CS 594 – Advanced Crypto (Spring 2026)

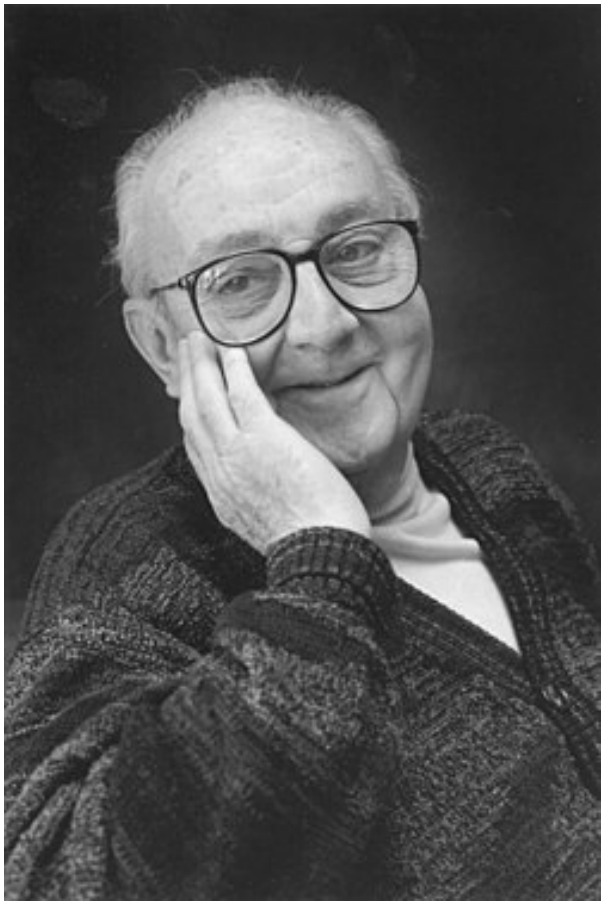
Alex Block

Lecture 5

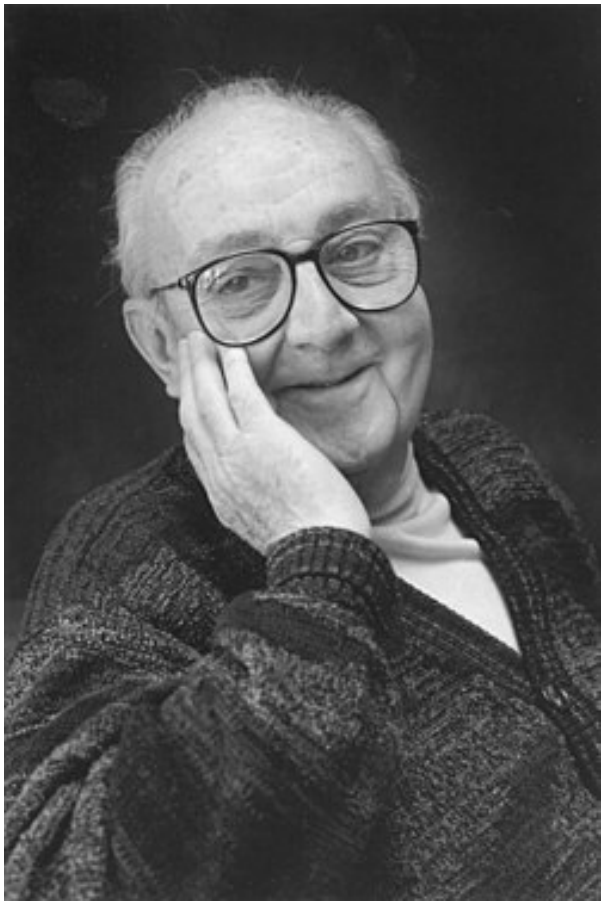
February 2, 2026

Notes on the Random Oracle Model

Notes on the Random Oracle Model



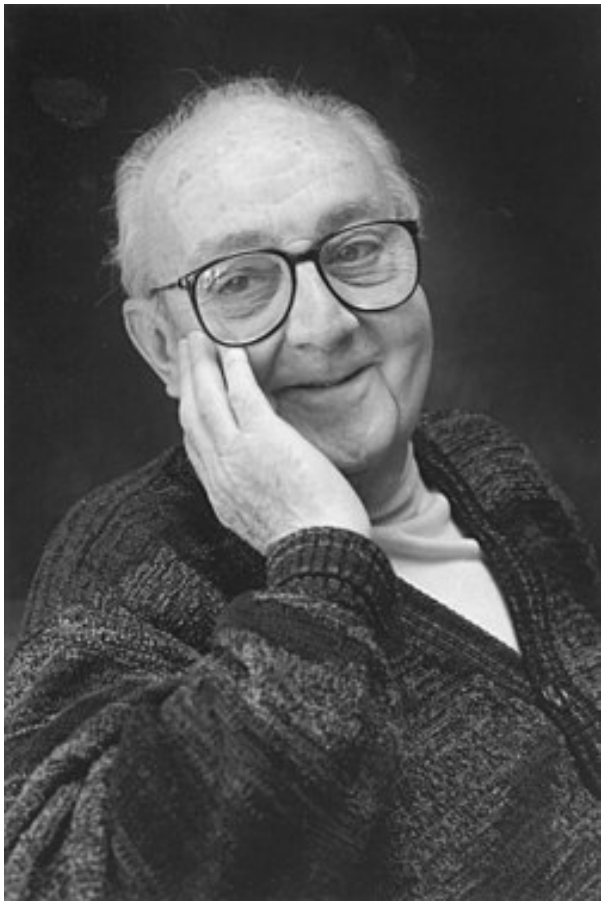
George Box
1919–2013



George Box
1919–2013

All models are wrong,

Notes on the Random Oracle Model



George Box
1919–2013

All models are wrong,

but some are useful.

