CS 594 – Advanced Crypto (Spring 2026)

Alex Block

Lecture 23

April 15, 2026

Wrapping up MHFs

Last Time: Cumulative Memory Complexity

Last Time: Cumulative Memory Complexity

- Since our model for MHFs utilizes a random oracle H, we need to define or memory model and costs with respect to a random oracle.

Last Time: Cumulative Memory Complexity

- Since our model for MHFs utilizes a random oracle H, we need to define or memory model and costs with respect to a random oracle.
- Let A_f^H be a *randomized* algorithm computing f for any input x that is allowed to make *parallel* calls to random oracle H.

Last Time: Cumulative Memory Complexity

- Since our model for MHFs utilizes a random oracle H, we need to define or memory model and costs with respect to a random oracle.
- Let A_f^H be a *randomized* algorithm computing f for any input x that is allowed to make *parallel* calls to random oracle H.
- Let $\text{Trace}(A_f^H, x) = (\sigma_0, \sigma_1, \ldots, \sigma_T)$ be the *execution trace* of $A_f^H(x)$ terminating in T steps.

Last Time: Cumulative Memory Complexity

- Since our model for MHFs utilizes a random oracle H, we need to define or memory model and costs with respect to a random oracle.
- Let A_f^H be a *randomized* algorithm computing f for any input x that is allowed to make *parallel* calls to random oracle H.
- Let $\text{Trace}(A_f^H, x) = (\sigma_0, \sigma_1, \ldots, \sigma_T)$ be the *execution trace* of $A_f^H(x)$ terminating in T steps.

Definition 1 (cmc)

The *cumulative memory complexity (cmc)* of A is defined as

$$\mathsf{cmc}(A_f) = \mathop{\mathbb{E}}_{H}\left[\min_{x,r} \sum_{\sigma \in \text{Trace}(A_f^H(r),x)} |\sigma|\right],$$

where $A_f^H(r)$ denotes that A's random coins are fixed to the string r.

Last Time: cmc and CC

Last Time: cmc and CC

- How can we relate cmc to CC?

Last Time: cmc and CC

- How can we relate cmc to CC?
  - Intuition: if A_f^H is computing $f_{G,H}$ (defined by DAG G and RO H), then the pebbling cost ($\mathsf{CC}(G)$) should translate to $\mathsf{cmc}(A_f)$.

Last Time: cmc and CC

- How can we relate cmc to CC?
  - Intuition: if A_f^H is computing $f_{G,H}$ (defined by DAG G and RO H), then the pebbling cost ($\mathsf{CC}(G)$) should translate to $\mathsf{cmc}(A_f)$.

Theorem 1 (Alwen-Serbinenko 2015)

Let A_f^H be an algorithm computing a function $f_{G,H}$ defined by a DAG G and random oracle $H\colon \{0,1\}^* \to \{0,1\}^\lambda$.

Last Time: cmc and CC

- How can we relate cmc to CC?
  - Intuition: if A_f^H is computing $f_{G,H}$ (defined by DAG G and RO H), then the pebbling cost ($\mathsf{CC}(G)$) should translate to $\mathsf{cmc}(A_f)$.

Theorem 1 (Alwen-Serbinenko 2015)

Let A_f^H be an algorithm computing a function $f_{G,H}$ defined by a DAG G and random oracle $H\colon \{0,1\}^* \to \{0,1\}^\lambda$. Then, in the parallel random oracle model (PROM), we have $\mathsf{cmc}(A_f) = \Omega(\lambda \cdot \mathsf{CC}(G))$.

Last Time: cmc and CC

- How can we relate cmc to CC?
  - Intuition: if A_f^H is computing $f_{G,H}$ (defined by DAG G and RO H), then the pebbling cost ($\mathsf{CC}(G)$) should translate to $\mathsf{cmc}(A_f)$.

Theorem 1 (Alwen-Serbinenko 2015)

Let A_f^H be an algorithm computing a function $f_{G,H}$ defined by a DAG G and random oracle $H\colon \{0,1\}^* \to \{0,1\}^\lambda$. Then, in the parallel random oracle model (PROM), we have $\mathsf{cmc}(A_f) = \Omega(\lambda \cdot \mathsf{CC}(G))$.

- We will give a very high-level overview of the proof.

Last Time: Label Distinctness

Last Time: Label Distinctness

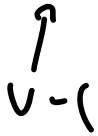
- Suppose we have a random oracle $H : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.

Last Time: Label Distinctness

- Suppose we have a random oracle $H\colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.
- Suppose we are given a DAG G on n vertices $V = \{1, \ldots, n\}$ such that $\delta(G) = 2$.

Last Time: Label Distinctness

- Suppose we have a random oracle $H\colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.
- Suppose we are given a DAG G on n vertices $V = \{1, \ldots, n\}$ such that $\delta(G) = 2$.
- Moreover, suppose that each vertex v has parents v and $r(v) < v - 1$ for some function r.



Last Time: Label Distinctness

- Suppose we have a random oracle $H\colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.
- Suppose we are given a DAG G on n vertices $V = \{1, \ldots, n\}$ such that $\delta(G) = 2$.
- Moreover, suppose that each vertex v has parents v and $r(v) < v - 1$ for some function r.
- Define $x = L_0 \in \{0,1\}^{\lambda}$ as the initial input.

Last Time: Label Distinctness

- Suppose we have a random oracle $H\colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.
- Suppose we are given a DAG G on n vertices $V = \{1, \ldots, n\}$ such that $\delta(G) = 2$.
- Moreover, suppose that each vertex v has parents v and $r(v) < v - 1$ for some function r.
- Define $x = L_0 \in \{0,1\}^{\lambda}$ as the initial input.
  - Define labels $L_1 = H(L_0, 0^{\lambda})$, $L_2 = H(L_1, 0^{\lambda})$, and

Last Time: Label Distinctness

- Suppose we have a random oracle $H\colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.
- Suppose we are given a DAG G on n vertices $V = \{1, \ldots, n\}$ such that $\delta(G) = 2$.
- Moreover, suppose that each vertex v has parents v and $r(v) < v - 1$ for some function r.
- Define $x = L_0 \in \{0,1\}^{\lambda}$ as the initial input.
  - Define labels $L_1 = H(L_0, 0^{\lambda})$, $L_2 = H(L_1, 0^{\lambda})$, and

$$L_3 = H(L_2, L_1)$$

Last Time: Label Distinctness

- Suppose we have a random oracle $H\colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.
- Suppose we are given a DAG G on n vertices $V = \{1, \ldots, n\}$ such that $\delta(G) = 2$.
- Moreover, suppose that each vertex v has parents v and $r(v) < v - 1$ for some function r.
- Define $x = L_0 \in \{0,1\}^{\lambda}$ as the initial input.
  - Define labels $L_1 = H(L_0, 0^{\lambda})$, $L_2 = H(L_1, 0^{\lambda})$, and

$$L_3 = H(L_2, L_1)$$

$$\ldots$$

$$L_v = H(L_{v-1}, L_{r(v)})$$

Last Time: Label Distinctness

- Suppose we have a random oracle $H\colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$.
- Suppose we are given a DAG G on n vertices $V = \{1, \ldots, n\}$ such that $\delta(G) = 2$.
- Moreover, suppose that each vertex v has parents v and $r(v) < v - 1$ for some function r.
- Define $x = L_0 \in \{0,1\}^{\lambda}$ as the initial input.
  - Define labels $L_1 = H(L_0, 0^{\lambda})$, $L_2 = H(L_1, 0^{\lambda})$, and

$$L_3 = H(L_2, L_1)$$
$$\ldots$$
$$L_v = H(L_{v-1}, L_{r(v)})$$
$$\ldots$$
$$L_n = H(L_{n-1}, L_{r(n)}).$$

Last Time: Pre-labeling and Label Collisions

Last Time: Pre-labeling and Label Collisions

- Define the *pre-labeling* of a node v as $\textbf{prelab}(v) = (L_{v-1}, L_{r(v)})$.

Last Time: Pre-labeling and Label Collisions

- Define the *pre-labeling* of a node v as $\textbf{prelab}(v) = (L_{v-1}, L_{r(v)})$.
- **Question**: suppose we can make at most q queries to the random oracle.

Last Time: Pre-labeling and Label Collisions

- Define the *pre-labeling* of a node v as $\textbf{prelab}(v) = (L_{v-1}, L_{r(v)})$.

- **Question**: suppose we can make at most q queries to the random oracle.
  - What is the probability we find z such that $L_v = H(z)$ but $z \neq \textbf{prelab}(v)$ for some node v?

Last Time: Pre-labeling and Label Collisions

- Define the *pre-labeling* of a node v as $\textbf{prelab}(v) = (L_{v-1}, L_{r(v)})$.

- **Question**: suppose we can make at most q queries to the random oracle.
  - What is the probability we find z such that $L_v = H(z)$ but $z \neq \textbf{prelab}(v)$ for some node v?

- **Answer**: probability at most $n \cdot q \cdot 2^{-\lambda}$.

Last Time: Pre-labeling and Label Collisions

- Define the *pre-labeling* of a node v as $\mathbf{prelab}(v) = (L_{v-1}, L_{r(v)})$.

- **Question**: suppose we can make at most q queries to the random oracle.
  - What is the probability we find z such that $L_v = H(z)$ but $z \neq \mathbf{prelab}(v)$ for some node v?

- **Answer**: probability at most $n \cdot q \cdot 2^{-\lambda}$.
  - Let z_i be the i-th query to the RO such that $z_i \neq \mathbf{prelab}(v)$ for any $v \leqslant n$.

Last Time: Pre-labeling and Label Collisions

- Define the *pre-labeling* of a node v as $\textbf{prelab}(v) = (L_{v-1}, L_{r(v)})$.

- **Question**: suppose we can make at most q queries to the random oracle.
  - What is the probability we find z such that $L_v = H(z)$ but $z \neq \textbf{prelab}(v)$ for some node v?

- **Answer**: probability at most $n \cdot q \cdot 2^{-\lambda}$.
  - Let z_i be the i-th query to the RO such that $z_i \neq \textbf{prelab}(v)$ for any $v \leqslant n$.
  - Then, we have

Last Time: Pre-labeling and Label Collisions

- Define the *pre-labeling* of a node v as $\mathbf{prelab}(v) = (L_{v-1}, L_{r(v)})$.

- **Question**: suppose we can make at most q queries to the random oracle.
  - What is the probability we find z such that $L_v = H(z)$ but $z \neq \mathbf{prelab}(v)$ for some node v?

- **Answer**: probability at most $n \cdot q \cdot 2^{-\lambda}$.
  - Let z_i be the i-th query to the RO such that $z_i \neq \mathbf{prelab}(v)$ for any $v \leqslant n$.
  - Then, we have

$$\Pr[H(z_i) \in \{L_1, \ldots, L_n\}] \leqslant n \cdot 2^{-\lambda}$$

Last Time: Pre-labeling and Label Collisions

- Define the *pre-labeling* of a node v as $\mathbf{prelab}(v) = (L_{v-1}, L_{r(v)})$.

- **Question**: suppose we can make at most q queries to the random oracle.
  - What is the probability we find z such that $L_v = H(z)$ but $z \neq \mathbf{prelab}(v)$ for some node v?

- **Answer**: probability at most $n \cdot q \cdot 2^{-\lambda}$.
  - Let z_i be the i-th query to the RO such that $z_i \neq \mathbf{prelab}(v)$ for any $v \leqslant n$.
  - Then, we have

$$\Pr[H(z_i) \in \{L_1, \ldots, L_n\}] \leqslant n \cdot 2^{-\lambda}$$
$$\Pr[\exists i \leqslant q \colon H(z_i) \in \{L_1, \ldots, L_n\}] \leqslant n \cdot q \cdot 2^{-\lambda}.$$

Ex Post Facto Pebbling

Ex Post Facto Pebbling

- We now describe the *ex post facto pebbling* strategy.

Ex Post Facto Pebbling

- We now describe the *ex post facto pebbling* strategy.

- Again, fix PROM algorithm A, input x, and RO H and recall the execution trace $\text{Trace}_{A,H}(x) = \{\sigma_i, \mathbf{q}_i, \mathbf{a}_i\}_{i \in [t]}$.

Ex Post Facto Pebbling

- We now describe the *ex post facto pebbling* strategy.
- Again, fix PROM algorithm A, input x, and RO H and recall the execution trace $\text{Trace}_{A,H}(x) = \{\sigma_i, \mathbf{q}_i, \mathbf{a}_i\}_{i \in [t]}$.
- For each node $v \in [n]$, track L_v as follows.

Ex Post Facto Pebbling

- We now describe the *ex post facto pebbling* strategy.
- Again, fix PROM algorithm A, input x, and RO H and recall the execution trace $\text{Trace}_{A,H}(x) = \{\sigma_i, \mathbf{q}_i, \mathbf{a}_i\}_{i \in [t]}$.
- For each node $v \in [n]$, track L_v as follows.
  - Note the rounds $i \in [t]$ where L_v appears as the *input* to the RO query (e.g., $L_v \in \mathbf{q}_i$).

Ex Post Facto Pebbling

- We now describe the *ex post facto pebbling* strategy.

- Again, fix PROM algorithm A, input x, and RO H and recall the execution trace $\text{Trace}_{A,H}(x) = \{\sigma_i, \mathbf{q}_i, \mathbf{a}_i\}_{i \in [t]}$.

- For each node $v \in [n]$, track L_v as follows.
  - Note the rounds $i \in [t]$ where L_v appears as the *input* to the RO query (e.g., $L_v \in \mathbf{q}_i$).
  - Note the rounds $i \in [t]$ where L_v appears at the *output* of the RO query (e.g., $L_v \in \mathbf{a}_i$).

