Pseudorandom Functions

CS 276: Introduction to Cryptography

Sanjam Garg

February 11, 2026

Overview

1. Pseudorandom Functions
   - Definitions
   - Construction of PRF from PRG
2. PRFs from DDH: Naor-Reingold

The Problem with Random Functions

Random Functions

Consider the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$:

- Total number: $(2^n)^{2^n} = 2^{n \cdot 2^n}$
- To describe a random function: need $n \cdot 2^n$ bits
- This is **exponential** in n!

The Problem with Random Functions

Random Functions

Consider the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$:

- Total number: $(2^n)^{2^n} = 2^{n \cdot 2^n}$
- To describe a random function: need $n \cdot 2^n$ bits
- This is **exponential** in n!

Goal

Can we have a function that:

- Looks random to any PPT adversary
- But can be described with only **polynomial** (in n) bits?
- And can be evaluated efficiently?

Pseudorandom Functions: Intuition

Key Idea

A pseudorandom function (PRF) is a function that:

- Can be described with a short **key** k (polynomial in n)
- Given key k, can evaluate $F_k(x)$ efficiently
- Cannot be distinguished from a truly random function by any PPT adversary
- Adversary can only query the function polynomially many times

Pseudorandom Functions: Intuition

Key Idea

A pseudorandom function (PRF) is a function that:

- Can be described with a short **key** k (polynomial in n)
- Given key k, can evaluate $F_k(x)$ efficiently
- Cannot be distinguished from a truly random function by any PPT adversary
- Adversary can only query the function polynomially many times

Applications

- Symmetric encryption
- Message authentication codes (MACs)
- Key derivation
- Many other cryptographic protocols

Ensemble

Definition 1 (Ensemble)

An **ensemble** is a family of objects indexed by the security parameter $n \in \mathbb{N}$, written as $\{X_n\}_{n\in\mathbb{N}}$ or $\{X_n\}_n$.
Typically each X_n is a random variable (or distribution) whose description or sample space may depend on n.

Ensemble

Definition 1 (Ensemble)

An **ensemble** is a family of objects indexed by the security parameter $n \in \mathbb{N}$, written as $\{X_n\}_{n \in \mathbb{N}}$ or $\{X_n\}_n$.
Typically each X_n is a random variable (or distribution) whose description or sample space may depend on n.

Examples

- **Distribution ensemble**: $\{X_n\}_n$ where X_n is a distribution over $\{0,1\}^{\ell(n)}$ for some polynomial ℓ
- **Function ensemble**: $\{F_n\}_n$ where F_n is a random variable over functions (defined next)

Function Ensemble

Definition 2 (Function Ensemble)

A **function ensemble** is a sequence of random variables $F_1, F_2, \ldots, F_n, \ldots$ denoted as $\{F_n\}_{n\in\mathbb{N}}$ such that F_n assumes values in the set of functions mapping n-bit input to n-bit output.

Function Ensemble

Definition 2 (Function Ensemble)

A **function ensemble** is a sequence of random variables $F_1, F_2, \ldots, F_n, \ldots$ denoted as $\{F_n\}_{n\in\mathbb{N}}$ such that F_n assumes values in the set of functions mapping n-bit input to n-bit output.

Notation

- We write $\{F_n\}_n$ or simply F_n when clear from context
- Each F_n is a random variable over functions
- Can generalize to functions mapping n-bit inputs to m-bit outputs

Random Function Ensemble

Definition 3 (Random Function Ensemble)

We denote a random function ensemble by $\{R_n\}_{n\in\mathbb{N}}$, where R_n is uniformly distributed over all functions from $\{0,1\}^n$ to $\{0,1\}^n$.

Random Function Ensemble

Definition 3 (Random Function Ensemble)

We denote a random function ensemble by $\{R_n\}_{n\in\mathbb{N}}$, where R_n is uniformly distributed over all functions from $\{0,1\}^n$ to $\{0,1\}^n$.

Key Properties

- A sampling of R_n requires $n \cdot 2^n$ bits to describe
- Truly random: each input-output pair is independent
- This is our “ideal” benchmark for PRFs

Efficiently Computable Function Ensemble

Definition 4 (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 4 (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)!