Notes on the Random Oracle Model

- **The RO model is a model**

Notes on the Random Oracle Model

- **The RO model is a model**
  - *all* security proofs are w.r.t. some simplified model of real-world attacks

Notes on the Random Oracle Model

- **The RO model is a model**
  - *all* security proofs are w.r.t. some simplified model of real-world attacks
  - Security proofs can closely model real-world attack scenarios, or potentially absurd ones

Notes on the Random Oracle Model

- **The RO model is a model**
  - *all* security proofs are w.r.t. some simplified model of real-world attacks
  - Security proofs can closely model real-world attack scenarios, or potentially absurd ones
- **The RO model is *demonstrably* "wrong"**

Notes on the Random Oracle Model

- **The RO model is a model**
  - *all* security proofs are w.r.t. some simplified model of real-world attacks
  - Security proofs can closely model real-world attack scenarios, or potentially absurd ones

- **The RO model is *demonstrably* "wrong"**
  - Real-world hash functions **cannot** behave like a RO

Notes on the Random Oracle Model

- **The RO model is a model**
  - *all* security proofs are w.r.t. some simplified model of real-world attacks
  - Security proofs can closely model real-world attack scenarios, or potentially absurd ones

- **The RO model is *demonstrably* "wrong"**
  - Real-world hash functions **cannot** behave like a RO
  - Security proofs that model hash functions as ROs do not necessarily guarantee security in the real-world

Notes on the Random Oracle Model

- **The RO model is a model**
  - *all* security proofs are w.r.t. some simplified model of real-world attacks
  - Security proofs can closely model real-world attack scenarios, or potentially absurd ones

- **The RO model is *demonstrably* "wrong"**
  - Real-world hash functions **cannot** behave like a RO
  - Security proofs that model hash functions as ROs do not necessarily guarantee security in the real-world
    - In fact, there are examples of schemes which are secure in the RO model **but completely insecure** with any hash function

Why Use the Random Oracle Model?

- **Despite previous slide, the ROM has proven to be very useful.**

Why Use the Random Oracle Model?

- **Despite previous slide, the ROM has proven to be very useful.**
  - Many RO constructions are simpler/more efficient than the best standard model (i.e., real world) constructions

Why Use the Random Oracle Model?

- **Despite previous slide, the ROM has proven to be very useful.**
  - Many RO constructions are simpler/more efficient than the best standard model (i.e., real world) constructions
  - For many real-world constructions, we only know how to show security in the RO model

Why Use the Random Oracle Model?

- **Despite previous slide, the ROM has proven to be very useful.**
  - Many RO constructions are simpler/more efficient than the best standard model (i.e., real world) constructions
  - For many real-world constructions, we only know how to show security in the RO model

- **Even though ROs are impossible, security in the ROM is still useful.**

Why Use the Random Oracle Model?

- **Despite previous slide, the ROM has proven to be very useful.**
  - Many RO constructions are simpler/more efficient than the best standard model (i.e., real world) constructions
  - For many real-world constructions, we only know how to show security in the RO model

- **Even though ROs are impossible, security in the ROM is still useful.**
  - Security proofs in the ROM tell us what security we can expect against attacks in the real world that do not depend on specific properties of the real-world hash function we are modeling as an RO.

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**
  - ROs are powerful, so constructions using ROs often are simpler and more efficient

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**
  - ROs are powerful, so constructions using ROs often are simpler and more efficient
  - Security proofs are often simpler, as any proof with a real-world hash function *cannot* observe how an adversary uses that hash function!

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**
  - ROs are powerful, so constructions using ROs often are simpler and more efficient
  - Security proofs are often simpler, as any proof with a real-world hash function *cannot* observe how an adversary uses that hash function!

- **The ROM is as "controversial" as any other model**

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**
  - ROs are powerful, so constructions using ROs often are simpler and more efficient
  - Security proofs are often simpler, as any proof with a real-world hash function *cannot* observe how an adversary uses that hash function!

- **The ROM is as "controversial" as any other model**
  - Requires a "leap of faith" when replacing a RO with a real-world hash function.

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**
  - ROs are powerful, so constructions using ROs often are simpler and more efficient
  - Security proofs are often simpler, as any proof with a real-world hash function *cannot* observe how an adversary uses that hash function!

- **The ROM is as "controversial" as any other model**
  - Requires a "leap of faith" when replacing a RO with a real-world hash function.
  - Same leap of faith required when talking about any cryptographic assumption in the standard model

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**
  - ROs are powerful, so constructions using ROs often are simpler and more efficient
  - Security proofs are often simpler, as any proof with a real-world hash function *cannot* observe how an adversary uses that hash function!

- **The ROM is as "controversial" as any other model**
  - Requires a "leap of faith" when replacing a RO with a real-world hash function.
  - Same leap of faith required when talking about any cryptographic assumption in the standard model
    - Is there really no algorithm to efficiently factor primes?

