CS 594 – Advanced Crypto (Spring 2026)

Alex Block

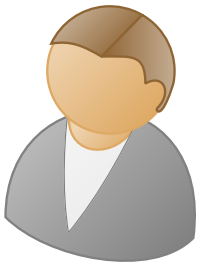
Lecture 2

January 21, 2026

Security Before Modern Cryptography

Security Before Modern Cryptography

Alice



Bob

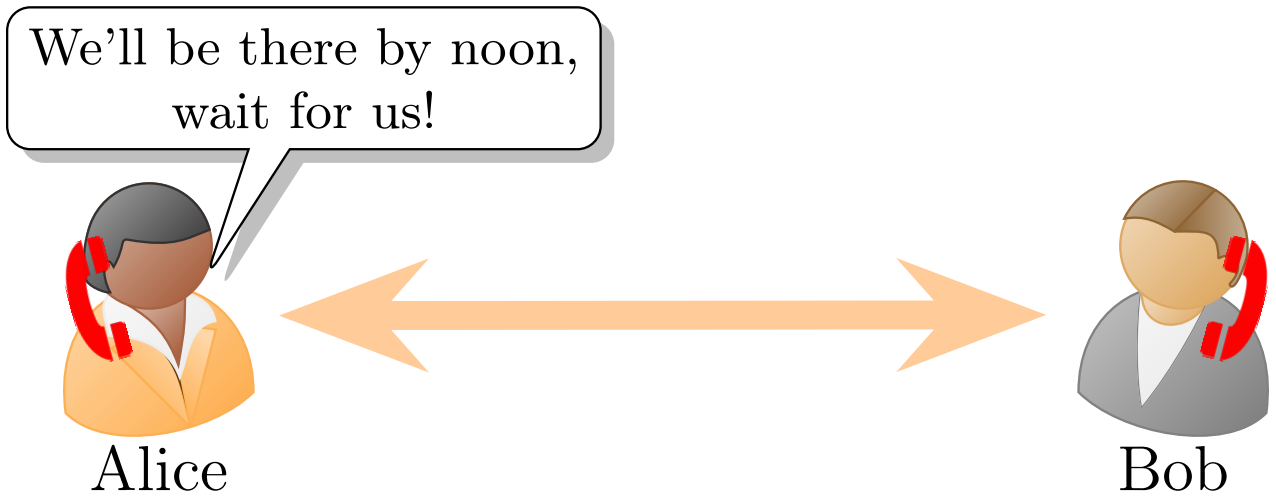
Security Before Modern Cryptography

Alice

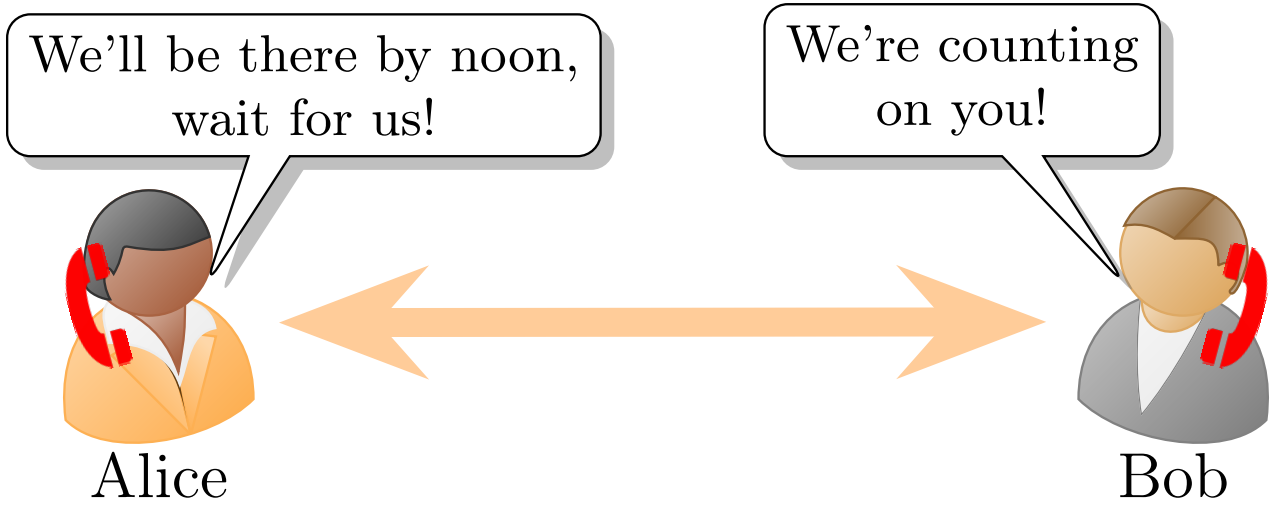
Bob

Security Before Modern Cryptography

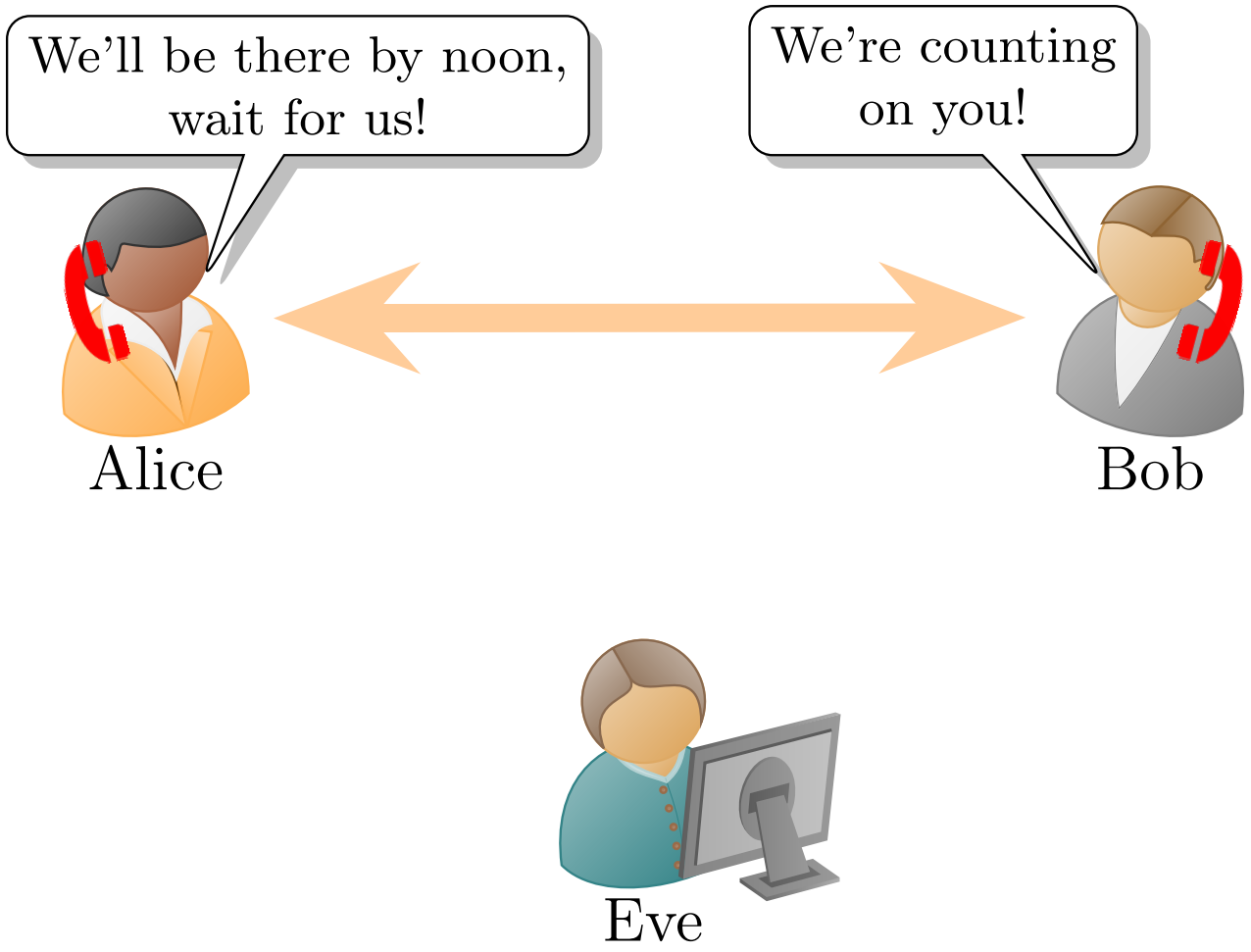
Security Before Modern Cryptography



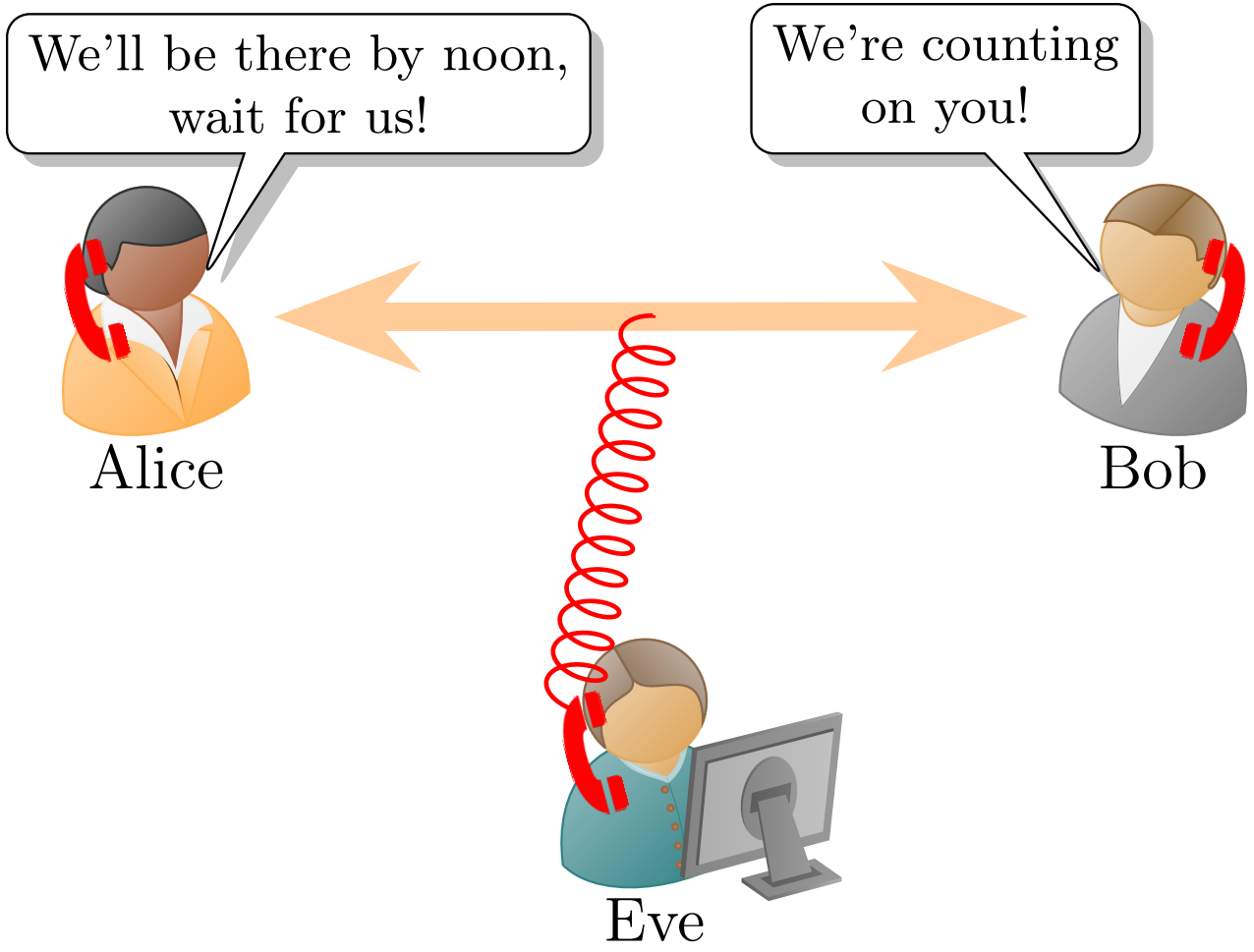
Security Before Modern Cryptography



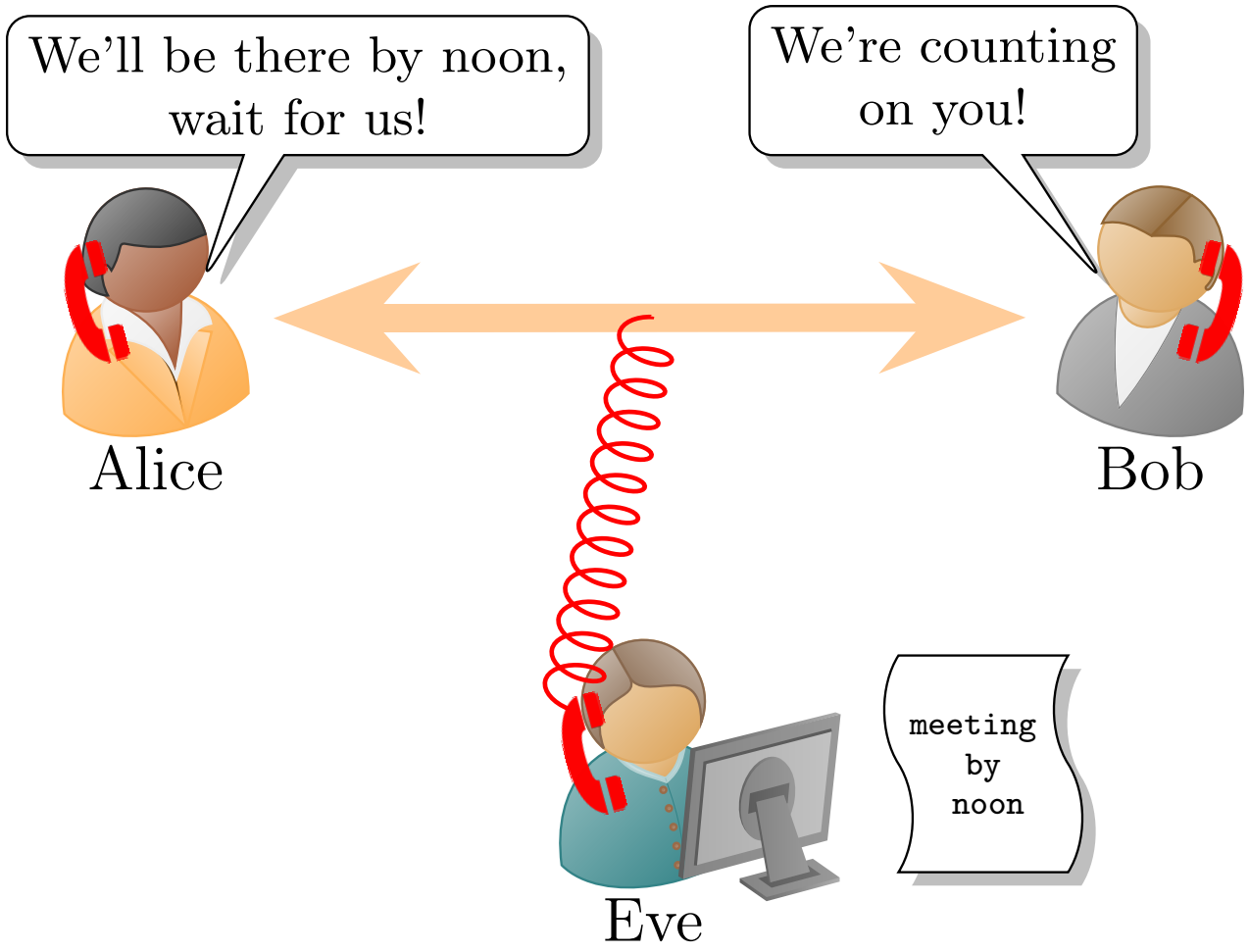
Security Before Modern Cryptography



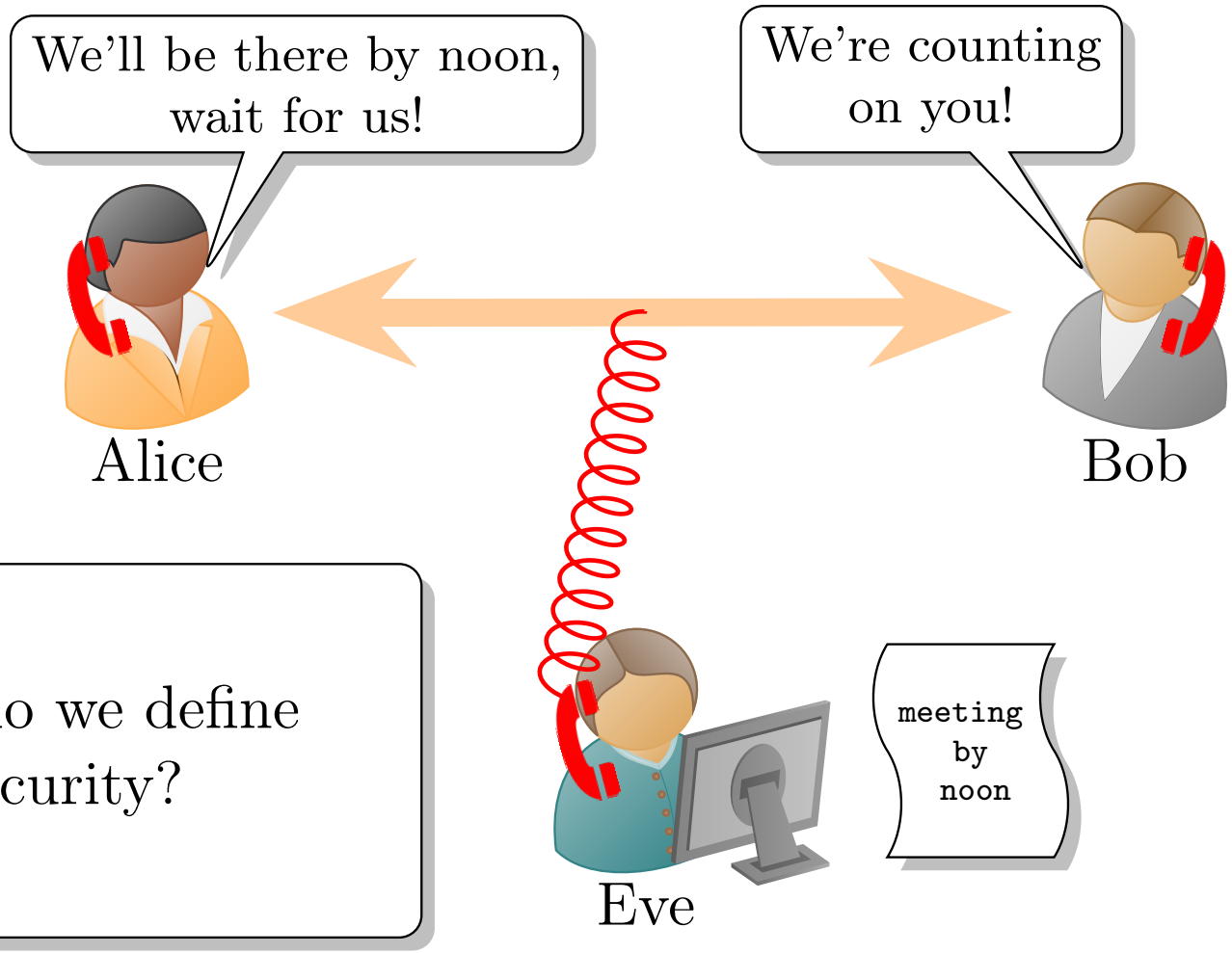
Security Before Modern Cryptography



Security Before Modern Cryptography

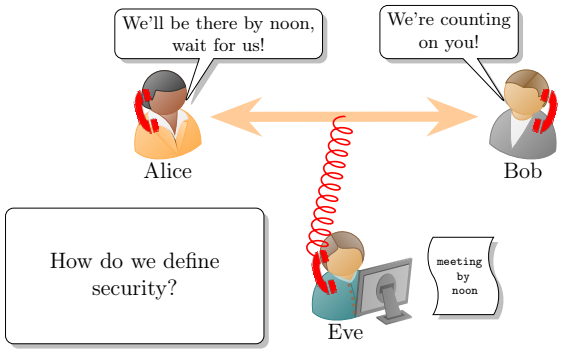


Security Before Modern Cryptography



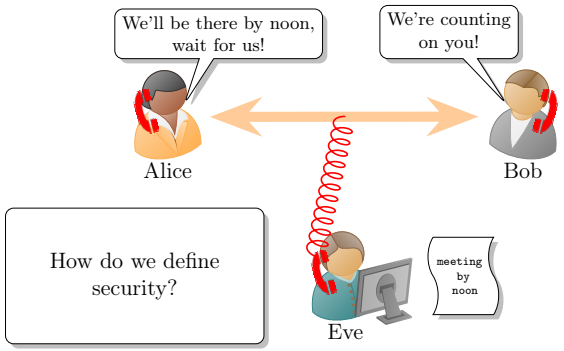
How do we define security?

Security Before Modern Cryptography



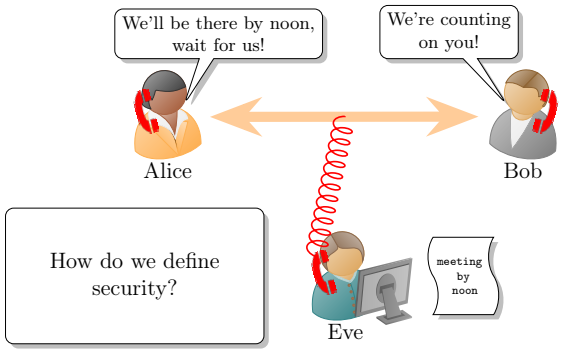