Ex Post Facto Pebbling

- We now describe the *ex post facto pebbling* strategy.

- Again, fix PROM algorithm A, input x, and RO H and recall the execution trace $\text{Trace}_{A,H}(x) = \{\sigma_i, \mathbf{q}_i, \mathbf{a}_i\}_{i \in [t]}$.

- For each node $v \in [n]$, track L_v as follows.
  - Note the rounds $i \in [t]$ where L_v appears as the *input* to the RO query (e.g., $L_v \in \mathbf{q}_i$).
  - Note the rounds $i \in [t]$ where L_v appears at the *output* of the RO query (e.g., $L_v \in \mathbf{a}_i$).

- Define indicator $\text{Need}(v, i) = 1$ if and only if the next time (after round i) L_v appears is an *input* to the RO; otherwise $\text{Need}(v, i) = 0$.

Ex Post Facto Pebbling

Ex Post Facto Pebbling

- Define the ex post facto pebbling as

Ex Post Facto Pebbling

- Define the ex post facto pebbling as

$$\mathbf{P} = \{P_i = \{v \colon \text{Need}(v, i) = 1\}\}_{i=1}^{t}.$$

Ex Post Facto Pebbling

- Define the ex post facto pebbling as

$$\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}.$$

- **Missing pieces**:

Ex Post Facto Pebbling

- Define the ex post facto pebbling as

$$\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}.$$

- **Missing pieces**:
  1. $\mathbf{P}$ is a valid pebbling of G, and

Ex Post Facto Pebbling

- Define the ex post facto pebbling as

$$\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}.$$

- **Missing pieces**:
  1. $\mathbf{P}$ is a valid pebbling of G, and
  2. $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.

Is Ex Post Facto Pebbling Legal?

Is Ex Post Facto Pebbling Legal?

- To show: $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$ is a legal pebbling.

Is Ex Post Facto Pebbling Legal?

- To show: $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$ is a legal pebbling.
- Define a bad event Order(v) as follows:

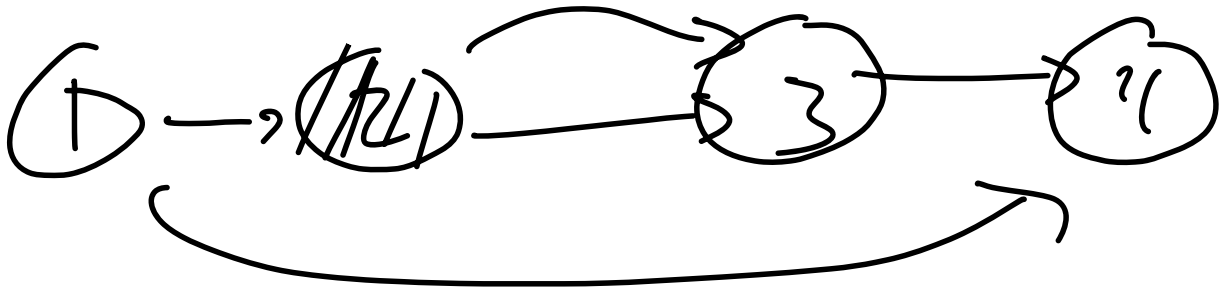
Is Ex Post Facto Pebbling Legal?

- To show: $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$ is a legal pebbling.

- Define a bad event Order(v) as follows:
  - Order(v) $= 1$ if L_v is used as a RO input *before* it has appeared as an output.

$L_1 = H(x, 0^n)$

$L_2 = H(L_1, 0^n)$



Is Ex Post Facto Pebbling Legal?

- To show: $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$ is a legal pebbling.

- Define a bad event $\text{Order}(v)$ as follows:
  - $\text{Order}(v) = 1$ if L_v is used as a RO input *before* it has appeared as an output.
  - I.e., there is some round i of the PROM computation where L_v is queried as an input and L_v does not appear as an output in any round $j < i$.

Is Ex Post Facto Pebbling Legal?

- To show: $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$ is a legal pebbling.

- Define a bad event $\text{Order}(v)$ as follows:
  - $\text{Order}(v) = 1$ if L_v is used as a RO input *before* it has appeared as an output.
  - I.e., there is some round i of the PROM computation where L_v is queried as an input and L_v does not appear as an output in any round $j < i$.

- Intuition: PROM algorithm A computing $f_{G,H}(x)$ should not be able to know/guess L_v before explicitly computing it!

Is Ex Post Facto Pebbling Legal?

- To show: $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$ is a legal pebbling.

- Define a bad event Order(v) as follows:
  - Order$(v) = 1$ if L_v is used as a RO input *before* it has appeared as an output.
  - I.e., there is some round i of the PROM computation where L_v is queried as an input and L_v does not appear as an output in any round $j < i$.

- Intuition: PROM algorithm A computing $f_{G,H}(x)$ should not be able to know/guess L_v before explicitly computing it!

- **Observation**: if $\mathsf{Order}(v) = 1$ for round i, then Need$(v, i) = 1$.

Is Ex Post Facto Pebbling Legal?

- To show: $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$ is a legal pebbling.

- Define a bad event $\text{Order}(v)$ as follows:
  - $\text{Order}(v) = 1$ if L_v is used as a RO input *before* it has appeared as an output.
  - I.e., there is some round i of the PROM computation where L_v is queried as an input and L_v does not appear as an output in any round $j < i$.

- Intuition: PROM algorithm A computing $f_{G,H}(x)$ should not be able to know/guess L_v before explicitly computing it!

- **Observation**: if $\mathsf{Order}(v) = 1$ for round i, then $\text{Need}(v, i) = 1$.
  - Moreover, this implies that P_i is *not* a valid pebbling step!

Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries.

Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \mathrm{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries. Then, $\mathbf{P}$ is a legal pebbling, except with probability $O(qn2^{-\lambda})$.

Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries. Then, $\mathbf{P}$ is a legal pebbling, except with probability $O(qn2^{-\lambda})$.

- Proof Sketch:

Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries. Then, $\mathbf{P}$ is a legal pebbling, except with probability $O(qn2^{-\lambda})$.

- Proof Sketch:
  - **Observation**: if $\mathsf{Order}(v) = 0$ for all v, then $\mathbf{P}$ is a legal pebbling by definition of $\text{Need}(v, i)$.

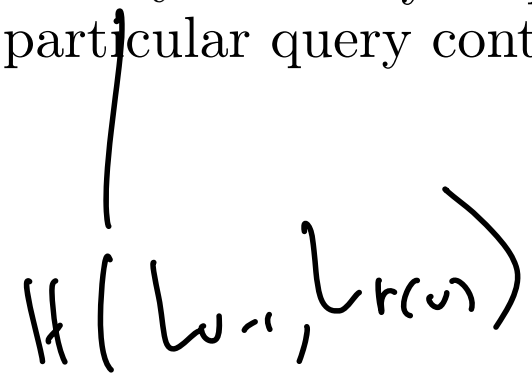
Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries. Then, $\mathbf{P}$ is a legal pebbling, except with probability $O(qn2^{-\lambda})$.

- Proof Sketch:
  - **Observation**: if $\mathsf{Order}(v) = 0$ for all v, then $\mathbf{P}$ is a legal pebbling by definition of $\text{Need}(v, i)$.
  - **Observation**: If L_v has not yet appeared as an output, then the probability a particular query contains L_v as an early input is at most $2^{-\lambda}$.



Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries. Then, $\mathbf{P}$ is a legal pebbling, except with probability $O(qn2^{-\lambda})$.

- Proof Sketch:
  - **Observation**: if $\mathsf{Order}(v) = 0$ for all v, then $\mathbf{P}$ is a legal pebbling by definition of $\text{Need}(v, i)$.
  - **Observation**: If L_v has not yet appeared as an output, then the probability a particular query contains L_v as an early input is at most $2^{-\lambda}$.
    - This implies $\Pr[\text{Order}(v)] \leqslant q2^{-\lambda}$ by a Union bound over all queries.

Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries. Then, $\mathbf{P}$ is a legal pebbling, except with probability $O(qn2^{-\lambda})$.

- Proof Sketch:
  - **Observation**: if $\mathsf{Order}(v) = 0$ for all v, then $\mathbf{P}$ is a legal pebbling by definition of $\text{Need}(v, i)$.
  - **Observation**: If L_v has not yet appeared as an output, then the probability a particular query contains L_v as an early input is at most $2^{-\lambda}$.
    - This implies $\Pr[\text{Order}(v)] \leqslant q2^{-\lambda}$ by a Union bound over all queries.
    - Here, we view L_v as a random λ-bit string before it first appears.

Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries. Then, $\mathbf{P}$ is a legal pebbling, except with probability $O(qn2^{-\lambda})$.

- Proof Sketch:
  - **Observation**: if $\mathsf{Order}(v) = 0$ for all v, then $\mathbf{P}$ is a legal pebbling by definition of $\text{Need}(v, i)$.
  - **Observation**: If L_v has not yet appeared as an output, then the probability a particular query contains L_v as an early input is at most $2^{-\lambda}$.
    - This implies $\Pr[\text{Order}(v)] \leqslant q2^{-\lambda}$ by a Union bound over all queries.
    - Here, we view L_v as a random λ-bit string before it first appears.
  - Taking a Union bound over all nodes $v \in [n]$ gives us the result.

Legal Ex Post Facto Pebbling

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

Claim 1

Let A be a PROM algorithm computing $f_{G,H}$ making at most q RO queries. Then, $\mathbf{P}$ is a legal pebbling, except with probability $O(qn2^{-\lambda})$.

- Proof Sketch:
  - **Observation**: if $\mathsf{Order}(v) = 0$ for all v, then $\mathbf{P}$ is a legal pebbling by definition of $\text{Need}(v, i)$.
  - **Observation**: If L_v has not yet appeared as an output, then the probability a particular query contains L_v as an early input is at most $2^{-\lambda}$.
    - This implies $\Pr[\text{Order}(v)] \leqslant q2^{-\lambda}$ by a Union bound over all queries.
    - Here, we view L_v as a random λ-bit string before it first appears.
  - Taking a Union bound over all nodes $v \in [n]$ gives us the result.
    - I.e., $\Pr[\exists v : \text{Order}(v)] \leqslant n \cdot \Pr[\text{Order}(1)] \leqslant nq2^{-\lambda}$.

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$
- **Observation**: $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$

- **Observation**: $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.
  - Here, $\mathsf{CC}(\mathbf{P}) = \sum_i |P_i|$.

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$

- **Observation**: $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.
  - Here, $\mathsf{CC}(\mathbf{P}) = \sum_i |P_i|$.
  - Follows by definition since $\mathsf{CC}(G)$ is the optimal pebbling.

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$ ⟵ $\text{cmc}(f_{G,H})$
- **Observation**: $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.
  - Here, $\mathsf{CC}(\mathbf{P}) = \sum_i |P_i|$.
  - Follows by definition since $\mathsf{CC}(G)$ is the optimal pebbling.

Lemma 1

For each round i, we have $|\sigma_i| + |\mathbf{a}_i| \geqslant \lambda |P_i|/2$.

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$

- **Observation**: $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.
  - Here, $\mathsf{CC}(\mathbf{P}) = \sum_i |P_i|$.
  - Follows by definition since $\mathsf{CC}(G)$ is the optimal pebbling.

Lemma 1

For each round i, we have $|\sigma_i| + |\mathbf{a}_i| \geqslant \lambda |P_i|/2$.

- Proof idea (we will not show this):

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$

- **Observation**: $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.
  - Here, $\mathsf{CC}(\mathbf{P}) = \sum_i |P_i|$.
  - Follows by definition since $\mathsf{CC}(G)$ is the optimal pebbling.

Lemma 1

For each round i, we have $|\sigma_i| + |\mathbf{a}_i| \geqslant \lambda |P_i|/2$.

- Proof idea (we will not show this):
  - Suppose not and that there is a round i^* such that $|\sigma_{i^*}| + |\mathbf{a}_{i^*}| < \lambda |P_{i^*}|/2$.

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$

- **Observation**: $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.
  - Here, $\mathsf{CC}(\mathbf{P}) = \sum_i |P_i|$.
  - Follows by definition since $\mathsf{CC}(G)$ is the optimal pebbling.

Lemma 1

For each round i, we have $|\sigma_i| + |\mathbf{a}_i| \geqslant \lambda|P_i|/2$.

- Proof idea (we will not show this):
  - Suppose not and that there is a round i^* such that $|\sigma_{i^*}| + |\mathbf{a}_{i^*}| < \lambda|P_{i^*}|/2$.
  - Then, we can build an extractor $\mathcal{E}$ which takes a hint h of size $\lambda|P_{i^*}|/2 + o(\lambda|P_{i^*}|)$ and outputs $p = |P_{i^*}|$ labels $L_{v_1}, \ldots, L_{v_p}$.

$\mathcal{E}$ extracts from RO

From Ex Post Facto Pebbling to cmc

- $\mathbf{P} = \{P_i = \{v : \text{Need}(v, i) = 1\}\}_{i=1}^{t}$
- Recall the trace with respect to A, x, and H.
  - $\text{Trace}_{A,H}(x) = \{(\sigma_i, \mathbf{q}_i, \mathbf{a}_i)\}_{i=1}^{t}$

- **Observation**: $\mathsf{CC}(\mathbf{P}) \geqslant \mathsf{CC}(G)$.
  - Here, $\mathsf{CC}(\mathbf{P}) = \sum_i |P_i|$.
  - Follows by definition since $\mathsf{CC}(G)$ is the optimal pebbling.

