CS 594 – Advanced Crypto (Spring 2026)

Alex Block

Lecture 3

January 26, 2026

Practice with Security Definitions

Practice with Security Definitions

Practice with Security Definitions

- Can you make a MAC secure against replay attacks?

Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?

Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?


Alice


Bob

Practice with Security Definitions

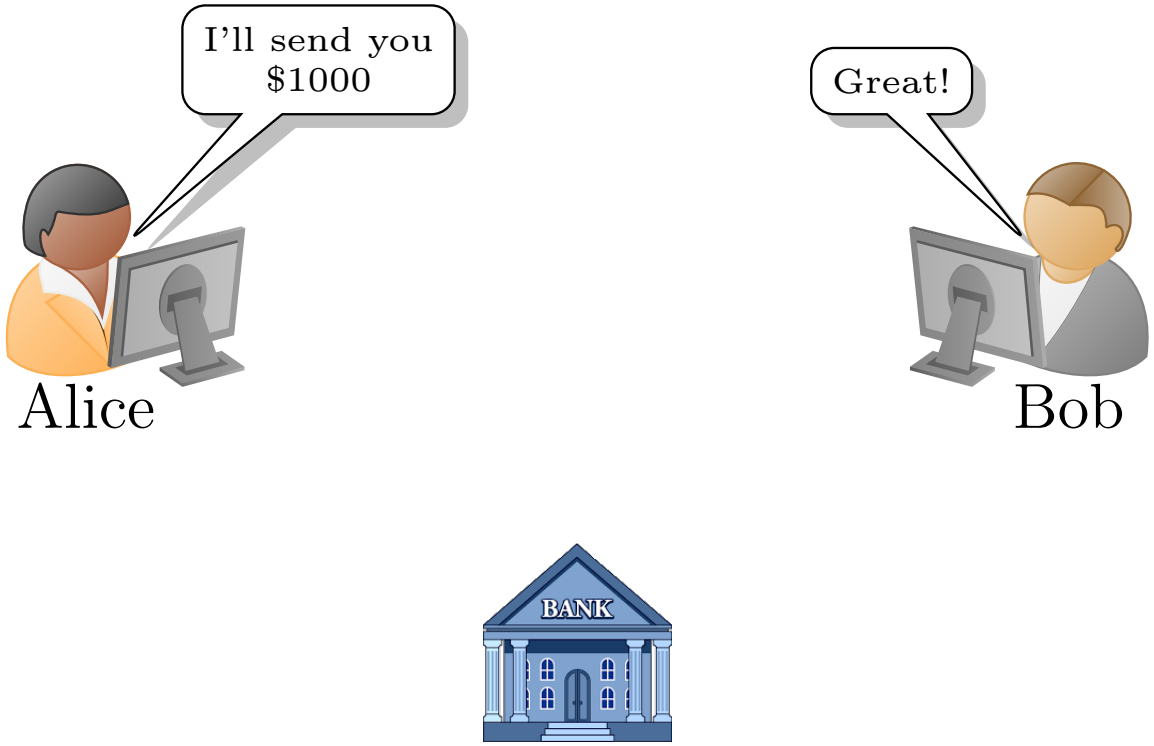
- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?

Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?

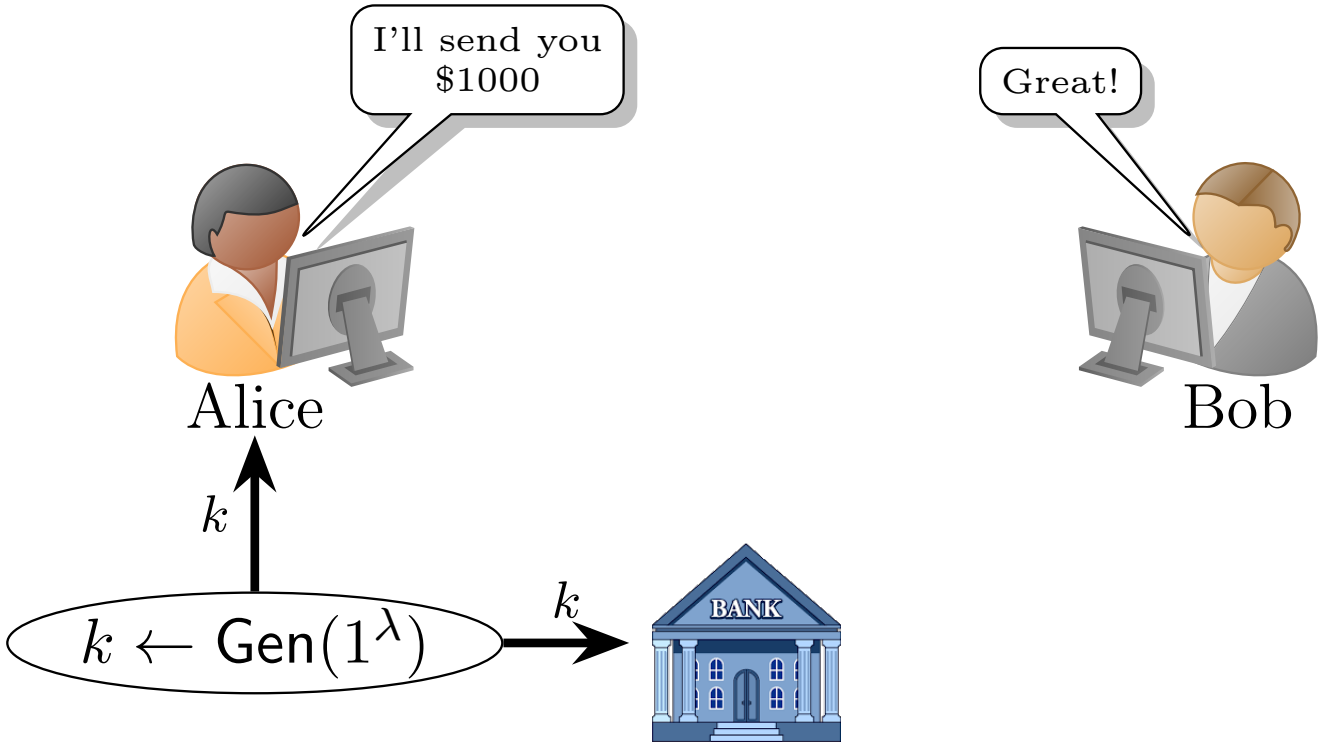
Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



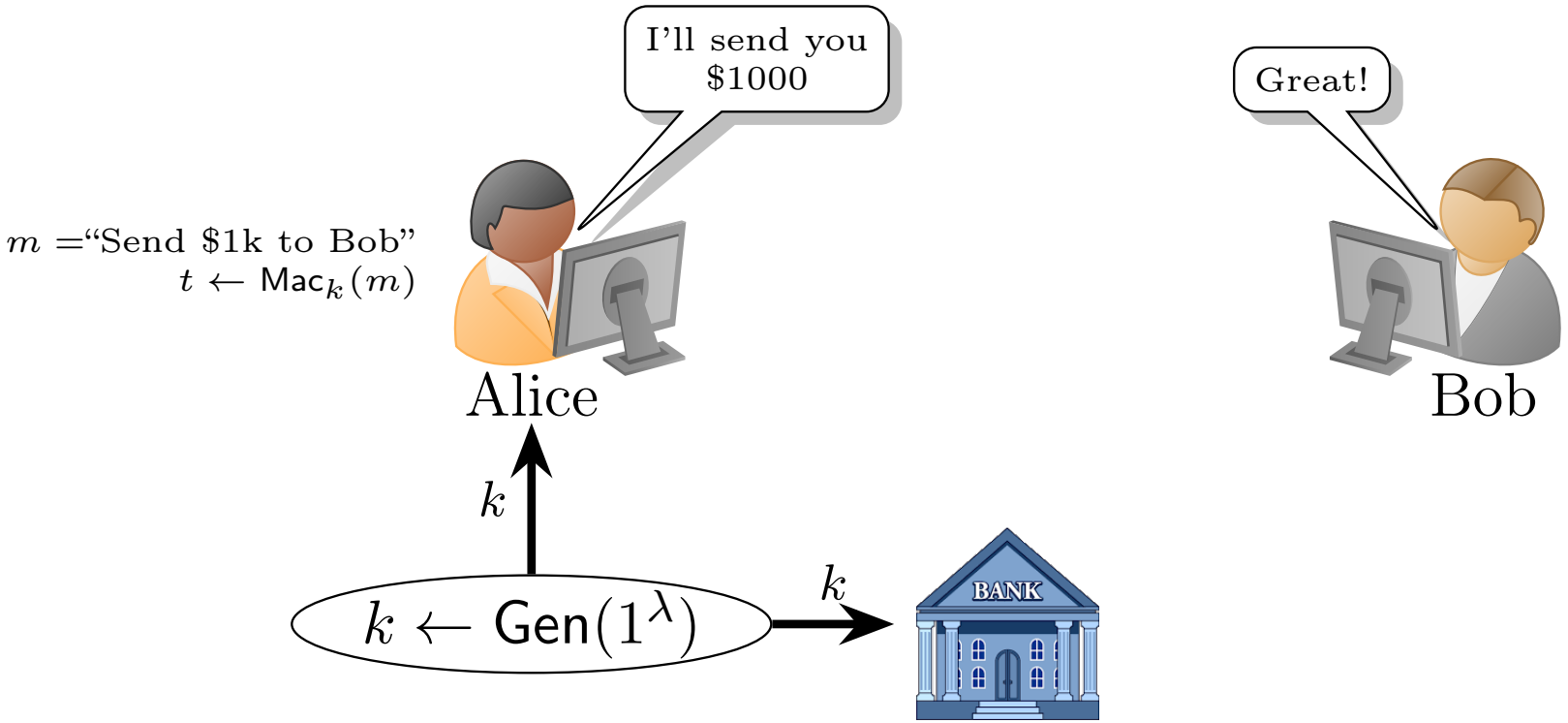
Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