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**
  - ROs are powerful, so constructions using ROs often are simpler and more efficient
  - Security proofs are often simpler, as any proof with a real-world hash function *cannot* observe how an adversary uses that hash function!

- **The ROM is as "controversial" as any other model**
  - Requires a "leap of faith" when replacing a RO with a real-world hash function.
  - Same leap of faith required when talking about any cryptographic assumption in the standard model
    - Is there really no algorithm to efficiently factor primes?
    - Do one-way functions really exist?

Why Use the Random Oracle Model?

- **There are advantages to using the ROM**
  - ROs are powerful, so constructions using ROs often are simpler and more efficient
  - Security proofs are often simpler, as any proof with a real-world hash function *cannot* observe how an adversary uses that hash function!

- **The ROM is as "controversial" as any other model**
  - Requires a "leap of faith" when replacing a RO with a real-world hash function.
  - Same leap of faith required when talking about any cryptographic assumption in the standard model
    - Is there really no algorithm to efficiently factor primes?
    - Do one-way functions really exist?
    - Does true randomness actually exist?

Why Use the Random Oracle Model?

A security proof in the random oracle model is *significantly* better than no proof at all!

Other Applications of the ROM

Other Applications of the ROM

- CRHFs

Other Applications of the ROM

- CRHFs
- PRGs

Other Applications of the ROM

- CRHFs
- PRGs
- Public-key Encryption

Other Applications of the ROM

- CRHFs
- PRGs
- Public-key Encryption
- Digital Signatures

Other Applications of the ROM

- CRHFs
- PRGs
- Public-key Encryption
- Digital Signatures
- Fiat-Shamir Transform

Other Applications of the ROM

- CRHFs
- PRGs
- Public-key Encryption
- Digital Signatures
- Fiat-Shamir Transform
- Zero-knowledge Proofs

Other Applications of the ROM

- CRHFs
- PRGs
- Public-key Encryption
- Digital Signatures
- Fiat-Shamir Transform
- Zero-knowledge Proofs
- Black-box separations

Other Applications of the ROM

- CRHFs
- PRGs
- Public-key Encryption
- Digital Signatures
- Fiat-Shamir Transform
- Zero-knowledge Proofs
- Black-box separations
- Complexity Theory

Collision Resistant Hash Functions

Collision Resistant Hash Functions

- A *hash function* h (informally) is a function which takes arbitrary length inputs and *compresses* them to some fixed-length output.

Collision Resistant Hash Functions

- A *hash function* h (informally) is a function which takes arbitrary length inputs and *compresses* them to some fixed-length output.

- Used in basically all real-world efficient crypto systems and beyond.

Collision Resistant Hash Functions

- A *hash function* h (informally) is a function which takes arbitrary length inputs and *compresses* them to some fixed-length output.
- Used in basically all real-world efficient crypto systems and beyond.
- Ideally, want these functions to be *collision resistant*

Collision Resistant Hash Functions

- A *hash function* h (informally) is a function which takes arbitrary length inputs and *compresses* them to some fixed-length output.
- Used in basically all real-world efficient crypto systems and beyond.
- Ideally, want these functions to be *collision resistant*
  - Infeasible for any *efficient* (i.e., PPT) adversary to find collisions.

Collision Resistant Hash Functions

- A *hash function* h (informally) is a function which takes arbitrary length inputs and *compresses* them to some fixed-length output.
- Used in basically all real-world efficient crypto systems and beyond.
- Ideally, want these functions to be *collision resistant*
  - Infeasible for any *efficient* (i.e., PPT) adversary to find collisions.
- Note: we are only interested in hash functions with input domains larger than the output.

Collision Resistant Hash Functions

Definition 1 (Hash Functions)

Collision Resistant Hash Functions

Definition 1 (Hash Functions)

A *hash function* with output length $\ell(\cdot)$ is a pair of PPT algorithms (Gen, H) such that

Collision Resistant Hash Functions

Definition 1 (Hash Functions)

A *hash function* with output length $\ell(\cdot)$ is a pair of PPT algorithms (Gen, H) such that

- Gen is a PPT algorithm which takes as input 1^λ for security parameter λ and outputs a key k (where λ is implicit in k); and

Collision Resistant Hash Functions

Definition 1 (Hash Functions)

A *hash function* with output length $\ell(\cdot)$ is a pair of PPT algorithms (Gen, H) such that

- Gen is a PPT algorithm which takes as input 1^λ for security parameter λ and outputs a key k (where λ is implicit in k); and
- H is *deterministic* that takes as input a key k and a string $x \in \{0,1\}^*$ and outputs a string $H_k(x) \in \{0,1\}^{\ell(\lambda)}$.

Collision Resistant Hash Functions

Definition 1 (Hash Functions)

A *hash function* with output length $\ell(\cdot)$ is a pair of PPT algorithms (Gen, H) such that

- Gen is a PPT algorithm which takes as input 1^λ for security parameter λ and outputs a key k (where λ is implicit in k); and
- H is *deterministic* that takes as input a key k and a string $x \in \{0,1\}^*$ and outputs a string $H_k(x) \in \{0,1\}^{\ell(\lambda)}$.