Lemma 1

For each round i, we have $|\sigma_i| + |\mathbf{a}_i| \geqslant \lambda|P_i|/2$.

- Proof idea (we will not show this):
  - Suppose not and that there is a round i^* such that $|\sigma_{i^*}| + |\mathbf{a}_{i^*}| < \lambda|P_{i^*}|/2$.
  - Then, we can build an extractor $\mathcal{E}$ which takes a hint h of size $\lambda|P_{i^*}|/2 + o(\lambda|P_{i^*}|)$ and outputs $p = |P_{i^*}|$ labels $L_{v_1}, \ldots, L_{v_p}$.
  - This contradicts incompressibility of the RO!

From Ex Post Facto Pebbling to cmc

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

$$\mathsf{cmc}(\mathrm{Trace}_{A,H}(x)) = \sum_{i=1}^{t}(|\sigma_i| + |\mathbf{a}_i|) \geqslant \frac{\lambda}{2}\sum_{i}^{t}|P_i|$$

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

$$\begin{aligned}\mathsf{cmc}(\mathrm{Trace}_{A,H}(x)) &= \sum_{i=1}^{t}(|\sigma_i| + |\mathbf{a}_i|) \geqslant \frac{\lambda}{2}\sum_{i}^{t}|P_i| \\ &= \frac{\lambda}{2}\mathsf{CC}(\mathbf{P}) \geqslant \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

$$\begin{aligned}\mathsf{cmc}(\mathrm{Trace}_{A,H}(x)) &= \sum_{i=1}^{t}(|\sigma_i| + |\mathbf{a}_i|) \geqslant \frac{\lambda}{2}\sum_{i}^{t}|P_i| \\ &= \frac{\lambda}{2}\mathsf{CC}(\mathbf{P}) \geqslant \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

- From our definition of cmc, we have

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

$$\begin{aligned}\mathsf{cmc}(\mathrm{Trace}_{A,H}(x)) &= \sum_{i=1}^{t}(|\sigma_i| + |\mathbf{a}_i|) \geqslant \frac{\lambda}{2}\sum_{i}^{t}|P_i| \\ &= \frac{\lambda}{2}\mathsf{CC}(\mathbf{P}) \geqslant \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

- From our definition of cmc, we have

$$\mathsf{cmc}(f_{G,H}) = \min_{A,x} \mathop{\mathbb{E}}_{H}[\mathsf{cmc}(\mathrm{Trace}_{A,H}(x))]$$

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

$$\begin{aligned}\mathsf{cmc}(\mathrm{Trace}_{A,H}(x)) &= \sum_{i=1}^{t}(|\sigma_i| + |\mathbf{a}_i|) \geqslant \frac{\lambda}{2}\sum_{i}^{t}|P_i| \\ &= \frac{\lambda}{2}\mathsf{CC}(\mathbf{P}) \geqslant \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

- From our definition of cmc, we have

$$\begin{aligned}\mathsf{cmc}(f_{G,H}) &= \min_{A,x}\mathop{\mathbb{E}}_{H}[\mathsf{cmc}(\mathrm{Trace}_{A,H}(x))] \\ &\geqslant \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\frac{\lambda}{2}\mathsf{CC}(\mathbf{P})\right] \geqslant \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\frac{\lambda}{2}\mathsf{CC}(G)\right]\end{aligned}$$

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

$$\begin{aligned}\mathsf{cmc}(\mathrm{Trace}_{A,H}(x)) &= \sum_{i=1}^{t}(|\sigma_i| + |\mathbf{a}_i|) \geqslant \frac{\lambda}{2}\sum_{i}^{t}|P_i| \\ &= \frac{\lambda}{2}\mathsf{CC}(\mathbf{P}) \geqslant \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

- From our definition of cmc, we have

$$\begin{aligned}\mathsf{cmc}(f_{G,H}) &= \min_{A,x}\mathop{\mathbb{E}}_{H}[\mathsf{cmc}(\mathrm{Trace}_{A,H}(x))] \\ &\geqslant \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\frac{\lambda}{2}\mathsf{CC}(\mathbf{P})\right] \geqslant \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\frac{\lambda}{2}\mathsf{CC}(G)\right] \\ &= \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

$$\begin{aligned}\mathsf{cmc}(\mathrm{Trace}_{A,H}(x)) &= \sum_{i=1}^{t}(|\sigma_i| + |\mathbf{a}_i|) \geqslant \frac{\lambda}{2}\sum_{i}^{t}|P_i| \\ &= \frac{\lambda}{2}\mathsf{CC}(\mathbf{P}) \geqslant \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

- From our definition of cmc, we have

$$\begin{aligned}\mathsf{cmc}(f_{G,H}) &= \min_{A,x}\mathop{\mathbb{E}}_{H}[\mathsf{cmc}(\mathrm{Trace}_{A,H}(x))] \\ &\geqslant \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\frac{\lambda}{2}\mathsf{CC}(\mathbf{P})\right] \geqslant \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\frac{\lambda}{2}\mathsf{CC}(G)\right] \\ &= \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

- This completes the proof.

From Ex Post Facto Pebbling to cmc

- By the Lemma, we have:

$$\begin{aligned}\mathsf{cmc}(\mathrm{Trace}_{A,H}(x)) &= \sum_{i=1}^{t}(|\sigma_i| + |\mathbf{a}_i|) \geqslant \frac{\lambda}{2}\sum_{i}^{t}|P_i| \\ &= \frac{\lambda}{2}\mathsf{CC}(\mathbf{P}) \geqslant \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

- From our definition of cmc, we have

$$\begin{aligned}\mathsf{cmc}(f_{G,H}) &= \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\mathsf{cmc}(\mathrm{Trace}_{A,H}(x))\right] \\ &\geqslant \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\frac{\lambda}{2}\mathsf{CC}(\mathbf{P})\right] \geqslant \min_{A,x}\mathop{\mathbb{E}}_{H}\left[\frac{\lambda}{2}\mathsf{CC}(G)\right] \\ &= \frac{\lambda}{2}\mathsf{CC}(G).\end{aligned}$$

- This completes the proof.
  - Any algorithm A_f computing $f_{G,H}$ has $\mathsf{cmc}(A_f) = \Omega(\lambda \cdot \mathsf{CC}(G))$.

Depth-Robustness: A Necessary Property

Depth-Robustness: A Necessary Property

- To conclude, we will discuss a necessary property of DAGs G for memory-hardness: *depth-robustness*.

Depth-Robustness: A Necessary Property

- To conclude, we will discuss a necessary property of DAGs G for memory-hardness: *depth-robustness*.

Definition 2

A DAG $G = (V, E)$ is (e, d)-*reducible* if there exists $S \subseteq V$ such that $|S| \leqslant e$ and $\mathsf{depth}(G \setminus S) \leqslant d$.

Depth-Robustness: A Necessary Property

- To conclude, we will discuss a necessary property of DAGs G for memory-hardness: *depth-robustness*.

Definition 2

A DAG $G = (V, E)$ is (e, d)-*reducible* if there exists $S \subseteq V$ such that $|S| \leqslant e$ and $\mathsf{depth}(G \setminus S) \leqslant d$. Otherwise, we say that G is (e, d) *depth-robust*.

Depth-Robustness: A Necessary Property

- To conclude, we will discuss a necessary property of DAGs G for memory-hardness: *depth-robustness*.

Definition 2

A DAG $G = (V, E)$ is (e, d)-*reducible* if there exists $S \subseteq V$ such that $|S| \leqslant e$ and $\mathsf{depth}(G \setminus S) \leqslant d$. Otherwise, we say that G is (e, d) *depth-robust*.

- Here, the graph $G' = G \setminus S$ is the graph obtained by removing all $s \in S$ from V and all edges of the form $(u, s) \in E$ and $(s, v) \in E$.

Depth-Robustness: A Necessary Property

- To conclude, we will discuss a necessary property of DAGs G for memory-hardness: *depth-robustness*.

Definition 2

A DAG $G = (V, E)$ is (e, d)-*reducible* if there exists $S \subseteq V$ such that $|S| \leqslant e$ and $\mathsf{depth}(G \setminus S) \leqslant d$. Otherwise, we say that G is (e, d) *depth-robust*.

- Here, the graph $G' = G \setminus S$ is the graph obtained by removing all $s \in S$ from V and all edges of the form $(u, s) \in E$ and $(s, v) \in E$.
- Example graph from before: $(1, 2)$-reducible.

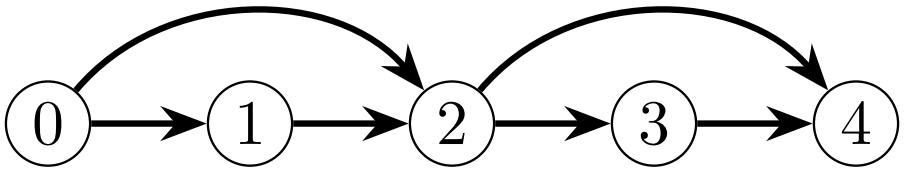
Depth-Robustness: A Necessary Property

- To conclude, we will discuss a necessary property of DAGs G for memory-hardness: *depth-robustness*.

Definition 2

A DAG $G = (V, E)$ is (e, d)-*reducible* if there exists $S \subseteq V$ such that $|S| \leqslant e$ and $\mathsf{depth}(G \setminus S) \leqslant d$. Otherwise, we say that G is (e, d) *depth-robust*.

- Here, the graph $G' = G \setminus S$ is the graph obtained by removing all $s \in S$ from V and all edges of the form $(u, s) \in E$ and $(s, v) \in E$.
- Example graph from before: $(1, 2)$-reducible.



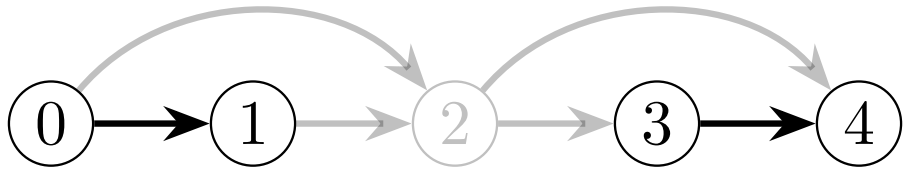
Depth-Robustness: A Necessary Property

- To conclude, we will discuss a necessary property of DAGs G for memory-hardness: *depth-robustness*.

Definition 2

A DAG $G = (V, E)$ is (e, d)-*reducible* if there exists $S \subseteq V$ such that $|S| \leqslant e$ and $\mathsf{depth}(G \setminus S) \leqslant d$. Otherwise, we say that G is (e, d) *depth-robust*.

- Here, the graph $G' = G \setminus S$ is the graph obtained by removing all $s \in S$ from V and all edges of the form $(u, s) \in E$ and $(s, v) \in E$.
- Example graph from before: $(1, 2)$-reducible.



Ideal iMHFs Don't Exist

Ideal iMHFs Don't Exist

Theorem 2 (Alwen-Blocki-Pietrzak 2017)

If G is a DAG on n nodes and is (e, d)-depth robust, then $\mathsf{CC}(G) \geqslant e \cdot d$.

Ideal iMHFs Don't Exist

Theorem 2 (Alwen-Blocki-Pietrzak 2017)

If G is a DAG on n nodes and is (e, d)-depth robust, then $\mathsf{CC}(G) \geqslant e \cdot d$.

Corollary 1

There exists a DAG G on n nodes such that $\delta(G) = O(1)$ with $\mathsf{CC}(G) \geqslant \Omega(n^2/\log(n))$.

Ideal iMHFs Don't Exist

Theorem 2 (Alwen-Blocki-Pietrzak 2017)

If G is a DAG on n nodes and is (e, d)-depth robust, then $\mathsf{CC}(G) \geqslant e \cdot d$.

Corollary 1

There exists a DAG G on n nodes such that $\delta(G) = O(1)$ with $\mathsf{CC}(G) \geqslant \Omega(n^2/\log(n))$.

Corollary 2

If G is a DAG on n nodes and is (e, d)-reducible, then $\mathsf{CC}(G) < e \cdot d$.

Ideal iMHFs Don't Exist

Ideal iMHFs Don't Exist

Theorem 3

If G is a DAG on n nodes with $\delta(G) = O(1)$, then G is

$$\left(O\left(\frac{n \operatorname{loglog}(n)}{\log(n)}\right), \frac{n}{\log^2(n)}\right) \text{-reducible.}$$

Ideal iMHFs Don't Exist

Theorem 3

If G is a DAG on n nodes with $\delta(G) = O(1)$, then G is

$$\left(O\left(\frac{n\,\mathrm{loglog}(n)}{\log(n)}\right), \frac{n}{\log^2(n)}\right)\text{-reducible.}$$

Theorem 4 (Erdős, Graham, and Szemeredi 1975)

There exists a $(\Omega(n), \Omega(n))$ depth-robust DAG G on n nodes with $\delta(G) = \not{\Theta}(\log(n))$.

Post-quantum Cryptography

Quantum Computing

Quantum Computing

- *Quantum* computers are special computers that use quantum mechanics to perform and compute operations.

Quantum Computing

- *Quantum* computers are special computers that use quantum mechanics to perform and compute operations.

- While we won't get into the specifics of quantum computers, we will discuss what types of problems are well-suited for quantum computers.

Quantum Computing

- *Quantum* computers are special computers that use quantum mechanics to perform and compute operations.
- While we won't get into the specifics of quantum computers, we will discuss what types of problems are well-suited for quantum computers.
- In a nutshell, there are certain "hard" problems on a classical computer that experience *exponential* speed-ups on a quantum computer.