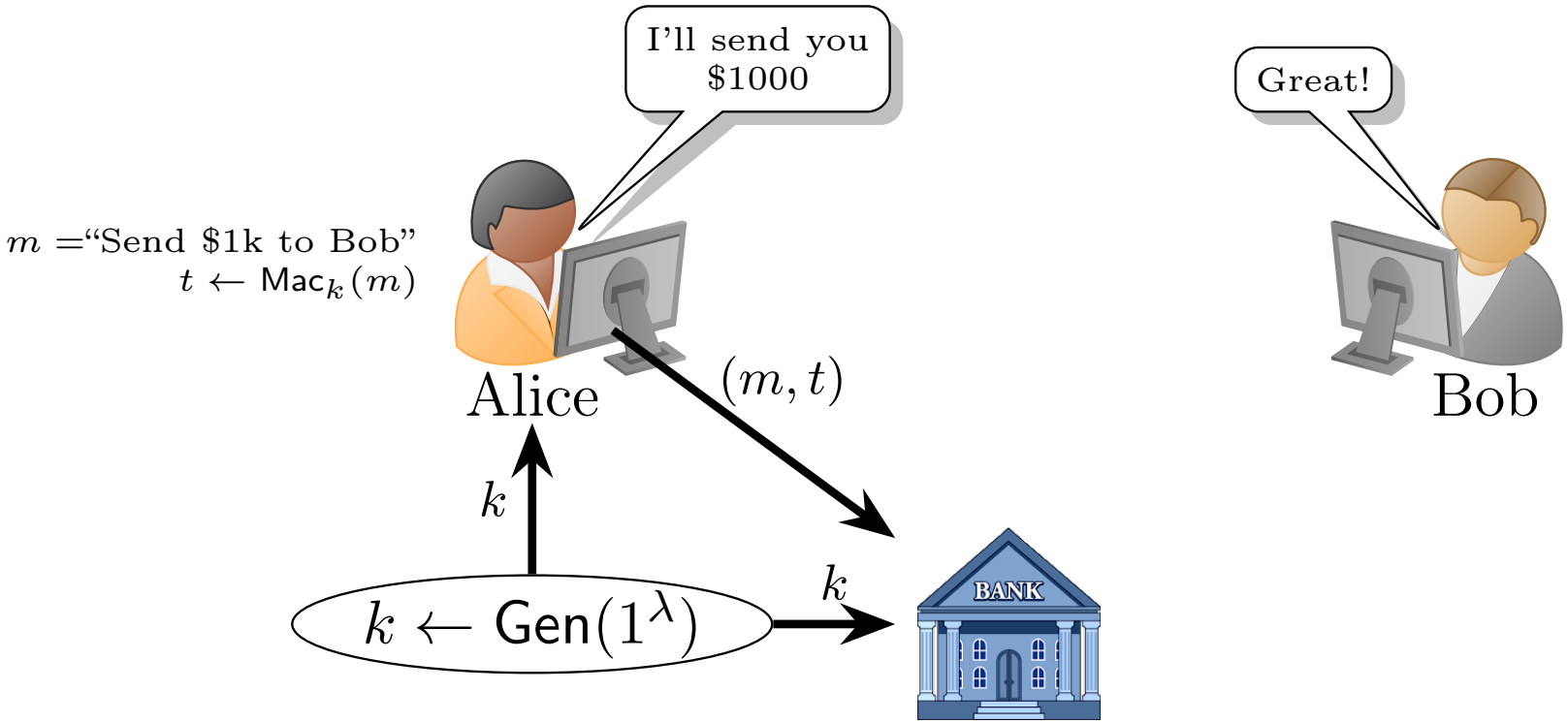
Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



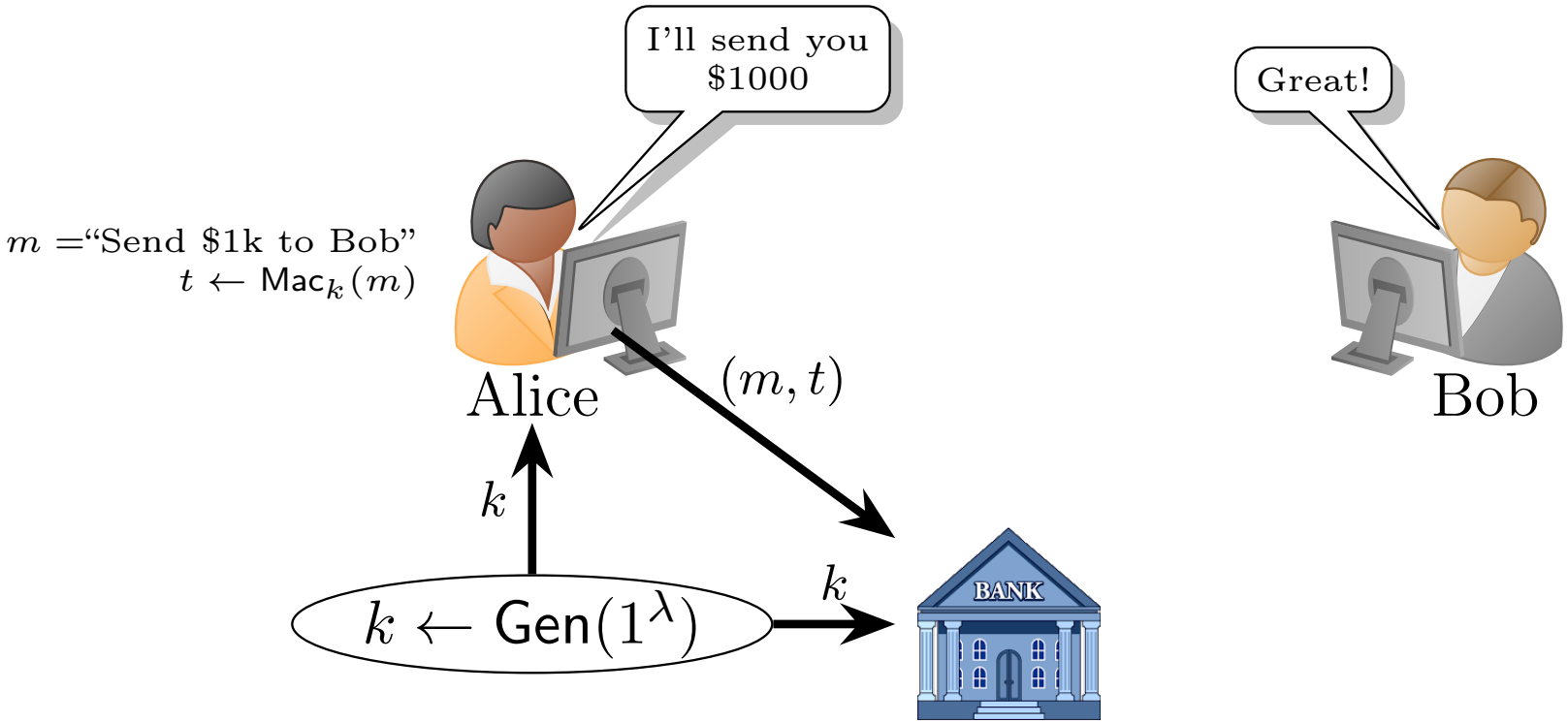
Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



