Purdue CS555: Cryptography
Lecture 6 Scribe Notes

Instructor: Hanshen Xiao
Teaching Assistant: Justin He

Fall 2025

Recap from Previous Lecture

- Implications of One-way Functions
- Construction of PRG from One-way Permutations
- Introduction to Goldreich-Levin (GL) Theorem

Topics Covered in This Lecture

1. Continuing: Proof of Goldreich-Levin (GL) Theorem
2. Stateless Encryption
3. Pseudorandom Functions (PRFs)

1 Completion of Goldreich-Levin Theorem

1.1 Quick Review of GL Theorem

Theorem 1 (Goldreich-Levin, Complete Statement)**.** *Let* $\{B_r : \{0,1\}^n \to \{0,1\}\}$ *where*

$$B_r(x) = \langle r, x \rangle = \sum_{i=1}^{n} r_i x_i \bmod 2$$

be a collection of predicates (one for each r*). Then, a random* B_r *is hardcore for every one-way function* F*. That is, for every one-way function* F*, every p.p.t. adversary* A*, there exists a negligible function* μ *such that:*

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

1.2 Alternative Interpretation

For every one-way function/permutation F, there is a related one-way function/permutation

$$F'(x, r) = (F(x), r)$$

which has a deterministic hardcore predicate: $B(x, r) = \langle r, x \rangle \bmod 2$.

This statement is sufficient to construct PRGs from any one-way permutation.

1.3 Proof Summary

The proof proceeds through several stages of refinement:

1.3.1 Stage 1: Perfect Predictor (Warmup)

Assumption: Perfect predictor P with

$$\Pr[x \leftarrow \{0,1\}^n; r \leftarrow \{0,1\}^n : P(F(x), r) = \langle r, x \rangle] = 1$$

Inverter: On input $y = F(x)$, run P on $(y, e_1), (y, e_2), \ldots, (y, e_n)$ where e_i are unit vectors. Since P is perfect, $P(y, e_i) = \langle e_i, x \rangle = x_i$.

1.3.2 Stage 2: Pretty Good Predictor

Assumption: P satisfies

$$\Pr[x \leftarrow \{0,1\}^n; r \leftarrow \{0,1\}^n : P(F(x), r) = \langle r, x \rangle] \geq \frac{3}{4} + \frac{1}{p(n)}$$

Claim 1 (Averaging Argument)**.** *For at least a $\frac{1}{2p(n)}$ fraction of x (call these "good x"):*

$$\Pr[r \leftarrow \{0,1\}^n : P(F(x), r) = \langle r, x \rangle] \geq \frac{3}{4} + \frac{1}{2p(n)}$$

Key Idea - Linearity: Pick random r, query P for $\langle r, x \rangle$ and $\langle r + e_i, x \rangle$. XOR the answers:

$$\langle r + e_i, x \rangle \oplus \langle r, x \rangle = \langle e_i, x \rangle = x_i$$

Analysis:

$$\begin{aligned}
\Pr[\text{compute } x_i \text{ correctly}] &\geq \Pr[P \text{ correct on both } r \text{ and } r + e_i] \\
&= 1 - \Pr[P \text{ wrong on } r \text{ or } r + e_i] \\
&\geq 1 - 2\left(\frac{1}{4} - \frac{1}{2p(n)}\right) = \frac{1}{2} + \frac{1}{p(n)}
\end{aligned}$$

Inverter: Repeat $\log n \cdot p(n)$ times, take majority vote for each bit. Analysis uses Chernoff + union bound.

1.3.3 Stage 3: Weak Predictor (The Real Challenge)

Assumption: P satisfies only

$$\Pr[x \leftarrow \{0,1\}^n; r \leftarrow \{0,1\}^n : P(F(x), r) = \langle r, x \rangle] \geq \frac{1}{2} + \frac{1}{2p(n)}$$

After averaging, for $\geq \frac{1}{2p(n)}$ fraction of x:

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

Problem: The union bound analysis from Stage 2 fails because we cannot guarantee both $\langle r, x \rangle$ and $\langle r + e_i, x \rangle$ are predicted correctly with high enough probability.