Quantum Computing

- *Quantum* computers are special computers that use quantum mechanics to perform and compute operations.
- While we won't get into the specifics of quantum computers, we will discuss what types of problems are well-suited for quantum computers.
- In a nutshell, there are certain "hard" problems on a classical computer that experience *exponential* speed-ups on a quantum computer.
- This is crucial to understand because modern cryptography (i.e., security against PPT adversaries) fundamentally relies on certain problems being difficult.

Quantum Computing

- *Quantum* computers are special computers that use quantum mechanics to perform and compute operations.
- While we won't get into the specifics of quantum computers, we will discuss what types of problems are well-suited for quantum computers.
- In a nutshell, there are certain "hard" problems on a classical computer that experience *exponential* speed-ups on a quantum computer.
- This is crucial to understand because modern cryptography (i.e., security against PPT adversaries) fundamentally relies on certain problems being difficult.
  - Large-scale quantum computers can completely break certain cryptosystems!

Quantum Cryptography vs. Post-quantum Cryptography

Quantum Cryptography vs. Post-quantum Cryptography

- There are two fundamental ways that Quantum Computers can affect cryptography.

Quantum Cryptography vs. Post-quantum Cryptography

- There are two fundamental ways that Quantum Computers can affect cryptography.

- **Quantum Cryptography**: using quantum computers and mechanics to construct and deploy *new cryptosystems*.

Quantum Cryptography vs. Post-quantum Cryptography

- There are two fundamental ways that Quantum Computers can affect cryptography.

- **Quantum Cryptography**: using quantum computers and mechanics to construct and deploy *new cryptosystems*.
  - E.g., quantum key-exchange (of classical keys) is possible using entangled qbits.

Quantum Cryptography vs. Post-quantum Cryptography

- There are two fundamental ways that Quantum Computers can affect cryptography.

- **Quantum Cryptography**: using quantum computers and mechanics to construct and deploy *new cryptosystems*.
  - E.g., quantum key-exchange (of classical keys) is possible using entangled qbits.
  - Interestingly, we can prove certain quantum cryptosystems are *unconditionally secure* (without any computational assumptions).

Quantum Cryptography vs. Post-quantum Cryptography

- There are two fundamental ways that Quantum Computers can affect cryptography.

- **Quantum Cryptography**: using quantum computers and mechanics to construct and deploy *new cryptosystems*.
  - E.g., quantum key-exchange (of classical keys) is possible using entangled qbits.
  - Interestingly, we can prove certain quantum cryptosystems are *unconditionally secure* (without any computational assumptions).
  - **Extremely limited**; specialized hardware is required.

Quantum Cryptography vs. Post-quantum Cryptography

- There are two fundamental ways that Quantum Computers can affect cryptography.

- **Quantum Cryptography**: using quantum computers and mechanics to construct and deploy *new cryptosystems*.
  - E.g., quantum key-exchange (of classical keys) is possible using entangled qbits.
  - Interestingly, we can prove certain quantum cryptosystems are *unconditionally secure* (without any computational assumptions).
  - **Extremely limited**; specialized hardware is required.

- **Post-quantum Cryptography**: standard classical cryptography that is secure against *quantum attackers* (though still PPT).

Quantum Cryptography vs. Post-quantum Cryptography

- There are two fundamental ways that Quantum Computers can affect cryptography.

- **Quantum Cryptography**: using quantum computers and mechanics to construct and deploy *new cryptosystems*.
  - E.g., quantum key-exchange (of classical keys) is possible using entangled qbits.
  - Interestingly, we can prove certain quantum cryptosystems are *unconditionally secure* (without any computational assumptions).
  - **Extremely limited**; specialized hardware is required.

- **Post-quantum Cryptography**: standard classical cryptography that is secure against *quantum attackers* (though still PPT).
  - This is what we (and most of the community) focus(es) on.

Urgency of Post-quantum Crypto

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.
  - Some people do not expect such a quantum computer to exist for several decades (or longer).

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.
  - Some people do not expect such a quantum computer to exist for several decades (or longer).

- Why the urgency now?

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.
  - Some people do not expect such a quantum computer to exist for several decades (or longer).

- Why the urgency now?
  - Paranoia is a feature in cryptography!

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.
  - Some people do not expect such a quantum computer to exist for several decades (or longer).

- Why the urgency now?
  - Paranoia is a feature in cryptography! Always prepare for the worst case.

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.
  - Some people do not expect such a quantum computer to exist for several decades (or longer).

- Why the urgency now?
  - Paranoia is a feature in cryptography! Always prepare for the worst case.
  - The theoretical impact of quantum computing on crypto has been recognized since the mid 1990s.

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.
  - Some people do not expect such a quantum computer to exist for several decades (or longer).
- Why the urgency now?
  - Paranoia is a feature in cryptography! Always prepare for the worst case.
  - The theoretical impact of quantum computing on crypto has been recognized since the mid 1990s.
- Personal quantum computing is far away, but nation-state actors are much more well funded.

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.
  - Some people do not expect such a quantum computer to exist for several decades (or longer).
- Why the urgency now?
  - Paranoia is a feature in cryptography! Always prepare for the worst case.
  - The theoretical impact of quantum computing on crypto has been recognized since the mid 1990s.
- Personal quantum computing is far away, but nation-state actors are much more well funded.
  - Consensus: such powerful groups may have quantum computers in the next 10–15 years.

Urgency of Post-quantum Crypto

- Large, scalable quantum computers required to break traditional cryptosystems **do not exist today**.
  - Some people do not expect such a quantum computer to exist for several decades (or longer).
- Why the urgency now?
  - Paranoia is a feature in cryptography! Always prepare for the worst case.
  - The theoretical impact of quantum computing on crypto has been recognized since the mid 1990s.
- Personal quantum computing is far away, but nation-state actors are much more well funded.
  - Consensus: such powerful groups may have quantum computers in the next 10–15 years.
  - Urgency is to standardize PQ crypto *now* so it is ready for the (optimistic) near-term future!

Quantum Computers vs. Symmetric-Key Crypto

Quantum Computers vs. Symmetric-Key Crypto

- Quantum Computers can speed up brute-force searches over $\{0,1\}^\lambda$ from 2^λ time to $2^{\lambda/2}$ time.

Quantum Computers vs. Symmetric-Key Crypto

- Quantum Computers can speed up brute-force searches over $\{0,1\}^\lambda$ from 2^λ time to $2^{\lambda/2}$ time.

Theorem 5 (Grover Search/Grover's Algorithm)

Let $F\colon \{0,1\}^\lambda \to \{0,1\}$ *be a function.*

Quantum Computers vs. Symmetric-Key Crypto

- Quantum Computers can speed up brute-force searches over $\{0,1\}^\lambda$ from 2^λ time to $2^{\lambda/2}$ time.

Theorem 5 (Grover Search/Grover's Algorithm)

Let $F\colon \{0,1\}^\lambda \to \{0,1\}$ be a function. There exists a quantum algorithm (Grover's Algorithm) that makes $O(2^{\lambda/2})$ quantum evaluations of F and outputs $x \in \{0,1\}^\lambda$ such that $F(x) = 1$, if such an x exists.

Quantum Computers vs. Symmetric-Key Crypto

- Quantum Computers can speed up brute-force searches over $\{0,1\}^\lambda$ from 2^λ time to $2^{\lambda/2}$ time.

Theorem 5 (Grover Search/Grover's Algorithm)

Let $F\colon \{0,1\}^\lambda \to \{0,1\}$ be a function. There exists a quantum algorithm (Grover's Algorithm) that makes $O(2^{\lambda/2})$ quantum evaluations of F and outputs $x \in \{0,1\}^\lambda$ such that $F(x) = 1$, if such an x exists.

- Note: you should think of any such F as having only a sparse number of x such that $F(x) = 1$.

Quantum Computers vs. Symmetric-Key Crypto

- Quantum Computers can speed up brute-force searches over $\{0,1\}^\lambda$ from 2^λ time to $2^{\lambda/2}$ time.

Theorem 5 (Grover Search/Grover's Algorithm)

Let $F\colon \{0,1\}^\lambda \to \{0,1\}$ be a function. There exists a quantum algorithm (Grover's Algorithm) that makes $O(2^{\lambda/2})$ quantum evaluations of F and outputs $x \in \{0,1\}^\lambda$ such that $F(x) = 1$, if such an x exists.

- Note: you should think of any such F as having only a sparse number of x such that $F(x) = 1$.

Question

Quantum Computers vs. Symmetric-Key Crypto

- Quantum Computers can speed up brute-force searches over $\{0,1\}^\lambda$ from 2^λ time to $2^{\lambda/2}$ time.

Theorem 5 (Grover Search/Grover's Algorithm)

Let $F\colon \{0,1\}^\lambda \to \{0,1\}$ be a function. There exists a quantum algorithm (Grover's Algorithm) that makes $O(2^{\lambda/2})$ quantum evaluations of F and outputs $x \in \{0,1\}^\lambda$ such that $F(x) = 1$, if such an x exists.

- Note: you should think of any such F as having only a sparse number of x such that $F(x) = 1$.

Question

How do we relate Grover's Algorithm to breaking Symmetric-Key Cryptography?

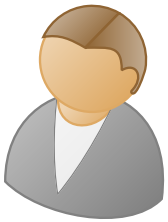
Quantum Computers vs. Symmetric-Key Crypto