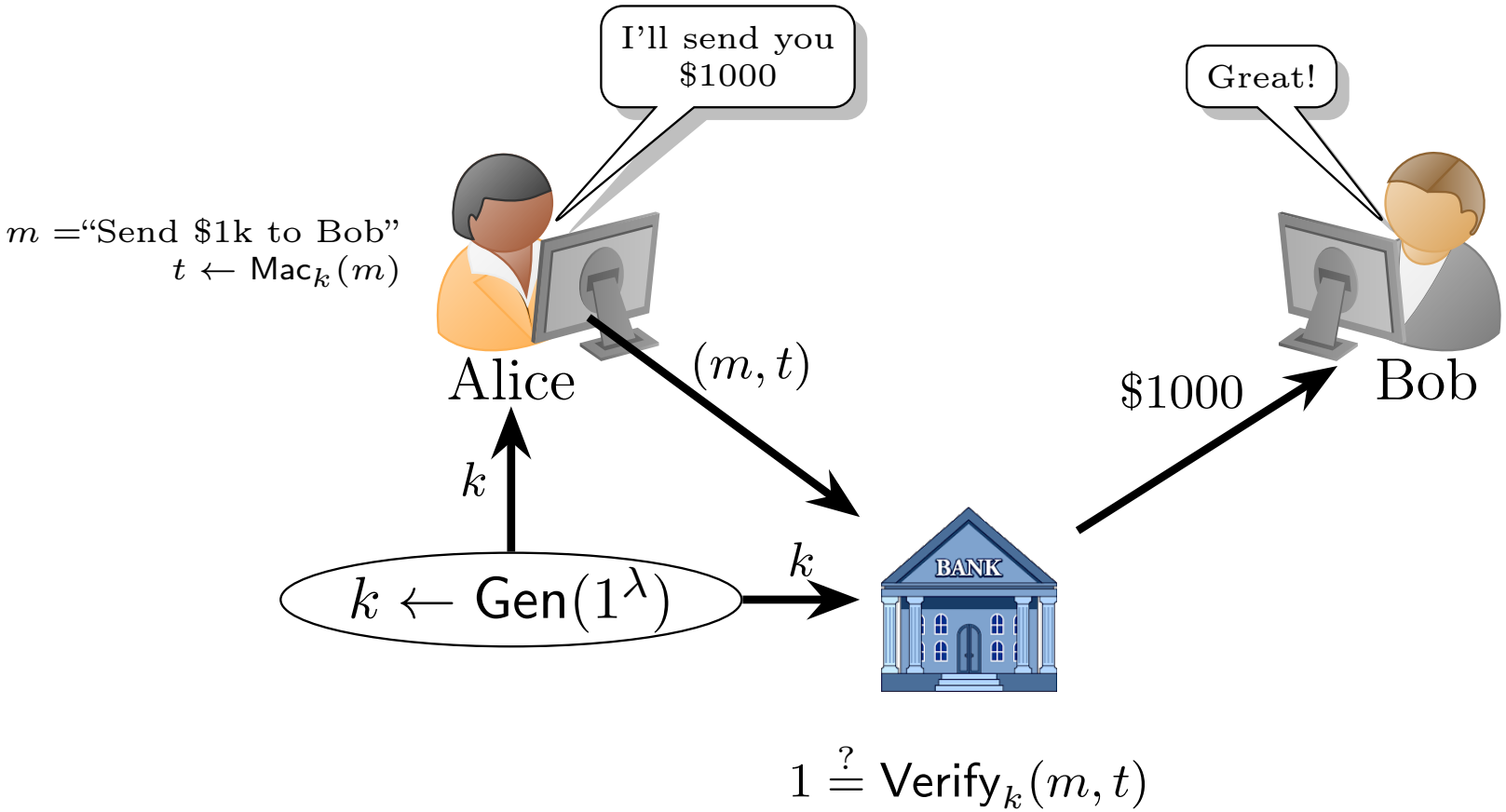
Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



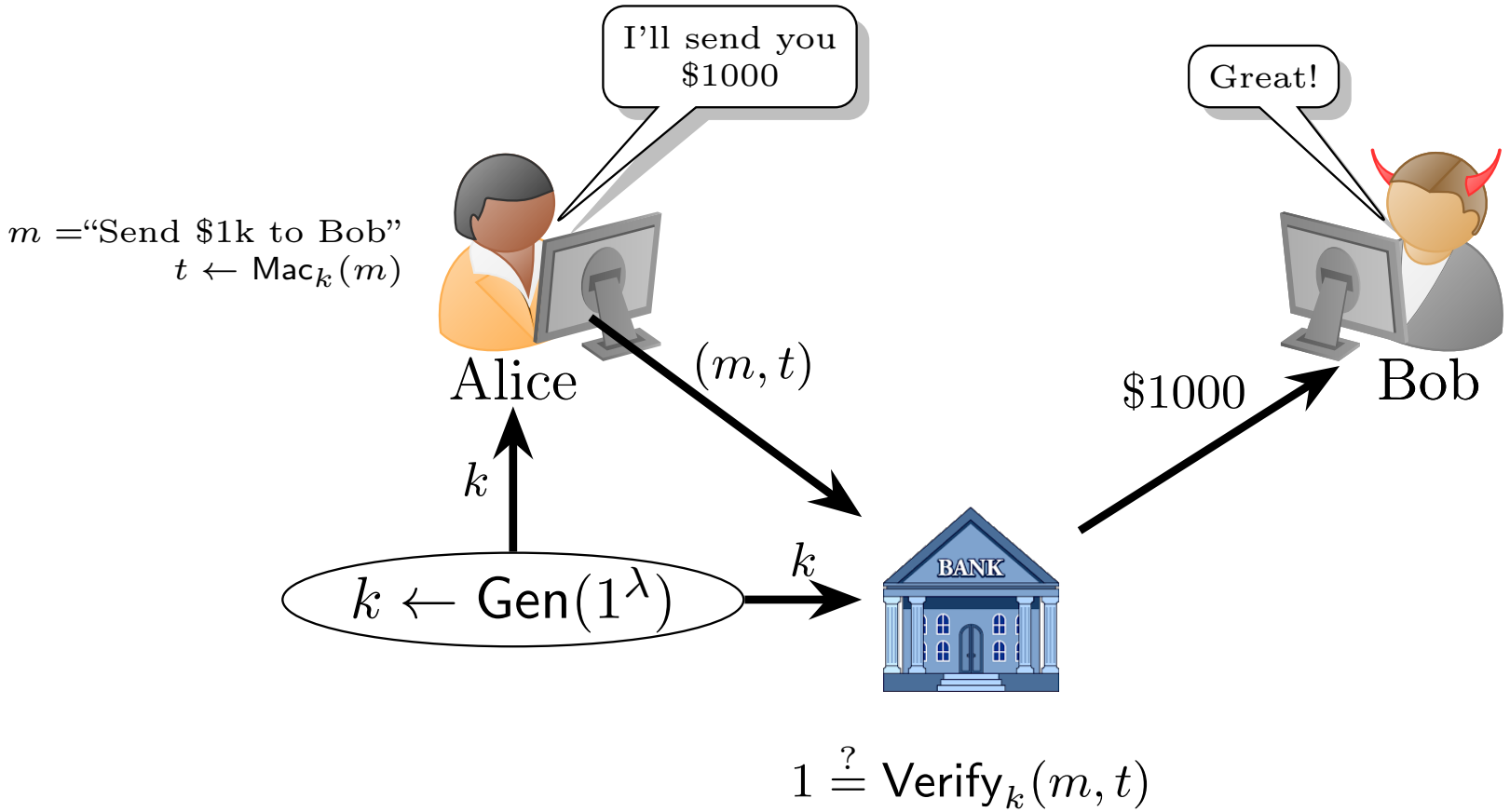
Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



