Computational Security

601.442/642 Modern Cryptography

29th January 2026

Announcement

- Homework 1 due **today**.
- Homework 2 will be out today and will be due next Thursday (5th Feb).

Recap: Limitations of Perfect Security

Theorem (Shannon): Any perfectly secure encryption scheme with key space $\mathscr{K}$ and message space $\mathscr{M}$ satisfies

$$|\mathscr{K}| \geq |\mathscr{M}|.$$

Perfect security is too strong. Can we weaken the definition?

Recap: What Makes Perfect Security So Strong?

- **Perfect Security:** Encryptions of any two messages are identically distributed i.e., the adversary cannot distinguish between the ciphertexts.

Recap: What Makes Perfect Security So Strong?

- **Perfect Security:** Encryptions of any two messages are identically distributed i.e., the adversary cannot distinguish between the ciphertexts.
  - **Statistical Security:** The adversary can distinguish between encryptions of two different messages with a small probability ϵ.

Recap: What Makes Perfect Security So Strong?

- **Perfect Security:** Encryptions of any two messages are identically distributed i.e., the adversary cannot distinguish between the ciphertexts.

  - **Statistical Security:** The adversary can distinguish between encryptions of two different messages with a small probability ϵ.

Probability (ϵ)	**Event**
2^{-10}	Full house in 5-card poker
2^{-20}	Royal flush in 5-card poker
2^{-28}	Winning this week's Powerball jackpot
2^{-40}	Royal flush in two consecutive poker games
2^{-60}	Next meteorite that hits Earth lands on this slide

Recap: What Makes Perfect Security So Strong?

- **Perfect Security:** Encryptions of any two messages are identically distributed i.e., the adversary cannot distinguish between the ciphertexts.

  - **Statistical Security:** The adversary can distinguish between encryptions of two different messages with a small probability ϵ.

Probability (ϵ)	**Event**
2^{-10}	Full house in 5-card poker
2^{-20}	Royal flush in 5-card poker
2^{-28}	Winning this week's Powerball jackpot
2^{-40}	Royal flush in two consecutive poker games
2^{-60}	Next meteorite that hits Earth lands on this slide

An attack that succeeds with small probability ($\approx 2^{-60}$) is not a practical threat.

Recap: What Makes Perfect Security So Strong?

- **Perfect Security:** Encryptions of any two messages are identically distributed i.e., the adversary cannot distinguish between the ciphertexts.

  - **Statistical Security:** The adversary can distinguish between encryptions of two different messages with a small probability ϵ.

  - Shannon's theorem can be extended to show that statistically secure encryption schemes still require long keys.

Recap: What Makes Perfect Security So Strong?

- **Perfect Security:** Encryptions of any two messages are identically distributed i.e., the adversary cannot distinguish between the ciphertexts.
  - **Statistical Security:** The adversary can distinguish between encryptions of two different messages with a small probability ϵ.
  - Shannon's theorem can be extended to show that statistically secure encryption schemes still require long keys.
- **Perfect Security:** Even an attacker that brute forces the key does not learn anything about the plaintext.

Recap: What Makes Perfect Security So Strong?

- **Perfect Security:** Encryptions of any two messages are identically distributed i.e., the adversary cannot distinguish between the ciphertexts.
  - **Statistical Security:** The adversary can distinguish between encryptions of two different messages with a small probability ϵ.
  - Shannon's theorem can be extended to show that statistically secure encryption schemes still require long keys.
- **Perfect Security:** Even an attacker that brute forces the key does not learn anything about the plaintext.
  - Brute forcing λ-bit keys requires $O(2^\lambda)$ computations.

Recap: What Makes Perfect Security So Strong?

- **Perfect Security:** Encryptions of any two messages are identically distributed i.e., the adversary cannot distinguish between the ciphertexts.
  - **Statistical Security:** The adversary can distinguish between encryptions of two different messages with a small probability ϵ.
  - Shannon's theorem can be extended to show that statistically secure encryption schemes still require long keys.
- **Perfect Security:** Even an attacker that brute forces the key does not learn anything about the plaintext.
  - Brute forcing λ-bit keys requires $O(2^\lambda)$ computations.
  - Are brute force attacks feasible?

Cost of Computation

- One way to measure the cost of computation is through the monetary value required to carry it out.

CPU Cycles	Approx. Cost	Reference
2^{50}	\$3.50	Cup of coffee
2^{55}	\$100	Tickets to Portland Trailblazers game
2^{65}	\$130,000	Median home price in Oshkosh, WI
2^{75}	\$130 million	Average budget of one of the Harry Potter movies
2^{92}	\$20 trillion	GDP of the United States
2^{99}	\$2 quadrillion	All human economic activity since 300,000 BC
2^{128}	???	A billion human civilizations' worth of effort

Cost of Computation

How large is 2^{128} computations?

Even if every atom in the observable universe ($\sim 10^{80}$) were a computer performing 10^9 computations per second, it would still take billions of years to complete 2^{128} computations.

Cost of Computation

How large is 2^{128} computations?

Even if every atom in the observable universe ($\sim 10^{80}$) were a computer performing 10^9 computations per second, it would still take billions of years to complete 2^{128} computations.

An attack that requires a large number of computations ($\approx 2^{128}$) is not a practical threat.

Computational Security

- Modern cryptography is based on computational security

Computational Security

- Modern cryptography is based on computational security
  - Security is only ensured against adversaries that run for a feasible amount of time.

Computational Security

- Modern cryptography is based on computational security
  - Security is only ensured against adversaries that run for a feasible amount of time.
  - Adversaries can potentially succeed with a very small probability.

Computational Security

- Modern cryptography is based on computational security
  - Security is only ensured against adversaries that run for a feasible amount of time.
  - Adversaries can potentially succeed with a very small probability.
- **Goal:** Overcome the limitations of perfect security (and much more!).