Security Before Modern Cryptography

- Crypto was more an "art" before it was a science



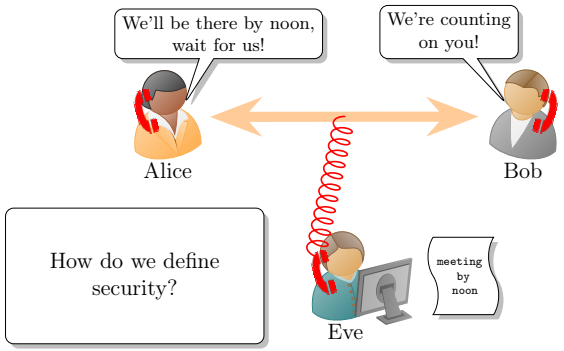
Security Before Modern Cryptography

- Crypto was more an "art" before it was a science
  - Heuristic Design and Analysis



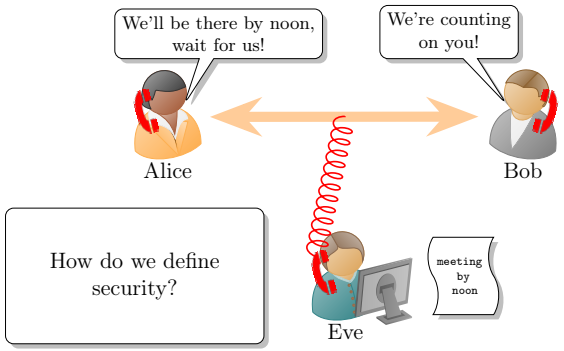
Security Before Modern Cryptography

- Crypto was more an "art" before it was a science
  - Heuristic Design and Analysis
  - Construct → Break → Patch → Repeat



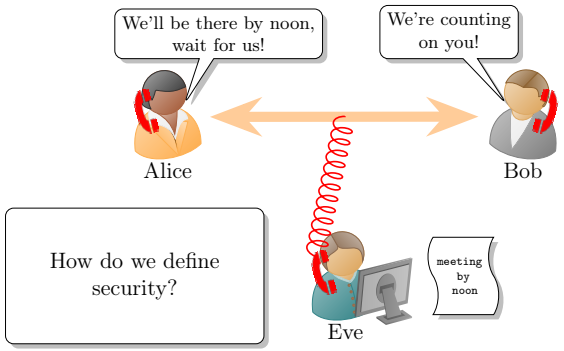
Security Before Modern Cryptography

- Crypto was more an "art" before it was a science
  - Heuristic Design and Analysis
  - Construct → Break → Patch → Repeat
- Can you formally prove a scheme is secure?



Security Before Modern Cryptography

- Crypto was more an "art" before it was a science
  - Heuristic Design and Analysis
  - Construct → Break → Patch → Repeat
- Can you formally prove a scheme is secure?
- What does "secure" even mean?



