

# Computational Intractability (II) & Pseudorandomness (I)

601.642/442: Modern Cryptography

Fall 2020

# Asymptotic Cost of an Attack

- As we saw in the last class, exponential-time (e.g.,  $\Theta(2^n)$ ) brute-force attacks do not scale well as we increase the length of the key or the security parameter. We view them as *infeasible*.
- To rule out only *feasible* attacks, we will focus on polynomial-time attacks (e.g.,  $n^2, n^5$ , etc) that scale reasonably well (especially if exponent is small). Recall that polynomial-time algorithms are also referred to as *efficient* algorithms.
- **Our goal:** Ensure that no polynomial-time attack can successfully break security.
- A useful property of polynomial-time algorithms is **closure**: *repeating a poly-time algorithm polynomial times is still polynomial time!*

- **Computational Security:** Security against adversaries that run in polynomial-time.

# Efficient Adversary

- **Computational Security:** Security against adversaries that run in polynomial-time.
- The adversary might be *randomized*.

- **Computational Security:** Security against adversaries that run in polynomial-time.
- The adversary might be *randomized*.
- It might also use a *different* algorithm for each input size, each of which might be efficient. This still counts as efficient since he is using polynomial-time resources!

- **Computational Security:** Security against adversaries that run in polynomial-time.
- The adversary might be *randomized*.
- It might also use a *different* algorithm for each input size, each of which might be efficient. This still counts as efficient since he is using polynomial-time resources!
- We call this a **non-uniform** adversary since the algorithm is not uniform across all input sizes.

## Definition (Non-Uniform PPT)

A non-uniform probabilistic polynomial time Turing machine  $A$  is a sequence of probabilistic machines  $A = \{A_1, A_2, \dots\}$  for which there exists a polynomial  $p(\cdot)$  such that for every  $A_i \in A$ , the description size  $|A_i|$  and the running time of  $A_i$  are at most  $p(i)$ . We write  $A(x)$  to denote the distribution obtained by running  $A_{|x|}(x)$ .

- Our adversary will usually be a non-uniform PPT Turing machine.  
(most general)

# Success Probability of an Attack

- Since our adversary is **probabilistic**, it is not enough to consider only the running time of an attack.



# Success Probability of an Attack

- Since our adversary is **probabilistic**, it is not enough to consider only the running time of an attack.
- For example, consider an attacker who just tries to guess a victim's secret key, making a single guess. This attack is extremely cheap, but it still has a nonzero chance (e.g.,  $2^{-128}$ ) of breaking security.

# Success Probability of an Attack

- Since our adversary is **probabilistic**, it is not enough to consider only the running time of an attack.
- For example, consider an attacker who just tries to guess a victim's secret key, making a single guess. This attack is extremely cheap, but it still has a nonzero chance (e.g.,  $2^{-128}$ ) of breaking security.
- Essentially, for security, we don't need to worry about the following:

# Success Probability of an Attack

- Since our adversary is **probabilistic**, it is not enough to consider only the running time of an attack.
- For example, consider an attacker who just tries to guess a victim's secret key, making a single guess. This attack is extremely cheap, but it still has a nonzero chance (e.g.,  $2^{-128}$ ) of breaking security.
- Essentially, for security, we don't need to worry about the following:
  - Attacks that are as expensive as a brute-force attack.

# Success Probability of an Attack

- Since our adversary is **probabilistic**, it is not enough to consider only the running time of an attack.
- For example, consider an attacker who just tries to guess a victim's secret key, making a single guess. This attack is extremely cheap, but it still has a nonzero chance (e.g.,  $2^{-128}$ ) of breaking security.
- Essentially, for security, we don't need to worry about the following:
  - Attacks that are as expensive as a brute-force attack.
  - Attacks whose success probability is as low as a blind-guess attack.

# Success Probability of an Attack

- Since our adversary is **probabilistic**, it is not enough to consider only the running time of an attack.
- For example, consider an attacker who just tries to guess a victim's secret key, making a single guess. This attack is extremely cheap, but it still has a nonzero chance (e.g.,  $2^{-128}$ ) of breaking security.
- Essentially, for security, we don't need to worry about the following:
  - Attacks that are as expensive as a brute-force attack.
  - Attacks whose success probability is as low as a blind-guess attack.
- While an attack with success probability  $2^{-128}$  should not really count as an attack, one with success probability  $1/2$  should. Where should we draw the line?

# Success Probability of an Attack

Probability	Equivalent
$2^{-10}$	full house in 5-card poker
$2^{-20}$	royal flush in 5-card poker
$2^{-28}$	you win this week's Powerball jackpot
$2^{-40}$	royal flush in 2 consecutive poker games
$2^{-60}$	the next meteorite that hits Earth lands on this slide

# Success Probability of an Attack

Probability	Equivalent
$2^{-10}$	full house in 5-card poker
$2^{-20}$	royal flush in 5-card poker
$2^{-28}$	you win this week's Powerball jackpot
$2^{-40}$	royal flush in 2 consecutive poker games
$2^{-60}$	the next meteorite that hits Earth lands on this slide

- While these examples are good to get some intuition, we need more concrete way to draw the line between *reasonable* and *unreasonable*.