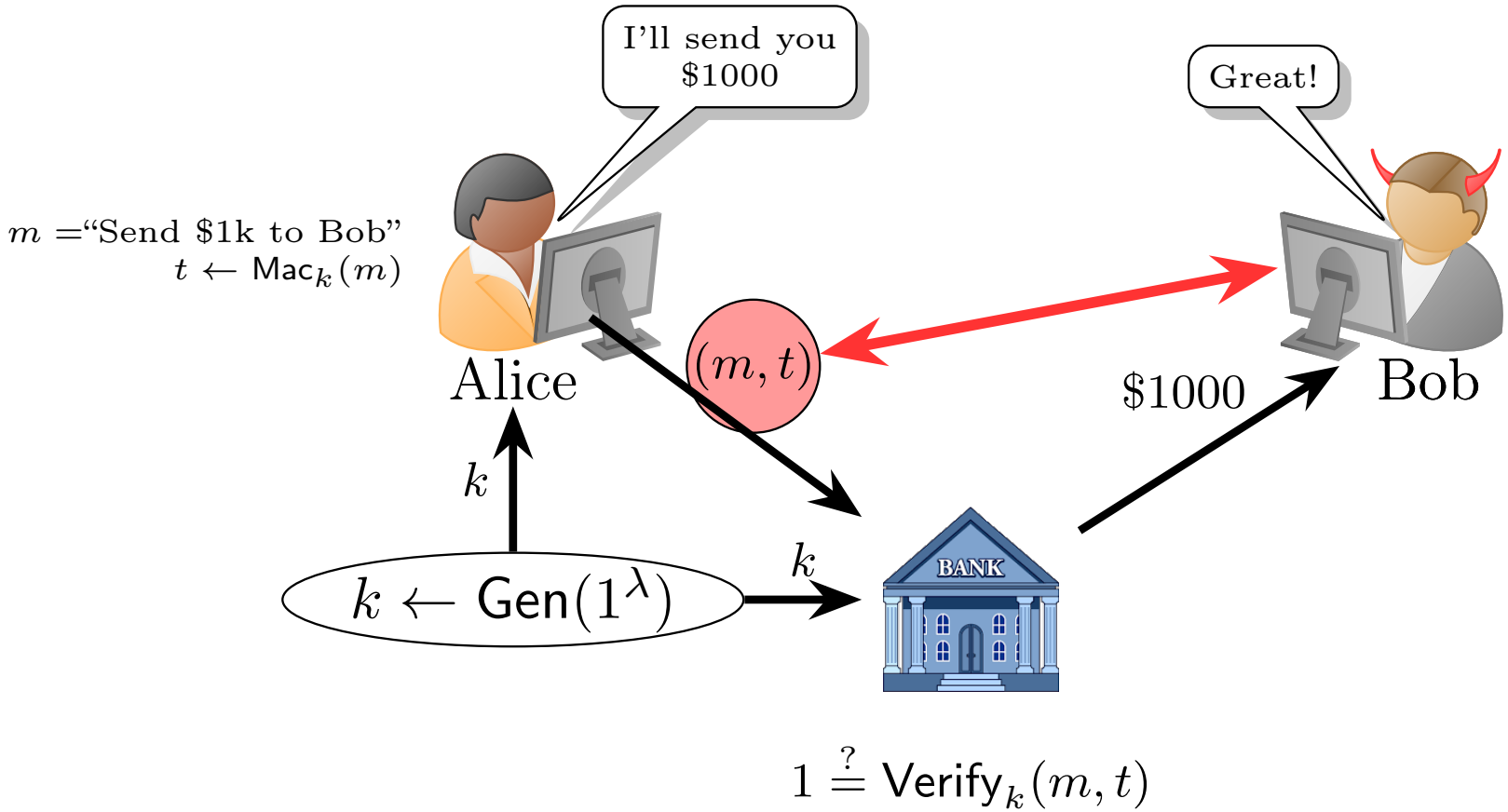
Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



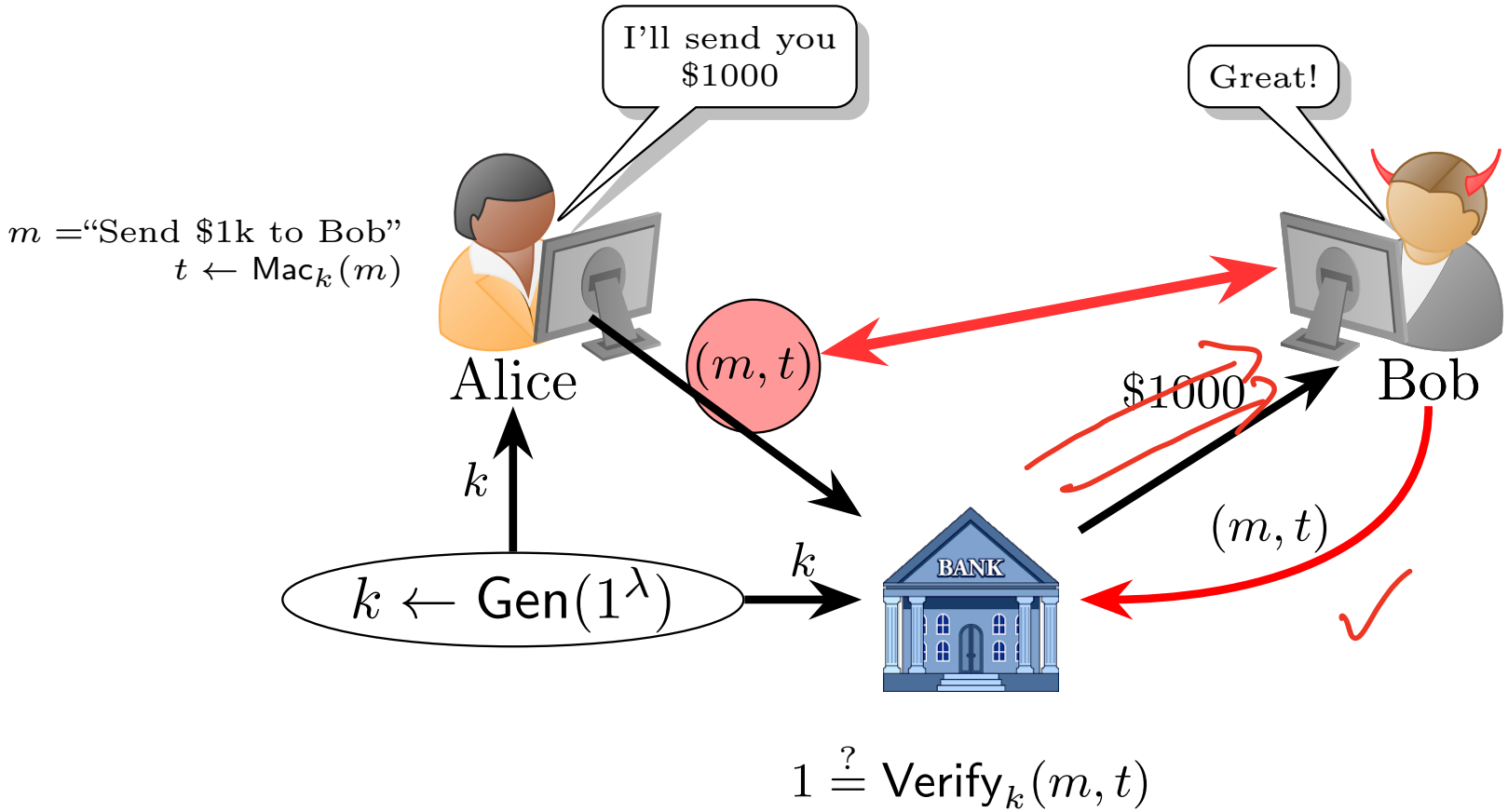
Practice with Security Definitions

- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



Practice with Security Definitions

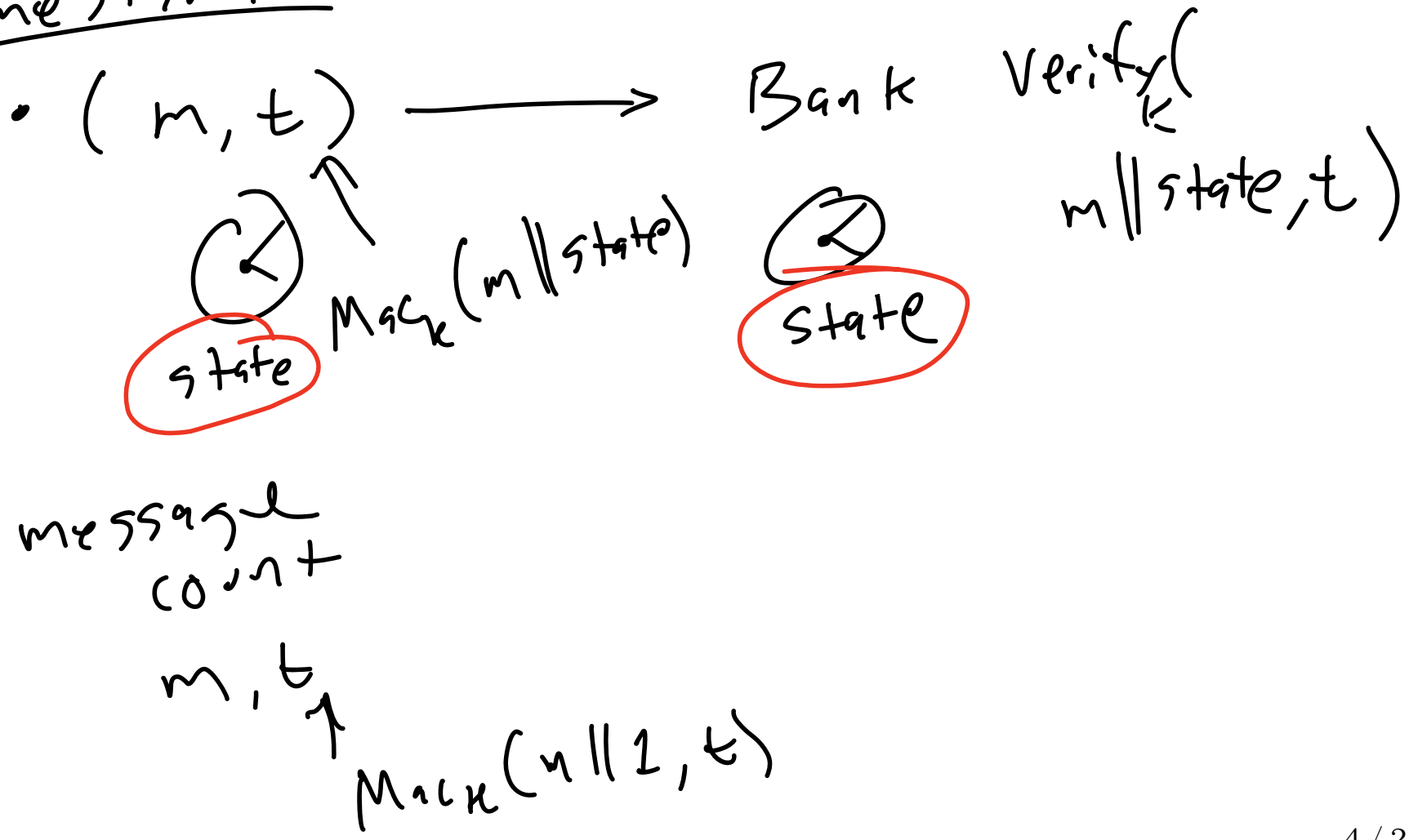
- Can you make a MAC secure against replay attacks?
- How would you define security against replay attacks?