Core Principles of Modern Cryptography

Core Principles of Modern Cryptography

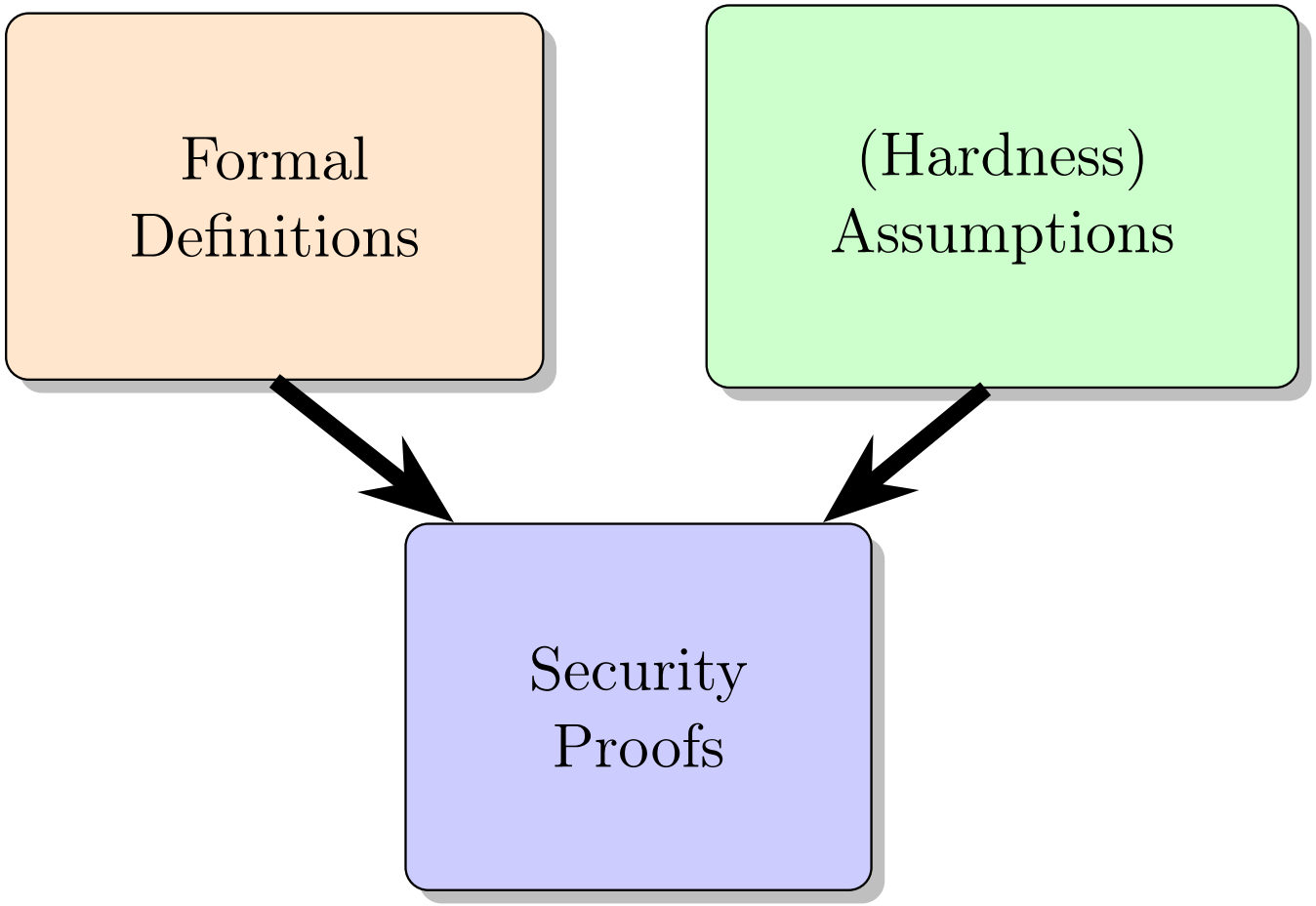
Formal Definitions

Core Principles of Modern Cryptography

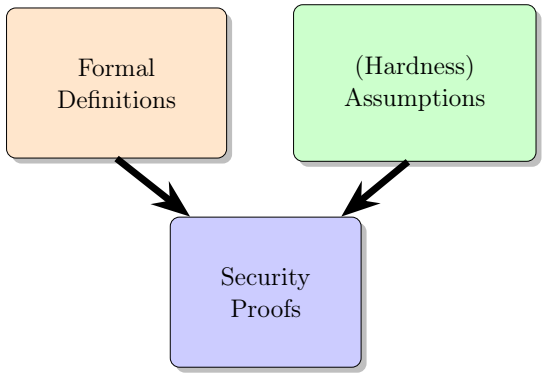
Formal Definitions

(Hardness) Assumptions

Core Principles of Modern Cryptography

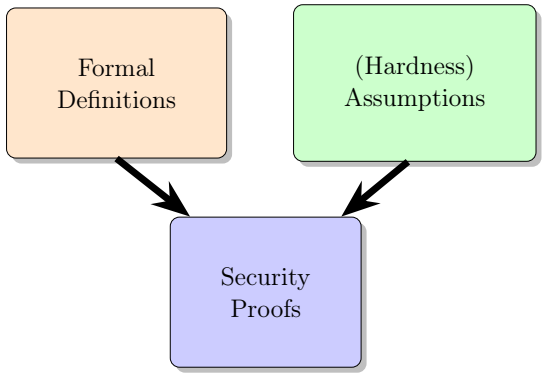


Core Principles of Modern Cryptography



Core Principles of Modern Cryptography

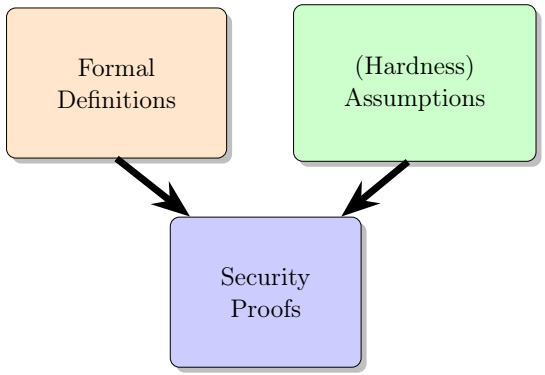
- Precise, mathematical models and definitions for what "security" really means



Core Principles of Modern Cryptography

- Precise, mathematical models and definitions for what "security" really means

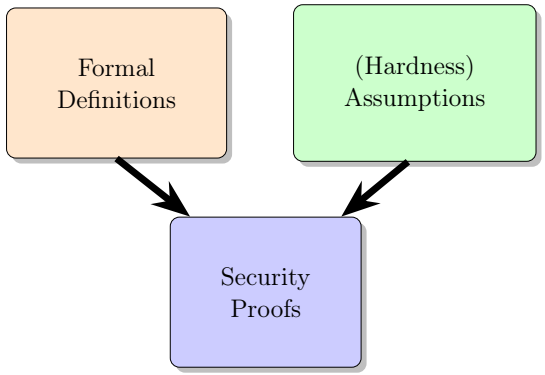
- Assumptions: no "modern" crypto without assuming hard problems, need to clearly state and understand these assumptions; no ambiguity!



Core Principles of Modern Cryptography

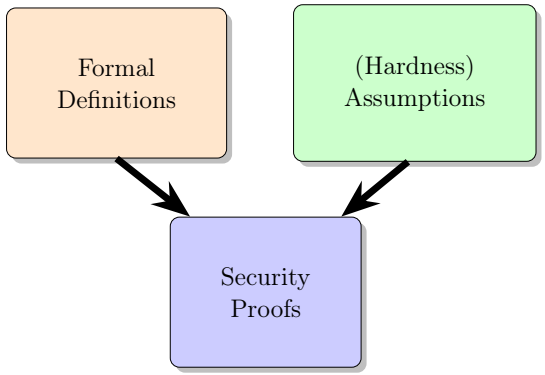
- Precise, mathematical models and definitions for what "security" really means

- Assumptions: no "modern" crypto without assuming hard problems, need to clearly state and understand these assumptions; no ambiguity!
  - Also build new schemes *assuming* others are secure



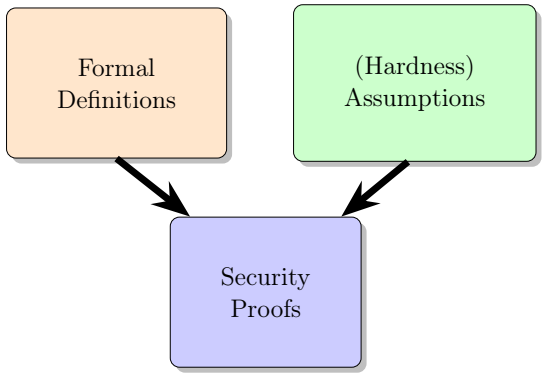
Core Principles of Modern Cryptography

- Precise, mathematical models and definitions for what "security" really means
- Assumptions: no "modern" crypto without assuming hard problems, need to clearly state and understand these assumptions; no ambiguity!
  - Also build new schemes *assuming* others are secure
- These two together give us proofs of security, moving away from design-break-patch



Core Principles of Modern Cryptography

- Precise, mathematical models and definitions for what "security" really means
- Assumptions: no "modern" crypto without assuming hard problems, need to clearly state and understand these assumptions; no ambiguity!
  - Also build new schemes *assuming* others are secure
- These two together give us proofs of security, moving away from design-break-patch
- Proof pattern: Construction X satisfies Security Definition Y, unless Hard Problem/Secure Construction Z is broken



Core Principles of Modern Cryptography

Pros

- Rigor
- Immune to tech improvements
- Robustness
- Tunable

.fast?

Cons

- Tech improvements (Quantum)
- Implementations
  - ↳ Legacy compat.
  - ↳ Impractical in real world

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems

Cons

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems
- Rigorous, provable, mathematical proofs of security

Cons

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems
- Rigorous, provable, mathematical proofs of security
- Pinning down what "security" means!

Cons

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems
- Rigorous, provable, mathematical proofs of security
- Pinning down what "security" means!
- Building secure systems on top of things we prove/believe are secure

Cons

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems
- Rigorous, provable, mathematical proofs of security
- Pinning down what "security" means!
- Building secure systems on top of things we prove/believe are secure

Cons

- Can be difficult to formalize what "security" means

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems
- Rigorous, provable, mathematical proofs of security
- Pinning down what "security" means!
- Building secure systems on top of things we prove/believe are secure

Cons

- Can be difficult to formalize what "security" means
- Mathematical definitions $\neq$ all possible real world attacks

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems
- Rigorous, provable, mathematical proofs of security
- Pinning down what "security" means!
- Building secure systems on top of things we prove/believe are secure

Cons

- Can be difficult to formalize what "security" means
- Mathematical definitions $\neq$ all possible real world attacks
- "Theoretical" design sometimes difficult to translate to real world systems

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems
- Rigorous, provable, mathematical proofs of security
- Pinning down what "security" means!
- Building secure systems on top of things we prove/believe are secure

Cons

- Can be difficult to formalize what "security" means
- Mathematical definitions $\neq$ all possible real world attacks
- "Theoretical" design sometimes difficult to translate to real world systems
- Crypto tends to move much slower than real world systems

Core Principles of Modern Cryptography

Pros

- Systematic, methodical way to design and build systems
- Rigorous, provable, mathematical proofs of security
- Pinning down what "security" means!
- Building secure systems on top of things we prove/believe are secure

Cons

- Can be difficult to formalize what "security" means
- Mathematical definitions $\neq$ all possible real world attacks
- "Theoretical" design sometimes difficult to translate to real world systems
- Crypto tends to move much slower than real world systems

Pros and Cons do not necessarily work against each other!

Defining Security

- What does it mean for something to be "secure"?

Defining Security

- What does it mean for something to be "secure"?
  - Confidentiality (Security/Privacy)

Defining Security

- What does it mean for something to be "secure"?
  - Confidentiality (Security/Privacy)
  - Integrity (Authenticity/Authentication/Correctness)

Defining Security

- What does it mean for something to be "secure"?
  - Confidentiality (Security/Privacy)
  - Integrity (Authenticity/Authentication/Correctness)
  - Can define more things (e.g., zero-knowledge as we'll see later)

Defining Security

- What does it mean for something to be "secure"?
  - Confidentiality (Security/Privacy)
  - Integrity (Authenticity/Authentication/Correctness)
  - Can define more things (e.g., zero-knowledge as we'll see later)
  - Intuitively: security = protecting information in some way

Defining Security

- What does it mean for something to be "secure"?
  - Confidentiality (Security/Privacy)
  - Integrity (Authenticity/Authentication/Correctness)
  - Can define more things (e.g., zero-knowledge as we'll see later)
  - Intuitively: security = protecting information in some way
  - What is the threat model? Who is attacking the scheme, or who are we trying to protect against? What kind of attacks?

Defining Security

- What does it mean for something to be "secure"?
  - Confidentiality (Security/Privacy)
  - Integrity (Authenticity/Authentication/Correctness)
  - Can define more things (e.g., zero-knowledge as we'll see later)
  - Intuitively: security = protecting information in some way
  - What is the threat model? Who is attacking the scheme, or who are we trying to protect against? What kind of attacks?

- Two flavors: concrete and asymptotic (of the above)

Defining Security

- What does it mean for something to be "secure"?
  - Confidentiality (Security/Privacy)
  - Integrity (Authenticity/Authentication/Correctness)
  - Can define more things (e.g., zero-knowledge as we'll see later)
  - Intuitively: security = protecting information in some way
  - What is the threat model? Who is attacking the scheme, or who are we trying to protect against? What kind of attacks?