Quantum Computers vs. Symmetric-Key Crypto

Alice

Quantum Computers vs. Symmetric-Key Crypto

Alice

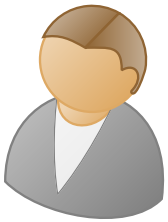


Bob

Quantum Computers vs. Symmetric-Key Crypto

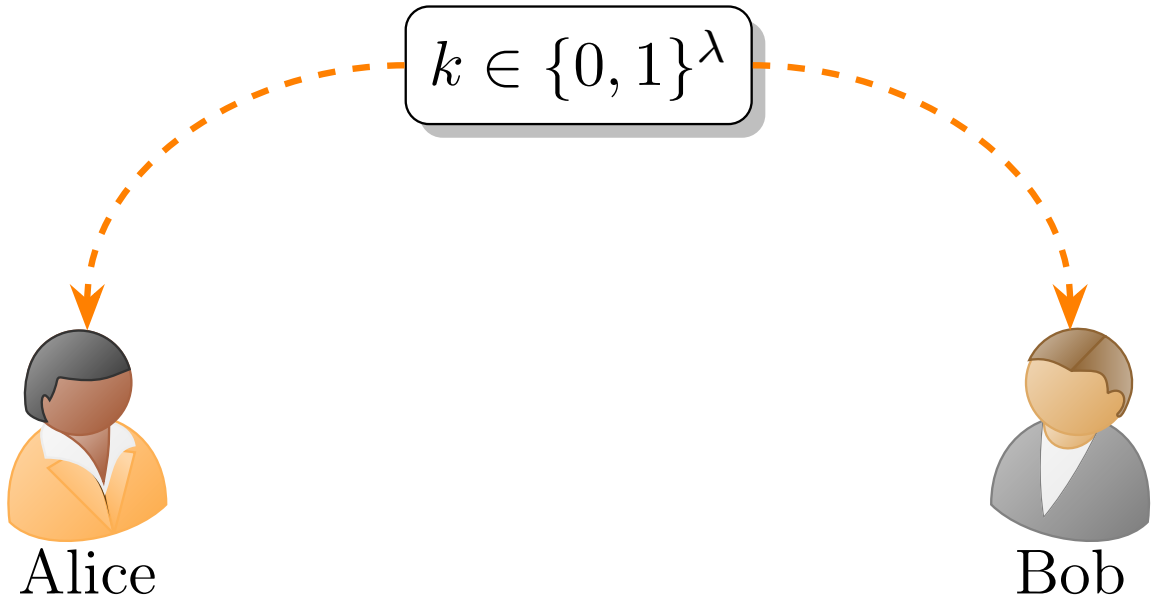
$$k \in \{0,1\}^\lambda$$

Alice

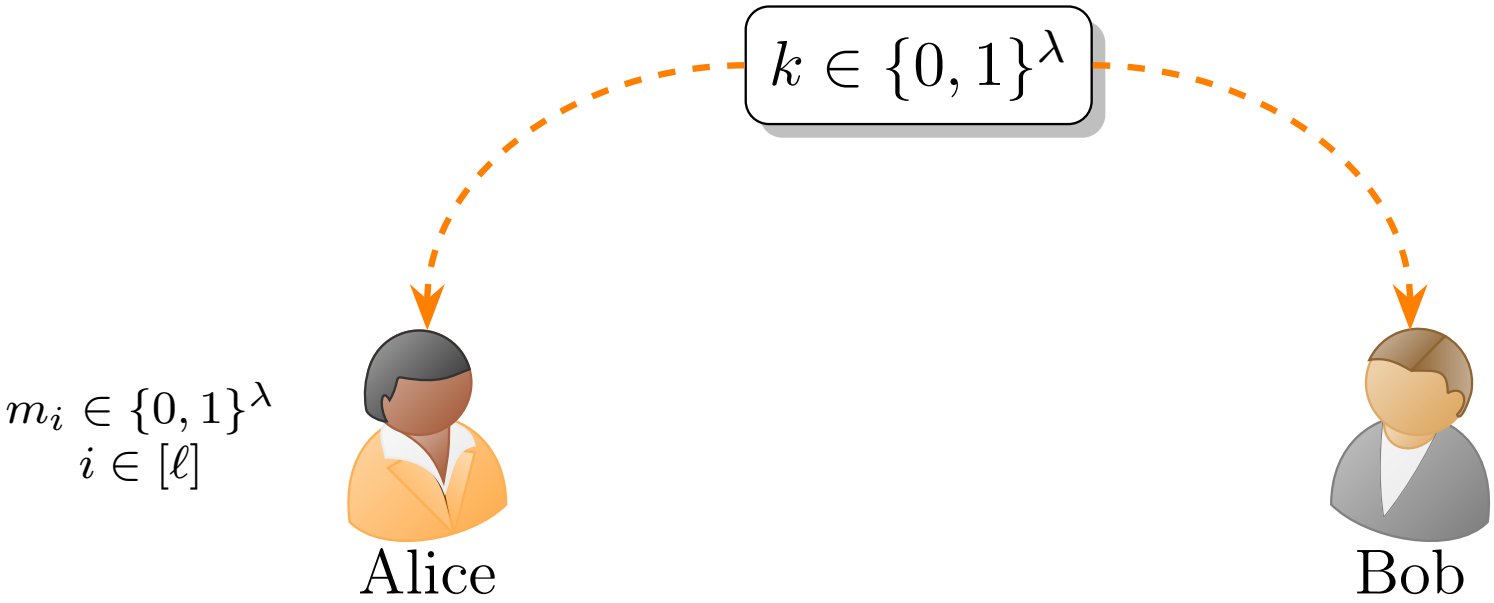


Bob

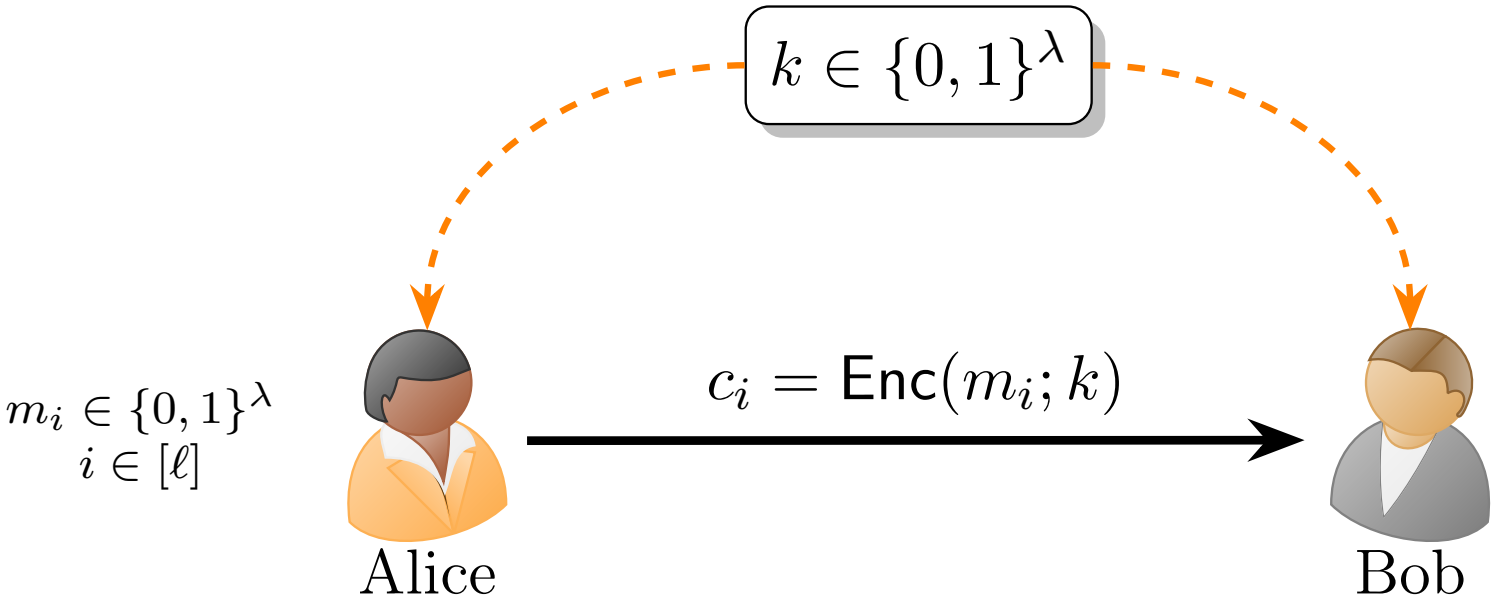
Quantum Computers vs. Symmetric-Key Crypto



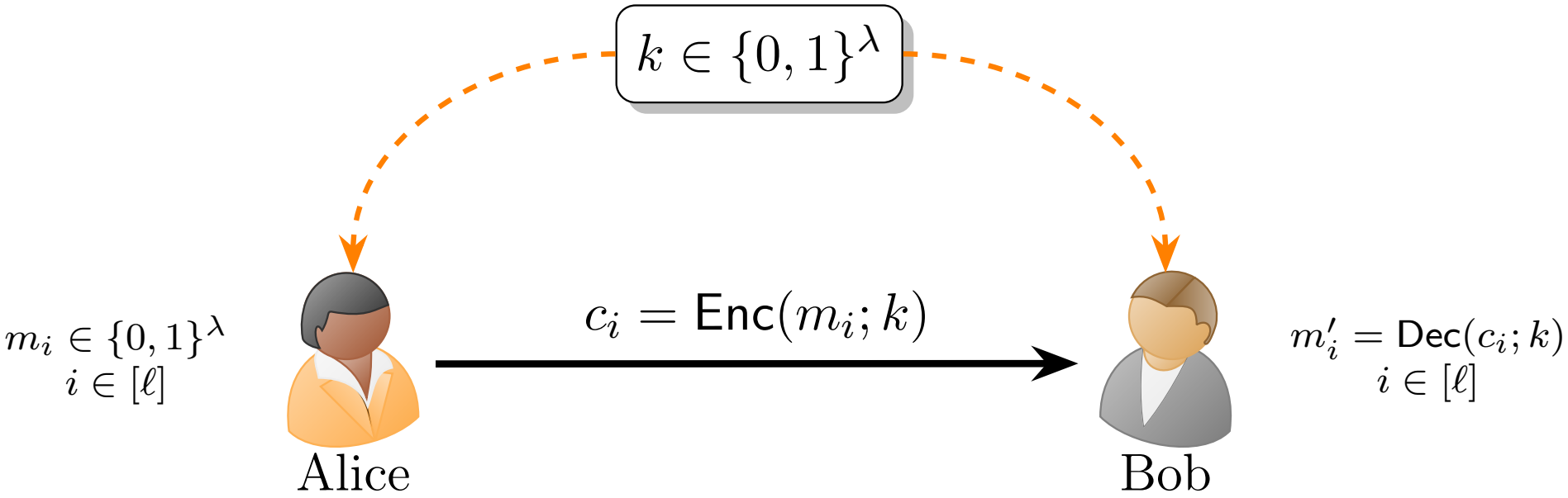
Quantum Computers vs. Symmetric-Key Crypto



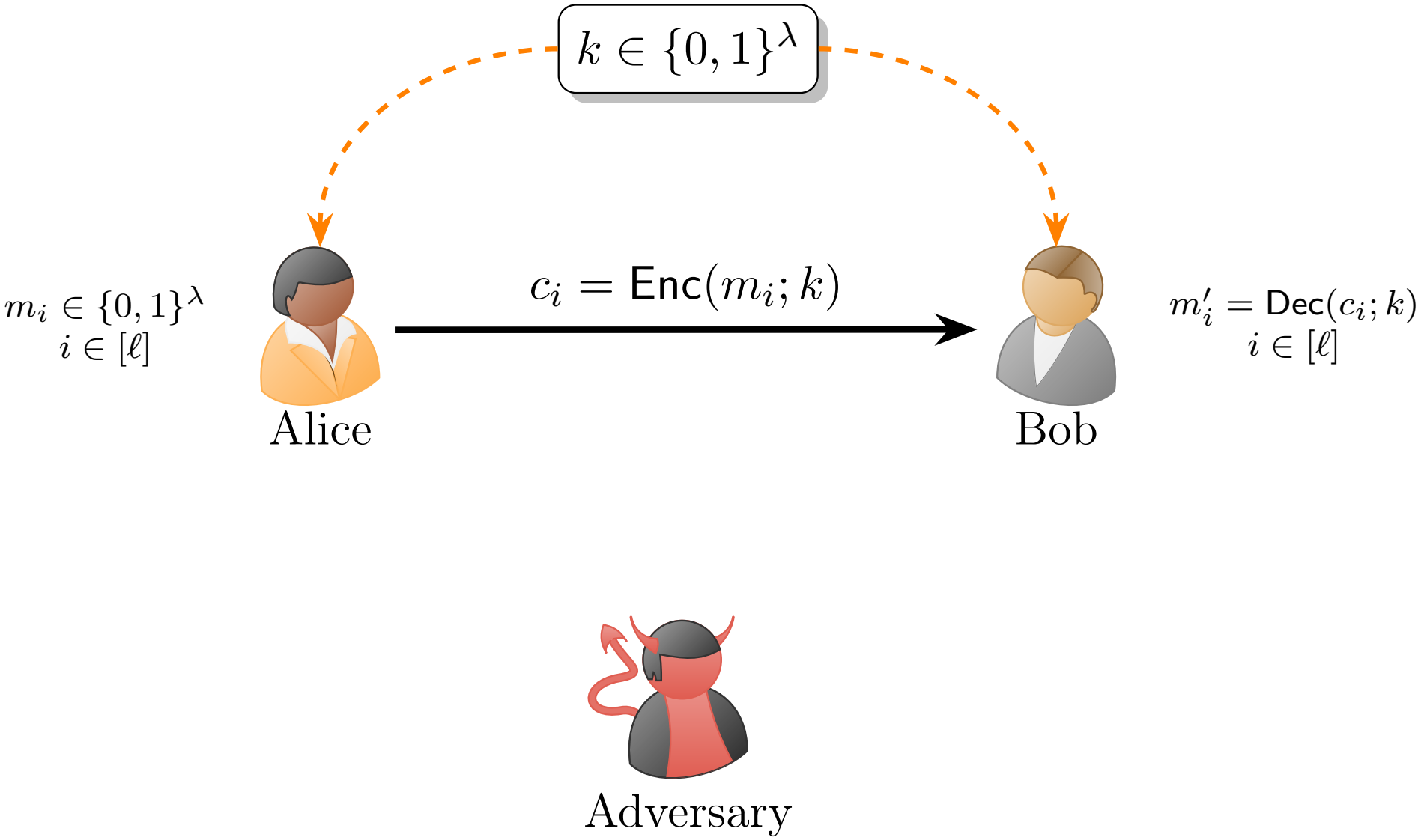
Quantum Computers vs. Symmetric-Key Crypto



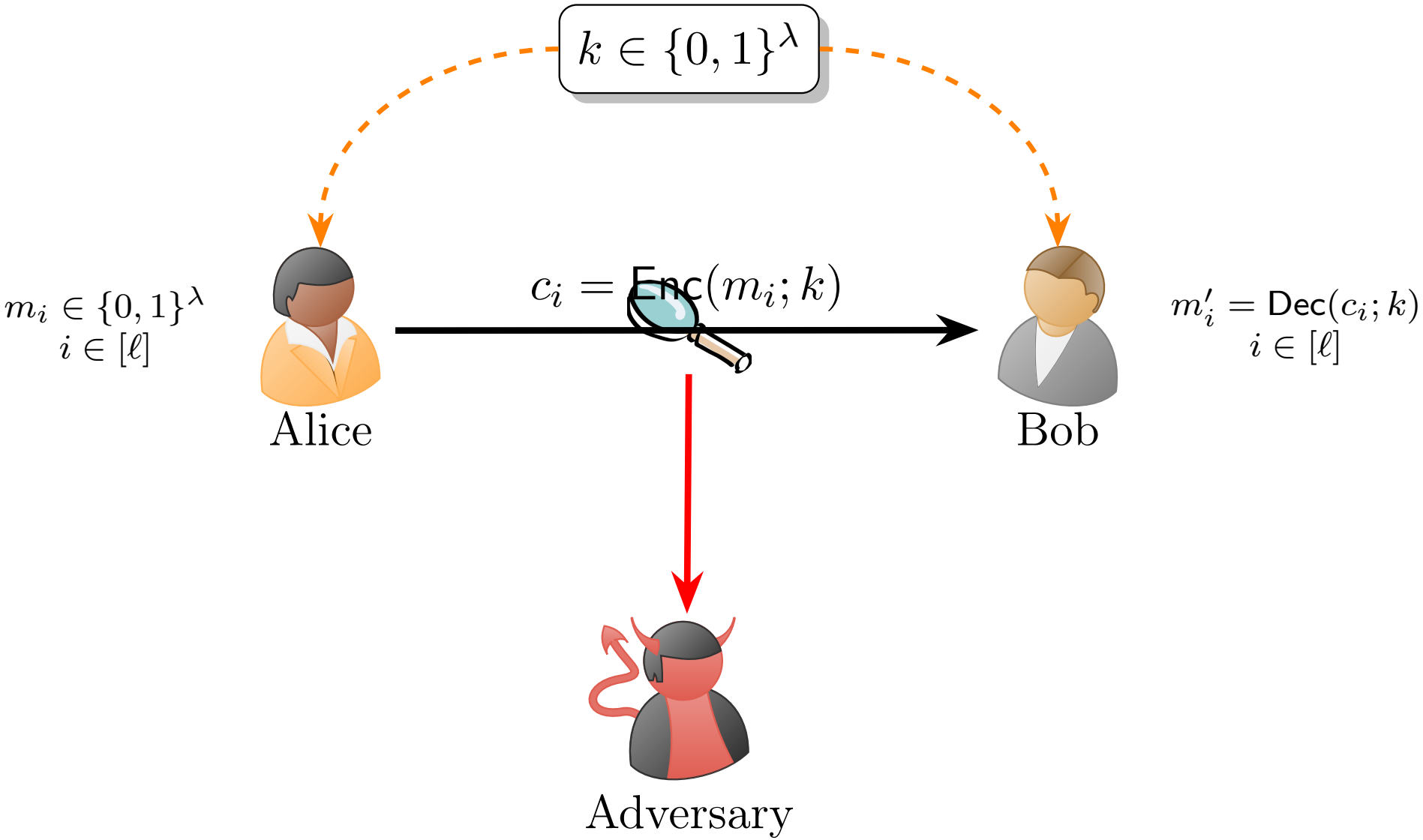
Quantum Computers vs. Symmetric-Key Crypto