Computational Security

- Modern cryptography is based on computational security
  - Security is only ensured against adversaries that run for a feasible amount of time.
  - Adversaries can potentially succeed with a very small probability.
- **Goal:** Overcome the limitations of perfect security (and much more!).
- Both relaxations of perfect security are necessary!

Computational Security

- Modern cryptography is based on computational security
  - Security is only ensured against adversaries that run for a feasible amount of time.
  - Adversaries can potentially succeed with a very small probability.
- **Goal:** Overcome the limitations of perfect security (and much more!).
- Both relaxations of perfect security are necessary!

  Why?

Computational Security

- Modern cryptography is based on computational security
  - Security is only ensured against adversaries that run for a feasible amount of time.

    Don’t worry about attacks that are as expensive as brute-force attacks!
  - Adversaries can potentially succeed with a very small probability.

- **Goal:** Overcome the limitations of perfect security (and much more!).
- Both relaxations of perfect security are necessary!

  Why?

Computational Security

- Modern cryptography is based on computational security
    - Security is only ensured against adversaries that run for a feasible amount of time.

      Don't worry about attacks that are as expensive as brute-force attacks!
    - Adversaries can potentially succeed with a very small probability.

      Don't worry about the adversary blindly guessing the key!
- **Goal:** Overcome the limitations of perfect security (and much more!).
- Both relaxations of perfect security are necessary!

  Why?

The Concrete Security Approach

A scheme is (T, ϵ)-secure if any adversary running for time at most T succeeds in breaking the scheme with probability at most ϵ.

The Concrete Security Approach

A scheme is (T, ϵ)-secure if any adversary running for time at most T succeeds in breaking the scheme with probability at most ϵ.

(T, ϵ)-Computational Indistinguishability

Two distributions X and Y are (T, ϵ)-computationally indistinguishable if for all adversaries A that run in time at most T,

$$\left| \Pr_{x \leftarrow X}\left[A(x) = 1\right] - \Pr_{y \leftarrow Y}\left[A(y) = 1\right] \right| \leq \epsilon,$$

where the probability is over sampling from the distributions X and Y, and the randomness of A.

The Concrete Security Approach

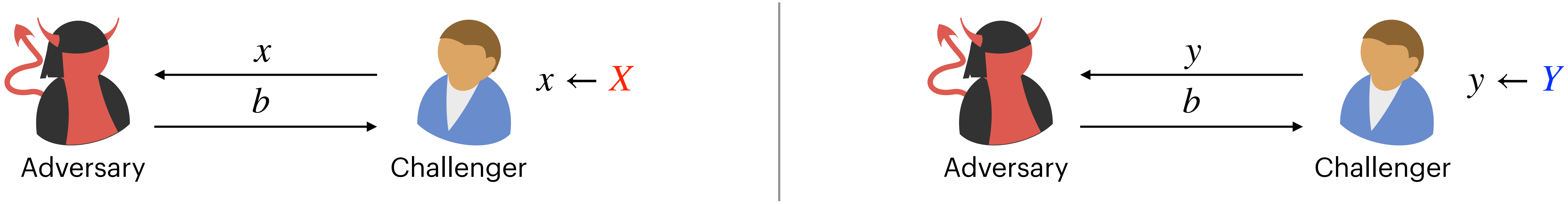
A scheme is (T, ϵ)-secure if any adversary running for time at most T succeeds in breaking the scheme with probability at most ϵ.

(T, ϵ)-Computational Indistinguishability

Two distributions X and Y are (T, ϵ)-computationally indistinguishable if for all adversaries A that run in time at most T,

$$\left| \Pr_{x \leftarrow X}\left[A(x) = 1\right] - \Pr_{y \leftarrow Y}\left[A(y) = 1\right] \right| \leq \epsilon,$$

where the probability is over sampling from the distributions X and Y, and the randomness of A.



Adversary cannot tell X and Y apart except with small probability.

The Concrete Security Approach

A scheme is (T, ϵ)-secure if any adversary running for time at most T succeeds in breaking the scheme with probability at most ϵ.

(T, ϵ)-Computational Indistinguishability

Two distributions X and Y are (T, ϵ)-computationally indistinguishable if for all adversaries A that run in time at most T,

$$\left| \Pr_{x \leftarrow X}\left[A(x) = 1\right] - \Pr_{y \leftarrow Y}\left[A(y) = 1\right] \right| \leq \epsilon,$$

where the probability is over sampling from the distributions X and Y, and the randomness of A.

The Concrete Security Approach

A scheme is (T, ϵ)-secure if any adversary running for time at most T succeeds in breaking the scheme with probability at most ϵ.

(T, ϵ)-Computational Indistinguishability

Two distributions X and Y are (T, ϵ)-computationally indistinguishable if for all adversaries A that run in time at most T,

$$\left| \Pr_{x \leftarrow X}\left[A(x) = 1\right] - \Pr_{y \leftarrow Y}\left[A(y) = 1\right] \right| \leq \epsilon,$$

where the probability is over sampling from the distributions X and Y, and the randomness of A.

(T, ϵ)-Computational Security

An encryption scheme is (T, ϵ)-computationally secure if for all $m_0, m_1 \in \mathcal{M}$, the following distributions are (T, ϵ)-computationally indistinguishable:

$$D_0 = \left\{ \mathsf{ct} : \begin{array}{c} k \leftarrow \mathsf{KeyGen}() \\ \mathsf{ct} \leftarrow \mathsf{Enc}(k, m_0) \end{array} \right\} \qquad D_1 = \left\{ \mathsf{ct} : \begin{array}{c} k \leftarrow \mathsf{KeyGen}() \\ \mathsf{ct} \leftarrow \mathsf{Enc}(k, m_1) \end{array} \right\}$$

The Concrete Security Approach

- $(2^{128}, 2^{-60})$**-computational security:** Attacks using at most 2^{128} cycles cannot break security with probability better than 2^{-40}.

