2 \text{ in } G\}$

Definition 2 (**NP**). *There exists a Turing machine V (verifier) and a polynomial function $p(n)$ such that V runs in time at most $p(|x|)$ for inputs x, and*

$$x \in L \iff \exists y \in \{0,1\}^{poly(|x|)} \text{ such that } V(x, y) = 1$$

Here y is called a witness/solution, and it is easy to verify whether a proposal is satisfied.
Example: $L = SAT = \{\phi : \exists y_1, \ldots, y_n \in \{0,1\}, \phi(y_1, \ldots, y_n) = 1\}$

Definition 3 (**BPP** (Bounded-error Probabilistic Polynomial time)). *The set of languages L such that there exists a randomized Turing machine M that runs in polynomial time such that:*

- $x \in L \Rightarrow \Pr[M(x; r) = 1] \geq 2/3$
- $x \notin L \Rightarrow \Pr[M(x; r) = 1] \leq 1/3$

where $|r| \leq p(|x|)$ for some polynomial function $p(n)$.
Examples: Primality testing, polynomial identity testing

1.3 Reduction

Definition 4 (Karp Reduction). *A reduction from language L_1 to language L_2 (denoted $L_1 \leq L_2$) means that L_2 is at least as hard as L_1. Formally, $x \in L_1 \iff R(x) \in L_2$ for some polynomial-time computable function $R(\cdot)$.*

If we have an algorithm A for L_2, we can solve L_1 by computing $R(x)$ and then running $A(R(x))$, achieving similar efficiency within polynomial overhead.

1.4 Completeness

Theorem 1 (Cook-Levin)**.** *SAT is NP-complete. That is, SAT $\in$ NP, and for all languages $L \in$ NP, there is a Karp reduction from L to SAT.*

A language H is **NP-hard** if $\forall L \in \text{NP} : L \leq H$ (note that H itself may not be in NP).

Theorem 2. *#SAT is #P-complete, where #SAT counts the number of satisfying assignments for a SAT instance.*

2 One-way Functions

2.1 Intuition

A one-way function is easy to compute but hard to invert. Given $y = F(x)$, it should be computationally infeasible to find any x' such that $F(x') = y$.

2.2 First Attempt (Flawed)

Definition 5 (One-way Function - Incorrect)**.** *A function family $\{F_n\}_{n\in\mathbb{N}}$ where $F_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ is one-way if for every p.p.t. adversary A, there is a negligible function μ such that:*

$$\Pr[x \leftarrow \{0,1\}^n; y = F_n(x) : A(1^n, y) = x] \leq \mu(n)$$

Problem: Consider $F_n(x) = 0$ for all x. This satisfies the definition (impossible to find the specific inverse even with unbounded time), but is not meaningful.

2.3 Correct Definition

Definition 6 (One-way Function)**.** *A function family $\{F_n\}_{n\in\mathbb{N}}$ where $F_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ is one-way if for every p.p.t. adversary A, there is a negligible function μ such that:*

$$\Pr[x \leftarrow \{0,1\}^n; y = F_n(x); x' \leftarrow A(1^n, y) : y = F_n(x')] \leq \mu(n)$$

The key difference is that the adversary must find *any* valid pre-image x', not necessarily the original x.

Definition 7 (One-way Permutation)**.** *A one-to-one one-way function with $m(n) = n$.*

2.4 Hardcore Predicates

If F is one-way, it's hard to compute a complete pre-image. But what about partial information?

Example 1. There exist one-way functions for which it is easy to compute the first half of the bits of an inverse. Consider $F'(x\|y) = x\|F(y)$ for a one-way function $F(\cdot)$.

2.4.1 First Attempt at Hardcore Bit

Definition 8 (Hardcore Bit - Initial Try)**.** *For a function family $F : \{0,1\}^n \to \{0,1\}^m$, a bit $i = i(n)$ is hardcore if for every p.p.t. adversary A, there is a negligible function μ such that:*

$$\Pr[x \leftarrow \{0,1\}^n; y = F(x) : A(y) = x_i] \leq \frac{1}{2} + \mu(n)$$

Problem: There exist one-way functions where every individual bit position is somewhat easy to predict (e.g., with probability $\frac{1}{2} + \frac{1}{n}$).

2.4.2 Correct Definition

Definition 9 (Hardcore Predicate)**.** *For a function family $F : \{0,1\}^n \to \{0,1\}^m$, a function $B : \{0,1\}^n \to \{0,1\}$ is a hardcore predicate if for every p.p.t. adversary A, there is a negligible function μ such that:*

$$\Pr[x \leftarrow \{0,1\}^n; y = F(x) : A(y) = B(x)] \leq \frac{1}{2} + \mu(n)$$

Key observations:

- The definition makes sense for any function family, not just one-way functions
- Some functions can have information-theoretically hard-to-guess predicates (e.g., compressing functions)
- We're interested in settings where x is uniquely determined given $F(x)$, yet $B(x)$ is hard to predict given $F(x)$

2.5 Relationship to P vs NP

Theorem 3. *If one-way functions exist, then $P \neq NP$.*

Proof Sketch. Assume f is one-way. Consider the language L of pairs (x^*, y) such that x^* is a prefix of some x satisfying $f(x) = y$.

Clearly $L \in$ NP (given a witness x, we can verify in polynomial time).

If $P = NP$, we could use a polynomial-time decider D for L to invert y by finding a pre-image bit by bit, contradicting the one-wayness of f. □

3 Construction of PRG from One-way Permutations

Theorem 4. *Let F be a one-way permutation and B an associated hardcore predicate for F. Define:*

$$G(x) = F(x) \| B(x)$$

Then G is a PRG (stretching input by one bit).