If H_k is defined only for inputs of length $\ell'(\lambda) > \ell(\lambda)$, then we say that (Gen, H) is a *fixed-length hash function* (for inputs of length $\ell'(\cdot)$).

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda),$$

where $\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$ is defined below.

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda),$$

where $\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$ is defined below.

$\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$:

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda),$$

where $\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$ is defined below.

$\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$:

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda),$$

where $\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$ is defined below.

$\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$:

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. $x, x' \leftarrow \mathcal{A}(1^\lambda, k)$.

Collision Resistant Hash Functions

Definition 2 (Collision Resistance 1)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda),$$

where $\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$ is defined below.

$H_k: \{0,1\}^{l'(\lambda)} \to \{0,1\}^{l(\lambda)}$

$\mathsf{Hash\text{-}Col}_{\mathcal{A},\mathcal{H}}(\lambda)$:

1. ~~$k \leftarrow \mathsf{Gen}(1^\lambda)$~~
2. $x, x' \leftarrow \mathcal{A}(1^\lambda, k)$.
3. Output 1 if and only if $x \neq x'$ and $H_k(x) == H_k(x')$; else output 0.

No key?

H has collisions ✓ keys.

Collision Resistant Hash Functions

Definition 3 (Collision Resistance 2)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if the following two distributions are computationally indistinguishable: for $k \leftarrow \mathsf{Gen}(1^\lambda)$

Collision Resistant Hash Functions

Definition 3 (Collision Resistance 2)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if the following two distributions are computationally indistinguishable: for $k \leftarrow \mathsf{Gen}(1^\lambda)$

$$\frac{D_{0,k}(x, x') :}{\text{return } H_k(x) == H_k(x')}$$

Collision Resistant Hash Functions

Definition 3 (Collision Resistance 2)

A (fixed-length) hash function $\mathcal{H} = (\mathsf{Gen}, H)$ is *collision resistant* if the following two distributions are computationally indistinguishable: for $k \leftarrow \mathsf{Gen}(1^\lambda)$

$\underline{D_{0,k}(x, x') :}$
return $H_k(x) == H_k(x')$

$\underline{D_{1,k}(x, x') :}$
return $x == x'$

Adv:

$x = x'$

- $D_{0,k}(x,x) = D_{1,k}(x,x)$

$x \neq x'$

$D_{1,k}(x,x') = 0$

$\Downarrow$

$D_{0,k}(x,x') = 1$ w.p. $\mathrm{negl}(\lambda)$

CRHFs from Random Oracles

CRHFs from Random Oracles

Claim 1

If $H : \{0,1\}^{\ell'} \to \{0,1\}^{\ell}$ *is a random oracle such that* $\ell' > \ell$*, then* H *is a CRHF.*

CRHFs from Random Oracles

Claim 1

If $H : \{0,1\}^{\ell'} \to \{0,1\}^{\ell}$ *is a random oracle such that* $\ell' > \ell$*, then* H *is a CRHF.*

Random Oracles do not need keys!

CRHFs Without Keys

CRHFs Without Keys

- Real-world hash functions are *unkeyed*

CRHFs Without Keys

- Real-world hash functions are *unkeyed*
- Theoretically, these functions *cannot* be CRHFs

CRHFs Without Keys

- Real-world hash functions are *unkeyed*
- Theoretically, these functions *cannot* be CRHFs
- Intuition: for keyed hash functions, the key is chosen *after* and adversary is fixed

CRHFs Without Keys

- Real-world hash functions are *unkeyed*
- Theoretically, these functions *cannot* be CRHFs
- Intuition: for keyed hash functions, the key is chosen *after* and adversary is fixed
- For unkeyed hash functions, *there exists* and adversary which trivially outputs a collision

CRHFs Without Keys

- Real-world hash functions are *unkeyed*
- Theoretically, these functions *cannot* be CRHFs
- Intuition: for keyed hash functions, the key is chosen *after* and adversary is fixed
- For unkeyed hash functions, *there exists* and adversary which trivially outputs a collision
  - We believe this adversary *cannot be found efficiently*

Other Idealized Models

Other Idealized Models

Other Idealized Models

- QROM

Other Idealized Models

- QROM
- Pseudorandom Oracle Model

Other Idealized Models

- QROM
- Pseudorandom Oracle Model
- AGM, GGM, GBM

Other Idealized Models

- QROM
- Pseudorandom Oracle Model
- AGM, GGM, GBM
- Ideal Functionalities (very general)

Other Idealized Models

- QROM
- Pseudorandom Oracle Model
- AGM, GGM, GBM
- Ideal Functionalities (very general)
  - Ideal Permutation Model

Other Idealized Models

- QROM
- Pseudorandom Oracle Model
- AGM, GGM, GBM
- Ideal Functionalities (very general)
  - Ideal Permutation Model
  - Ideal Cipher Model

Other Idealized Models

- QROM
- Pseudorandom Oracle Model
- AGM, GGM, GBM
- Ideal Functionalities (very general)
  - Ideal Permutation Model
  - Ideal Cipher Model
  - Ideal Obfuscation

Other Idealized Models

- QROM
- Pseudorandom Oracle Model
- AGM, GGM, GBM
- Ideal Functionalities (very general)
  - Ideal Permutation Model
  - Ideal Cipher Model
  - Ideal Obfuscation
  - Ideal <Insert Useful Crypto Functionality>

Other Idealized Models

- QROM
- Pseudorandom Oracle Model
- AGM, GGM, GBM
- Ideal Functionalities (very general)
  - Ideal Permutation Model
  - Ideal Cipher Model
  - Ideal Obfuscation
  - Ideal <Insert Useful Crypto Functionality>
  - ...