The Concrete Security Approach

- $(2^{128}, 2^{-60})$**-computational security:** Attacks using at most 2^{128} cycles cannot break security with probability better than 2^{-40}.
  - This is the type of guarantee we want to give for crypto systems deployed in the real-world.

The Concrete Security Approach

- $(2^{128}, 2^{-60})$**-computational security:** Attacks using at most 2^{128} cycles cannot break security with probability better than 2^{-40}.
  - This is the type of guarantee we want to give for crypto systems deployed in the real-world.
- **Limitation:**

The Concrete Security Approach

- $(2^{128}, 2^{-60})$**-computational security:** Attacks using at most 2^{128} cycles cannot break security with probability better than 2^{-40}.
  - This is the type of guarantee we want to give for crypto systems deployed in the real-world.
- **Limitation:**
  - What type of computing power do we assume the adversary uses? (e.g., GPUs, super-computers)

The Concrete Security Approach

- $(2^{128}, 2^{-60})$**-computational security:** Attacks using at most 2^{128} cycles cannot break security with probability better than 2^{-40}.
  - This is the type of guarantee we want to give for crypto systems deployed in the real-world.
- **Limitation:**
  - What type of computing power do we assume the adversary uses? (e.g., GPUs, super-computers)
  - How to account for future advances in computing power?

The Concrete Security Approach

- $(2^{128}, 2^{-60})$**-computational security:** Attacks using at most 2^{128} cycles cannot break security with probability better than 2^{-40}.
  - This is the type of guarantee we want to give for crypto systems deployed in the real-world.
- **Limitation:**
  - What type of computing power do we assume the adversary uses? (e.g., GPUs, super-computers)
  - How to account for future advances in computing power?
- We need a “knob” to tune the security level

The Concrete Security Approach

- $(2^{128}, 2^{-60})$**-computational security:** Attacks using at most 2^{128} cycles cannot break security with probability better than 2^{-40}.
    - This is the type of guarantee we want to give for crypto systems deployed in the real-world.
- **Limitation:**
    - What type of computing power do we assume the adversary uses? (e.g., GPUs, super-computers)
    - How to account for future advances in computing power?
- We need a “knob” to tune the security level
    - **Analogy:** We consider asymptotic growth in runtime for sorting algorithms; not their runtime on lists of 10,000 values i.e., we have a “knob” to tune the runtime for lists of different length.

The Asymptotic Approach

- Security parameter $\lambda \in \mathbb{N}$ used to parametrize algorithms and adversaries.

The Asymptotic Approach

- Security parameter $\lambda \in \mathbb{N}$ used to parametrize algorithms and adversaries.
  - Knob for tuning security level of the scheme.

The Asymptotic Approach

- Security parameter $\lambda \in \mathbb{N}$ used to parametrize algorithms and adversaries.
  - Knob for tuning security level of the scheme.
  - **Intuition:** Length of the key.

The Asymptotic Approach

- Security parameter $\lambda \in \mathbb{N}$ used to parametrize algorithms and adversaries.
    - Knob for tuning security level of the scheme.
    - **Intuition:** Length of the key.
    - Set by honest parties when deploying the scheme in the real world. Also known to the adversary.

The Asymptotic Approach

- Security parameter $\lambda \in \mathbb{N}$ used to parametrize algorithms and adversaries.
  - Knob for tuning security level of the scheme.
  - **Intuition:** Length of the key.
  - Set by honest parties when deploying the scheme in the real world. Also known to the adversary.
  - We will analyze the runtime of algorithms, the adversary's runtime, and the adversary's success probability in terms of the security parameter.

The Asymptotic Approach

- Security parameter $\lambda \in \mathbb{N}$ used to parametrize algorithms and adversaries.
  - Knob for tuning security level of the scheme.
  - **Intuition:** Length of the key.
  - Set by honest parties when deploying the scheme in the real world. Also known to the adversary.
  - We will analyze the runtime of algorithms, the adversary's runtime, and the adversary's success probability in terms of the security parameter.
- **Illustrative Example:** Encryption requires $10^6 \cdot \lambda$ CPU cycles. Adversary running for $10^8 \cdot \lambda^4$ CPU cycles can succeed in breaking the scheme with probability at most $2^{-\lambda/2}$.

The Asymptotic Approach

- Security parameter $\lambda \in \mathbb{N}$ used to parametrize algorithms and adversaries.
  - Knob for tuning security level of the scheme.
  - **Intuition:** Length of the key.
  - Set by honest parties when deploying the scheme in the real world. Also known to the adversary.
  - We will analyze the runtime of algorithms, the adversary's runtime, and the adversary's success probability in terms of the security parameter.
- **Illustrative Example:** Encryption requires $10^6 \cdot \lambda$ CPU cycles. Adversary running for $10^8 \cdot \lambda^4$ CPU cycles can succeed in breaking the scheme with probability at most $2^{-\lambda/2}$.
  - **2GHz Computers with $\lambda = 80$:** Encryption takes 3.2 seconds. Adversary that runs for ~3 weeks can break security with probability at most 2^{-40}.

