Pseudorandom Functions - Wrapup

CS 276: Introduction to Cryptography

Sanjam Garg

February 11, 2026

Definition of PRG

Definition 1 (Pseudorandom Generator)

A function $G : \{0,1\}^n \to \{0,1\}^{n+m}$ with $m = \text{poly}(n)$ is called a **pseudorandom generator** (PRG) if:

1. G is computable in polynomial time
2. $U_{n+m} \approx G(U_n)$, where U_k denotes the uniform distribution on $\{0,1\}^k$ and $\approx$ denotes computational indistinguishability

Definition of PRG

Definition 1 (Pseudorandom Generator)

A function $G : \{0,1\}^n \to \{0,1\}^{n+m}$ with $m = \text{poly}(n)$ is called a **pseudorandom generator** (PRG) if:

1. G is computable in polynomial time
2. $U_{n+m} \approx G(U_n)$, where U_k denotes the uniform distribution on $\{0,1\}^k$ and $\approx$ denotes computational indistinguishability

Intuition

Short seed $\to$ longer output that looks random to any PPT distinguisher.

Efficiently Computable Function Ensemble

Definition 2 (Efficiently Computable Function Ensemble)

A function ensemble $\{F_n\}_n$ is **efficiently computable** if:

1. **Succinct**: $\exists$ PPT algorithm I and mapping ϕ such that $\phi(I(1^n))$ and F_n are identically distributed
2. **Efficient**: $\exists$ poly-time machine V such that $V(i, x) = f_i(x)$ for every $x \in \{0, 1\}^n$, where i is in the range of $I(1^n)$ and $f_i = \phi(i)$

Efficiently Computable Function Ensemble

Definition 2 (Efficiently Computable Function Ensemble)

A function ensemble $\{F_n\}_n$ is **efficiently computable** if:

1. **Succinct**: $\exists$ PPT algorithm I and mapping ϕ such that $\phi(I(1^n))$ and F_n are identically distributed
2. **Efficient**: $\exists$ poly-time machine V such that $V(i, x) = f_i(x)$ for every $x \in \{0, 1\}^n$, where i is in the range of $I(1^n)$ and $f_i = \phi(i)$

Key Insight

- A sample from F_n can be generated by sampling a key $k \in \{0, 1\}^n$
- The key k is the description of the function
- Only n bits needed (vs $n \cdot 2^n$ for random functions)!

Definition of PRF

Definition 3 (Pseudorandom Function Ensemble)

A function ensemble $F = \{F_n\}_{n \in \mathbb{N}}$ is **pseudorandom** if for every non-uniform PPT oracle adversary $\mathcal{A}$, there exists a negligible function $\epsilon(n)$ such that:

$$\left| \Pr[\mathcal{A}^{F_n}(1^n) = 1] - \Pr[\mathcal{A}^{R_n}(1^n) = 1] \right| \leq \epsilon(n)$$

Definition of PRF

Definition 3 (Pseudorandom Function Ensemble)

A function ensemble $F = \{F_n\}_{n \in \mathbb{N}}$ is **pseudorandom** if for every non-uniform PPT oracle adversary $\mathcal{A}$, there exists a negligible function $\epsilon(n)$ such that:

$$\left| \Pr[\mathcal{A}^{F_n}(1^n) = 1] - \Pr[\mathcal{A}^{R_n}(1^n) = 1] \right| \leq \epsilon(n)$$

Recall

- Adversary has **oracle access** to the function (queries only)
- R_n: uniformly random function from $\{0,1\}^n$ to $\{0,1\}^n$
- Cannot distinguish PRF from random function

GGM Construction: Setup

Notation

Given PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$:

- $G_0(x), G_1(x)$: first and last n bits of $G(x)$ respectively

GGM Construction: Setup

Notation

Given PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$:

- $G_0(x), G_1(x)$: first and last n bits of $G(x)$ respectively

Construction

For key $K \in \{0,1\}^n$ and input $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$:

$$F_n^{(K)}(x_1 x_2 \cdots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_1}(K))\cdots))$$

GGM Construction: Setup

Notation

Given PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$:

- $G_0(x), G_1(x)$: first and last n bits of $G(x)$ respectively

Construction

For key $K \in \{0,1\}^n$ and input $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$:

$$F_n^{(K)}(x_1 x_2 \cdots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_1}(K))\cdots))$$

Proof: Hybrid H_i

For $i \in \{0,1,\ldots,n\}$, define hybrid H_i:

$$H_i^{(K_i)}(x_1 x_2 \ldots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_{i+1}}(R_i(x_1 \ldots x_i)))\cdots))$$

where K_i is a random function from $\{0,1\}^i$ to $\{0,1\}^n$.