Practice with Security Definitions

- How would you prevent replay attacks?
- How would you define security against replay attacks?

Timestamps



Strong MAC Security

$\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with.
   - **Goal**: Output (m, t) (message and tag).
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m, t) = 1$ and (2) $m \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

Strong MAC Security

$\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with.
   - **Goal**: Output (m, t) (message and tag).
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m, t) = 1$ and (2) $m \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

- MAC Security: adversary cannot generate a valid tag for a new message that was *never previously* authenticated

Strong MAC Security

$\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with.
   - **Goal**: Output (m, t) (message and tag).
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m, t) = 1$ and (2) $m \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

- MAC Security: adversary cannot generate a valid tag for a new message that was *never previously* authenticated
- Says nothing about an attacker being able to generate a new tag for a previously authenticated message

Strong MAC Security

$\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with.
   - **Goal**: Output (m, t) (message and tag).
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m, t) = 1$ and (2) $m \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

- MAC Security: adversary cannot generate a valid tag for a new message that was *never previously* authenticated
- Says nothing about an attacker being able to generate a new tag for a previously authenticated message
  - Note: you'd want this for replay attack security!

Strong MAC Security

$\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with.
   - **Goal**: Output (m, t) (message and tag).
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m, t) = 1$ and (2) $m \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

- MAC Security: adversary cannot generate a valid tag for a new message that was *never previously* authenticated
- Says nothing about an attacker being able to generate a new tag for a previously authenticated message
  - Note: you'd want this for replay attack security!
- Gives us notion of *strong MAC security*

Strong MAC Security

$\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(\lambda)$

Strong MAC Security

Mac-sforge$_{\mathcal{A},\Pi}(\lambda)$

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

Strong MAC Security

$\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with *and their tags*.

Strong MAC Security

$\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with *and their tags*.
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m,t) = 1$ and (2) $(m,t) \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

Strong MAC Security

$\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with *and their tags*.
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m,t) = 1$ and (2) $(m,t) \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

Definition 1 (Strong MAC Security)

Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. Then Π is a *strong MAC* if for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Mac\text{-}sforge}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda).$$

$\mathcal{A}, \Pi$

General Methods for Proving Security

General Methods for Proving Security

General Methods for Proving Security

Reductions

General Methods for Proving Security

Reductions

Indistinguishability

Reductions

Reductions

- Proofs are via contradiction.

Reductions

- Proofs are via contradiction.
- Theorem: Assume Problem/Construction X is hard/secure.

Reductions

- Proofs are via contradiction.
- Theorem: Assume Problem/Construction X is hard/secure. Then, Construction Y is secure.

Reductions

- Proofs are via contradiction.
- Theorem: Assume Problem/Construction X is hard/secure. Then, Construction Y is secure.
- Proof: Suppose not, that is, construction Y is not secure.

Reductions

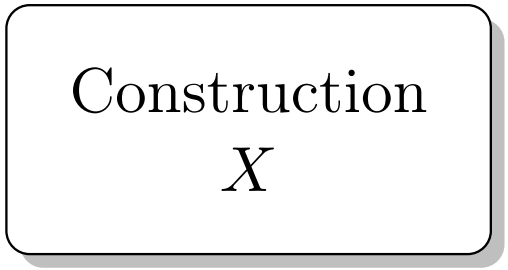
- Proofs are via contradiction.
- Theorem: Assume Problem/Construction X is hard/secure. Then, Construction Y is secure.
- Proof: Suppose not, that is, construction Y is not secure. Then we can construct an adversary to break problem/construction X.

Reductions

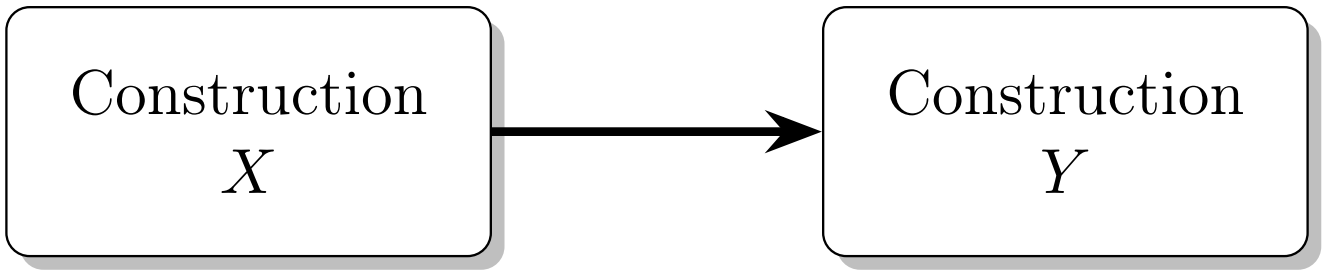
- Proofs are via contradiction.
- Theorem: Assume Problem/Construction X is hard/secure. Then, Construction Y is secure.
- Proof: Suppose not, that is, construction Y is not secure. Then we can construct an adversary to break problem/construction X.
  - Since we assumed problem/construction X was hard/secure, this is a contradiction!