The Asymptotic Approach

- Security parameter $\lambda \in \mathbb{N}$ used to parametrize algorithms and adversaries.
  - Knob for tuning security level of the scheme.
  - **Intuition:** Length of the key.
  - Set by honest parties when deploying the scheme in the real world. Also known to the adversary.
  - We will analyze the runtime of algorithms, the adversary's runtime, and the adversary's success probability in terms of the security parameter.
- **Illustrative Example:** Encryption requires $10^6 \cdot \lambda$ CPU cycles. Adversary running for $10^8 \cdot \lambda^4$ CPU cycles can succeed in breaking the scheme with probability at most $2^{-\lambda/2}$.
  - **2GHz Computers with $\lambda = 80$:** Encryption takes 3.2 seconds. Adversary that runs for ~3 weeks can break security with probability at most 2^{-40}.
  - **8GHz Computers with $\lambda = 160$:** Encryption takes 3.2 seconds. Adversary that runs for ~13 weeks can break security with probability at most 2^{-80}!

The Asymptotic Approach

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

Efficient Adversaries

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

Efficient Adversaries

> Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.

Efficient Adversaries

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.
  - All standard (classical) models of computation are equivalent up to polynomial time.

Efficient Adversaries

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.
  - All standard (classical) models of computation are equivalent up to polynomial time.
  - **Closure property:** Repeating a poly-time algorithm polynomially many times is still poly-time.

Efficient Adversaries

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.
  - All standard (classical) models of computation are equivalent up to polynomial time.
  - **Closure property:** Repeating a poly-time algorithm polynomially many times is still poly-time.
- We will require security against adversaries that run in polynomial-time i.e., poly-time attacks are feasible.

Efficient Adversaries

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.
  - All standard (classical) models of computation are equivalent up to polynomial time.
  - **Closure property:** Repeating a poly-time algorithm polynomially many times is still poly-time.
- We will require security against adversaries that run in polynomial-time i.e., poly-time attacks are feasible.
  - The adversary can be randomized.

Efficient Adversaries

> Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.
  - All standard (classical) models of computation are equivalent up to polynomial time.
  - **Closure property:** Repeating a poly-time algorithm polynomially many times is still poly-time.
- We will require security against adversaries that run in polynomial-time i.e., poly-time attacks are feasible.
  - The adversary can be randomized.
  - The adversary is non-uniform.

Efficient Adversaries

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.
  - All standard (classical) models of computation are equivalent up to polynomial time.
  - **Closure property:** Repeating a poly-time algorithm polynomially many times is still poly-time.
- We will require security against adversaries that run in polynomial-time i.e., poly-time attacks are feasible.
  - The adversary can be randomized.
  - The adversary is non-uniform.
  - **Efficient Adversary:** A non-uniform PPT Turing machine.

Efficient Adversaries

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.
  - All standard (classical) models of computation are equivalent up to polynomial time.
  - **Closure property:** Repeating a poly-time algorithm polynomially many times is still poly-time.
- We will require security against adversaries that run in polynomial-time i.e., poly-time attacks are feasible.
  - The adversary can be randomized.
  - The adversary is non-uniform.
  - **Efficient Adversary:** A non-uniform PPT Turing machine.
- Cryptographic primitives will also be PPT algorithms.

Efficient Adversaries

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- **Efficient Algorithms:** Algorithms that have polynomial runtime in the security parameter λ.
  - All standard (classical) models of computation are equivalent up to polynomial time.
  - **Closure property:** Repeating a poly-time algorithm polynomially many times is still poly-time.
- We will require security against adversaries that run in polynomial-time i.e., poly-time attacks are feasible.
  - The adversary can be randomized.
  - The adversary is non-uniform.
  - **Efficient Adversary:** A non-uniform PPT Turing machine.
- Cryptographic primitives will also be PPT algorithms.
  - Primitives have a fixed (small) polynomial runtime and the adversary can run for much longer (arbitrary polynomial runtime).

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- How to define “negligible” probability

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- How to define "negligible" probability
  - Eve successfully attacks an encryption scheme with probability at most $2^{-\lambda}$.

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- How to define "negligible" probability
  - Eve successfully attacks an encryption scheme with probability at most $2^{-\lambda}$.
  - If she repeats the attack polynomially many times (say λ^c), the probability that at least one of them is successful is at least

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- How to define "negligible" probability
  - Eve successfully attacks an encryption scheme with probability at most $2^{-\lambda}$.
  - If she repeats the attack polynomially many times (say λ^c), the probability that at least one of them is successful is at least $\lambda^c \cdot 2^{-\lambda}$.

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- How to define "negligible" probability
  - Eve successfully attacks an encryption scheme with probability at most $2^{-\lambda}$.
  - If she repeats the attack polynomially many times (say λ^c), the probability that at least one of them is successful is at least $\lambda^c \cdot 2^{-\lambda}$.
  - This is still low for sufficiently large λ; no polynomial can "rescue" $2^{-\lambda}$ from approaching zero.

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- How to define "negligible" probability
  - Eve successfully attacks an encryption scheme with probability at most $2^{-\lambda}$.
  - If she repeats the attack polynomially many times (say λ^c), the probability that at least one of them is successful is at least $\lambda^c \cdot 2^{-\lambda}$.
  - This is still low for sufficiently large λ; no polynomial can "rescue" $2^{-\lambda}$ from approaching zero.
  - Our definition of "negligible" probability should be robust to such amplification strategies by efficient adversaries.

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- How to define "negligible" probability
  - Eve successfully attacks an encryption scheme with probability at most $2^{-\lambda}$.
  - If she repeats the attack polynomially many times (say λ^c), the probability that at least one of them is successful is at least $\lambda^c \cdot 2^{-\lambda}$.
  - This is still low for sufficiently large λ; no polynomial can "rescue" $2^{-\lambda}$ from approaching zero.
  - Our definition of "negligible" probability should be robust to such amplification strategies by efficient adversaries.
- **Negligible Function:** A function $\nu(\cdot)$ is negligible if for every polynomial $p(\cdot)$, we have $\lim_{\lambda\to\infty} p(\lambda) \cdot \nu(\lambda) = 0$.