GGM Construction: Setup

Notation

Given PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$:

- $G_0(x), G_1(x)$: first and last n bits of $G(x)$ respectively

Construction

For key $K \in \{0,1\}^n$ and input $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$:

$$F_n^{(K)}(x_1 x_2 \cdots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_1}(K))\cdots))$$

Proof: Hybrid H_i

For $i \in \{0,1,\ldots,n\}$, define hybrid H_i:

$$H_i^{(K_i)}(x_1 x_2 \ldots x_n) := G_{x_n}(G_{x_{n-1}}(\cdots(G_{x_{i+1}}(R_i(x_1 \ldots x_i)))\cdots))$$

where K_i is a random function from $\{0,1\}^i$ to $\{0,1\}^n$.

GGM Proof: The Challenge

Problem

We cannot directly reduce H_i vs H_{i+1} to PRG security.

GGM Proof: The Challenge

Problem

We cannot directly reduce H_i vs H_{i+1} to PRG security.

Solution: Sub-Hybrids

- Define sub-hybrids $H_{i,j}$ for $j \in \{0, \ldots, q\}$ where q is number of queries
- $H_{i,0} = H_i$ and $H_{i,q} = H_{i+1}$
- Each sub-hybrid handles queries one at a time

GGM Proof: Sub-Hybrids

Sub-Hybrid $H_{i,j}$

Let $R_i : \{0,1\}^i \to \{0,1\}^n$ and $S_i : \{0,1\}^{i+1} \to \{0,1\}^n$ be random functions. Initialize list L of i-bit prefixes seen

For query $x = x_1 \dots x_n$:

1. If $|L| < j$ or (or earlier query in this case):
   - Set $y \leftarrow S_i(x_1 \dots x_{i+1})$ (random)
   - Append $(x_1 \dots x_i)$ to L
2. Else:
   - Set $y \leftarrow R_i(x_1 \dots x_i)$ (random)
   - Update $y \leftarrow G_{x_{i+1}}(y)$
3. For $k = i+2$ to n: update $y \leftarrow G_{x_k}(y)$
4. Output y

GGM Proof: Outer Adversary

Construction of $\mathcal{B}$ (same cases as $H_{i,j}$ on previous slide)

$\mathcal{B}$ on input $z \in \{0,1\}^{2n}$ (from U_{2n} or $G(U_n)$). Parse z as $z_0 \| z_1$. Initialize list L of i-bit prefixes seen.

For each query $x = x_1 \ldots x_n$:

1. If $|L| < j - 1$ (or earlier query in this case):
   - Set $y \leftarrow S_i(x_1 \ldots x_{i+1})$ (random); Append $(x_1 \ldots x_i)$ to L
2. If $|L| = j - 1$ (or earlier query in this case) :
   - Set $y \leftarrow z_{x_{i+1}}$ (embed PRG); Append $(x_1 \ldots x_i)$ to L
3. Else:
   - Set $y \leftarrow R_i(x_1 \ldots x_i)$ (random)
   - Update $y \leftarrow G_{x_{i+1}}(y)$
4. For $k = i + 2$ to n: update $y \leftarrow G_{x_k}(y)$
5. Respond with y

$\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

GGM Proof: Outer Adversary

Construction of $\mathcal{B}$ (same cases as $H_{i,j}$ on previous slide)

$\mathcal{B}$ on input $z \in \{0,1\}^{2n}$ (from U_{2n} or $G(U_n)$). Parse z as $z_0 \| z_1$. Initialize list L of i-bit prefixes seen.

For each query $x = x_1 \ldots x_n$:

1. If $|L| < j - 1$ (or earlier query in this case):
   - Set $y \leftarrow S_i(x_1 \ldots x_{i+1})$ (random); Append $(x_1 \ldots x_i)$ to L
2. If $|L| = j - 1$ (or earlier query in this case) :
   - Set $y \leftarrow z_{x_{i+1}}$ (embed PRG); Append $(x_1 \ldots x_i)$ to L
3. Else:
   - Set $y \leftarrow R_i(x_1 \ldots x_i)$ (random)
   - Update $y \leftarrow G_{x_{i+1}}(y)$
4. For $k = i + 2$ to n: update $y \leftarrow G_{x_k}(y)$
5. Respond with y

$\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Naor-Reingold PRF: Motivation

Question

Can we construct PRFs from number-theoretic assumptions like DDH?

Naor-Reingold PRF: Motivation

Question

Can we construct PRFs from number-theoretic assumptions like DDH?