Pseudorandom Function Ensemble

Definition 5 (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:

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

Pseudorandom Function Ensemble

Definition 5 (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:

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

Key Points

- Adversary $\mathcal{A}$ has **oracle access** to the function
- Can query the function polynomially many times
- Each oracle call costs 1 unit of time
- Cannot distinguish PRF from truly random function

PRF from PRG: The GGM Construction

Goal

Construct a PRF from a length-doubling PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$.

PRF from PRG: The GGM Construction

Goal

Construct a PRF from a length-doubling PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$.

Key Idea: Binary Tree

- View the PRF evaluation as traversing a binary tree
- Root: the key K
- Each level: use PRG to generate two children
- Leaf: the output

GGM Construction: Setup

Notation

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

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

GGM Construction: Setup

Notation

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

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

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

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

Construction

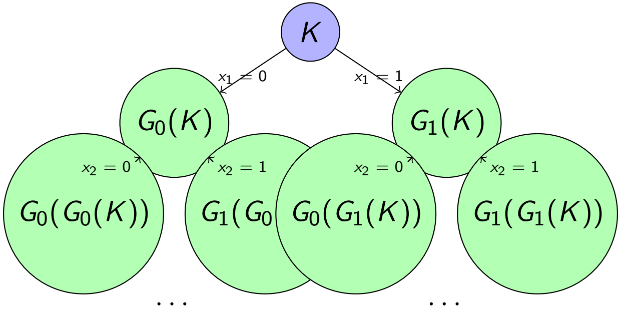
For key $K \in \{0,1\}^n$ and input $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$:

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

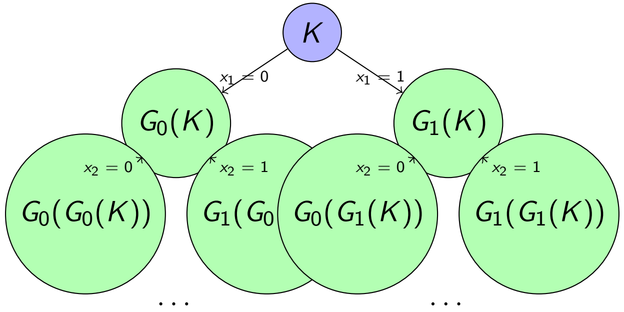
Algorithm

1. Set $y \leftarrow K$
2. For $i = 1$ to n: update $y \leftarrow G_{x_i}(y)$
3. Output y

GGM Construction: Binary Tree View



GGM Construction: Binary Tree View



Intuition

- To evaluate $F_K(x_1 \cdots x_n)$, follow path $x_1, x_2, \ldots, x_n$
- At each level, use G_0 if bit is 0, G_1 if bit is 1
- Final node is the output

GGM Construction: Theorem

Theorem 6 (GGM)

The function ensemble $\{F_n\}_{n\in\mathbb{N}}$ constructed above is pseudorandom.

GGM Construction: Theorem

Theorem 6 (GGM)

The function ensemble $\{F_n\}_{n\in\mathbb{N}}$ *constructed above is pseudorandom.*

Proof Strategy

- Assume for contradiction that $\{F_n\}$ is not a PRF
- Use hybrid argument to show this breaks the PRG
- Key challenge: adversary can make multiple queries
- Need sub-hybrids to handle this

GGM Proof: Hybrids

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: Hybrids

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 $\{0,1\}^n$.

Key Observations

- H_0: PRF (all levels use PRG)
- H_n: Random function (all levels random)
- H_i: Levels 0 to i random, levels $i+1$ to n use PRG

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 $j \in \{0, \dots, q\}$ 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 $x = x_1 \dots x_n$:

1. If $|L| < j - 1$ (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. If $|L| = j - 1$ (or earlier query in this case) :
   - Set $y \leftarrow z_{x_{i+1}}$ (embed PRG); Append $(x_1 \dots x_i)$ to L
3. Else:
   - Set $y \leftarrow R_i(x_1 \dots x_i)$ (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:

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

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:

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

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 7

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

Naor-Reingold PRF: Security

Lemma 7

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 $R_i : \{0,1\}^i \rightarrow \mathbb{G}$ and $S_i : \{0,1\}^{i+1} \rightarrow \mathbb{G}$ be random functions. Initialize list L of i-bit prefixes seen. Sample $u_\ell \overset{\$}{\leftarrow} \{0, \ldots, |\mathbb{G}| - 1\}$ 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 g^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 g^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 g^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 g^y

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