Quantum Random Oracle Model

Quantum Random Oracle Model

- Random Oracle Model but for *quantum* adversaries.

Quantum Random Oracle Model

- Random Oracle Model but for *quantum* adversaries.
- Quantum Adversaries are allowed to make *super-position queries*.

Quantum Random Oracle Model

- Random Oracle Model but for *quantum* adversaries.
- Quantum Adversaries are allowed to make *super-position queries*.
- We'll discuss this more in post-quantum crypto at the end of the course.

Idealized Group Models

Idealized Group Models

- Cryptography uses many *group-based assumptions*

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA
  - Unknown Order

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA
  - Unknown Order
  - Pairings/Bilinear Maps

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA
  - Unknown Order
  - Pairings/Bilinear Maps
  - DDH

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA
  - Unknown Order
  - Pairings/Bilinear Maps
  - DDH
  - Group Actions (PQ Crypto)

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA
  - Unknown Order
  - Pairings/Bilinear Maps
  - DDH
  - Group Actions (PQ Crypto)
- Sometimes we want *idealized models* for representing access to group elements

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA
  - Unknown Order
  - Pairings/Bilinear Maps
  - DDH
  - Group Actions (PQ Crypto)
- Sometimes we want *idealized models* for representing access to group elements
- Useful for *lower bounds*

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA
  - Unknown Order
  - Pairings/Bilinear Maps
  - DDH
  - Group Actions (PQ Crypto)
- Sometimes we want *idealized models* for representing access to group elements
- Useful for *lower bounds*
  - What are the *fastest* algorithms for breaking discrete-log given idealized access to group elements?

Idealized Group Models

- Cryptography uses many *group-based assumptions*
  - Discrete-log
  - RSA
  - Unknown Order
  - Pairings/Bilinear Maps
  - DDH
  - Group Actions (PQ Crypto)
- Sometimes we want *idealized models* for representing access to group elements
- Useful for *lower bounds*
  - What are the *fastest* algorithms for breaking discrete-log given idealized access to group elements?
- So security in an idealized group model gives some confidence in the real-world security of group-based crypto.

Generic Group Model

Generic Group Model

- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.

$(\mathbb{Z}_p, 1)$ example

Generic Group Model

- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.
- The *Generic Group Model* (of Shoup) assumes the following about the group $\mathbb{G}$:

Generic Group Model

- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.
- The *Generic Group Model* (of Shoup) assumes the following about the group $\mathbb{G}$:
  1. The elements of $\mathbb{G}$ are labeled *uniformly at random*; and

Generic Group Model

- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.
- The *Generic Group Model* (of Shoup) assumes the following about the group $\mathbb{G}$:
  1. The elements of $\mathbb{G}$ are labeled *uniformly at random*; and
  2. There is an oracle $\mathcal{O}$ which computes the group operation on two elements.

Generic Group Model

- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.
- The *Generic Group Model* (of Shoup) assumes the following about the group $\mathbb{G}$:
  1. The elements of $\mathbb{G}$ are labeled *uniformly at random*; and
  2. There is an oracle $\mathcal{O}$ which computes the group operation on two elements.

$$(\mathbb{G}, +)$$
$$L \xleftarrow{\$} \{f : \mathbb{Z}_p \to \{0,1\}^n \mid f \text{ is injective}\}$$

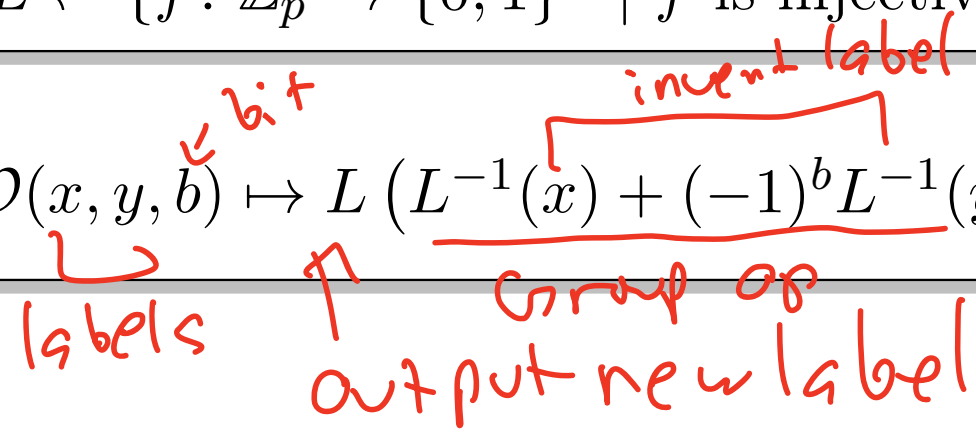
Generic Group Model

- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.
- The *Generic Group Model* (of Shoup) assumes the following about the group $\mathbb{G}$:
  1. The elements of $\mathbb{G}$ are labeled *uniformly at random*; and
  2. There is an oracle $\mathcal{O}$ which computes the group operation on two elements.