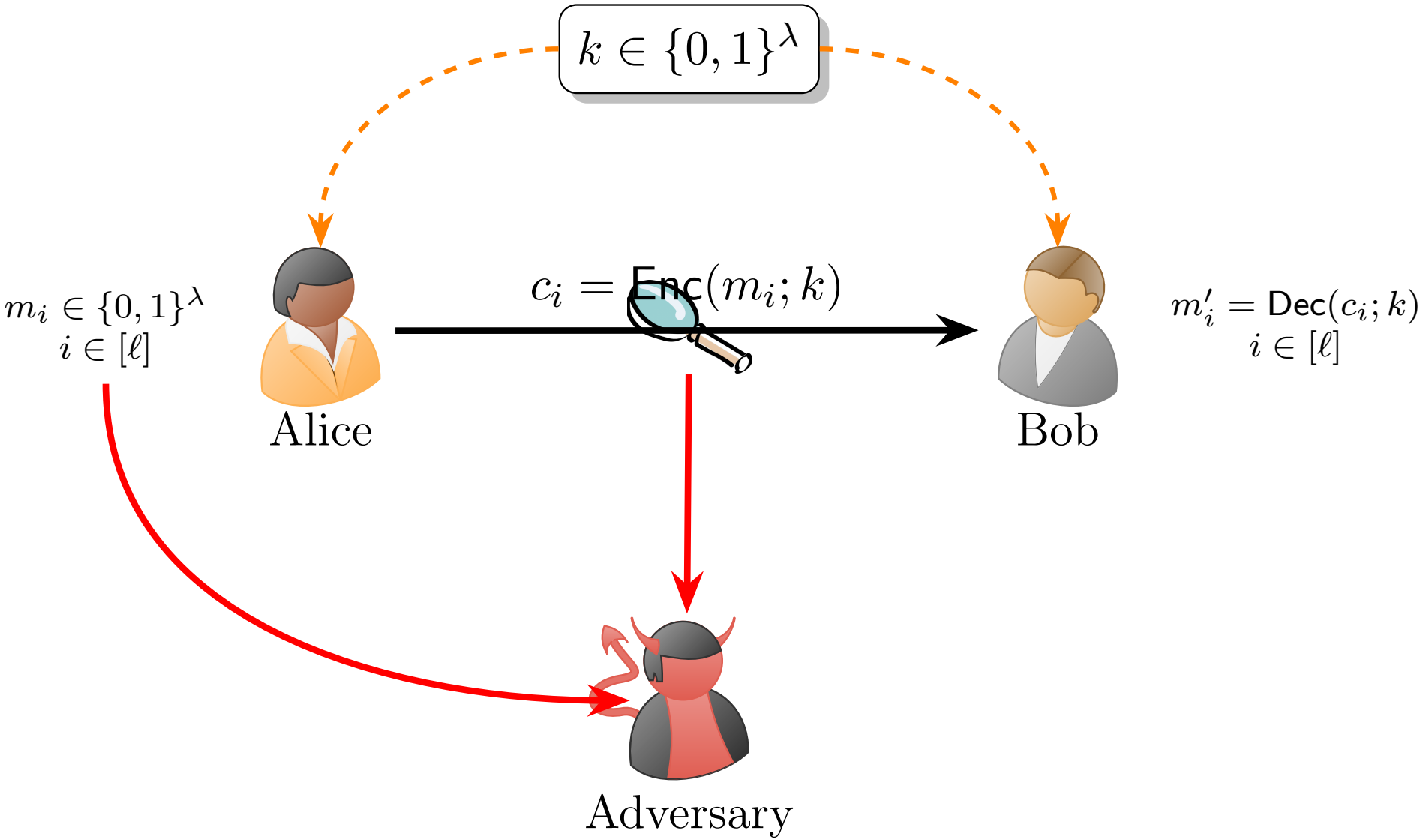
Quantum Computers vs. Symmetric-Key Crypto



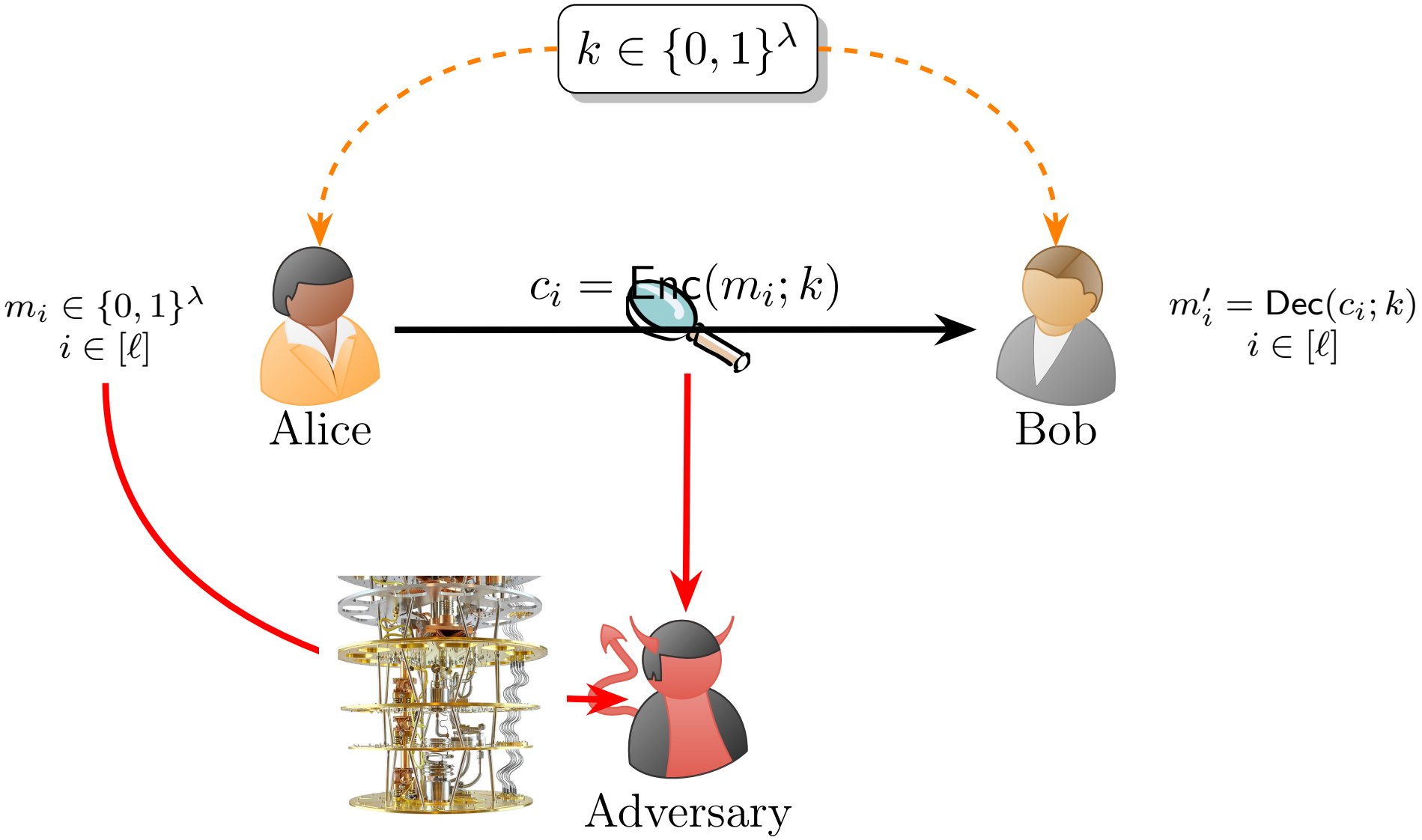
Quantum Computers vs. Symmetric-Key Crypto



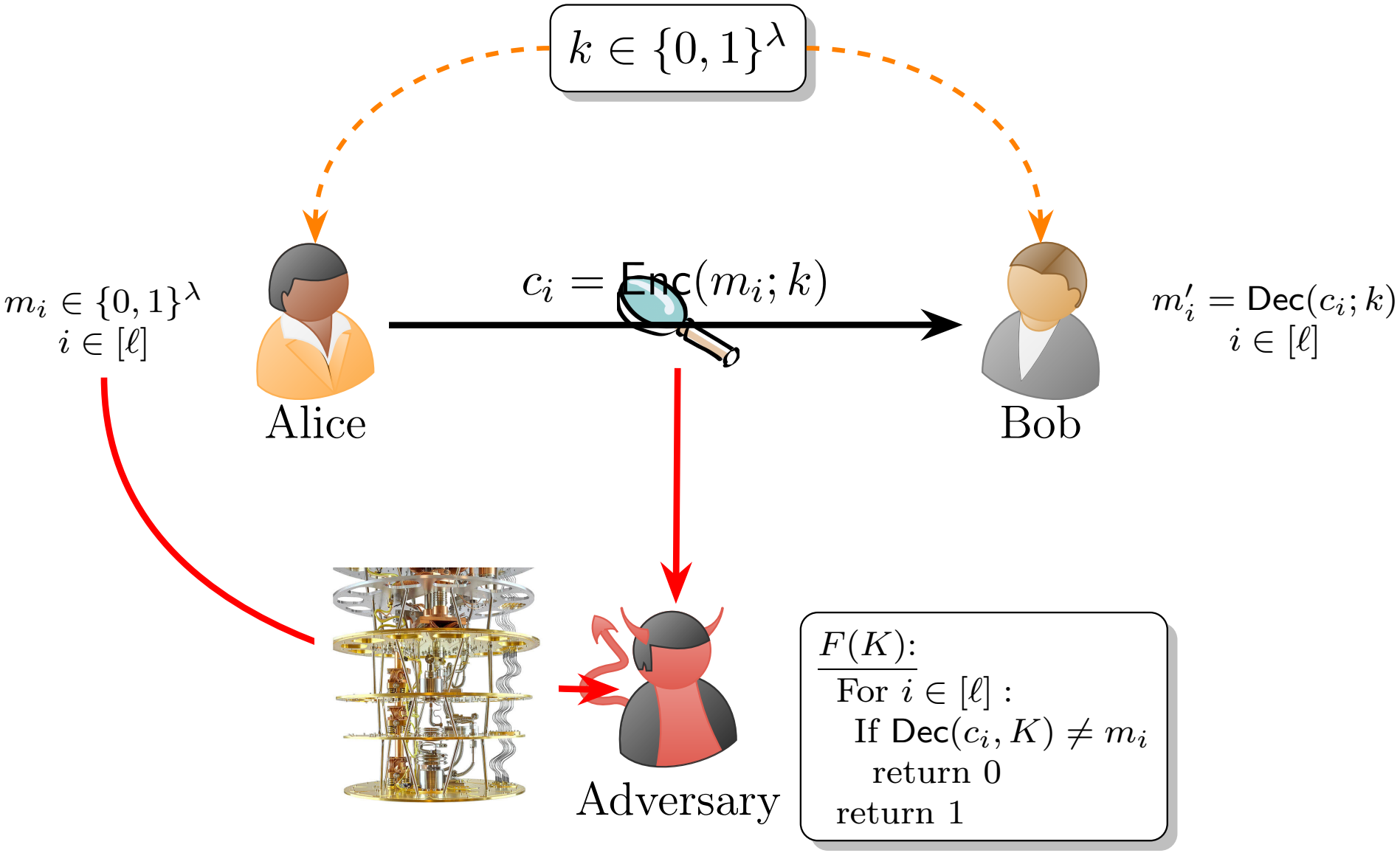
Quantum Computers vs. Symmetric-Key Crypto



Quantum Computers vs. Symmetric-Key Crypto



Quantum Computers vs. Symmetric-Key Crypto



Quantum Computers vs. Symmetric-Key Crypto

- In picture on previous slide, need ℓ large enough so that there is a unique key k such that $c_i = \mathsf{Enc}(m_i, k)$ for all k.

Quantum Computers vs. Symmetric-Key Crypto

- In picture on previous slide, need ℓ large enough so that there is a unique key k such that $c_i = \mathsf{Enc}(m_i, k)$ for all k.

- How would the adversary get both plaintexts and ciphertexts?

Quantum Computers vs. Symmetric-Key Crypto

- In picture on previous slide, need ℓ large enough so that there is a unique key k such that $c_i = \mathsf{Enc}(m_i, k)$ for all k.

- How would the adversary get both plaintexts and ciphertexts?
  - This is exactly what CPA secure encryption allows!

Quantum Computers vs. Symmetric-Key Crypto

- In picture on previous slide, need ℓ large enough so that there is a unique key k such that $c_i = \mathsf{Enc}(m_i, k)$ for all k.

- How would the adversary get both plaintexts and ciphertexts?
  - This is exactly what CPA secure encryption allows!

- For symmetric-key encryption like AES, the previous attack brute-forces the key with Grover Search in $O(2^{\lambda/2})$ time on a quantum computer.