- Two flavors: concrete and asymptotic (of the above)

- Intuitively: concrete, there is a specific type of adversary you want to protect against (this is very hand-wavy and oversimplified)

Defining Security

- What does it mean for something to be "secure"?
  - Confidentiality (Security/Privacy)
  - Integrity (Authenticity/Authentication/Correctness)
  - Can define more things (e.g., zero-knowledge as we'll see later)
  - Intuitively: security = protecting information in some way
  - What is the threat model? Who is attacking the scheme, or who are we trying to protect against? What kind of attacks?
- Two flavors: concrete and asymptotic (of the above)
- Intuitively: concrete, there is a specific type of adversary you want to protect against (this is very hand-wavy and oversimplified)
- Asymptotic: trying to catch-all for all possible adversaries (for some class of adversaries)

Example: Message Authentication Codes

Example: Message Authentication Codes

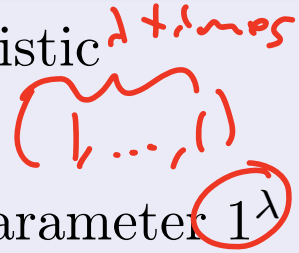
Definition 1 (Message Authentication Code (MAC))

A *message authentication code* (or *MAC*) consists of three probabilistic polynomial-time (PPT) algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ such that:

Example: Message Authentication Codes

Definition 1 (Message Authentication Code (MAC))

A *message authentication code* (or *MAC*) consists of three probabilistic polynomial-time (PPT) algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ such that:



1. The *key-generation algorithm* Gen takes as input the security parameter 1^λ and outputs a key k such that $|k| \geqslant \lambda$, which we denote as $k \leftarrow \mathsf{Gen}(1^\lambda)$;

Example: Message Authentication Codes

Definition 1 (Message Authentication Code (MAC))

A *message authentication code* (or *MAC*) consists of three probabilistic polynomial-time (PPT) algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ such that:

1. The *key-generation algorithm* Gen takes as input the security parameter 1^λ and outputs a key k such that $|k| \geqslant \lambda$, which we denote as $k \leftarrow \mathsf{Gen}(1^\lambda)$;
2. The *tag-generation algorithm* Mac takes as input a key k and a message $m \in \{0,1\}^*$ and outputs a tag t, which we denote as $t \leftarrow \mathsf{Mac}_k(m)$; and

Probabilistic

Example: Message Authentication Codes

Definition 1 (Message Authentication Code (MAC))

A *message authentication code* (or *MAC*) consists of three probabilistic polynomial-time (PPT) algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ such that:

1. The *key-generation algorithm* Gen takes as input the security parameter 1^λ and outputs a key k such that $|k| \geqslant \lambda$, which we denote as $k \leftarrow \mathsf{Gen}(1^\lambda)$;
2. The *tag-generation algorithm* Mac takes as input a key k and a message $m \in \{0,1\}^*$ and outputs a tag t, which we denote as $t \leftarrow \mathsf{Mac}_k(m)$; and
3. The deterministic *verification algorithm* Verify takes as input a key k, a message m, and a tag t, and outputs a bit b such that $b = 1$ denotes that t is a *valid* tag for message m under key k, and otherwise $b = 0$ denotes that t is an *invalid* tag. We denote this as $b := \mathsf{Verify}_k(m, t)$.

Example: Message Authentication Codes

Definition 1 (Message Authentication Code (MAC))

A *message authentication code* (or *MAC*) consists of three probabilistic polynomial-time (PPT) algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ such that:

1. The *key-generation algorithm* Gen takes as input the security parameter 1^λ and outputs a key k such that $|k| \geqslant \lambda$, which we denote as $k \leftarrow \mathsf{Gen}(1^\lambda)$;
2. The *tag-generation algorithm* Mac takes as input a key k and a message $m \in \{0,1\}^*$ and outputs a tag t, which we denote as $t \leftarrow \mathsf{Mac}_k(m)$; and
3. The deterministic *verification algorithm* Verify takes as input a key k, a message m, and a tag t, and outputs a bit b such that $b = 1$ denotes that t is a *valid* tag for message m under key k, and otherwise $b = 0$ denotes that t is an *invalid* tag. We denote this as $b := \mathsf{Verify}_k(m, t)$.

We require that for every $\lambda \in \mathbb{Z}^+$, every $k \in \mathsf{Gen}(1^\lambda)$, and every $m \in \{0,1\}^*$, it holds that $\mathsf{Verify}_k(m, \mathsf{Mac}_k(m)) = 1$.

same message

= key

Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.

Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

To verify that a message is *authentic*

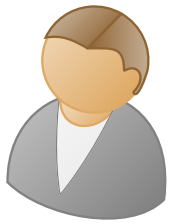
Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

To verify that a message is *authentic*

Alice



Bob

Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

To verify that a message is *authentic*

Alice

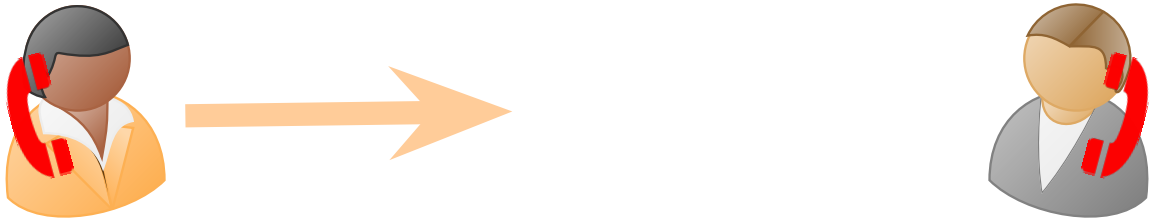
Bob

Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

To verify that a message is *authentic*

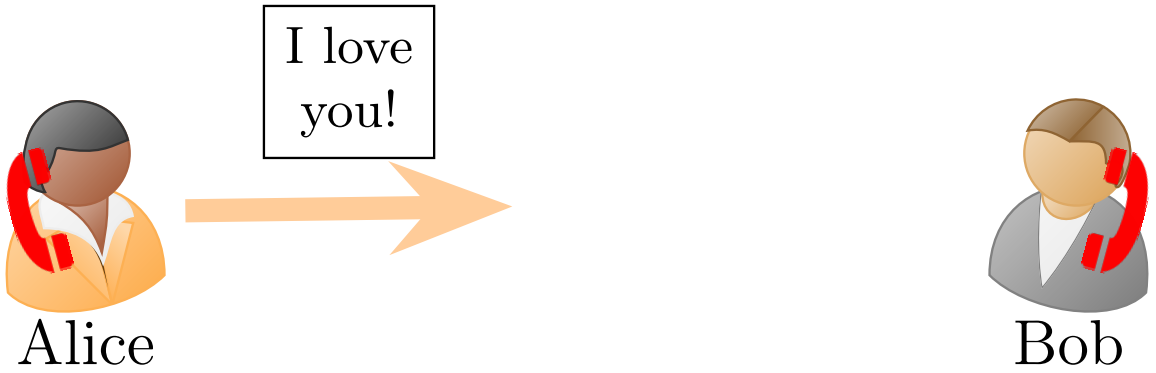


Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

To verify that a message is *authentic*

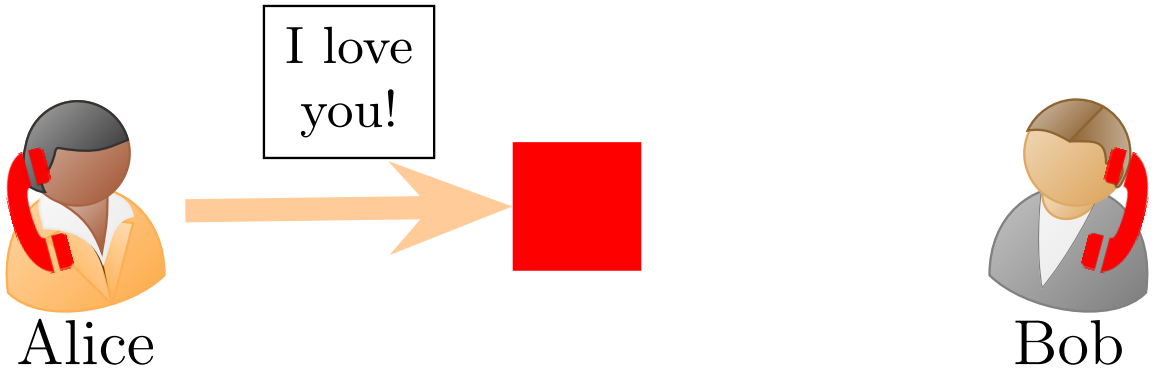


Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

To verify that a message is *authentic*

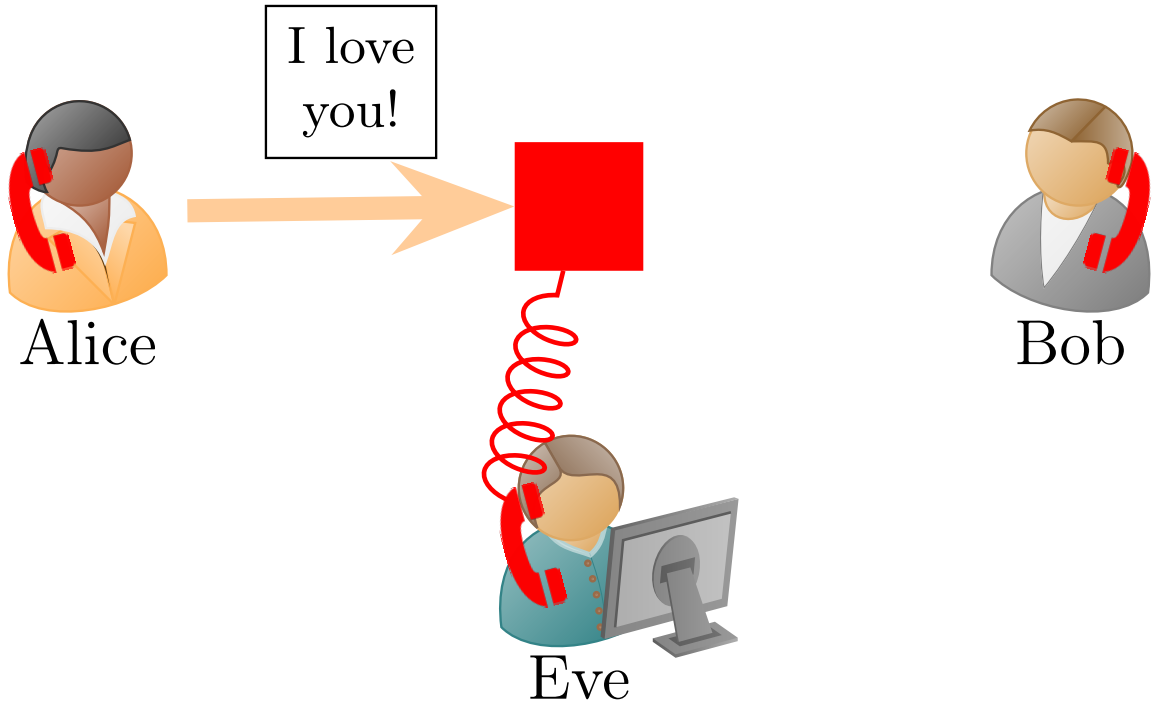


Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

To verify that a message is *authentic*

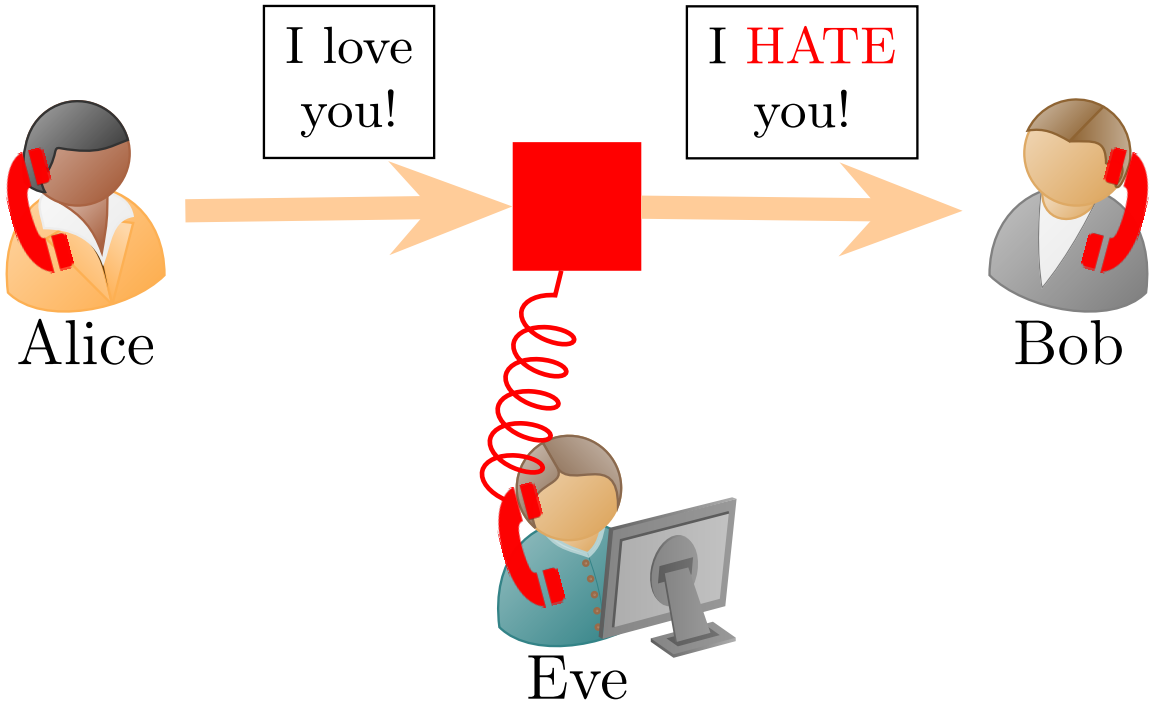


Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.
- How would you define security for a MAC?

Why use a MAC?

To verify that a message is *authentic*



Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.