$$(\mathbb{G}, +)$$

$$L \xleftarrow{\$} \{f : \mathbb{Z}_p \to \{0,1\}^n \mid f \text{ is injective}\}$$

$$\mathcal{O}(x, y, b) \mapsto L\left(L^{-1}(x) + (-1)^b L^{-1}(y)\right)$$



Generic Group Model

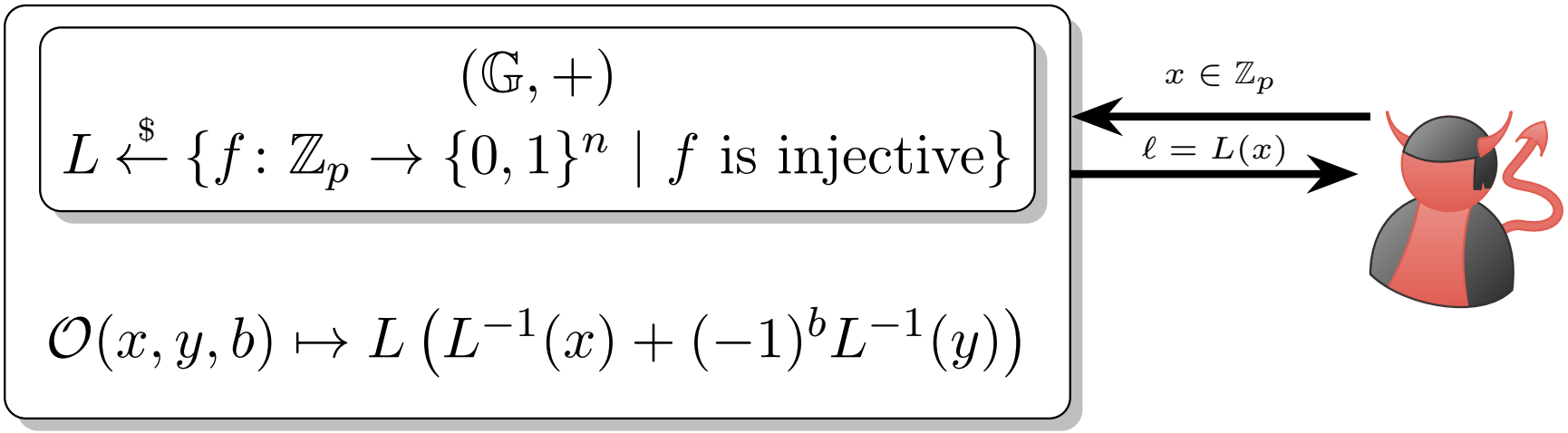
- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.
- The *Generic Group Model* (of Shoup) assumes the following about the group $\mathbb{G}$:
  1. The elements of $\mathbb{G}$ are labeled *uniformly at random*; and
  2. There is an oracle $\mathcal{O}$ which computes the group operation on two elements.

$$(\mathbb{G}, +)$$
$$L \xleftarrow{\$} \{f : \mathbb{Z}_p \to \{0,1\}^n \mid f \text{ is injective}\}$$

$$\mathcal{O}(x, y, b) \mapsto L\left(L^{-1}(x) + (-1)^b L^{-1}(y)\right)$$

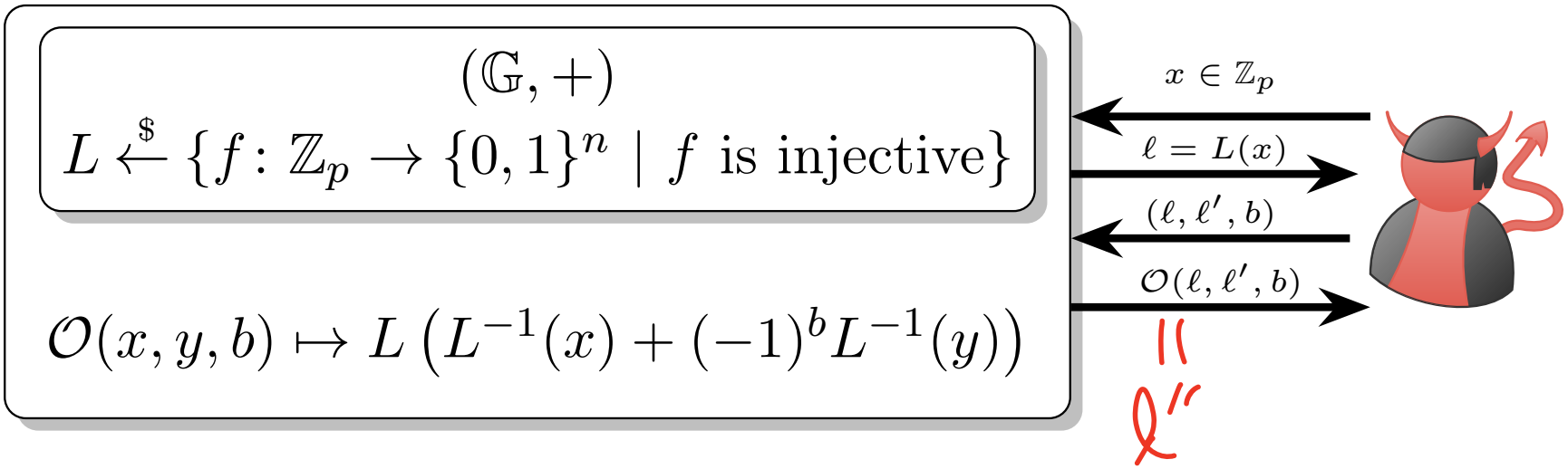
Generic Group Model

- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.
- The *Generic Group Model* (of Shoup) assumes the following about the group $\mathbb{G}$:
  1. The elements of $\mathbb{G}$ are labeled *uniformly at random*; and
  2. There is an oracle $\mathcal{O}$ which computes the group operation on two elements.