Quantum Computers vs. Symmetric-Key Crypto

- Another application of Grover Search: collisions in a hash function $H\colon \{0,1\}^n \to \{0,1\}^\lambda$.

Quantum Computers vs. Symmetric-Key Crypto

- Another application of Grover Search: collisions in a hash function $H : \{0,1\}^n \to \{0,1\}^\lambda$.

- **Issue**: there is already an $O(2^{\lambda/2})$ time algorithm for finding collisions.

Quantum Computers vs. Symmetric-Key Crypto

- Another application of Grover Search: collisions in a hash function $H : \{0,1\}^n \to \{0,1\}^\lambda$.

- **Issue**: there is already an $O(2^{\lambda/2})$ time algorithm for finding collisions.
  - Birthday attacks!

Quantum Computers vs. Symmetric-Key Crypto

- Another application of Grover Search: collisions in a hash function $H : \{0,1\}^n \to \{0,1\}^\lambda$.

- **Issue**: there is already an $O(2^{\lambda/2})$ time algorithm for finding collisions.
  - Birthday attacks!

Question

Can we do better with Grover Search?

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!
- Let $\ell \ll 2^\lambda$ be a parameter (to be determined later).

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!
- Let $\ell \ll 2^\lambda$ be a parameter (to be determined later).
- Model $H\colon \{0,1\}^n \to \{0,1\}^\lambda$ as a random function.

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!
- Let $\ell \ll 2^\lambda$ be a parameter (to be determined later).
- Model $H\colon \{0,1\}^n \to \{0,1\}^\lambda$ as a random function.
- Let $C, D \subset \{0,1\}^n$ such that $|C| = \ell$, $|D| = \ell^2$, and $C \cap D = \emptyset$.

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!
- Let $\ell \ll 2^\lambda$ be a parameter (to be determined later).
- Model $H\colon \{0,1\}^n \to \{0,1\}^\lambda$ as a random function.
- Let $C, D \subset \{0,1\}^n$ such that $|C| = \ell$, $|D| = \ell^2$, and $C \cap D = \emptyset$.
- Define $y_i = H(x_i)$ for all $x_i \in C$, and let $C' = \{y_i\}_i$.

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!
- Let $\ell \ll 2^\lambda$ be a parameter (to be determined later).
- Model $H\colon \{0,1\}^n \to \{0,1\}^\lambda$ as a random function.
- Let $C, D \subset \{0,1\}^n$ such that $|C| = \ell$, $|D| = \ell^2$, and $C \cap D = \emptyset$.
- Define $y_i = H(x_i)$ for all $x_i \in C$, and let $C' = \{y_i\}_i$.
  - If $\exists i, j$ such that $y_i = y_j$, then we are done!

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!
- Let $\ell \ll 2^\lambda$ be a parameter (to be determined later).
- Model $H\colon \{0,1\}^n \to \{0,1\}^\lambda$ as a random function.
- Let $C, D \subset \{0,1\}^n$ such that $|C| = \ell$, $|D| = \ell^2$, and $C \cap D = \emptyset$.
- Define $y_i = H(x_i)$ for all $x_i \in C$, and let $C' = \{y_i\}_i$.
  - If $\exists i, j$ such that $y_i = y_j$, then we are done!
  - Otherwise, define $f\colon D \to \{0,1\}$ as

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!
- Let $\ell \ll 2^\lambda$ be a parameter (to be determined later).
- Model $H\colon \{0,1\}^n \to \{0,1\}^\lambda$ as a random function.
- Let $C, D \subset \{0,1\}^n$ such that $|C| = \ell$, $|D| = \ell^2$, and $C \cap D = \emptyset$.
- Define $y_i = H(x_i)$ for all $x_i \in C$, and let $C' = \{y_i\}_i$.
  - If $\exists i, j$ such that $y_i = y_j$, then we are done!
  - Otherwise, define $f\colon D \to \{0,1\}$ as

$$f(x) = 1 \iff H(x) \in C'.$$

Quantum Computers vs. Symmetric-Key Crypto

- Yes, we can do better!
- Let $\ell \ll 2^\lambda$ be a parameter (to be determined later).
- Model $H\colon \{0,1\}^n \to \{0,1\}^\lambda$ as a random function.
- Let $C, D \subset \{0,1\}^n$ such that $|C| = \ell$, $|D| = \ell^2$, and $C \cap D = \emptyset$.
- Define $y_i = H(x_i)$ for all $x_i \in C$, and let $C' = \{y_i\}_i$.
  - If $\exists i, j$ such that $y_i = y_j$, then we are done!
  - Otherwise, define $f\colon D \to \{0,1\}$ as

$$f(x) = 1 \iff H(x) \in C'.$$

else 0

- Run Grover Search on the function f above.

Quantum Computers vs. Symmetric-Key Crypto

Quantum Computers vs. Symmetric-Key Crypto

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time

Quantum Computers vs. Symmetric-Key Crypto

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).

Quantum Computers vs. Symmetric-Key Crypto

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.

Quantum Computers vs. Symmetric-Key Crypto

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?

Quantum Computers vs. Symmetric-Key Crypto

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?
  - We only run Grover search if C' contains no collisions, which means $|C'| = \ell$.

Quantum Computers vs. Symmetric-Key Crypto

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?
  - We only run Grover search if C' contains no collisions, which means $|C'| = \ell$.
  - Since H is a random function, $\Pr[H(x) \in C' \mid x \in D] = \ell \cdot 2^{-\lambda}$.

Quantum Computers vs. Symmetric-Key Crypto

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?
  - We only run Grover search if C' contains no collisions, which means $|C'| = \ell$.
  - Since H is a random function, $\Pr[H(x) \in C' \mid x \in D] = \ell \cdot 2^{-\lambda}$.
  - Equivalently, $\Pr[H(x) \notin C' \mid x \in D] = 1 - \ell \cdot 2^{-\lambda}$.

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?
  - We only run Grover search if C' contains no collisions, which means $|C'| = \ell$.
  - Since H is a random function, $\Pr[H(x) \in C' \mid x \in D] = \ell \cdot 2^{-\lambda}$.
  - Equivalently, $\Pr[H(x) \notin C' \mid x \in D] = 1 - \ell \cdot 2^{-\lambda}$.
  - Therefore:

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?
  - We only run Grover search if C' contains no collisions, which means $|C'| = \ell$.
  - Since H is a random function, $\Pr[H(x) \in C' \mid x \in D] = \ell \cdot 2^{-\lambda}$.
  - Equivalently, $\Pr[H(x) \notin C' \mid x \in D] = 1 - \ell \cdot 2^{-\lambda}$.
  - Therefore:

$$\Pr[\exists x \in D\colon H(x) \in C'] = 1 - \Pr[\forall x \in D\colon H(x) \notin C']$$

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?
  - We only run Grover search if C' contains no collisions, which means $|C'| = \ell$.
  - Since H is a random function, $\Pr[H(x) \in C' \mid x \in D] = \ell \cdot 2^{-\lambda}$.
  - Equivalently, $\Pr[H(x) \notin C' \mid x \in D] = 1 - \ell \cdot 2^{-\lambda}$.
  - Therefore:

$$\begin{aligned}\Pr[\exists x \in D\colon H(x) \in C'] &= 1 - \Pr[\forall x \in D\colon H(x) \notin C'] \\ &= 1 - (1 - \ell \cdot 2^{-\lambda})^{\ell^2}\end{aligned}$$

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?
  - We only run Grover search if C' contains no collisions, which means $|C'| = \ell$.
  - Since H is a random function, $\Pr[H(x) \in C' \mid x \in D] = \ell \cdot 2^{-\lambda}$.
  - Equivalently, $\Pr[H(x) \notin C' \mid x \in D] = 1 - \ell \cdot 2^{-\lambda}$.
  - Therefore:

$$\begin{aligned}\Pr[\exists x \in D\colon H(x) \in C'] &= 1 - \Pr[\forall x \in D\colon H(x) \notin C'] \\ &= 1 - (1 - \ell \cdot 2^{-\lambda})^{\ell^2} \\ &\geqslant 1 - e^{-\ell^3/2^{\lambda}}\end{aligned}$$

- **Analysis:** if $\exists x \in D$ such that $f(x) = 1$, Grover Search finds it in $O(\sqrt{|D|}) = O(\ell)$ time (equivalently $O(\ell)$ evaluations of H).
- Constructing C' also requires $O(\ell)$ evaluations of H, giving us $O(\ell)$ time/evaluations total.
- **Question:** what is the probability such an $x \in D$ exists?
  - We only run Grover search if C' contains no collisions, which means $|C'| = \ell$.
  - Since H is a random function, $\Pr[H(x) \in C' \mid x \in D] = \ell \cdot 2^{-\lambda}$.
  - Equivalently, $\Pr[H(x) \notin C' \mid x \in D] = 1 - \ell \cdot 2^{-\lambda}$.
  - Therefore:

$$\begin{aligned}\Pr[\exists x \in D\colon H(x) \in C'] &= 1 - \Pr[\forall x \in D\colon H(x) \notin C'] \\ &= 1 - (1 - \ell \cdot 2^{-\lambda})^{\ell^2} \\ &\geqslant 1 - e^{-\ell^3/2^\lambda}\end{aligned}$$

- Setting $\ell = \Theta(2^{\lambda/3})$ gives us a constant probability of finding a collision using only $O(2^{\lambda/3})$ evaluations of H!

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.
- However, it has important limitations to understand.

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

- However, it has important limitations to understand.
  1. Grover Search is *optimal*: no quantum algorithm can do better than $O(2^{\lambda/2})$ time to solve the same problem as Grover Search.

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

- However, it has important limitations to understand.
  1. Grover Search is *optimal*: no quantum algorithm can do better than $O(2^{\lambda/2})$ time to solve the same problem as Grover Search.
  2. Grover Search is *inherently sequential*: you cannot have a parallel quantum version of it.

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

- However, it has important limitations to understand.
  1. Grover Search is *optimal*: no quantum algorithm can do better than $O(2^{\lambda/2})$ time to solve the same problem as Grover Search.
  2. Grover Search is *inherently sequential*: you cannot have a parallel quantum version of it.
     - In contrast, classical brute-force can easily be parallelized!

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

- However, it has important limitations to understand.
  1. Grover Search is *optimal*: no quantum algorithm can do better than $O(2^{\lambda/2})$ time to solve the same problem as Grover Search.
  2. Grover Search is *inherently sequential*: you cannot have a parallel quantum version of it.
     - In contrast, classical brute-force can easily be parallelized!
  3. To counter Grover Search, one only needs to double the key-length!

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

- However, it has important limitations to understand.
  1. Grover Search is *optimal*: no quantum algorithm can do better than $O(2^{\lambda/2})$ time to solve the same problem as Grover Search.
  2. Grover Search is *inherently sequential*: you cannot have a parallel quantum version of it.
     - In contrast, classical brute-force can easily be parallelized!
  3. To counter Grover Search, one only needs to double the key-length!
     - Grover Search is still exponential time (on a quantum computer no less).

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

- However, it has important limitations to understand.
  1. Grover Search is *optimal*: no quantum algorithm can do better than $O(2^{\lambda/2})$ time to solve the same problem as Grover Search.
  2. Grover Search is *inherently sequential*: you cannot have a parallel quantum version of it.
     - In contrast, classical brute-force can easily be parallelized!
  3. To counter Grover Search, one only needs to double the key-length!
     - Grover Search is still exponential time (on a quantum computer no less).
     - Setting the key-length of, e.g., AES, to 2λ gives us λ-bits of security versus a quantum computer.

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

- However, it has important limitations to understand.
  1. Grover Search is *optimal*: no quantum algorithm can do better than $O(2^{\lambda/2})$ time to solve the same problem as Grover Search.
  2. Grover Search is *inherently sequential*: you cannot have a parallel quantum version of it.
     - In contrast, classical brute-force can easily be parallelized!
  3. To counter Grover Search, one only needs to double the key-length!
     - Grover Search is still exponential time (on a quantum computer no less).
     - Setting the key-length of, e.g., AES, to 2λ gives us λ-bits of security versus a quantum computer.
  4. Above counter only works if *brute-force search* is the best possible attack.

Limitations of Grover Search

- Grover Search seems like a very powerful tool we can use to break cryptosystems.

- However, it has important limitations to understand.
  1. Grover Search is *optimal*: no quantum algorithm can do better than $O(2^{\lambda/2})$ time to solve the same problem as Grover Search.
  2. Grover Search is *inherently sequential*: you cannot have a parallel quantum version of it.
     - In contrast, classical brute-force can easily be parallelized!
  3. To counter Grover Search, one only needs to double the key-length!
     - Grover Search is still exponential time (on a quantum computer no less).
     - Setting the key-length of, e.g., AES, to 2λ gives us λ-bits of security versus a quantum computer.
  4. Above counter only works if *brute-force search* is the best possible attack.
     - We will see shortly that there are better algorithms for certain problems!

Quantum Computers vs. Asymmetric-key Crypto

Quantum Computers vs. Asymmetric-key Crypto

- In 1994, Peter Shor gave a quantum algorithm that could break most (at the time) asymmetric-key cryptography in *polynomial time*.

Quantum Computers vs. Asymmetric-key Crypto

- In 1994, Peter Shor gave a quantum algorithm that could break most (at the time) asymmetric-key cryptography in *polynomial time*.

Theorem 6 (Shor's Algorithm)

There exists a quantum algorithm with $\text{poly}(\lambda)$ *run-time for solving the following problems:*

Quantum Computers vs. Asymmetric-key Crypto

- In 1994, Peter Shor gave a quantum algorithm that could break most (at the time) asymmetric-key cryptography in *polynomial time*.