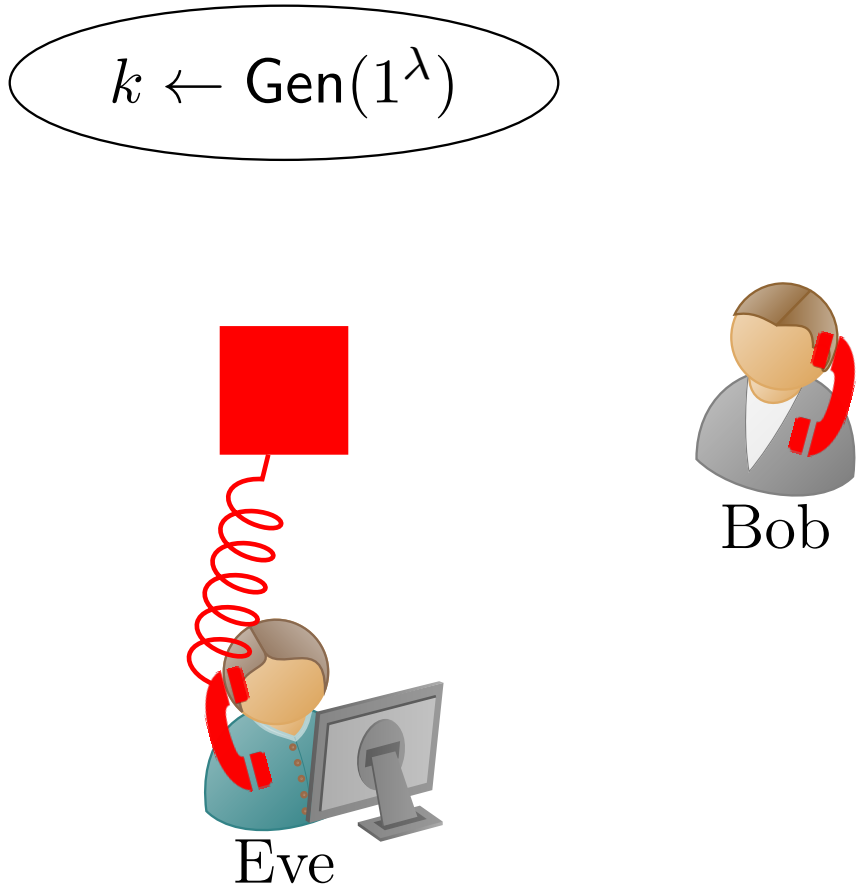
Alice

Eve

Bob

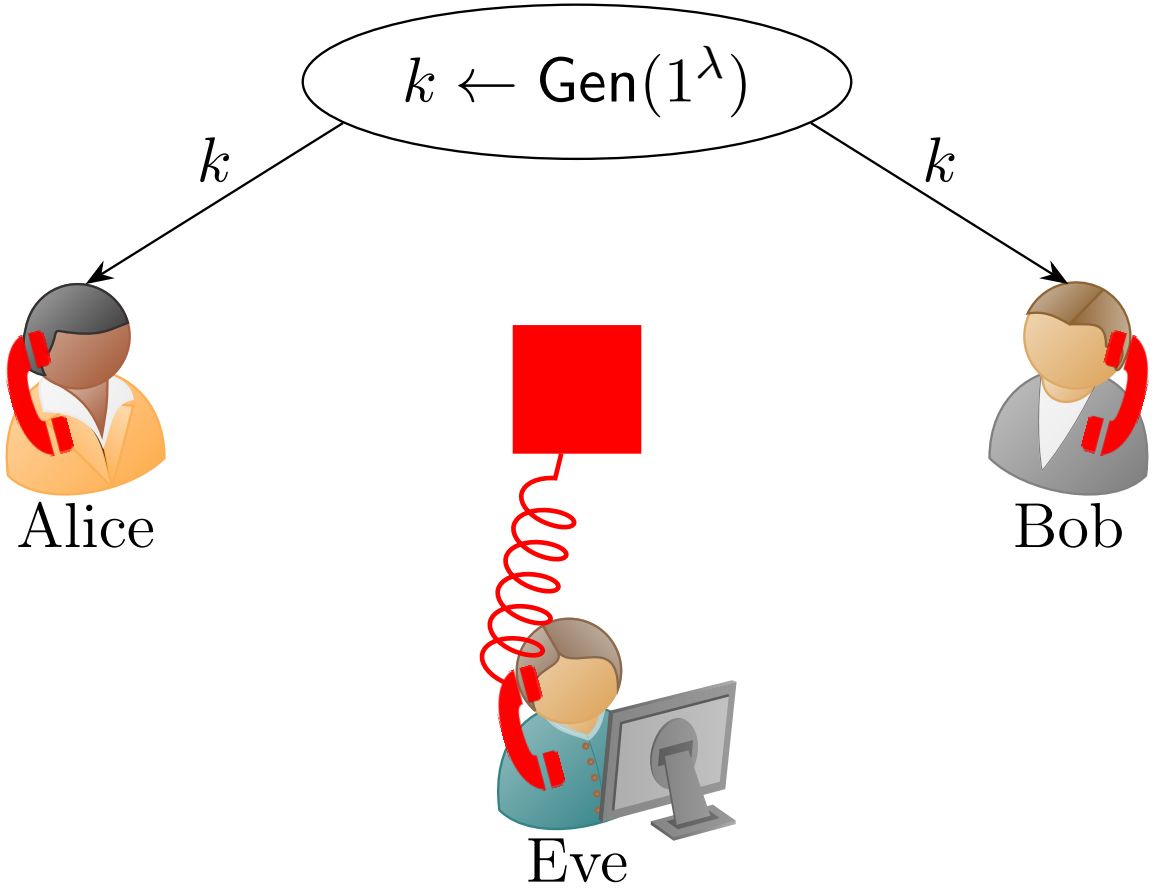
Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.



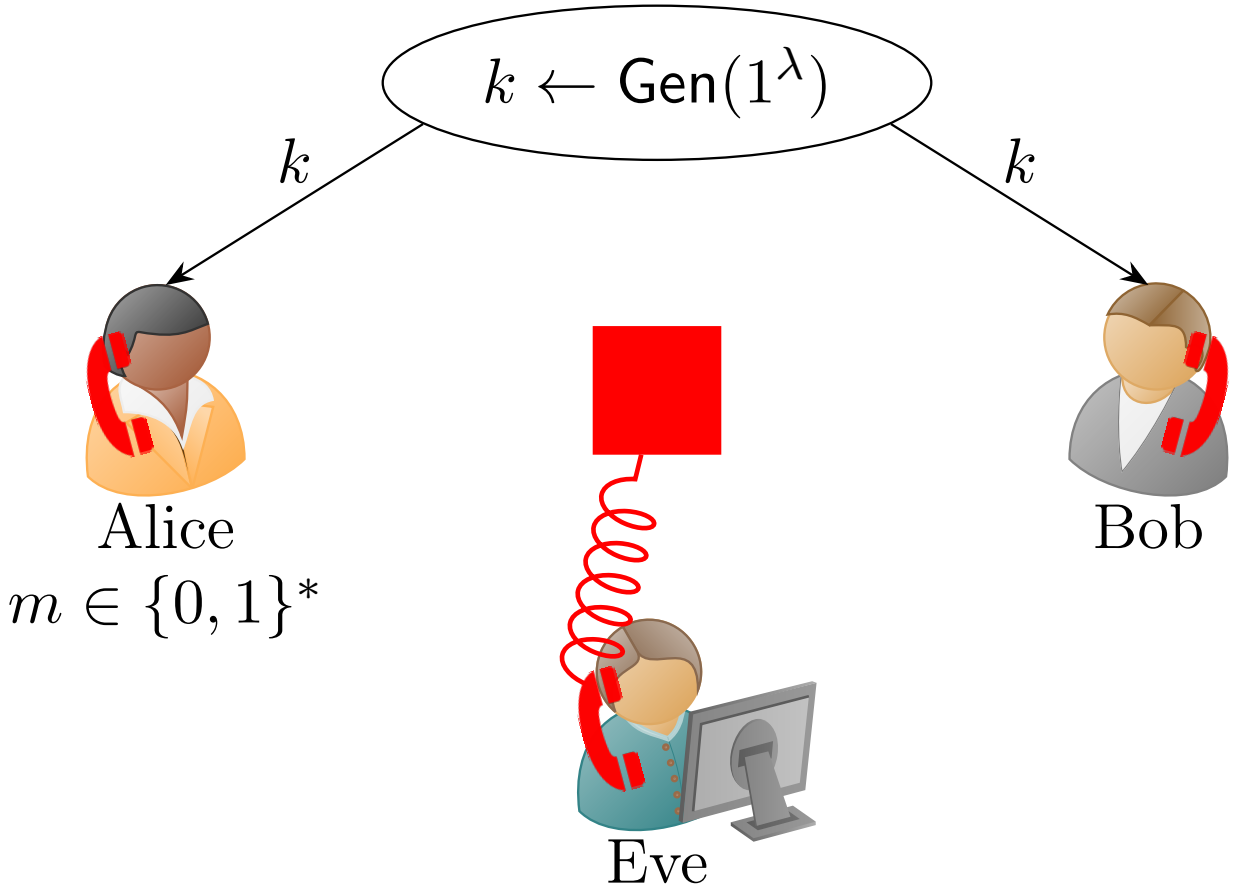
Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.



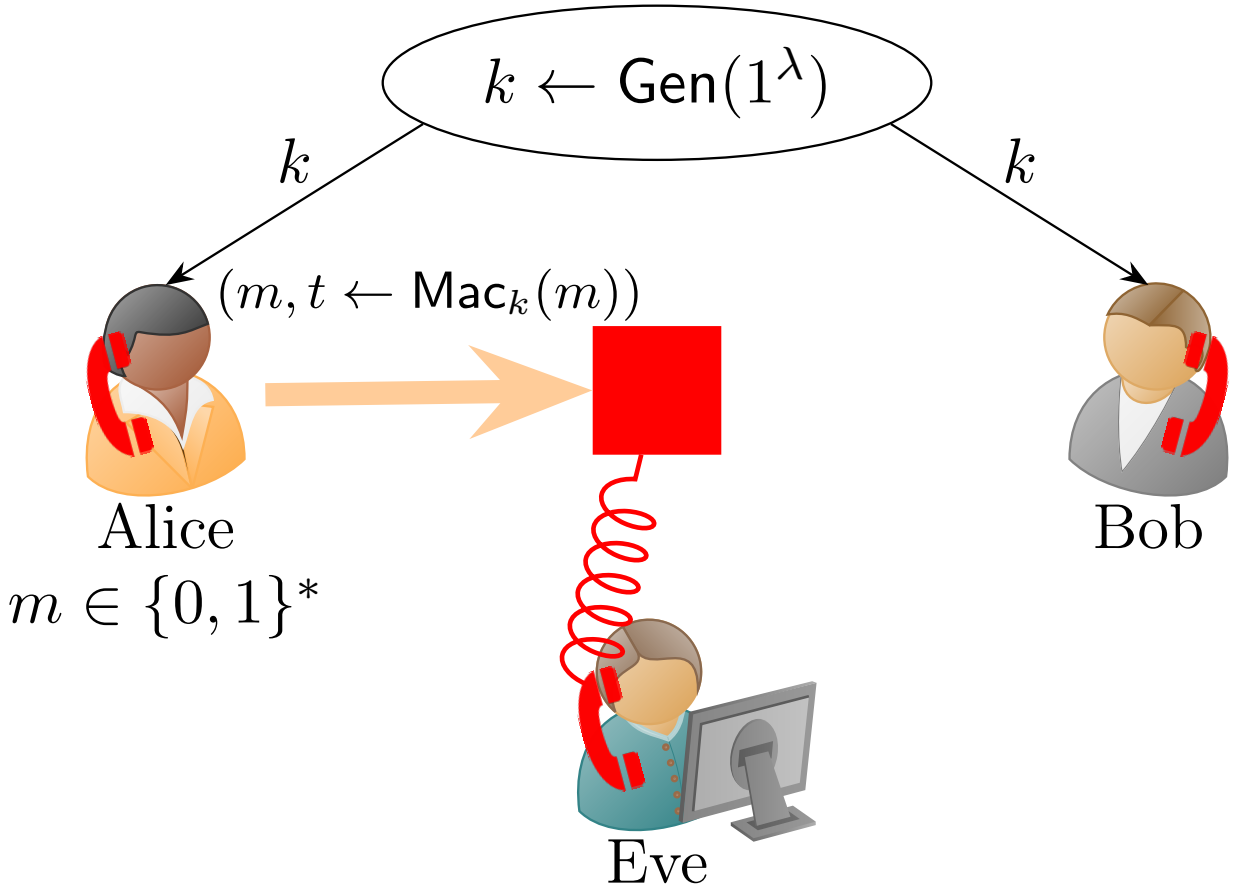
Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.