Negligible Functions

Any efficient adversary should succeed in attacking the scheme with at most negligible probability.

- How to define "negligible" probability
  - Eve successfully attacks an encryption scheme with probability at most $2^{-\lambda}$.
  - If she repeats the attack polynomially many times (say λ^c), the probability that at least one of them is successful is at least $\lambda^c \cdot 2^{-\lambda}$.
  - This is still low for sufficiently large λ; no polynomial can "rescue" $2^{-\lambda}$ from approaching zero.
  - Our definition of "negligible" probability should be robust to such amplification strategies by efficient adversaries.
- **Negligible Function:** A function $\nu(\cdot)$ is negligible if for every polynomial $p(\cdot)$, we have $\lim_{\lambda \to \infty} p(\lambda) \cdot \nu(\lambda) = 0$.

For poly-time algorithms, events that occur with negligible probability look like they never occur.

Negligible Functions

- Alternatively, a negligible function decays faster than all inverse polynomial functions.

Negligible Functions

- Alternatively, a negligible function decays faster than all inverse polynomial functions.

- That is, for all $c > 0$, $\nu(\lambda) = O(\lambda^{-c})$.

Negligible Functions

- Alternatively, a negligible function decays faster than all inverse polynomial functions.

- That is, for all $c > 0$, $\nu(\lambda) = O(\lambda^{-c})$.

Negligible Function

A function $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is negligible if $\forall c \in \mathbb{Z}_{\geq 0}$, $\exists \Lambda \in \mathbb{N}$ such that $\forall \lambda \in \mathbb{N}$ and $\lambda > \Lambda$, it holds that

$$\nu(\lambda) \leq \frac{1}{\lambda^c}.$$

Negligible Functions

- Alternatively, a negligible function decays faster than all inverse polynomial functions.
- That is, for all $c > 0$, $\nu(\lambda) = O(\lambda^{-c})$.

<u>Negligible Function</u>

A function $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is negligible if $\forall c \in \mathbb{Z}_{\geq 0}$, $\exists \Lambda \in \mathbb{N}$ such that $\forall \lambda \in \mathbb{N}$ and $\lambda > \Lambda$, it holds that

$$\nu(\lambda) \leq \frac{1}{\lambda^c}.$$

Examples: $\nu(\lambda) = 2^{-\lambda}$

Negligible Functions

- Alternatively, a negligible function decays faster than all inverse polynomial functions.
- That is, for all $c > 0$, $\nu(\lambda) = O(\lambda^{-c})$.

Negligible Function

A function $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is negligible if $\forall c \in \mathbb{Z}_{\geq 0}$, $\exists \Lambda \in \mathbb{N}$ such that $\forall \lambda \in \mathbb{N}$ and $\lambda > \Lambda$, it holds that

$$\nu(\lambda) \leq \frac{1}{\lambda^c}.$$

Examples: $\nu(\lambda) = 2^{-\lambda}$ $\nu(\lambda) = \lambda^{-\log \lambda}$

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \dfrac{1}{\lambda^c}$.

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \frac{1}{\lambda^c}$.

- Fix an arbitrary c.

- Since f and g are negligible

  - $\exists \Lambda_f$ such that $\forall \lambda > \Lambda_f,\ \ f(\lambda) \leq \frac{1}{\lambda^{c+1}}$ and $\exists \Lambda_g$ such that $\forall \lambda > \Lambda_g,\ \ g(\lambda) \leq \frac{1}{\lambda^{c+1}}$.

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since f and g are negligible
  - $\exists \Lambda_f$ such that $\forall \lambda > \Lambda_f,\ \ f(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$ and $\exists \Lambda_g$ such that $\forall \lambda > \Lambda_g,\ \ g(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$.
- Let $\Lambda = \max(\Lambda_f, \Lambda_g, 2)$. For all $\lambda > \Lambda$ we have

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since f and g are negligible
  - $\exists \Lambda_f$ such that $\forall \lambda > \Lambda_f,\ \ f(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$ and $\exists \Lambda_g$ such that $\forall \lambda > \Lambda_g,\ \ g(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$.
- Let $\Lambda = \max(\Lambda_f, \Lambda_g, 2)$. For all $\lambda > \Lambda$ we have

$$f(\lambda) + g(\lambda) \quad \leq \frac{1}{\lambda^{c+1}} + \frac{1}{\lambda^{c+1}}$$

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since f and g are negligible
  - $\exists \Lambda_f$ such that $\forall \lambda > \Lambda_f,\ \ f(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$ and $\exists \Lambda_g$ such that $\forall \lambda > \Lambda_g,\ \ g(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$.
- Let $\Lambda = \max(\Lambda_f, \Lambda_g, 2)$. For all $\lambda > \Lambda$ we have

$$f(\lambda) + g(\lambda) \quad \leq \frac{1}{\lambda^{c+1}} + \frac{1}{\lambda^{c+1}} \quad \leq \frac{2}{\lambda^{c+1}}$$

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since f and g are negligible
  - $\exists \Lambda_f$ such that $\forall \lambda > \Lambda_f,\ \ f(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$ and $\exists \Lambda_g$ such that $\forall \lambda > \Lambda_g,\ \ g(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$.
- Let $\Lambda = \max(\Lambda_f, \Lambda_g, 2)$. For all $\lambda > \Lambda$ we have

$$f(\lambda) + g(\lambda) \quad \leq \frac{1}{\lambda^{c+1}} + \frac{1}{\lambda^{c+1}} \quad \leq \frac{2}{\lambda^{c+1}} \quad \leq \frac{\lambda}{\lambda^{c+1}}$$