Reductions

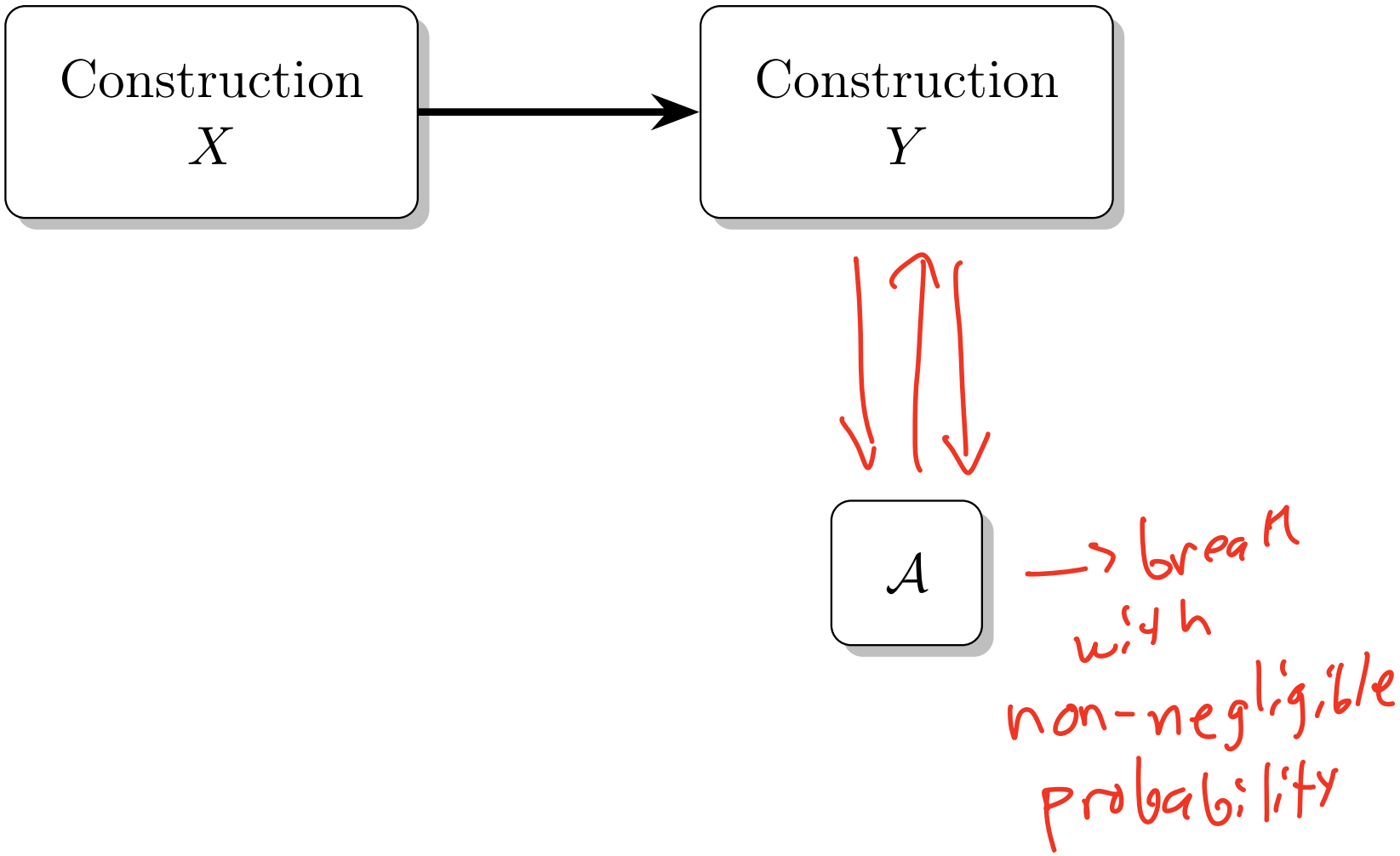
Reductions



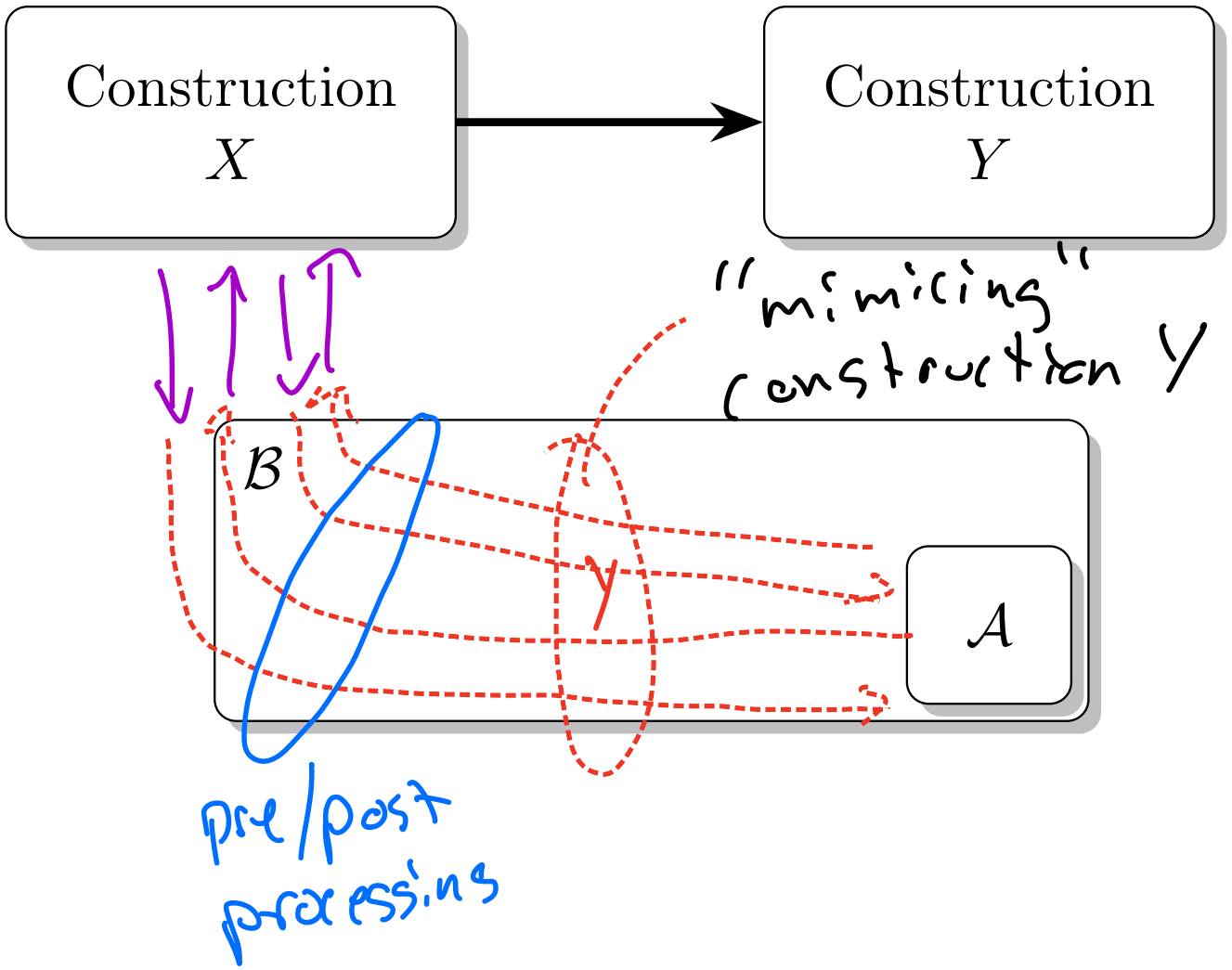
Reductions



Reductions


Construction
X
Construction
Y
$\mathcal{A}$
→ break
with
non-negligible
probability

Reductions


Construction
X
Construction
Y
"mimicing"
construction Y
$\mathcal{B}$
γ
$\mathcal{A}$
pre/post
processing

Example: Strong MACs via Canonical Verification

Theorem 1 (Strong MACs from MACs with Canonical Verification)

Example: Strong MACs via Canonical Verification

Theorem 1 (Strong MACs from MACs with Canonical Verification)

Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ *be a secure MAC with canonical verification.*

Example: Strong MACs via Canonical Verification

Theorem 1 (Strong MACs from MACs with Canonical Verification)

Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ *be a secure MAC with canonical verification. Then,* Π *is a strong MAC.*

Example: Strong MACs via Canonical Verification

Theorem 1 (Strong MACs from MACs with Canonical Verification)

Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ *be a secure MAC with canonical verification. Then,* Π *is a strong MAC.*

Canonical Verification

$\mathsf{Verify}_k(m, t) :=$

- $t' \leftarrow \mathsf{Mac}_k(m)$
- Output $t' == t$

Deterministic

Example: Strong MACs via Canonical Verification

Theorem 1 (Strong MACs from MACs with Canonical Verification)

Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ *be a secure MAC with canonical verification. Then,* Π *is a strong MAC.*

Canonical Verification

$\mathsf{Verify}_k(m, t) :=$

- $t' \leftarrow \mathsf{Mac}_k(m)$
- Output $t' == t$

Mac_k is deterministic given a fixed key k!

Playing with Reductions: Strong MACs

First recall the two security games:

$\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with.
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m,t) = 1$ and (2) $m \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

$\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(\lambda)$

1. $k \leftarrow \mathsf{Gen}(1^\lambda)$.
2. Adversary $\mathcal{A}$ receives 1^λ as input and *oracle access* to $\mathsf{Mac}_k(\cdot)$. Let $\mathcal{Q}$ denote the set of all messages that $\mathcal{A}$ queries its oracle with *and their tags*.
3. $\mathcal{A}$ *wins* if and only if (1) $\mathsf{Verify}_k(m,t) = 1$ and (2) $(m,t) \notin \mathcal{Q}$ ($\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1$ in this case and 0 otherwise).

Playing with Reductions: Strong MACs

Proof Overview

Playing with Reductions: Strong MACs

Proof Overview

- Let Π be a MAC with canonical verification, but suppose via contradiction that Π is *not* a strong MAC.

Playing with Reductions: Strong MACs

Proof Overview

- Let Π be a MAC with canonical verification, but suppose via contradiction that Π is *not* a strong MAC.

- Suppose we have an adversary $\mathcal{A}$ which can win $\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(\lambda)$ with *non-negligible* probability $p(\lambda)$.

Playing with Reductions: Strong MACs

Proof Overview

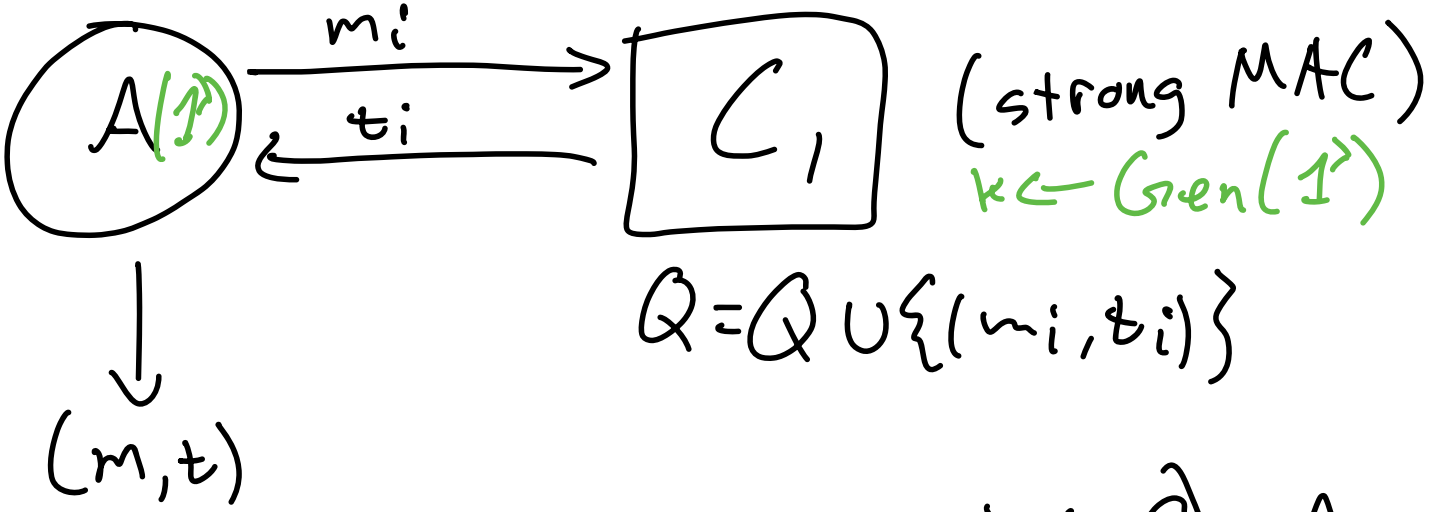
- Let Π be a MAC with canonical verification, but suppose via contradiction that Π is *not* a strong MAC.
- Suppose we have an adversary $\mathcal{A}$ which can win $\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(\lambda)$ with *non-negligible* probability $p(\lambda)$.
- We will use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that wins $\mathsf{Mac\text{-}forge}_{\mathcal{B},\Pi}(\lambda)$ with non-negligible probability.