Generic Group Model

- Let $(\mathbb{G}, +)$ be a cyclic group of prime order p, where $n := \lceil \log(p) \rceil$.
- The *Generic Group Model* (of Shoup) assumes the following about the group $\mathbb{G}$:
  1. The elements of $\mathbb{G}$ are labeled *uniformly at random*; and
  2. There is an oracle $\mathcal{O}$ which computes the group operation on two elements.



Generic Bilinear Group Model

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Definition 4 (Bilinear/Pairing-friendly Groups)

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Definition 4 (Bilinear/Pairing-friendly Groups)

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p and operator $+$.

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Definition 4 (Bilinear/Pairing-friendly Groups)

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p and operator $+$. Let $g_i \in \mathbb{G}_i$ for $i \in [2]$ each be a generator of the (respective) group.

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Definition 4 (Bilinear/Pairing-friendly Groups)

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p and operator $+$. Let $g_i \in \mathbb{G}_i$ for $i \in [2]$ each be a generator of the (respective) group. We say that $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are *bilinear groups* (or *pairing-friendly groups*) if

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Definition 4 (Bilinear/Pairing-friendly Groups)

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p and operator $+$. Let $g_i \in \mathbb{G}_i$ for $i \in [2]$ each be a generator of the (respective) group. We say that $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are *bilinear groups* (or *pairing-friendly groups*) if there exists an efficiently computable function (called a *pairing*) $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that the following hold:

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Definition 4 (Bilinear/Pairing-friendly Groups)

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p and operator $+$. Let $g_i \in \mathbb{G}_i$ for $i \in [2]$ each be a generator of the (respective) group. We say that $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are *bilinear groups* (or *pairing-friendly groups*) if there exists an efficiently computable function (called a *pairing*) $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that the following hold:

1. (Bilinear map) For all $u, u' \in \mathbb{G}_1$ and $v, v' \in \mathbb{G}_2$, we have $e(u + u', v) = e(u, v) + e(u', v)$ and $e(u, v + v') = e(u, v) + e(u, v')$;

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Definition 4 (Bilinear/Pairing-friendly Groups)

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p and operator $+$. Let $g_i \in \mathbb{G}_i$ for $i \in [2]$ each be a generator of the (respective) group. We say that $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are *bilinear groups* (or *pairing-friendly groups*) if there exists an efficiently computable function (called a *pairing*) $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that the following hold:

1. (Bilinear map) For all $u, u' \in \mathbb{G}_1$ and $v, v' \in \mathbb{G}_2$, we have $e(u + u', v) = e(u, v) + e(u', v)$ and $e(u, v + v') = e(u, v) + e(u, v')$;
2. (Non-degenerate) $g_T := e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.

Generic Bilinear Group Model

- Extension of GGM to *bilinear groups*.

Definition 4 (Bilinear/Pairing-friendly Groups)

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p and operator $+$. Let $g_i \in \mathbb{G}_i$ for $i \in [2]$ each be a generator of the (respective) group. We say that $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ are *bilinear groups* (or *pairing-friendly groups*) if there exists an efficiently computable function (called a *pairing*) $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that the following hold:

1. (Bilinear map) For all $u, u' \in \mathbb{G}_1$ and $v, v' \in \mathbb{G}_2$, we have $e(u + u', v) = e(u, v) + e(u', v)$ and $e(u, v + v') = e(u, v) + e(u, v')$;
2. (Non-degenerate) $g_T := e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.

Theorem 1 (Informal; Zhandry & Zhang, ASIACRYPT'23)

The GBM is strictly stronger than the GGM, which is strictly stronger than the ROM.

Algebraic Group Model

;

Algebraic Group Model

- Somewhat milder model than GGM

;

Algebraic Group Model

- Somewhat milder model than GGM
- Intuitive explanation: Adversaries get explicit access to group elements but are "algebraic"

;

Algebraic Group Model

- Somewhat milder model than GGM
- Intuitive explanation: Adversaries get explicit access to group elements but are "algebraic"
  - If adversary queries some group elements $g_0, \ldots, g_i$, then outputs an element g, it *must* be "explained" by $g_0, \ldots, g_i$;

Algebraic Group Model

- Somewhat milder model than GGM
- Intuitive explanation: Adversaries get explicit access to group elements but are "algebraic"
  - If adversary queries some group elements $g_0, \ldots, g_i$, then outputs an element g, it *must* be "explained" by $g_0, \ldots, g_i$; i.e., it must output $a_0, \ldots, a_i$ such that

$$g = \sum_{i=0}^{i} g_i \cdot a_i.$$

Algebraic Group Model

- Somewhat milder model than GGM
- Intuitive explanation: Adversaries get explicit access to group elements but are "algebraic"
  - If adversary queries some group elements $g_0, \ldots, g_i$, then outputs an element g, it *must* be "explained" by $g_0, \ldots, g_i$; i.e., it must output $a_0, \ldots, a_i$ such that

$$g = \sum_{i=0}^{i} g_i \cdot a_i.$$

AGM Requirement

Non-group elements given as input to an oracle must not depend on group elements.