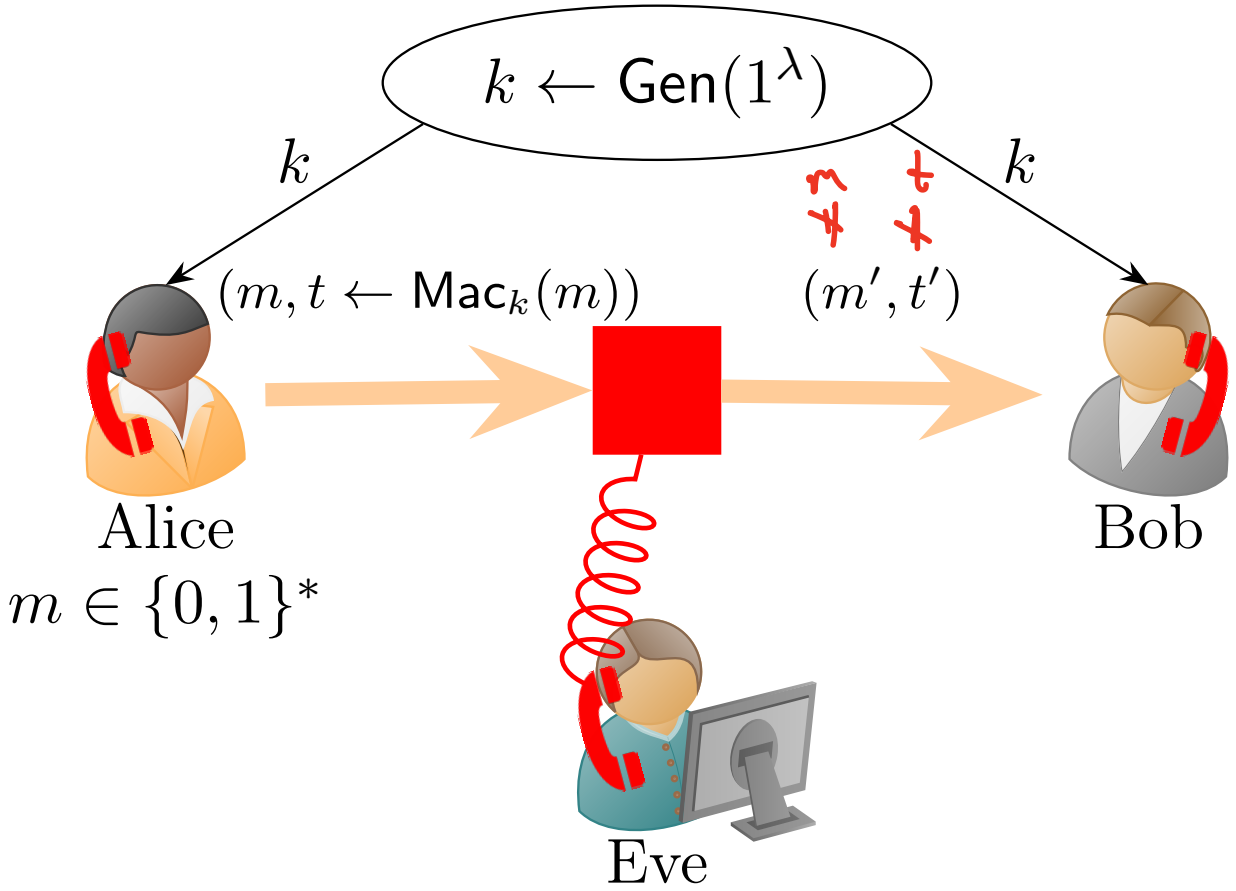
Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.



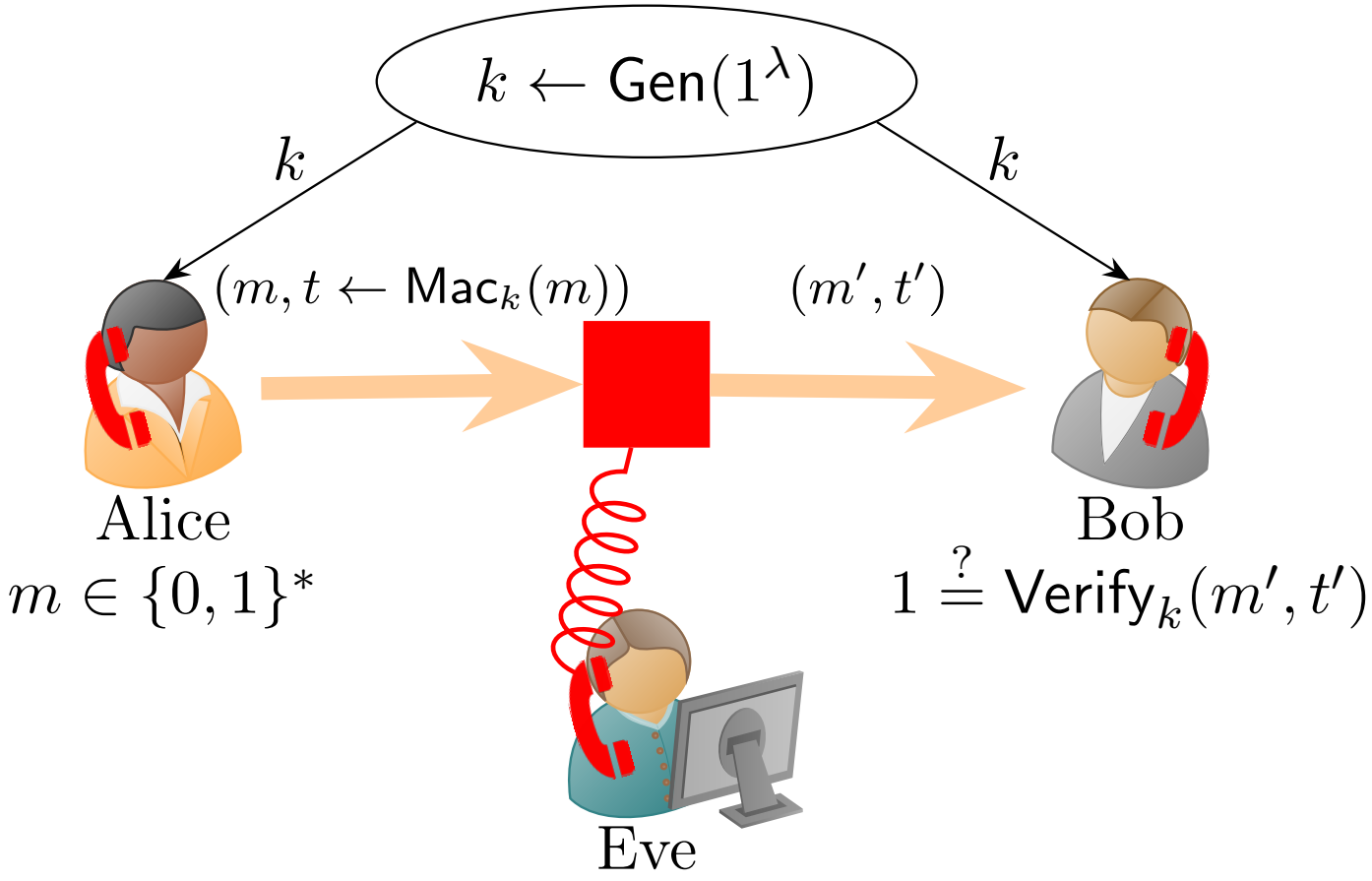
Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.



Security of MACs

- Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC.



Eve does not have Key K

Security of MACs

Intuition

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if an adversary cannot forge a tag for a message of their choice *without* access to the secret key.

Security of MACs

Intuition

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if an adversary cannot forge a tag for a message of their choice *without* access to the secret key.

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

Security of MACs

Intuition

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if an adversary cannot forge a tag for a message of their choice *without* access to the secret key.

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

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

Security of MACs

Intuition

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if an adversary cannot forge a tag for a message of their choice *without* access to the secret key.

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

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.

Security of MACs

Intuition

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if an adversary cannot forge a tag for a message of their choice *without* access to the secret key.

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

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).

Security of MACs

Intuition

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if an adversary cannot forge a tag for a message of their choice *without* access to the secret key.

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

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).

Security of MACs

Intuition

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if an adversary cannot forge a tag for a message of their choice *without* access to the secret key.

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

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).

- Also known as *existential unforgeability under adaptive chosen-message attacks*