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since f and g are negligible
  - $\exists \Lambda_f$ such that $\forall \lambda > \Lambda_f,\ \ f(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$ and $\exists \Lambda_g$ such that $\forall \lambda > \Lambda_g,\ \ g(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$.
- Let $\Lambda = \max(\Lambda_f, \Lambda_g, 2)$. For all $\lambda > \Lambda$ we have

$$f(\lambda) + g(\lambda) \quad \leq \frac{1}{\lambda^{c+1}} + \frac{1}{\lambda^{c+1}} \quad \leq \frac{2}{\lambda^{c+1}} \quad \leq \frac{\lambda}{\lambda^{c+1}}$$

$\lambda \geq \Lambda \geq 2$

Negligible Functions: Properties

Lemma: If $f(\lambda)$ and $g(\lambda)$ are negligible functions, then $f(\lambda) + g(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ f(\lambda) + g(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since f and g are negligible
  - $\exists \Lambda_f$ such that $\forall \lambda > \Lambda_f,\ \ f(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$ and $\exists \Lambda_g$ such that $\forall \lambda > \Lambda_g,\ \ g(\lambda) \leq \dfrac{1}{\lambda^{c+1}}$.
- Let $\Lambda = \max(\Lambda_f, \Lambda_g, 2)$. For all $\lambda > \Lambda$ we have

$$f(\lambda) + g(\lambda) \quad \leq \frac{1}{\lambda^{c+1}} + \frac{1}{\lambda^{c+1}} \quad \leq \frac{2}{\lambda^{c+1}} \quad \leq \frac{\lambda}{\lambda^{c+1}} \quad \leq \frac{1}{\lambda^c}.$$

$\lambda \geq \Lambda \geq 2$

Negligible Functions: Properties

Lemma: If $\nu(\lambda)$ be a negligible function and $p(\lambda)$ be a polynomial such that $p(\lambda) \geq 0$ for all $\lambda \geq 0$.
Then $\nu(\lambda) \cdot p(\lambda)$ is also negligible.

Negligible Functions: Properties

Lemma: If $\nu(\lambda)$ be a negligible function and $p(\lambda)$ be a polynomial such that $p(\lambda) \geq 0$ for all $\lambda \geq 0$.
Then $\nu(\lambda) \cdot p(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ \nu(\lambda) \cdot p(\lambda) \leq \dfrac{1}{\lambda^c}$.

Negligible Functions: Properties

Lemma: If $\nu(\lambda)$ be a negligible function and $p(\lambda)$ be a polynomial such that $p(\lambda) \geq 0$ for all $\lambda \geq 0$.
Then $\nu(\lambda) \cdot p(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ \nu(\lambda) \cdot p(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.

Negligible Functions: Properties

Lemma: If $\nu(\lambda)$ be a negligible function and $p(\lambda)$ be a polynomial such that $p(\lambda) \geq 0$ for all $\lambda \geq 0$.
Then $\nu(\lambda) \cdot p(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ \nu(\lambda) \cdot p(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since p is a polynomial, $\exists \Lambda_p, c_p$ such that $\forall \lambda > \Lambda_p,\ \ p(\lambda) \leq \lambda^{c_p}$.

Negligible Functions: Properties

Lemma: If $\nu(\lambda)$ be a negligible function and $p(\lambda)$ be a polynomial such that $p(\lambda) \geq 0$ for all $\lambda \geq 0$.
Then $\nu(\lambda) \cdot p(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ \nu(\lambda) \cdot p(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since p is a polynomial, $\exists \Lambda_p, c_p$ such that $\forall \lambda > \Lambda_p,\ \ p(\lambda) \leq \lambda^{c_p}$.
- Since $\nu(\lambda)$ is negligible, $\exists \Lambda_\nu$ such that $\forall \lambda > \Lambda_\nu,\ \ \nu(\lambda) \leq \dfrac{1}{\lambda^{c+c_p}}$.

Negligible Functions: Properties

Lemma: If $\nu(\lambda)$ be a negligible function and $p(\lambda)$ be a polynomial such that $p(\lambda) \geq 0$ for all $\lambda \geq 0$.
Then $\nu(\lambda) \cdot p(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ \nu(\lambda) \cdot p(\lambda) \leq \frac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since p is a polynomial, $\exists \Lambda_p, c_p$ such that $\forall \lambda > \Lambda_p,\ \ p(\lambda) \leq \lambda^{c_p}$.
- Since $\nu(\lambda)$ is negligible, $\exists \Lambda_\nu$ such that $\forall \lambda > \Lambda_\nu,\ \ \nu(\lambda) \leq \frac{1}{\lambda^{c+c_p}}$.
- Let $\Lambda = \max(\Lambda_p, \Lambda_\nu)$. For all $\lambda > \Lambda$ we have

Negligible Functions: Properties

Lemma: If $\nu(\lambda)$ be a negligible function and $p(\lambda)$ be a polynomial such that $p(\lambda) \geq 0$ for all $\lambda \geq 0$.
Then $\nu(\lambda) \cdot p(\lambda)$ is also negligible.

Proof:

- We want to show that $\forall c,\ \exists \Lambda$, such that $\forall \lambda > \Lambda,\ \ \nu(\lambda) \cdot p(\lambda) \leq \dfrac{1}{\lambda^c}$.
- Fix an arbitrary c.
- Since p is a polynomial, $\exists \Lambda_p, c_p$ such that $\forall \lambda > \Lambda_p,\ \ p(\lambda) \leq \lambda^{c_p}$.
- Since $\nu(\lambda)$ is negligible, $\exists \Lambda_\nu$ such that $\forall \lambda > \Lambda_\nu,\ \ \nu(\lambda) \leq \dfrac{1}{\lambda^{c+c_p}}$.
- Let $\Lambda = \max(\Lambda_p, \Lambda_\nu)$. For all $\lambda > \Lambda$ we have

$$\nu(\lambda) \cdot p(\lambda) \quad \leq \frac{1}{\lambda^{c+c_p}} \cdot \lambda^{c_p} \quad \leq \frac{1}{\lambda^c}\,.$$

Ensembles

Our goal is to give an asymptotic definition of computational indistinguishability.