Adv.

Ideal Permutation Model

Ideal Permutation Model

- Sometimes, instead of PRFs we want *pseudorandom permutations*

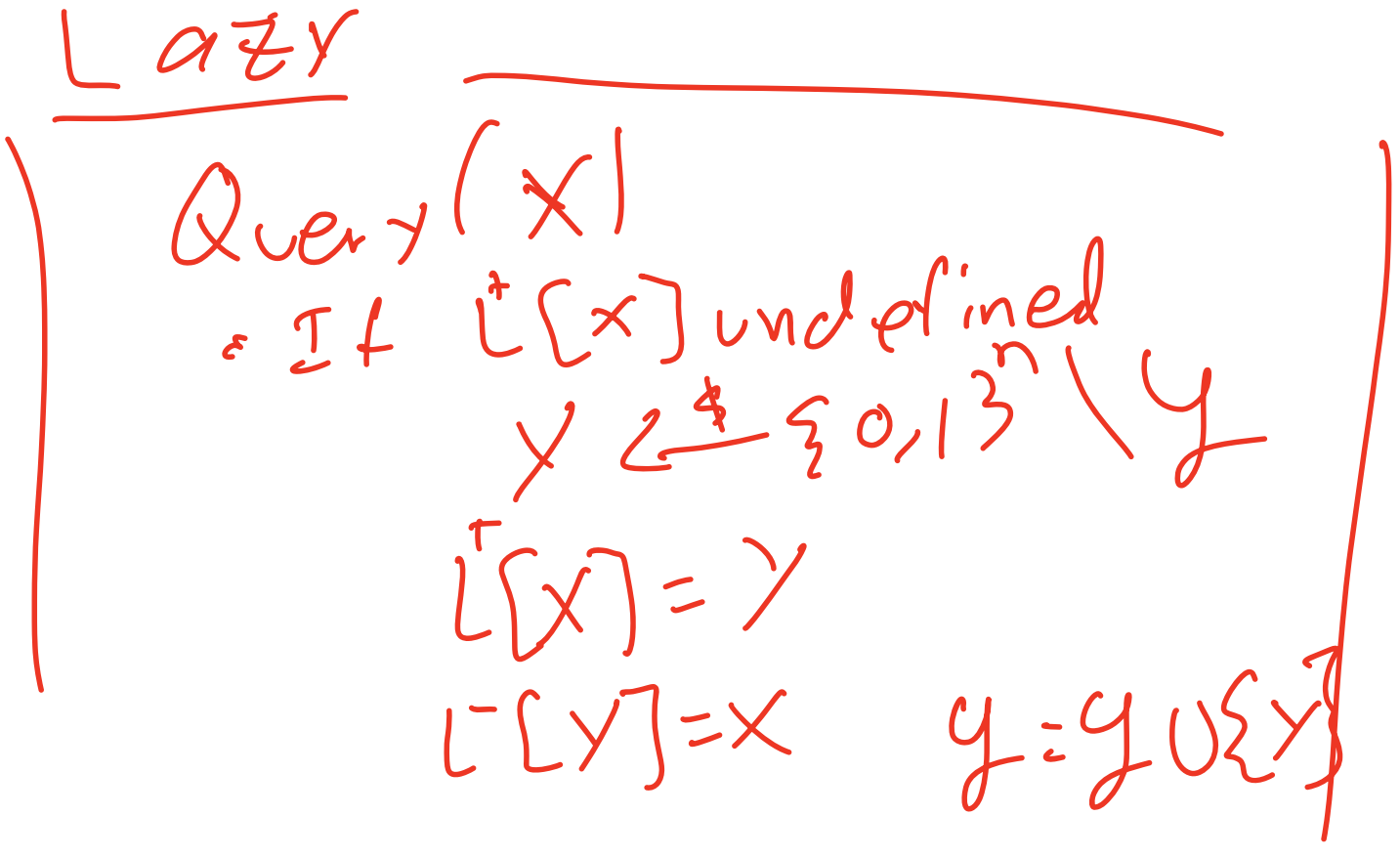
Ideal Permutation Model

- Sometimes, instead of PRFs we want *pseudorandom permutations*
  - PRFs that are *bijections*.

Ideal Permutation Model

- Sometimes, instead of PRFs we want *pseudorandom permutations*
  - PRFs that are *bijections*.

- *Ideal Permutation Model*: ROM but you have oracle access to a random permutation *and its inverse*.



Ideal Cipher Model

Ideal Cipher Model

- Stronger version of Ideal Permutation Model.

Ideal Cipher Model

- Stronger version of Ideal Permutation Model.

- PRPs: F_k and F_k^{-1} are indistinguishable from random *when the key k is uniform and secret.*

Ideal Cipher Model

- Stronger version of Ideal Permutation Model.
- PRPs: F_k and F_k^{-1} are indistinguishable from random *when the key k is uniform and secret.*
- Ideal Cipher Model: F_k and F_k^{-1} are indistinguishable from random *for every key k.*

Next Time: Secret Sharing