Playing with Reductions: Strong MACs

Proof Overview

- Let Π be a MAC with canonical verification, but suppose via contradiction that Π is *not* a strong MAC.
- Suppose we have an adversary $\mathcal{A}$ which can win $\mathsf{Mac\text{-}sforge}_{\mathcal{A},\Pi}(\lambda)$ with *non-negligible* probability $p(\lambda)$.
- We will use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that wins $\mathsf{Mac\text{-}forge}_{\mathcal{B},\Pi}(\lambda)$ with non-negligible probability.
  - We will need to use the canonical verification property in our proof.

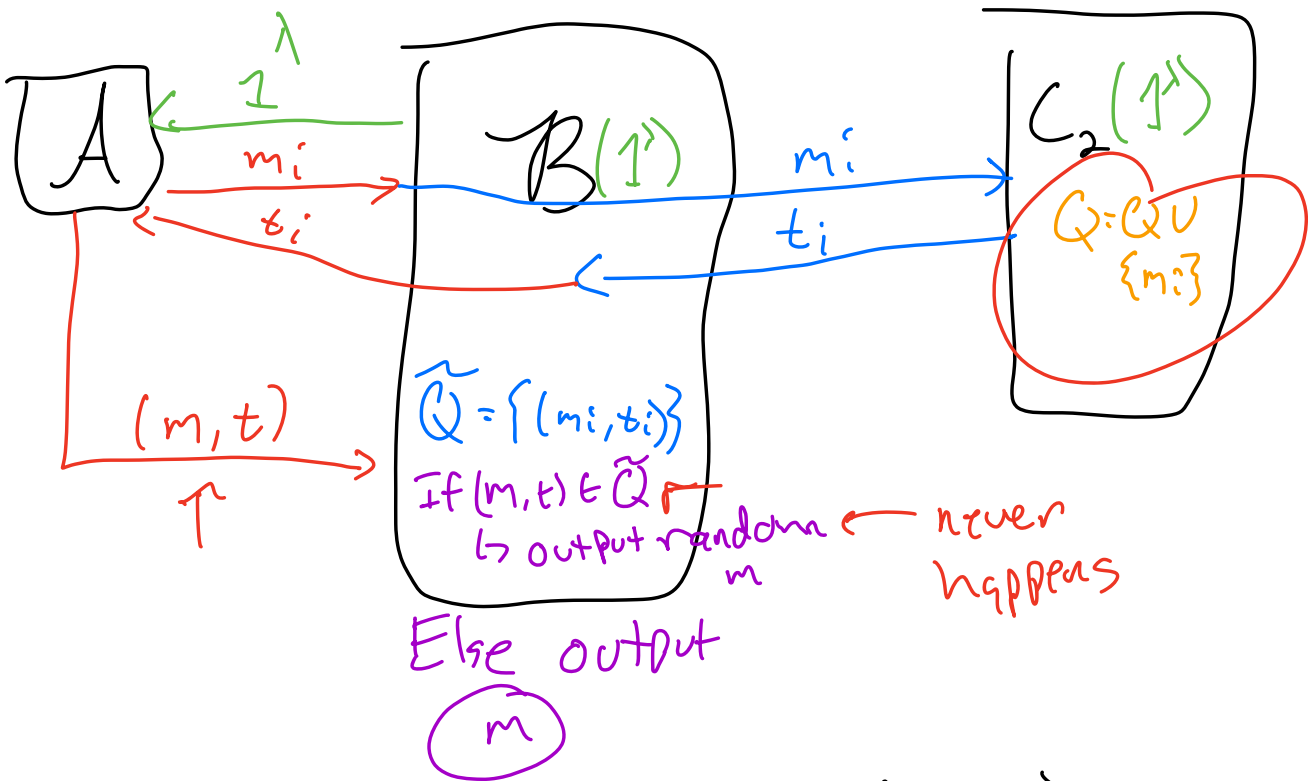
Playing with Reductions: Strong MACs



Proof:

A wins $\iff (m,t) \notin Q \wedge Verify_k(m,t) = 1$

Assumption: A wins w.p. $\geq p(\lambda)$ for non-negl. $p(\lambda)$



Question: Can A output (m,t) s.t. $\exists\, t'$ s.t. $(m,t') \in \tilde{Q}$ but $(m,t) \notin \tilde{Q}$?

NO! • Every m has a unique tag

Question: Can A output (m,t) s.t. $\exists\, m'$ s.t. $(m',t) \in \tilde{Q}$ but $(m,t) \notin \tilde{Q}$?

Yes!

Indistinguishability

Indistinguishability

- *Indistinguishability* offers another way to define and prove security.

Indistinguishability

- *Indistinguishability* offers another way to define and prove security.
- Proofs again are usually via contradiction, but you can also do *direct proofs* more easily (in my opinion) via indistingushability (in some cases).

Indistinguishability

- *Indistinguishability* offers another way to define and prove security.
- Proofs again are usually via contradiction, but you can also do *direct proofs* more easily (in my opinion) via indistingushability (in some cases).
- Theorem: Construction X and Construction Y are (computationally) indistinguishable.

Indistinguishability

- *Indistinguishability* offers another way to define and prove security.
- Proofs again are usually via contradiction, but you can also do *direct proofs* more easily (in my opinion) via indistingushability (in some cases).
- Theorem: Construction X and Construction Y are (computationally) indistinguishable.
- Proof: Suppose not; that is, we can distinguish X and Y.

Indistinguishability

- *Indistinguishability* offers another way to define and prove security.
- Proofs again are usually via contradiction, but you can also do *direct proofs* more easily (in my opinion) via indistingushability (in some cases).
- Theorem: Construction X and Construction Y are (computationally) indistinguishable.
- Proof: Suppose not; that is, we can distinguish X and Y. Then, we can use the *distinguisher* which does this to break some security property of X or Y, or break some underlying hard problem.

Indistinguishability: "Real" vs. "Random"

Indistinguishability: "Real" vs. "Random"

- "Real" vs. "Random": show that the actual construction X is computationally/statistically indistinguishable from a *uniformly* random object.

Indistinguishability: "Real" vs. "Random"

- "Real" vs. "Random": show that the actual construction X is computationally/statistically indistinguishable from a *uniformly* random object.

Pseudorandom Function
$F_k : \{0,1\}^m \to \{0,1\}^n$

Indistinguishability: "Real" vs. "Random"

- "Real" vs. "Random": show that the actual construction X is computationally/statistically indistinguishable from a *uniformly* random object.

Pseudorandom Function
$F_k : \{0,1\}^m \to \{0,1\}^n$

Uniform Distribution

$\mathbb{U}_{\{0,1\}^n}$

Indistinguishability: "Real" vs. "Random"

- "Real" vs. "Random": show that the actual construction X is computationally/statistically indistinguishable from a *uniformly* random object.

$n \geq m+1$

Pseudorandom Function	$\approx_\varepsilon$	Uniform Distribution
$F_k : \{0,1\}^m \to \{0,1\}^n$		$\mathbb{U}_{\{0,1\}^n}$

Computationally indistinguishable

Efficient (PPT) cannot distinguish except w.p. $\leq \varepsilon(n) + \frac{1}{2}$

Indistinguishability: "Left" vs. "Right"

Indistinguishability: "Left" vs. "Right"

- "Left" vs. "Right": show that the actual construction X is computationally/statistically indistinguishable from a *specific distribution*.

Indistinguishability: "Left" vs. "Right"

- "Left" vs. "Right": show that the actual construction X is computationally/statistically indistinguishable from a *specific distribution*.