<u>(T, ϵ)-**Computational Indistinguishability**</u>

Two distributions X and Y are (T, ϵ)-computationally indistinguishable if for all adversaries A that run in time at most T,

$$\left| \Pr_{x \leftarrow X}\left[A(x) = 1\right] - \Pr_{y \leftarrow Y}\left[A(y) = 1\right] \right| \leq \epsilon,$$

where the probability is over sampling from the distributions X and Y, and the randomness of A.

- It is not very meaningful to talk about individual distributions when we want to capture asymptotic behavior.
- For example, using longer keys leads to distributions over longer bit strings.

Ensembles

Probability Ensemble

Let $\mathscr{I}$ be a countable index set. An ensemble indexed by $\mathscr{I}$ is a sequence of random variables $\{X_i\}_{i \in \mathscr{I}}$.

- In most cases, $\mathscr{I}$ will be the set of natural numbers.

Ensembles

Probability Ensemble

Let $\mathscr{I}$ be a countable index set. An ensemble indexed by $\mathscr{I}$ is a sequence of random variables $\{X_i\}_{i \in \mathscr{I}}$.

- In most cases, $\mathscr{I}$ will be the set of natural numbers.
- An ensemble is simply sequence of random variables $X_1, X_2, \ldots$

Ensembles

Probability Ensemble

Let $\mathcal{I}$ be a countable index set. An ensemble indexed by $\mathcal{I}$ is a sequence of random variables $\{X_i\}_{i \in \mathcal{I}}$.

- In most cases, $\mathcal{I}$ will be the set of natural numbers.
- An ensemble is simply sequence of random variables $X_1, X_2, \ldots$
  - Allows us to focus on asymptotic behavior of distributions e.g., what happens when the key is a sufficiently long, uniformly random bit string.

Computational Indistinguishability

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i \in \mathbb{N}}$ and $Y = \{Y_i\}_{i \in \mathbb{N}}$ are computationally indistinguishable if

for all $\lambda \in \mathbb{N}$

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i \in \mathbb{N}}$ and $Y = \{Y_i\}_{i \in \mathbb{N}}$ are computationally indistinguishable if

for all $\lambda \in \mathbb{N}$

$$\left| \Pr_{x \leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y \leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right|$$

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i \in \mathbb{N}}$ and $Y = \{Y_i\}_{i \in \mathbb{N}}$ are computationally indistinguishable if every non-uniform PPT adversary A, for all $\lambda \in \mathbb{N}$

$$\left| \Pr_{x \leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y \leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right|$$

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i \in \mathbb{N}}$ and $Y = \{Y_i\}_{i \in \mathbb{N}}$ are computationally indistinguishable if every non-uniform PPT adversary A, for all $\lambda \in \mathbb{N}$

$$\left| \Pr_{x \leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y \leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right|$$

Denotes string of λ ones.

Ensures A is polynomial in λ.

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i \in \mathbb{N}}$ and $Y = \{Y_i\}_{i \in \mathbb{N}}$ are computationally indistinguishable if every non-uniform PPT adversary A, there exists a negligible function $\nu(\lambda)$ such that for all $\lambda \in \mathbb{N}$

$$\left| \Pr_{x \leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y \leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right| \leq \nu(\lambda),$$

Denotes string of λ ones.
Ensures A is polynomial in λ.

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i\in\mathbb{N}}$ and $Y = \{Y_i\}_{i\in\mathbb{N}}$ are computationally indistinguishable if every non-uniform PPT adversary A, there exists a negligible function $\nu(\lambda)$ such that for all $\lambda \in \mathbb{N}$

$$\left|\Pr_{x\leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y\leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right]\right| \leq \nu(\lambda),$$

where the probability is over sampling from the distributions X_λ and Y_λ, and the randomness of A.