Security of MACs

Mac-forge$_{\mathcal{A},\Pi}(\lambda)$ as a Security Game

Security of MACs

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

Security of MACs

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

Security of MACs

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

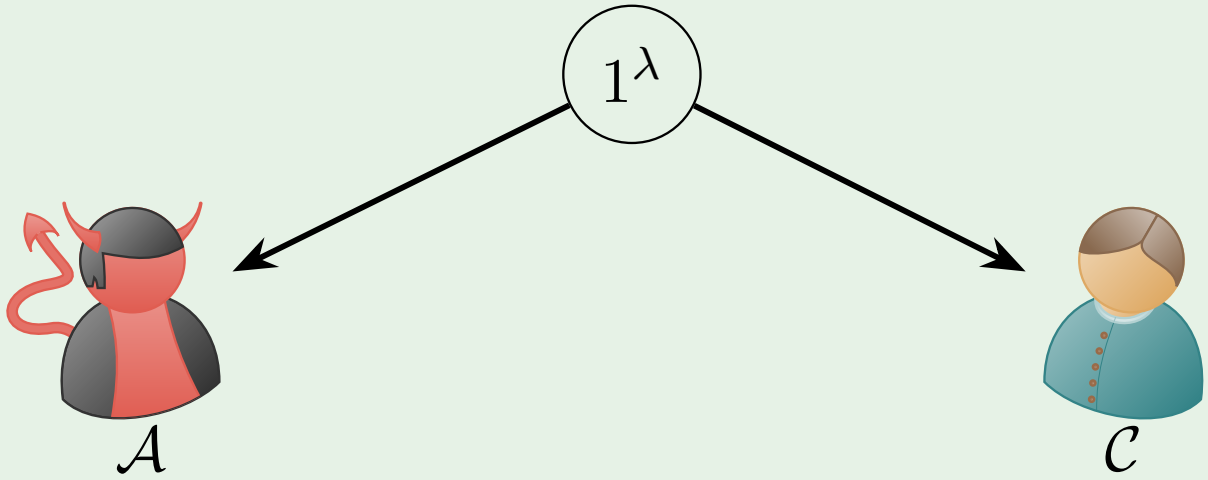
1^λ

$\mathcal{A}$

$\mathcal{C}$

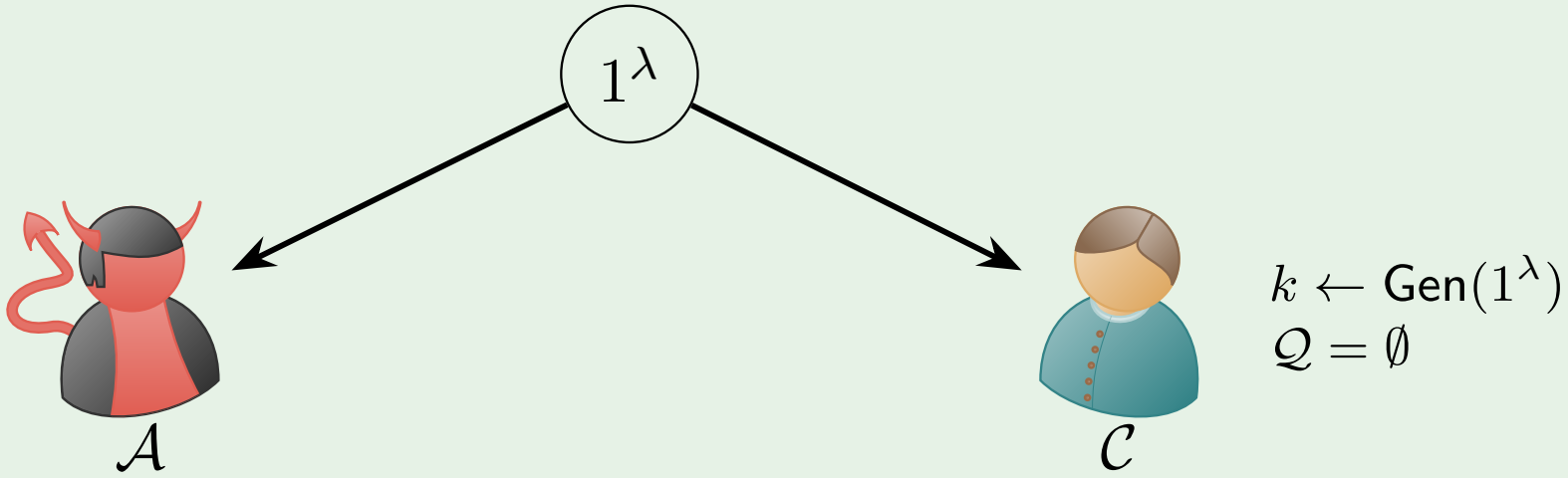
Security of MACs

Mac-forge$_{\mathcal{A},\Pi}(\lambda)$ as a Security Game



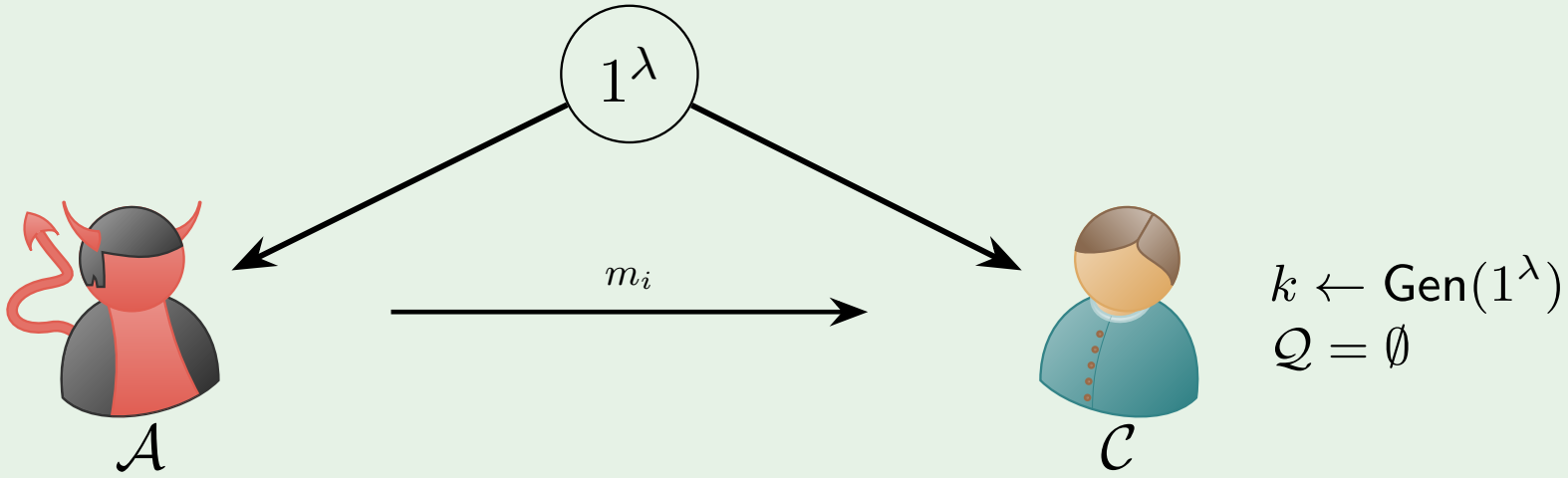
Security of MACs

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



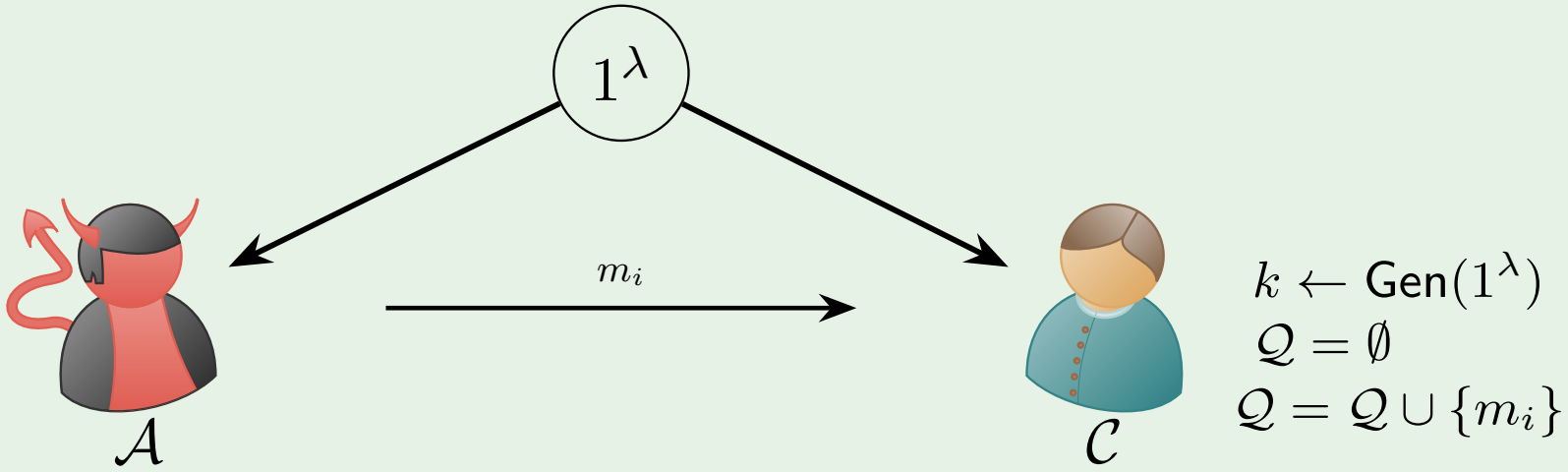
Security of MACs

Mac-forge$_{\mathcal{A},\Pi}(\lambda)$ as a Security Game



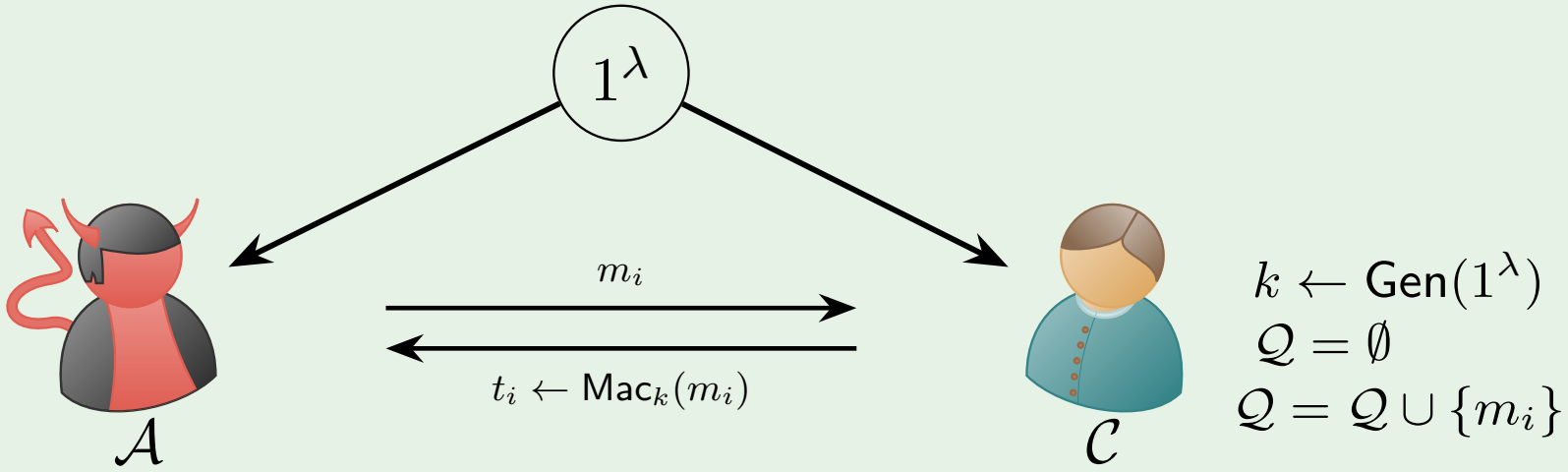
Security of MACs

Mac-forge$_{\mathcal{A},\Pi}(\lambda)$ as a Security Game



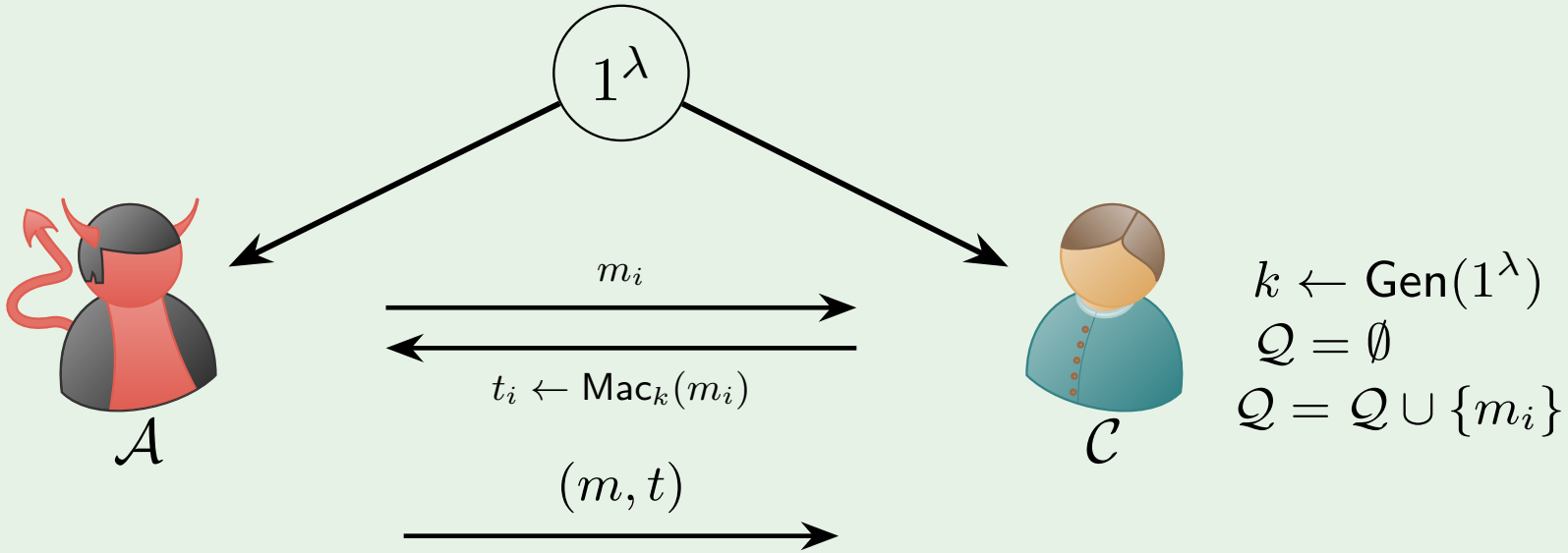
Security of MACs

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



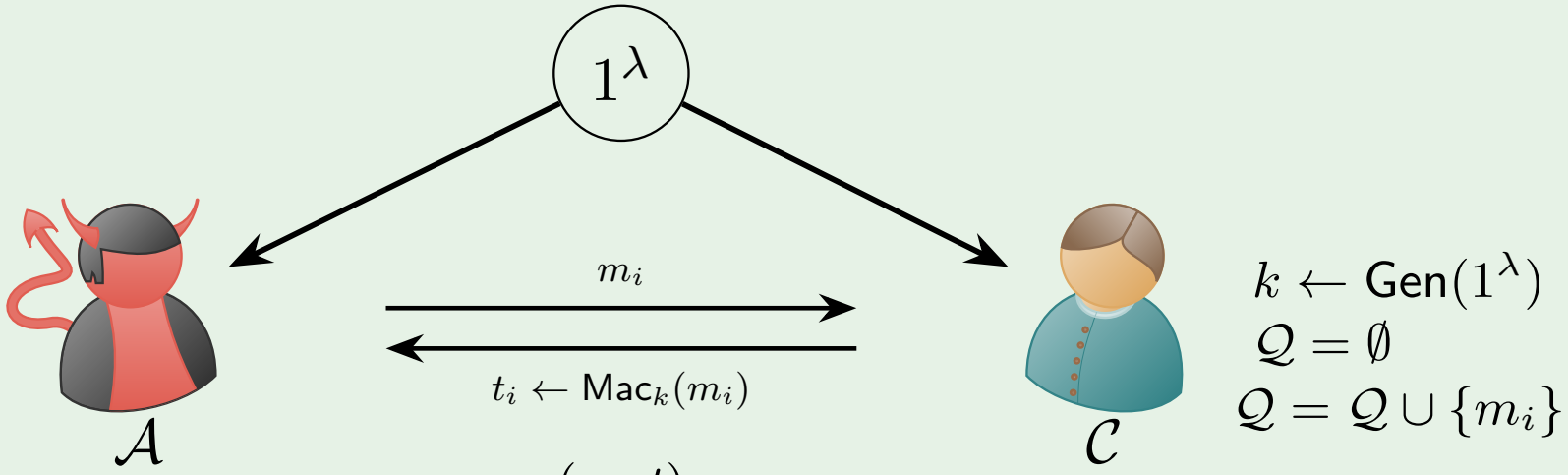
Security of MACs

Mac-forge$_{\mathcal{A},\Pi}(\lambda)$ as a Security Game



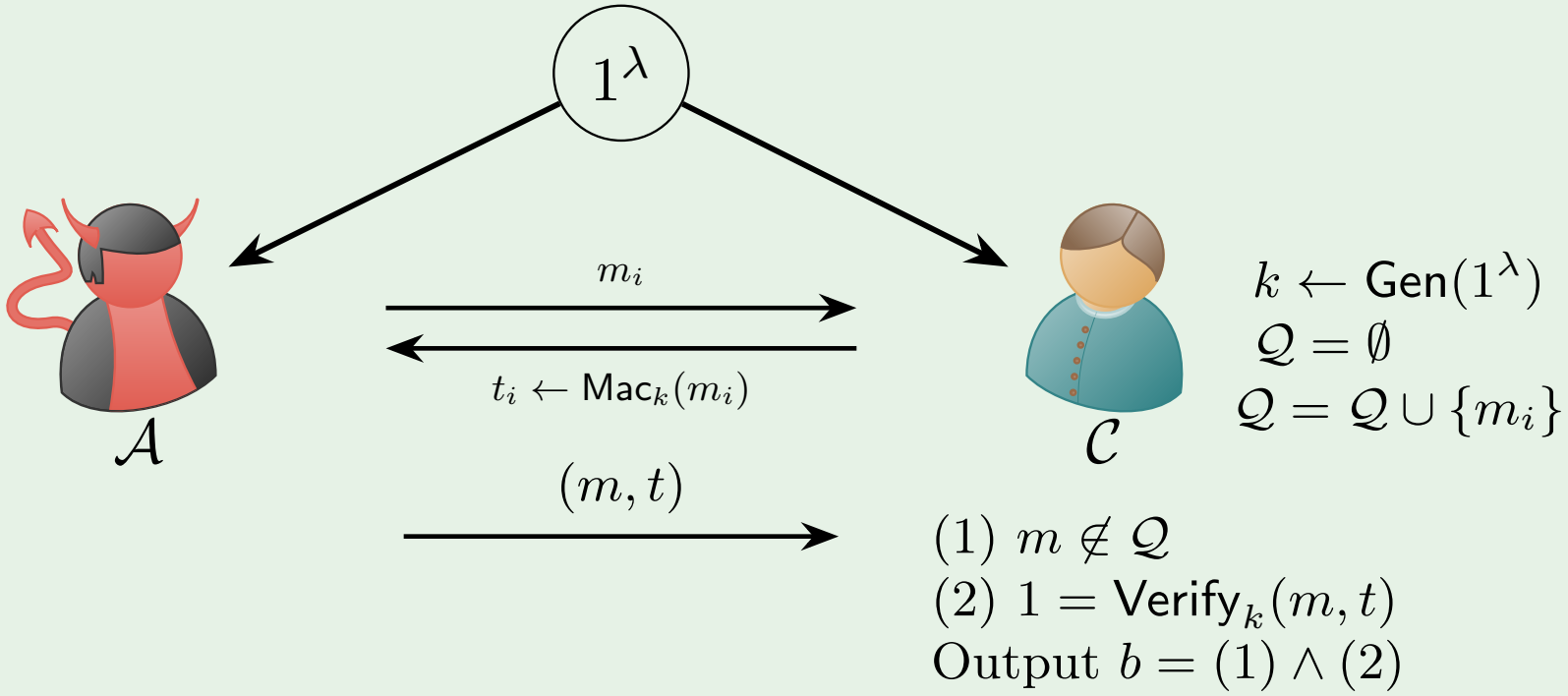
Security of MACs

Mac-forge$_{\mathcal{A},\Pi}(\lambda)$ as a Security Game



Security of MACs

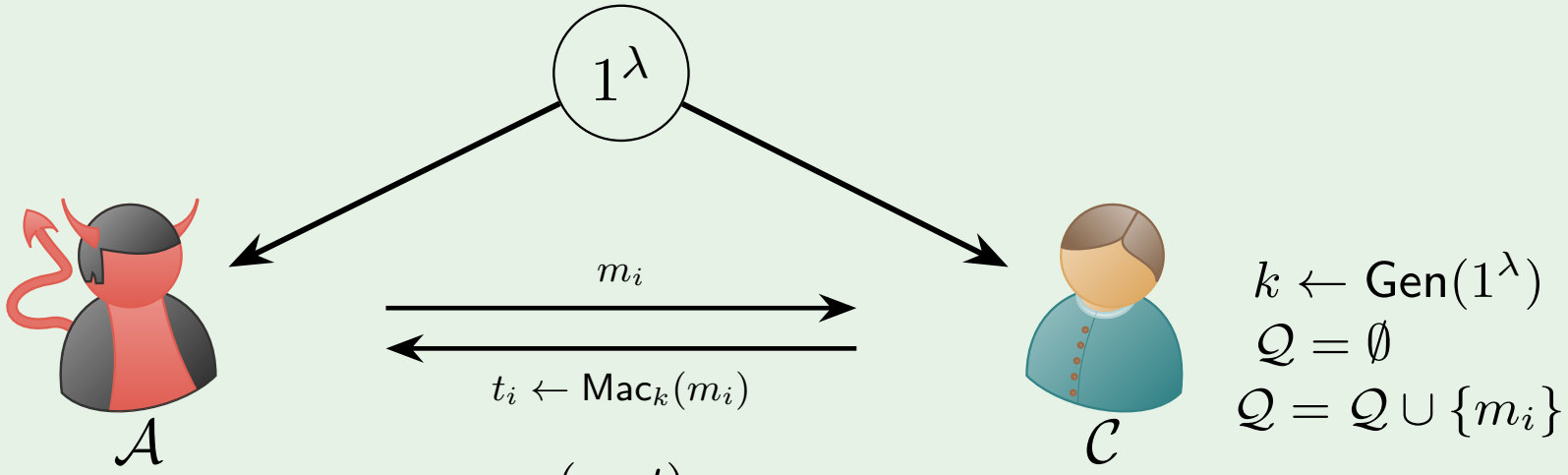
$\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda)$ as a Security Game



- How many queries?

Security of MACs

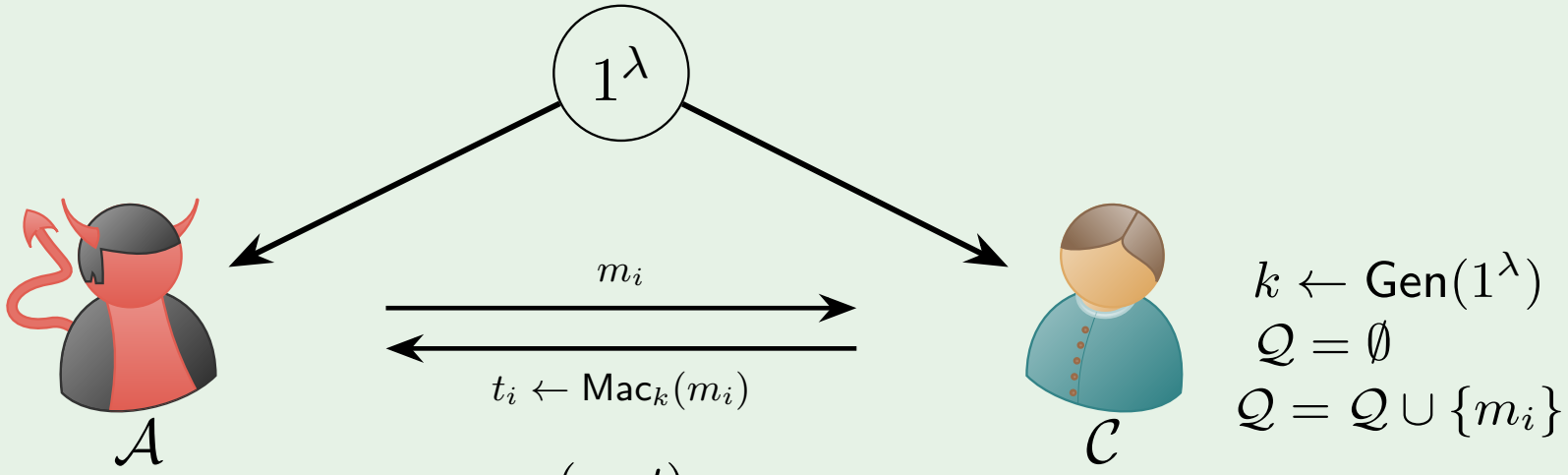
Mac-forge$_{\mathcal{A},\Pi}(\lambda)$ as a Security Game



- How many queries?
- How "powerful" is $\mathcal{A}$?

Security of MACs

Mac-forge$_{\mathcal{A},\Pi}(\lambda)$ as a Security Game



- How many queries?
- How "powerful" is $\mathcal{A}$?
- How likely is it for $\mathcal{A}$ to win the game?

Concrete Security

Concrete Security

Concrete Security (Informal)

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$,

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$, *every* adversary running in time at most $t(\lambda)$

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$, *every* adversary running in time at most $t(\lambda)$ succeeds in breaking the scheme with probability at most $\varepsilon(\lambda)$.

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$, *every* adversary running in time at most $t(\lambda)$ succeeds in breaking the scheme with probability at most $\varepsilon(\lambda)$.

- Example: $t = 2^{60}$ CPU cycles

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$, *every* adversary running in time at most $t(\lambda)$ succeeds in breaking the scheme with probability at most $\varepsilon(\lambda)$.

- Example: $t = 2^{60}$ CPU cycles
  - Roughly 9 years on a 4GHz processor
  - $<$ 1 minute on fastest supercomputer (in parallel) (as of 2023)

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$, *every* adversary running in time at most $t(\lambda)$ succeeds in breaking the scheme with probability at most $\varepsilon(\lambda)$.

- Example: $t = 2^{60}$ CPU cycles
  - Roughly 9 years on a 4GHz processor
  - $<$ 1 minute on fastest supercomputer (in parallel) (as of 2023)

- Need to specify what "break" means

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$, *every* adversary running in time at most $t(\lambda)$ succeeds in breaking the scheme with probability at most $\varepsilon(\lambda)$.

- Example: $t = 2^{60}$ CPU cycles
  - Roughly 9 years on a 4GHz processor
  - $<$ 1 minute on fastest supercomputer (in parallel) (as of 2023)
- Need to specify what "break" means
- Important metric in practice

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$, *every* adversary running in time at most $t(\lambda)$ succeeds in breaking the scheme with probability at most $\varepsilon(\lambda)$.

- Example: $t = 2^{60}$ CPU cycles
  - Roughly 9 years on a 4GHz processor
  - $<$ 1 minute on fastest supercomputer (in parallel) (as of 2023)
- Need to specify what "break" means
- Important metric in practice
  - Caveat 1: difficult to prove/provide these precise statements!

Concrete Security

Concrete Security (Informal)

A scheme Π is (t, ε)-secure if for every $\lambda \in \mathbb{Z}^+$, *every* adversary running in time at most $t(\lambda)$ succeeds in breaking the scheme with probability at most $\varepsilon(\lambda)$.