Naor-Reingold PRF

- Based on DDH assumption
- Output is a group element (not a bit string)
- More efficient in some settings
- Key is longer: $(n+1)$ elements

Naor-Reingold PRF: Construction

Setup

- Group ensemble $\{\mathbb{G}_n\}$ where DDH is hard
- Generator g of $\mathbb{G}_n$
- Key space: $\mathcal{K}$

Naor-Reingold PRF: Construction

Setup

- Group ensemble $\{\mathbb{G}_n\}$ where DDH is hard
- Generator g of $\mathbb{G}_n$
- Key space: $\mathcal{K}$

Key Generation

Key $K = (h, u_1, u_2, \ldots, u_n)$ where:

- $u, u_1, \ldots, u_n \xleftarrow{\$} \{0, \ldots, |\mathbb{G}_n| - 1\}$
- $h = g^u$

Naor-Reingold PRF: Construction

Setup

- Group ensemble $\{\mathbb{G}_n\}$ where DDH is hard
- Generator g of $\mathbb{G}_n$
- Key space: $\mathcal{K}$

Key Generation

Key $K = (h, u_1, u_2, \ldots, u_n)$ where:

- $u, u_1, \ldots, u_n \xleftarrow{\$} \{0, \ldots, |\mathbb{G}_n| - 1\}$
- $h = g^u$

Function Evaluation

For input $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$:

$$F_n(K, x) = h^{\prod_{\ell=1}^{n} u_\ell^{x_\ell}} = g^{u \cdot \prod_{\ell=1}^{n} u_\ell^{x_\ell}}$$

Naor-Reingold PRF: Properties

Key Differences from GGM

- **Key length**: $(n+1)$ elements vs n bits
- **Output**: Group element vs bit string
- **Assumption**: DDH vs PRG (which needs OWP)
- **Structure**: Algebraic vs tree-based

Naor-Reingold PRF: Properties

Key Differences from GGM

- **Key length**: $(n + 1)$ elements vs n bits
- **Output**: Group element vs bit string
- **Assumption**: DDH vs PRG (which needs OWP)
- **Structure**: Algebraic vs tree-based

Advantages

- Can be more efficient in practice
- Natural for group-based cryptography
- Useful for certain applications

Naor-Reingold PRF: Security

Lemma 4

Assuming the DDH assumption holds for $\{\mathbb{G}_n\}$, the function ensemble $\{F_n\}$ is pseudorandom.

Naor-Reingold PRF: Security

Lemma 4

Assuming the DDH assumption holds for $\{\mathbb{G}_n\}$, the function ensemble $\{F_n\}$ is pseudorandom.

Proof Strategy

- Similar to GGM proof: hybrid argument
- Key difference: nodes in tree are not independent
- Must handle DDH relations carefully
- Use DDH challenge to embed in reduction

Naor-Reingold Proof: Hybrids

Hybrid H_i

For $i \in \{0, \ldots, n\}$, let $R_i : \{0,1\}^i \to \mathbb{G}$ be a random function.

$$H_i((u, u_{i+1} \ldots u_n), x) = R_i(x_1 \ldots x_i)^{\prod_{\ell=i+1}^{n} u_\ell^{x_\ell}}$$

where $R_0(\cdot) = h$ (constant function).

Naor-Reingold Proof: Hybrids

Hybrid H_i

For $i \in \{0, \ldots, n\}$, let $R_i : \{0,1\}^i \to \mathbb{G}$ be a random function.

$$H_i((u, u_{i+1} \ldots u_n), x) = R_i(x_1 \ldots x_i)^{\prod_{\ell=i+1}^{n} u_\ell^{x_\ell}}$$

where $R_0(\cdot) = h$ (constant function).

Observations

- H_0: Naor-Reingold PRF
- H_n: Random function (uniform group element)
- H_i: First i bits use random function, rest use key

Naor-Reingold Proof: Sub-Hybrids

Sub-Hybrid $H_{i,j}$

Let $R_i : \{0,1\}^i \to \mathbb{G}$ and $S_i : \{0,1\}^{i+1} \to \mathbb{G}$ be random functions. Initialize list L of i-bit prefixes seen. Sample $u_\ell \xleftarrow{\$} \{0,\ldots,|\mathbb{G}|-1\}$ for $\ell = i+1$ to n.

For query $x = x_1 \ldots x_n$:

1. If $|L| < j$ (or earlier query in this case):
   - Set $y \leftarrow S_i(x_1 \ldots x_{i+1})$ (random); Append $(x_1 \ldots x_i)$ to L
2. Else:
   - Set $y \leftarrow R_i(x_1 \ldots x_i)$ (random)
   - Update $y \leftarrow y^{u_{i+1}^{x_{i+1}}}$.
3. Update $y \leftarrow y^{u_\ell^{x_\ell}}$ for $\ell = i+2$ to n.
4. Output y

$H_{i,0} = H_i$ and $H_{i,q} = H_{i+1}$.