Theorem 6 (Shor's Algorithm)

There exists a quantum algorithm with $\text{poly}(\lambda)$ *run-time for solving the following problems:*

1. *Factoring* λ*-bit numbers.*

Quantum Computers vs. Asymmetric-key Crypto

- In 1994, Peter Shor gave a quantum algorithm that could break most (at the time) asymmetric-key cryptography in *polynomial time*.

Theorem 6 (Shor's Algorithm)

There exists a quantum algorithm with $\text{poly}(\lambda)$ *run-time for solving the following problems:*

1. *Factoring* λ*-bit numbers.*
2. *Solving the discrete-log problem in any cyclic group of order* $< 2^{\lambda}$*, even elliptic curve groups.*

Quantum Computers vs. Asymmetric-key Crypto

- In 1994, Peter Shor gave a quantum algorithm that could break most (at the time) asymmetric-key cryptography in *polynomial time*.

Theorem 6 (Shor's Algorithm)

There exists a quantum algorithm with $\text{poly}(\lambda)$ *run-time for solving the following problems:*

1. *Factoring* λ*-bit numbers.*
2. *Solving the discrete-log problem in any cyclic group of order* $< 2^{\lambda}$*, even elliptic curve groups.*

- Implications: the following cryptosystems are broken by quantum computers (this list is non-exhaustive).

Quantum Computers vs. Asymmetric-key Crypto

- In 1994, Peter Shor gave a quantum algorithm that could break most (at the time) asymmetric-key cryptography in *polynomial time*.

Theorem 6 (Shor's Algorithm)

There exists a quantum algorithm with $\text{poly}(\lambda)$ *run-time for solving the following problems:*

1. *Factoring* λ*-bit numbers.*
2. *Solving the discrete-log problem in any cyclic group of order* $< 2^{\lambda}$*, even elliptic curve groups.*

- Implications: the following cryptosystems are broken by quantum computers (this list is non-exhaustive).
  - **Factoring-based:** RSA encryption and signatures.

Quantum Computers vs. Asymmetric-key Crypto

- In 1994, Peter Shor gave a quantum algorithm that could break most (at the time) asymmetric-key cryptography in *polynomial time*.

Theorem 6 (Shor's Algorithm)

There exists a quantum algorithm with $\text{poly}(\lambda)$ *run-time for solving the following problems:*

1. *Factoring* λ*-bit numbers.*
2. *Solving the discrete-log problem in any cyclic group of order* $< 2^{\lambda}$*, even elliptic curve groups.*

- Implications: the following cryptosystems are broken by quantum computers (this list is non-exhaustive).
  - **Factoring-based:** RSA encryption and signatures.
  - **DLog-based:** DDH and El-Gamal encryption, Schnorr signatures.

Shor's Algorithm: High-level Overview

- We give a high-level overview of Shor's algorithm and how it can be used to break some cryptosystems.

Shor's Algorithm: High-level Overview

- We give a high-level overview of Shor's algorithm and how it can be used to break some cryptosystems.
- We begin by discussing an abstract math problem that does not have an explicit connection to cryptography.

Shor's Algorithm: High-level Overview

- We give a high-level overview of Shor's algorithm and how it can be used to break some cryptosystems.
- We begin by discussing an abstract math problem that does not have an explicit connection to cryptography.

Period-Finding Problem

Shor's Algorithm: High-level Overview

- We give a high-level overview of Shor's algorithm and how it can be used to break some cryptosystems.
- We begin by discussing an abstract math problem that does not have an explicit connection to cryptography.

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.

Shor's Algorithm: High-level Overview

- We give a high-level overview of Shor's algorithm and how it can be used to break some cryptosystems.
- We begin by discussing an abstract math problem that does not have an explicit connection to cryptography.

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.

Shor's Algorithm: High-level Overview

- We give a high-level overview of Shor's algorithm and how it can be used to break some cryptosystems.
- We begin by discussing an abstract math problem that does not have an explicit connection to cryptography.

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.
  - $\exists \delta \in \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ such that $\forall x \in \mathbb{G}$:

Shor's Algorithm: High-level Overview

- We give a high-level overview of Shor's algorithm and how it can be used to break some cryptosystems.
- We begin by discussing an abstract math problem that does not have an explicit connection to cryptography.

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.
  - $\exists \delta \in \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ such that $\forall x \in \mathbb{G}$:

$$f(x) = f(x + \delta).$$

Shor's Algorithm: High-level Overview

- We give a high-level overview of Shor's algorithm and how it can be used to break some cryptosystems.
- We begin by discussing an abstract math problem that does not have an explicit connection to cryptography.

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.
  - $\exists \delta \in \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ such that $\forall x \in \mathbb{G}$:

$$f(x) = f(x + \delta).$$

- The *period-finding problem* is: find δ given *oracle access* to f.

Shor's Algorithm: High-level Overview

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.
  - $\exists \delta \in \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ such that $\forall x \in \mathbb{G}$:

$$f(x) = f(x + \delta).$$

- The *period-finding problem* is: find δ given *oracle access* to f.

Shor's Algorithm: High-level Overview

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.
  - $\exists \delta \in \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ such that $\forall x \in \mathbb{G}$:

$$f(x) = f(x + \delta).$$

- The *period-finding problem* is: find δ given *oracle access* to f.

- Classically, it is unclear how to solve this problem efficiently.

Shor's Algorithm: High-level Overview

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.
  - $\exists \delta \in \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ such that $\forall x \in \mathbb{G}$:

$$f(x) = f(x + \delta).$$

- The *period-finding problem* is: find δ given *oracle access* to f.

- Classically, it is unclear how to solve this problem efficiently.
  - Seems difficult *given* δ to verify that δ is a period of f given oracle access to f.

Shor's Algorithm: High-level Overview

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.
  - $\exists \delta \in \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ such that $\forall x \in \mathbb{G}$:

$$f(x) = f(x + \delta).$$

- The *period-finding problem* is: find δ given *oracle access* to f.

- Classically, it is unclear how to solve this problem efficiently.
  - Seems difficult *given* δ to verify that δ is a period of f given oracle access to f.

- Shor's Algorithm is a quantum algorithm that solves this problem in *polynomial time* for certain groups $\mathbb{G}$.

Shor's Algorithm: High-level Overview

Period-Finding Problem

- Let $f : \mathbb{G} \to R$ be an Abelian group with arbitrary co-domain R.
- Assume that f is *periodic*.
  - $\exists \delta \in \mathbb{G} \setminus \{0_{\mathbb{G}}\}$ such that $\forall x \in \mathbb{G}$:

$$f(x) = f(x + \delta).$$

- The *period-finding problem* is: find δ given *oracle access* to f.

- Classically, it is unclear how to solve this problem efficiently.
  - Seems difficult *given* δ to verify that δ is a period of f given oracle access to f.

- Shor's Algorithm is a quantum algorithm that solves this problem in *polynomial time* for certain groups $\mathbb{G}$.
  - Extended/generalized to other groups by subsequent work.

Shor's Algorithm: Breaking Factoring

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!
- First, we consider the factoring problem.

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!
- First, we consider the factoring problem.
- Let $N = pq$ for two sufficiently large primes $p \neq q$.

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!
- First, we consider the factoring problem.
- Let $N = pq$ for two sufficiently large primes $p \neq q$.
- Take any $x \in \mathbb{Z}_N^\times$ and define $f_{x,N} \colon \mathbb{Z} \to \mathbb{Z}_N^\times$ as

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!
- First, we consider the factoring problem.
- Let $N = pq$ for two sufficiently large primes $p \neq q$.
- Take any $x \in \mathbb{Z}_N^\times$ and define $f_{x,N} \colon \mathbb{Z} \to \mathbb{Z}_N^\times$ as

$$f_{x,N}(r) = x^r \bmod N.$$

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!
- First, we consider the factoring problem.
- Let $N = pq$ for two sufficiently large primes $p \neq q$.
- Take any $x \in \mathbb{Z}_N^\times$ and define $f_{x,N} \colon \mathbb{Z} \to \mathbb{Z}_N^\times$ as

$$f_{x,N}(r) = x^r \bmod N.$$

- **Key Observation:** f is periodic with period $\delta = \phi(N) = (p-1)(q-1)$.

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!
- First, we consider the factoring problem.
- Let $N = pq$ for two sufficiently large primes $p \neq q$.
- Take any $x \in \mathbb{Z}_N^\times$ and define $f_{x,N} \colon \mathbb{Z} \to \mathbb{Z}_N^\times$ as

$$f_{x,N}(r) = x^r \bmod N.$$

- **Key Observation:** f is periodic with period $\delta = \phi(N) = (p-1)(q-1)$.

$$f_{x,N}(r + \phi(N)) = x^{r+\phi(N)} \bmod N = x^r \cdot x^{\phi(N)} \bmod N$$

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!
- First, we consider the factoring problem.
- Let $N = pq$ for two sufficiently large primes $p \neq q$.
- Take any $x \in \mathbb{Z}_N^\times$ and define $f_{x,N} \colon \mathbb{Z} \to \mathbb{Z}_N^\times$ as

$$f_{x,N}(r) = x^r \bmod N.$$

- **Key Observation:** f is periodic with period $\delta = \phi(N) = (p-1)(q-1)$.

$$\begin{aligned} f_{x,N}(r + \phi(N)) &= x^{r+\phi(N)} \bmod N = x^r \cdot x^{\phi(N)} \bmod N \\ &= x^r \bmod N = f_{x,N}(r). \end{aligned}$$

Shor's Algorithm: Breaking Factoring

- Let's see how this period finding problem can be used to break crypto!
- First, we consider the factoring problem.
- Let $N = pq$ for two sufficiently large primes $p \neq q$.
- Take any $x \in \mathbb{Z}_N^\times$ and define $f_{x,N} \colon \mathbb{Z} \to \mathbb{Z}_N^\times$ as

$$f_{x,N}(r) = x^r \bmod N.$$

- **Key Observation:** f is periodic with period $\delta = \phi(N) = (p-1)(q-1)$.

$$\begin{aligned} f_{x,N}(r + \phi(N)) &= x^{r+\phi(N)} \bmod N = x^r \cdot x^{\phi(N)} \bmod N \\ &= x^r \bmod N = f_{x,N}(r). \end{aligned}$$

- Use Shor's Algorithm to find this period δ, can be used to factor N in *classical* polynomial time since $x^\delta = 1 \bmod N$.

Shor's Algorithm: Breaking Discrete-log

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.
- Our goal is to find some k such that $g^k = h$. $\log_g(h) = k$

$k \in \mathbb{Z}_q$

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.
- Our goal is to find some k such that $g^k = h$.
- Define $f_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ as

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.
- Our goal is to find some k such that $g^k = h$.
- Define $f_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ as

$$f_{g,h}(a, b) = g^a \cdot h^{-b}.$$

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.
- Our goal is to find some k such that $g^k = h$.
- Define $f_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ as

$$f_{g,h}(a, b) = g^a \cdot h^{-b}.$$

- **Key Observation:** f has period $(x, 1)$ for $x = \log_g(h)$ since

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.
- Our goal is to find some k such that $g^k = h$.
- Define $f_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ as

$$f_{g,h}(a, b) = g^a \cdot h^{-b}.$$

- **Key Observation:** f has period $(x, 1)$ for $x = \log_g(h)$ since

$$f_{g,h}(a + x, b + 1) = g^{a+x} \cdot h^{-b-1} = g^a g^x h^{-b} h^{-1}$$

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.
- Our goal is to find some k such that $g^k = h$.
- Define $f_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ as

$$f_{g,h}(a, b) = g^a \cdot h^{-b}.$$

- **Key Observation:** f has period $(x, 1)$ for $x = \log_g(h)$ since

$$\begin{aligned} f_{g,h}(a + x, b + 1) &= g^{a+x} \cdot h^{-b-1} = g^a g^x h^{-b} h^{-1} \\ &= g^a \cdot h \cdot h^{-1} \cdot h^{-b} = g^a h^{-b} \end{aligned}$$

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.
- Our goal is to find some k such that $g^k = h$.
- Define $f_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ as

$$f_{g,h}(a, b) = g^a \cdot h^{-b}.$$

- **Key Observation:** f has period $(x, 1)$ for $x = \log_g(h)$ since

$$\begin{aligned} f_{g,h}(a + x, b + 1) &= g^{a+x} \cdot h^{-b-1} = g^a g^x h^{-b} h^{-1} \\ &= g^a \cdot h \cdot h^{-1} \cdot h^{-b} = g^a h^{-b} \\ &= f_{g,h}(a, b). \end{aligned}$$

Shor's Algorithm: Breaking Discrete-log

- We also can break discrete-log via period finding.
- Let $\mathbb{G}$ be a cyclic group of order q and generator g.
- Suppose we are given some $h \in \mathbb{G}$.
- Our goal is to find some k such that $g^k = h$.
- Define $f_{g,h} \colon \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ as

$$f_{g,h}(a,b) = g^a \cdot h^{-b}.$$

- **Key Observation:** f has period $(x,1)$ for $x = \log_g(h)$ since

$$\begin{aligned} f_{g,h}(a+x, b+1) &= g^{a+x} \cdot h^{-b-1} = g^a g^x h^{-b} h^{-1} \\ &= g^a \cdot h \cdot h^{-1} \cdot h^{-b} = g^a h^{-b} \\ &= f_{g,h}(a,b). \end{aligned}$$

- Can use Shor's Algorithm to find period $(\delta, 1)$, can be used to compute $\log_g(h)$ in *classical* polynomial-time since $f_{g,h}(\delta, 1) = 1_{\mathbb{G}}$.

Next Time: Post-quantum Crypto Assumptions and Constructions