- Example: $t = 2^{60}$ CPU cycles
  - Roughly 9 years on a 4GHz processor
  - $<$ 1 minute on fastest supercomputer (in parallel) (as of 2023)
- Need to specify what "break" means
- Important metric in practice
  - Caveat 1: difficult to prove/provide these precise statements!
  - Caveat 2: hardware continuously improves over time

Concrete Security of MACs

Definition 2 ((t, q, ε)-secure MAC)

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is (t, q, ε)-secure if for every $\lambda \in \mathbb{Z}^+$ and every adversary $\mathcal{A}$ running in time at most $t(\lambda)$ and making at most $q(\lambda)$ oracle queries/challenges, it holds that

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1] \leqslant \varepsilon(\lambda).$$

Asymptotic Security

Asymptotic Security

- Concrete Security is often *messy*

Asymptotic Security

- Concrete Security is often *messy*
  - Recall the caveats from before

Asymptotic Security

- Concrete Security is often *messy*
  - Recall the caveats from before
- Would much rather have a catch-all statement for security

Asymptotic Security

- Concrete Security is often *messy*
  - Recall the caveats from before
- Would much rather have a catch-all statement for security

Asymptotic Security (Informal)

A scheme Π is secure if every probabilistic polynomial-time (PPT) adversary succeeds in breaking the scheme with negligible probability.

Asymptotic Security

- Concrete Security is often *messy*
  - Recall the caveats from before
- Would much rather have a catch-all statement for security

Asymptotic Security (Informal)

A scheme Π is secure if every probabilistic polynomial-time (PPT) adversary succeeds in breaking the scheme with negligible probability.

- Intuition: if we choose the security parameter λ to be sufficiently large, then *any* PPT adversary will succeed with very small (negligible) probability!

Asymptotic Security

- Concrete Security is often *messy*
  - Recall the caveats from before
- Would much rather have a catch-all statement for security

Asymptotic Security (Informal)

A scheme Π is secure if every probabilistic polynomial-time (PPT) adversary succeeds in breaking the scheme with negligible probability.

- Intuition: if we choose the security parameter λ to be sufficiently large, then *any* PPT adversary will succeed with very small (negligible) probability!
  - Statements like this are much easier to prove (and we'll focus on this in the course)

Asymptotic Security

- Concrete Security is often *messy*
  - Recall the caveats from before
- Would much rather have a catch-all statement for security

Asymptotic Security (Informal)

A scheme Π is secure if every probabilistic polynomial-time (PPT) adversary succeeds in breaking the scheme with negligible probability.

- Intuition: if we choose the security parameter λ to be sufficiently large, then *any* PPT adversary will succeed with very small (negligible) probability!
  - Statements like this are much easier to prove (and we'll focus on this in the course)
  - Don't have to worry about hardware improvements.

Asymptotic Security of MACs

Definition 3 (Secure MAC)

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda).$$

Asymptotic Security of MACs

Definition 3 (Secure MAC)

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda).$$

Notice:

Asymptotic Security of MACs

Definition 3 (Secure MAC)

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda).$$

Notice:

- We don't have to worry about the number of queries, they are implicitly bounded by some polynomial

Asymptotic Security of MACs

Definition 3 (Secure MAC)

A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is secure if for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{Mac\text{-}forge}_{\mathcal{A},\Pi}(\lambda) = 1] \leqslant \mathsf{negl}(\lambda).$$

Notice:

- We don't have to worry about the number of queries, they are implicitly bounded by some polynomial
- We just need to choose the security parameter λ large enough for the adversary to basically have no chance of winning!