Note: We already know how to extend a 1-bit stretch PRG to a PRG that stretches to any polynomial number of bits.

Proof. We use the next-bit unpredictability characterization of PRGs.

Assume for contradiction that G is not a PRG. Then there exists a next-bit predictor D and a polynomial function p such that:

$$\Pr[x \leftarrow \{0,1\}^n; y = G(x) : D(y_{1\ldots n}) = y_{n+1}] \geq \frac{1}{2} + \frac{1}{p(n)}$$

Since $G(x) = F(x) \| B(x)$, this means:

$$\Pr[x \leftarrow \{0,1\}^n : D(F(x)) = B(x)] \geq \frac{1}{2} + \frac{1}{p(n)}$$

Therefore, D is a predictor for the hardcore predicate B, contradicting the assumption that B is hardcore for F. □

4 Average-Case vs Worst-Case Hardness

4.1 Definitions

Average-case hardness: For randomly chosen instances (with entropy), the problem is hard. One-way functions capture this notion.

Worst-case hardness: The meaning depends on context:

- *IND-CPA security:* The adversary can arbitrarily select the challenge; worst-case indistinguishability holds for any input candidate pair
- *Lattice problems:* There exists some instance that is hard to solve

4.2 Lattice Problems

Definition 10 (Lattice)**.** *A lattice $L \subseteq \mathbb{R}^d$ is the set of all integer linear combinations of some basis (linearly independent) vectors $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathbb{R}^d$:*

$$L = \left\{ \sum_{i=1}^{m} z_i \mathbf{b}_i : z_i \in \mathbb{Z} \right\}$$

Important lattice problems:

- **Shortest Vector Problem (SVP):** Find the shortest nonzero vector in L
- **Closest Vector Problem (CVP):** Given a target vector t, find the lattice vector closest to t
- **GapSVP:** Given a number p, decide whether the length of the shortest nonzero lattice vector is $\leq p$ or $\geq \gamma(d) \cdot p$

4.3 Examples of Average-Case to Worst-Case Reductions

- **Short Integer Solution (SIS):** Finding a short, non-zero vector such that $Ax = 0 \pmod{q}$ is hard for randomly-selected matrix A. This problem has worst-case to average-case reductions from lattice problems.
- **Learning with Errors (LWE):** Given a random matrix A and a vector $b = As + e \pmod{q}$ for some secret s and small random noise e, it is hard to recover the secret s. This also admits worst-case to average-case reductions.

4.4 Another PRG Construction

PRG candidate from Subset-Sum:

$$G(a_1, \ldots, a_n, x_1, \ldots, x_n) = \left(a_1, \ldots, a_n, \sum_{i=1}^{n} x_i a_i \bmod 2^{n+1} \right)$$

where a_i are random $(n+1)$-bit numbers and x_i are random bits.

If lattice problems are hard in the worst case, then G is a PRG.

5 Applications of PRG: Derandomization

Theorem 5. *If PRGs exist, then BPP $\subseteq \bigcap_{\epsilon>0}$ TIME(2^{m^ϵ}).*

In other words, every randomized polynomial-time algorithm can be simulated in deterministic sub-exponential time.

Proof Sketch. Use a PRG that expands from $n = m^\epsilon$ bits to m bits.

Consider a BPP language L with randomized algorithm M using m random bits. By definition:

$$x \in L \Rightarrow \Pr_{r \leftarrow \{0,1\}^m}[M(x, r) \text{ accepts}] \geq \frac{2}{3}$$

$$x \notin L \Rightarrow \Pr_{r \leftarrow \{0,1\}^m}[M(x, r) \text{ accepts}] \leq \frac{1}{3}$$

Using the PRG G that expands seeds of length $n = m^\epsilon$ to length m:

$$x \in L \Rightarrow \Pr_{y \leftarrow \{0,1\}^n}[M(x, G(y)) \text{ accepts}] > \frac{2}{3} - \mu(n)$$

$$x \notin L \Rightarrow \Pr_{y \leftarrow \{0,1\}^n}[M(x, G(y)) \text{ accepts}] < \frac{1}{3} + \mu(n)$$

Why? If this weren't true, M would be a distinguisher for the PRG!

Deterministic algorithm: Enumerate over all $2^n = 2^{m^\epsilon}$ seeds y and run $M(x, G(y))$. If the number of accepting runs is $> 0.65 \cdot 2^n$, accept; otherwise reject.

This gives:

$$x \in L \Rightarrow \#\{y : M(x, G(y)) \text{ accepts}\} > 0.65 \cdot 2^{m^\epsilon}$$
$$x \notin L \Rightarrow \#\{y : M(x, G(y)) \text{ accepts}\} < 0.35 \cdot 2^{m^\epsilon}$$

The algorithm runs in time $2^{m^\epsilon} \cdot \text{poly}(m) = 2^{m^\epsilon}$ for any $\epsilon > 0$. □

Theorem 6. *If "exponentially secure" PRGs exist, then BPP = P.*

Proof Sketch. Use a PRG that expands from $n = O(\log m)$ bits to m bits that is indistinguishable not just by $\text{poly}(n)$-time algorithms but also by $2^{c_1 n} = m^{c_2}$-time algorithms (for some $c_1 < 1$).

The previous proof goes through with the same structure, but now the enumeration over all seeds takes only $2^{O(\log m)} = \text{poly}(m)$ time, yielding a polynomial-time deterministic algorithm. □