- Example: certain Encryption Schemes

Indistinguishability: "Left" vs. "Right"

- "Left" vs. "Right": show that the actual construction X is computationally/statistically indistinguishable from a *specific distribution*.

- Example: certain Encryption Schemes

Encryption
of m_1
$\mathsf{Enc}_k(m_1)$

Indistinguishability: "Left" vs. "Right"

- "Left" vs. "Right": show that the actual construction X is computationally/statistically indistinguishable from a *specific distribution*.

- Example: certain Encryption Schemes

Encryption of m_1
$\mathsf{Enc}_k(m_1)$

Encryption of m_2
$\mathsf{Enc}_k(m_2)$

Indistinguishability: "Left" vs. "Right"

- "Left" vs. "Right": show that the actual construction X is computationally/statistically indistinguishable from a *specific distribution*.

- Example: certain Encryption Schemes

$m_1 \neq m_2$

Encryption of m_1 $\mathsf{Enc}_k(m_1)$	$\approx_\varepsilon$	Encryption of m_2 $\mathsf{Enc}_k(m_2)$

Idealized Models

Idealized Models

Idealized Models

- We've seen that security can be tricky!

Idealized Models

- We've seen that security can be tricky!
- We've also been ignoring *implementations* of cryptographic objects

Idealized Models

- We've seen that security can be tricky!
- We've also been ignoring *implementations* of cryptographic objects
  - How do we know an object behaves *randomly*?

Idealized Models

- We've seen that security can be tricky!
- We've also been ignoring *implementations* of cryptographic objects
  - How do we know an object behaves *randomly*?
  - How do we know the specified protocol outputs *correctly*?

Idealized Models

- We've seen that security can be tricky!

- We've also been ignoring *implementations* of cryptographic objects
  - How do we know an object behaves *randomly*?
  - How do we know the specified protocol outputs *correctly*?
  - How do we know that an adversary *cannot distinguish* the protocol from random?

Idealized Models

- We've seen that security can be tricky!
- We've also been ignoring *implementations* of cryptographic objects
  - How do we know an object behaves *randomly*?
  - How do we know the specified protocol outputs *correctly*?
  - How do we know that an adversary *cannot distinguish* the protocol from random?
- *Idealized Models* offer one of many ways to *prove* these properties.

Idealized Models: What are they?

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation.*

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation.*

- Examples of idealized models:

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation.*

- Examples of idealized models:
  - Random Oracle Model

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation.*

- Examples of idealized models:
  - Random Oracle Model
  - Ideal Permutation Model

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation.*

- Examples of idealized models:
  - Random Oracle Model
  - Ideal Permutation Model
  - Ideal Cipher Model

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation.*

- Examples of idealized models:
  - Random Oracle Model
  - Ideal Permutation Model
  - Ideal Cipher Model
  - Generic Group Model

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation.*

- Examples of idealized models:
  - Random Oracle Model
  - Ideal Permutation Model
  - Ideal Cipher Model
  - Generic Group Model
  - Algebraic Group Model

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation*.

- Examples of idealized models:
  - Random Oracle Model
  - Ideal Permutation Model
  - Ideal Cipher Model
  - Generic Group Model
  - Algebraic Group Model
  - Ideal Functionalities

Idealized Models: What are they?

- In a nutshell: *a perfect implementation of your function* or a *specific implementation.*

- Examples of idealized models:
  - Random Oracle Model
  - Ideal Permutation Model
  - Ideal Cipher Model
  - Generic Group Model
  - Algebraic Group Model
  - Ideal Functionalities

Random Oracle Model

Random Oracle Model

- Everyone has access to an *unkeyed* hash function that is *truly random* (implemented as a lazy dictionary).

Random Oracle Model

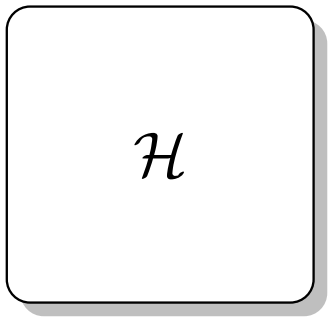
- Everyone has access to an *unkeyed* hash function that is *truly random* (implemented as a lazy dictionary).

- $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^n$.

Random Oracle Model

- Everyone has access to an *unkeyed* hash function that is *truly random* (implemented as a lazy dictionary).

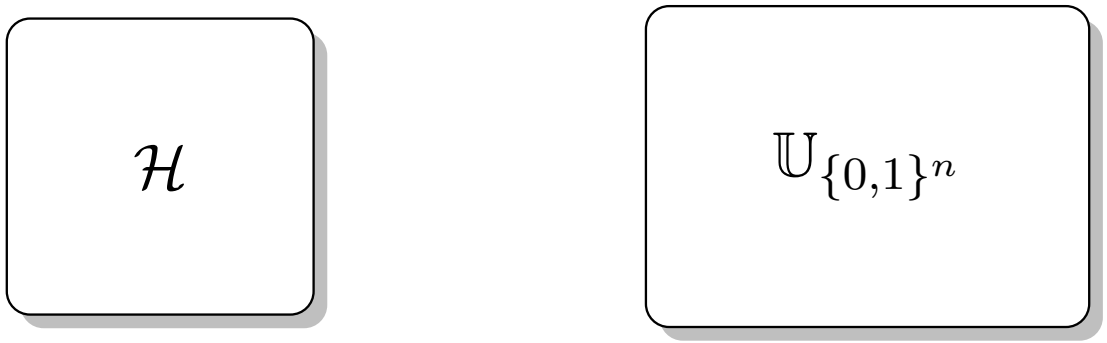
- $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^n$.



Random Oracle Model

- Everyone has access to an *unkeyed* hash function that is *truly random* (implemented as a lazy dictionary).

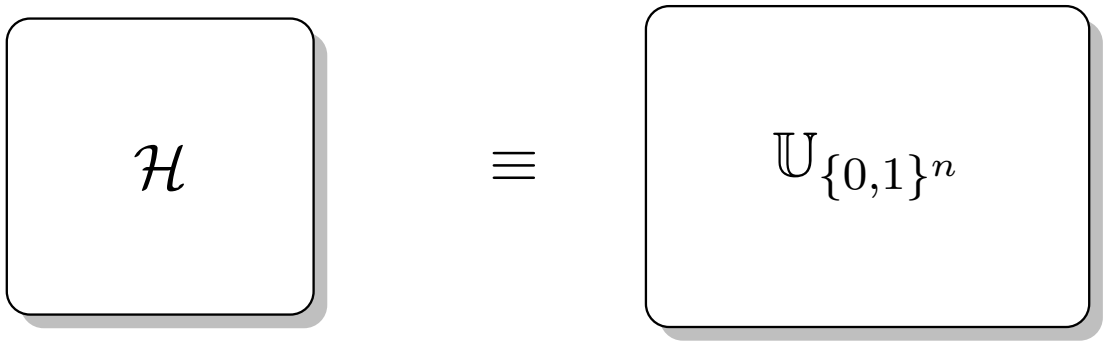
- $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^n$.



Random Oracle Model

- Everyone has access to an *unkeyed* hash function that is *truly random* (implemented as a lazy dictionary).

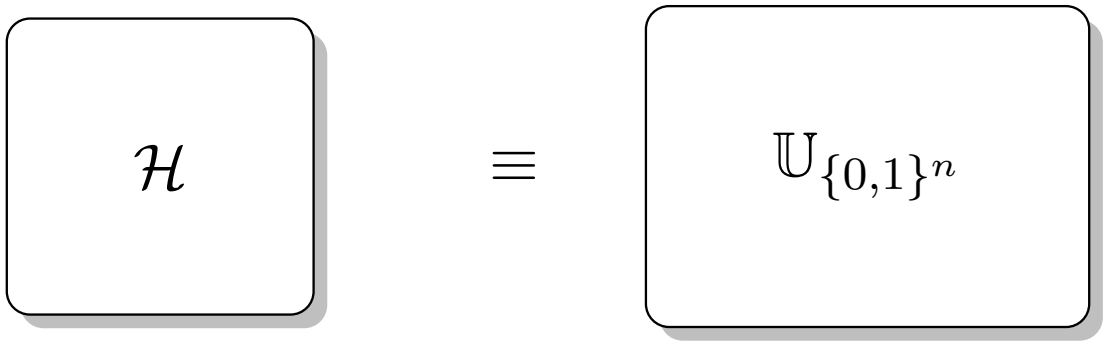
- $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^n$.



Random Oracle Model

- Everyone has access to an *unkeyed* hash function that is *truly random* (implemented as a lazy dictionary).

- $\mathcal{H}\colon \{0,1\}^* \to \{0,1\}^n$.



$\mathcal{H}$ does not output random values!

If you call $h_1 = \mathcal{H}(x)$ at 10am, then $h_2 = \mathcal{H}(x)$ at 2pm, $h_1 = h_2$!

Random Oracle Model

- Next time, we'll use the Random Oracle Model to build simple and secure *pseudorandom functions*