1.4 The Rackoff Trick

Key Insight (attributed to Charlie Rackoff): If we had an oracle that gave us $\langle r, x \rangle$ for free, then:

- Pick random r, get $\langle r, x \rangle$ from oracle
- Query P for $\langle r + e_i, x \rangle$
- XOR to get x_i

Success probability: $\Pr[\text{correct}] \geq \frac{1}{2} + \frac{1}{2p(n)}$

Solution: Since we don't have an oracle, we *guess* the values.

1.5 Parsimony in Guessing

Instead of guessing $\langle r, x\rangle$ for every r, we use a clever construction.

Parameters: Let $m = O(n \log n \cdot (p(n))^2)$.

Strategy:

1. Pick random seed vectors $s_1, \dots, s_{\log(m+1)}$
2. Guess $c_j = \langle s_j, x\rangle$ for all $j \in \{1, \dots, \log(m+1)\}$
3. Probability all guesses correct: $\frac{1}{2^{\log(m+1)}} = \frac{1}{m+1}$

Key Construction: From seed vectors, generate many r_i values.

Let $T_1, \dots, T_m$ denote all non-empty subsets of $\{1, 2, \dots, \log(m+1)\}$. Define:

$$r_i = \bigoplus_{j \in T_i} s_j \quad \text{and} \quad b_i = \bigoplus_{j \in T_i} c_j$$

Lemma 1 (Key Observation)**.** *If guesses $c_1, \dots, c_{\log(m+1)}$ are all correct, then so are $b_1, \dots, b_m$, because:*

$$b_i = \bigoplus_{j \in T_i} c_j = \bigoplus_{j \in T_i} \langle s_j, x\rangle = \left\langle \bigoplus_{j \in T_i} s_j, x \right\rangle = \langle r_i, x\rangle$$

1.6 Complete Inverter Algorithm

Algorithm 1 OWF Inverter for GL Theorem

Input: $y = F(x)$ for unknown x
Generate random seed vectors $s_1, \dots, s_{\log(m+1)}$ where $m = O(n \log n \cdot (p(n))^2)$
Generate random bits $c_1, \dots, c_{\log(m+1)}$ (guesses for $\langle s_j, x\rangle$)
Derive $r_1, \dots, r_m$ and $b_1, \dots, b_m$: $r_i = \bigoplus_{j \in T_i} s_j$, $b_i = \bigoplus_{j \in T_i} c_j$
for $i = 1$ to n **do**
    Repeat $100n(p(n))^2$ times:
        Pick random index $\ell \in \{1, \dots, m\}$
        Query $P(F(x), r_\ell + e_i)$ to get answer a
        Compute guess: $g = a \oplus b_\ell$
    Compute majority of all guesses to determine x_i
end for
Output: $x = x_1 x_2 \cdots x_n$

1.7 Analysis

Condition on: Guesses $c_1, \dots, c_{\log(m+1)}$ all correct, and x is good.

Key Issue: The r_i are *not* independent, so we cannot use Chernoff bound.

Critical Observation: The r_i are *pairwise independent*.

Therefore, we can apply Chebyshev's inequality.

1.7.1 Single Bit Analysis

For a fixed bit position i, let E_ℓ be the event that iteration ℓ gives correct x_i.

$$\Pr[E_\ell] \geq \frac{1}{2} + \frac{1}{2p(n)}$$

Expected number of successes:

$$\mathbb{E}[\#\text{correct}] = \left(\frac{1}{2} + \frac{1}{2p(n)}\right) \cdot 100n(p(n))^2 = 50n(p(n))^2 + 50np(n)$$

Variance (using pairwise independence):

$$\text{Var}[\#\text{correct}] \approx \frac{1}{4} \cdot 100n(p(n))^2 = 25n(p(n))^2$$

Applying Chebyshev:

$$\Pr[\text{majority decision for } x_i \text{ incorrect}] \leq \frac{25n(p(n))^2}{(50np(n))^2} = \frac{1}{100n}$$

Union bound over all n bits:

$$\Pr[\text{any } x_i \text{ incorrect}] \leq n \cdot \frac{1}{100n} = \frac{1}{100}$$

Therefore: $p := \Pr[\text{Inverter succeeds} \mid \text{all guesses correct, good } x] \geq 0.99$