Naor-Reingold Proof: Key Insight

DDH Relation

$\mathcal{B}$ receives DDH challenge $(g, A = g^a, B = g^b, C)$ where C is either g^{ab} (DDH tuple) or g^c (random).

Analysis of $\mathcal{B}$

- If $C = g^{ab}$: responses match $H_{i,j}$
- If $C = g^c$: responses match $H_{i,j+1}$
- $\mathcal{B}$ can distinguish DDH tuples from random
- This contradicts DDH assumption

Naor-Reingold Proof: Complexity

Complexity

- Unlike GGM, nodes are not independent
- Must maintain DDH relations across all queries
- More careful handling needed

Naor-Reingold Proof: Outer Adversary

Construction of $\mathcal{B}$ (same cases as $H_{i,j}$ on previous slide)

$\mathcal{B}$ gets DDH challenge $(g, A = g^a, B = g^b, C)$. Sample $u_\ell \xleftarrow{\$} \{0, \ldots, |\mathbb{G}| - 1\}$ for $\ell = i + 2$ to n. We set unknown u_{i+1} to be dlog of A. Initialize list L of i-bit prefixes seen

For each query $x = x_1 \ldots x_n$:

1. If $|L| < j - 1$ (or earlier query in this case):
   - Set $y \leftarrow S_i(x_1 \ldots x_{i+1})$ (random); Append $(x_1 \ldots x_i)$ to L
2. If $|L| = j - 1$ (or earlier query in this case):
   - Set $y \leftarrow B$ if $x_{i+1} = 0$ else $y \leftarrow C$; Append $(x_1 \ldots x_i)$ to L
3. Else:
   - Set $y \leftarrow R_i(x_1 \ldots x_i)$ (random)
   - Update $y \leftarrow y^{u_{i+1}^{x_{i+1}}}$
4. Update $y \leftarrow y^{u_\ell^{x_\ell}}$ for $\ell = i + 2$ to n. Respond with y

$\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Naor-Reingold Proof: Outer Adversary

Construction of $\mathcal{B}$ (same cases as $H_{i,j}$ on previous slide)

$\mathcal{B}$ gets DDH challenge $(g, A = g^a, B = g^b, C)$. Sample $u_\ell \xleftarrow{\$} \{0, \ldots, |\mathbb{G}| - 1\}$ for $\ell = i + 2$ to n. We set unknown u_{i+1} to be dlog of A. Initialize list L of i-bit prefixes seen

For each query $x = x_1 \ldots x_n$:

1. If $|L| < j - 1$ (or earlier query in this case):
   - Set $y \leftarrow S_i(x_1 \ldots x_{i+1})$ (random); Append $(x_1 \ldots x_i)$ to L
2. If $|L| = j - 1$ (or earlier query in this case):
   - Set $y \leftarrow B$ if $x_{i+1} = 0$ else $y \leftarrow C$; Append $(x_1 \ldots x_i)$ to L
3. Else:
   - Set $y \leftarrow R_i(x_1 \ldots x_i)$ (random)
   - Update $y \leftarrow y^{u_{i+1}^{x_{i+1}}}$ (we set u_{i+1} to be dlog of A so can't do this)
4. Update $y \leftarrow y^{u_\ell^{x_\ell}}$ for $\ell = i + 2$ to n. Respond with y

$\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Naor-Reingold Proof: Outer Adversary

Construction of $\mathcal{B}$ (same cases as $H_{i,j}$ on previous slide)

$\mathcal{B}$ gets DDH challenge $(g, A = g^a, B = g^b, C)$. Sample $u_\ell \xleftarrow{\$} \{0, \ldots, |\mathbb{G}| - 1\}$ for $\ell = i + 2$ to n. We set unknown u_{i+1} to be dlog of A. Initialize list L of i-bit prefixes seen

For each query $x = x_1 \ldots x_n$:

1. If $|L| < j - 1$ (or earlier query in this case):
   - Set $y \leftarrow S_i(x_1 \ldots x_{i+1})$ (random); Append $(x_1 \ldots x_i)$ to L
2. If $|L| = j - 1$ (or earlier query in this case):
   - Set $y \leftarrow B$ if $x_{i+1} = 0$ else $y \leftarrow C$; Append $(x_1 \ldots x_i)$ to L
3. Else:
   - Sample $\gamma \leftarrow R_i(x_1 \ldots x_i)$ (random exponent)
   - Set $y \leftarrow g^\gamma$ if $x_{i+1} = 0$ else $y \leftarrow A^\gamma$.
4. Update $y \leftarrow y^{u_\ell^{x_\ell}}$ for $\ell = i + 2$ to n. Respond with y

$\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Questions?