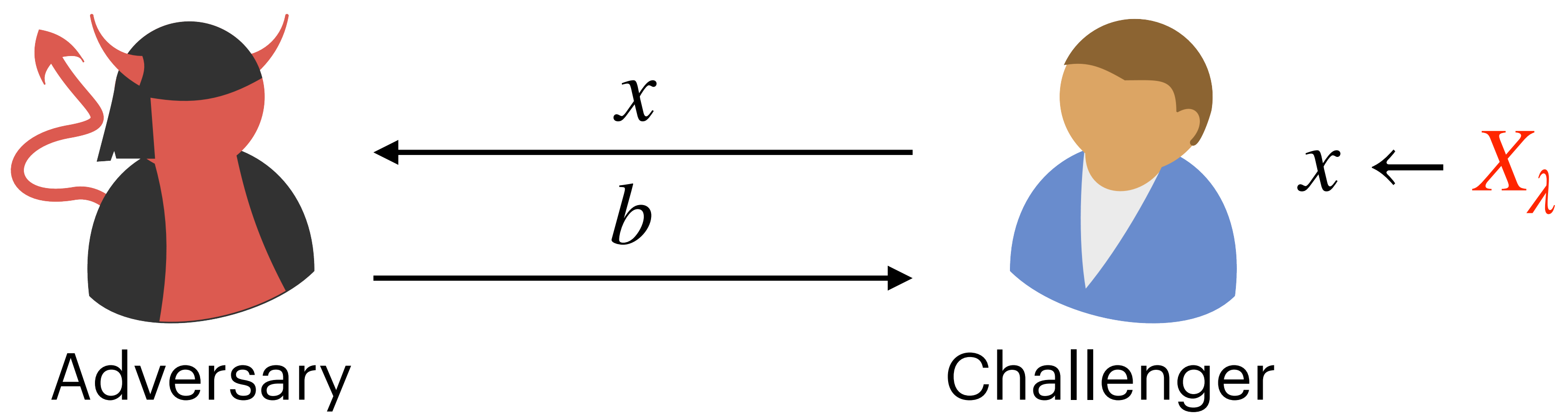
Computational Indistinguishability

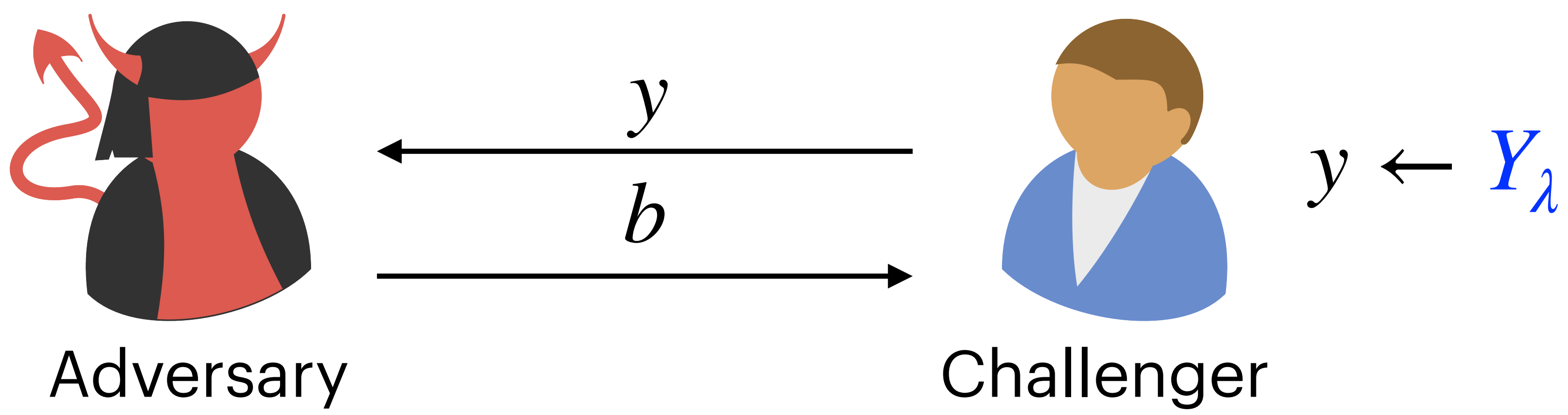
Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i \in \mathbb{N}}$ and $Y = \{Y_i\}_{i \in \mathbb{N}}$ are computationally indistinguishable if every non-uniform PPT adversary A, there exists a negligible function $\nu(\lambda)$ such that for all $\lambda \in \mathbb{N}$

$$\left| \Pr_{x \leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y \leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right| \leq \nu(\lambda),$$

where the probability is over sampling from the distributions X_λ and Y_λ, and the randomness of A.





No efficient test can distinguish between the ensembles X and Y.

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i \in \mathbb{N}}$ and $Y = \{Y_i\}_{i \in \mathbb{N}}$ are computationally indistinguishable if every non-uniform PPT adversary A, there exists a negligible function $\nu(\lambda)$ such that for all $\lambda \in \mathbb{N}$

$$\left| \Pr_{x \leftarrow X_\lambda} \left[A(1^\lambda, x) = 1 \right] - \Pr_{y \leftarrow Y_\lambda} \left[A(1^\lambda, y) = 1 \right] \right| \leq \nu(\lambda),$$

where the probability is over sampling from the distributions X_λ and Y_λ, and the randomness of A.

- We use $X \overset{c}{\approx} Y$ as a shorthand to denote that the two ensembles are computationally indistinguishable.

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i \in \mathbb{N}}$ and $Y = \{Y_i\}_{i \in \mathbb{N}}$ are computationally indistinguishable if every non-uniform PPT adversary A, there exists a negligible function $\nu(\lambda)$ such that for all $\lambda \in \mathbb{N}$

$$\left| \Pr_{x \leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y \leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right| \leq \nu(\lambda),$$

where the probability is over sampling from the distributions X_λ and Y_λ, and the randomness of A.

- We use $X \overset{c}{\approx} Y$ as a shorthand to denote that the two ensembles are computationally indistinguishable.
- The value

$$\left| \Pr_{x \leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y \leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right|$$

is called the adversary's **advantage** in distinguishing between X and Y.

Computational Indistinguishability

Computational Indistinguishability

Two probability ensembles $X = \{X_i\}_{i\in\mathbb{N}}$ and $Y = \{Y_i\}_{i\in\mathbb{N}}$ are computationally indistinguishable if every non-uniform PPT adversary A, there exists a negligible function $\nu(\lambda)$ such that for all $\lambda \in \mathbb{N}$

$$\left| \Pr_{x\leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y\leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right| \le \nu(\lambda),$$

where the probability is over sampling from the distributions X_λ and Y_λ, and the randomness of A.

- We use $X \overset{c}{\approx} Y$ as a shorthand to denote that the two ensembles are computationally indistinguishable.
- The value

$$\left| \Pr_{x\leftarrow X_\lambda}\left[A(1^\lambda, x) = 1\right] - \Pr_{y\leftarrow Y_\lambda}\left[A(1^\lambda, y) = 1\right] \right|$$

  is called the adversary's **advantage** in distinguishing between X and Y.

- $X \overset{c}{\approx} Y$ if all non-uniform PPT adversaries have negligible advantage in distinguishing between the two ensembles.