1.8 Overall Success Probability

$$\begin{aligned}
\Pr[\text{Inverter succeeds}] &\geq \Pr[\text{succeeds} \mid \text{guesses correct, good } x] \cdot \Pr[\text{guesses correct}] \cdot \Pr[\text{good } x] \\
&= p \cdot \frac{1}{m+1} \cdot \frac{1}{2p(n)} \\
&= p \cdot \frac{1}{2n^2 p(n)^3} \\
&\geq \frac{0.99}{2n^2 p(n)^3}
\end{aligned}$$

This is non-negligible, contradicting one-wayness of F. □

Remark 1. We can amplify success probability to $\approx \frac{1}{p(n)}$ by enumerating over all $2^{\log(m+1)} = m+1$ possible guesses. Each guess yields a candidate inverse, but we can verify which is correct by checking $F(x') = y$.

1.9 Coding-Theoretic View

The mapping $x \mapsto (\langle x, r\rangle)_{r \in \{0,1\}^n}$ is the **Hadamard code**, a highly redundant encoding.

- $P(F(x), r)$ provides access to a *noisy* codeword
- Our proof = list-decoding algorithm for Hadamard code with error rate $\frac{1}{2} - \frac{1}{p(n)}$
- The "pretty good predictor" case = unique decoding with error rate $\frac{1}{4} - \frac{1}{p(n)}$

1.10 General Framework: List-Decodable Codes

Framework (Impagliazzo-Sudan):

Definition 1. *Let $x \to C(x)$ be an encoding. A list-decoder takes a corrupted codeword (incorrect at $\frac{1}{2} - \epsilon$ fraction of locations) and outputs a list $\{x_1, \ldots, x_m\}$ of possible values for x.*

Hardcore Predicate Construction: Define $B_i(x) = C(x)_i$ (the i-th bit of the codeword).
How it works:

1. Hardcore-bit predictor gives access to corrupted codeword
2. Run list-decoder to get candidate inverses
3. Filter candidates by checking $F(x_j) = y$ (since F is efficiently computable)

This provides a general method for constructing hardcore predicates from any list-decodable code.

2 From Stateful to Stateless Encryption

2.1 Stateful Encryption Using PRGs

Setup: Alice and Bob share initial state s_0 (a random seed).
Protocol:

- Both parties maintain synchronized state s_i
- To encrypt: compute $G(s_i) = (s_{i+1}, b_i)$, send $m \oplus b_i$
- Update state to s_{i+1}
- To decrypt: compute $G(s_i) = (s_{i+1}, b_i)$, recover $m = (m \oplus b_i) \oplus b_i$

Advantages:

- Can encrypt arbitrarily many bits
- Each bit uses fresh pseudorandom bit

Disadvantages:

- Alice and Bob must keep states in *perfect synchrony*
- Cannot transmit simultaneously
- If synchronization is lost, both correctness and security fail

2.2 Naive Attempt: Random Index Selection

Idea: Pre-generate a long pseudorandom string $b_1, b_2, \ldots, b_{n^{100}}$ from key $k = s_0$.
Encryption: Pick random index i, send $(i, m \oplus b_i)$.
Problem - Birthday Paradox:

$$\Pr[\text{Alice's first two indices collide}] \geq \frac{1}{n^{100}}$$

This is *not* negligible. Reusing the same one-time pad bit twice completely breaks security.

2.3 Second Attempt: Exponential-Length String

Idea: Pre-generate pseudorandom string $b_1, b_2, \ldots, b_{2^n}$ from key $k = s_0$.
Encryption: Pick random index $i \in [2^n]$, send $(i, m \oplus b_i)$.
Collision Analysis:

$$\Pr[\exists \text{ collision in } t = \text{poly}(n) \text{ indices}] \leq \frac{t^2}{2^n} = \text{negl}(n)$$

Problem: Alice and Bob are *not* polynomial-time. Cannot generate or store exponential-length string.

2.4 The Right Idea: Pseudorandom Functions

Key Insight: Never compute the exponentially long string explicitly.
Instead, we want a function $f_k : \{0,1\}^n \to \{0,1\}^m$ such that:

- $f_k(x) = b_x$, the x-th bit of the implicit pseudorandom string
- Computable in time $\text{poly}(|x|) = \text{poly}(n)$
- For random (or distinct) $x_1, x_2, \ldots$, the values $f_k(x_1), f_k(x_2), \ldots$ are computationally indistinguishable from random bits

This is precisely a **Pseudorandom Function (PRF)**.

3 Pseudorandom Functions (PRFs)

3.1 Definition

Definition 2 (Pseudorandom Function Family)**.** *A collection of functions*

$$\mathcal{F}_\ell = \{f_k : \{0,1\}^\ell \to \{0,1\}^m\}_{k \in \{0,1\}^n}$$

consists of:

- *n: key length (security parameter)*
- *ℓ: input length*
- *m: output length*
- *All parameters are polynomial in security parameter: $\ell, m = poly(n)$*

Key Generation: *$Gen(1^n)$ generates a random n-bit key k.*
Evaluation: *$Eval(k, x)$ is a polynomial-time algorithm that outputs $f_k(x)$.*
Size: *$|\mathcal{F}_\ell| \le 2^n$ (singly exponential in n)*

Definition 3 (All Functions)**.** *The set of all functions:*

$$ALL_\ell = \{f : \{0,1\}^\ell \to \{0,1\}^m\}$$

Size: *$|ALL_\ell| = 2^{m \cdot 2^\ell}$ (doubly exponential in ℓ)*

3.2 Pseudorandomness Property

Definition 4 (Pseudorandom Function)**.** *$\mathcal{F}_\ell$ is a pseudorandom function family if for all p.p.t. distinguishers D with oracle access, there exists a negligible function μ such that:*

$$\left|\Pr[f \leftarrow \mathcal{F}_\ell : D^f(1^n) = 1] - \Pr[f \leftarrow ALL_\ell : D^f(1^n) = 1]\right| \le \mu(n)$$

Visual Representation:

Pseudorandom World	**Random World**
$f_k \leftarrow \mathcal{F}_\ell$	$f \leftarrow \mathrm{ALL}_\ell$
Distinguisher D queries x	Distinguisher D queries x
Receives $f_k(x)$	Receives $f(x)$
Outputs 0 or 1	Outputs 0 or 1

The distinguisher cannot tell which world it's in with non-negligible advantage.

3.3 Key Properties

- **Efficiency:** f_k is computable in polynomial time
- **Compactness:** Only 2^n functions in $\mathcal{F}_\ell$ vs. $2^{m \cdot 2^\ell}$ in ALL_ℓ
- **Pseudorandomness:** Output on any polynomial number of queries looks random
- **Domain size:** Must have 2^ℓ be super-polynomially large in n to avoid collisions

4 Application: PRF $\Rightarrow$ Stateless Encryption

4.1 Construction

Let $f_k : \{0,1\}^\ell \to \{0,1\}^m$ be a PRF where 2^ℓ is super-polynomially large in n.
Key Generation: $\mathrm{Gen}(1^n)$ generates random n-bit key k defining f_k.
Encryption: $\mathrm{Enc}(k, m)$ where $|m| = m$ bits:

1. Pick random $x \leftarrow \{0,1\}^\ell$

2. Output ciphertext $c = (x, y = f_k(x) \oplus m)$

Decryption: $\text{Dec}(k, c)$ where $c = (x, y)$:

1. Compute $f_k(x)$
2. Output $m = f_k(x) \oplus y$

4.2 Correctness

$$\begin{aligned}\text{Dec}(k, \text{Enc}(k, m)) &= \text{Dec}(k, (x, f_k(x) \oplus m)) \\ &= f_k(x) \oplus (f_k(x) \oplus m) \\ &= m\end{aligned}$$

4.3 Security Intuition

Why it's secure:

- Each encryption uses a fresh random $x \leftarrow \{0,1\}^\ell$
- Since 2^ℓ is super-polynomial, collision probability is negligible
- Each message is essentially encrypted with a fresh one-time pad $f_k(x)$
- The PRF property ensures $f_k(x)$ looks random to the adversary

Advantages over stateful encryption:

- No synchronization required
- Can encrypt messages in any order
- Multiple parties can encrypt simultaneously
- No state to maintain or lose

4.4 Requirements

For security, we need:

1. 2^ℓ is super-polynomially large (to avoid collisions)
2. f_k is a secure PRF
3. Random x is chosen independently for each encryption