# Success Probability of an Attack

Probability	Equivalent
$2^{-10}$	full house in 5-card poker
$2^{-20}$	royal flush in 5-card poker
$2^{-28}$	you win this week's Powerball jackpot
$2^{-40}$	royal flush in 2 consecutive poker games
$2^{-60}$	the next meteorite that hits Earth lands on this slide

- While these examples are good to get some intuition, we need more concrete way to draw the line between *reasonable* and *unreasonable*.
- Just like running time, we will use an asymptotic approach to capture **low success probability** so that it can be tweaked as desired by changing the security parameter.



# Negligible Functions

- A blind guessing adversary may “amplify” his success probability by guessing polynomial (say  $n^c$ ) number of times, to achieve  $\frac{n^c}{2^n}$  probability of success.

# Negligible Functions

- A blind guessing adversary may “amplify” his success probability by guessing polynomial (say  $n^c$ ) number of times, to achieve  $\frac{n^c}{2^n}$  probability of success.
- This is still very low when  $n$  is large. Indeed,  $\frac{1}{2^n}$  approaches zero so fast that it cannot be “rescued” by any polynomial

# Negligible Functions

- A blind guessing adversary may “amplify” his success probability by guessing polynomial (say  $n^c$ ) number of times, to achieve  $\frac{n^c}{2^n}$  probability of success.
- This is still very low when  $n$  is large. Indeed,  $\frac{1}{2^n}$  approaches zero so fast that it cannot be “rescued” by any polynomial
- We want to formalize this property so as to rule out the possibility that a PPT (i.e., probabilistic polynomial-time) adversary is able to amplify very low success probability to “non-trivial” success probability

# Negligible Functions

- A blind guessing adversary may “amplify” his success probability by guessing polynomial (say  $n^c$ ) number of times, to achieve  $\frac{n^c}{2^n}$  probability of success.
- This is still very low when  $n$  is large. Indeed,  $\frac{1}{2^n}$  approaches zero so fast that it cannot be “rescued” by any polynomial
- We want to formalize this property so as to rule out the possibility that a PPT (i.e., probabilistic polynomial-time) adversary is able to amplify very low success probability to “non-trivial” success probability

# Negligible Functions

- A blind guessing adversary may “amplify” his success probability by guessing polynomial (say  $n^c$ ) number of times, to achieve  $\frac{n^c}{2^n}$  probability of success.
- This is still very low when  $n$  is large. Indeed,  $\frac{1}{2^n}$  approaches zero so fast that it cannot be “rescued” by any polynomial
- We want to formalize this property so as to rule out the possibility that a PPT (i.e., probabilistic polynomial-time) adversary is able to amplify very low success probability to “non-trivial” success probability

## Definition (Negligible Function)

A function  $\nu(\cdot)$  is negligible if for every polynomial  $p(\cdot)$ , we have  $\lim_{n \rightarrow \infty} p(n)\nu(n) = 0$

## Negligible Functions (contd.)

- A negligible function approaches zero so fast that you can never catch up when multiplying by a polynomial

## Negligible Functions (contd.)

- A negligible function approaches zero so fast that you can never catch up when multiplying by a polynomial
- **Alternatively:** A negligible function *decays faster than all “inverse-polynomial” functions*

## Negligible Functions (contd.)

- A negligible function approaches zero so fast that you can never catch up when multiplying by a polynomial
- **Alternatively:** A negligible function *decays faster than all “inverse-polynomial” functions*



# Negligible Functions (contd.)

- A negligible function approaches zero so fast that you can never catch up when multiplying by a polynomial
- **Alternatively:** A negligible function *decays faster than all “inverse-polynomial” functions*

## Definition (Negligible Function)

A function  $\nu(n)$  is negligible if  $\forall c, \exists n_0$  such that  $\forall n > n_0, \nu(n) \leq \frac{1}{n^c}$ .

# Negligible Functions: Examples

## Problem

Let  $f(\cdot)$  and  $g(\cdot)$  be negligible functions. Show that  $f(n) + g(n)$  is negligible.

# Negligible Functions: Examples

## Problem

Let  $f(\cdot)$  and  $g(\cdot)$  be negligible functions. Show that  $f(n) + g(n)$  is negligible.

We need to show that  $\forall c, \exists n_0$ , such that  $\forall n > n_0, f(n) + g(n) \leq \frac{1}{n^c}$ .

# Negligible Functions: Examples

## Problem

Let  $f(\cdot)$  and  $g(\cdot)$  be negligible functions. Show that  $f(n) + g(n)$  is negligible.

We need to show that  $\forall c, \exists n_0$ , such that  $\forall n > n_0, f(n) + g(n) \leq \frac{1}{n^c}$ .

- Since  $f(\cdot)$  and  $g(\cdot)$  are both negligible functions, we know that  $\exists n_f, n_g$  corresponding to  $c + 1$ , such that  $\forall n > n_f, f(n) \leq \frac{1}{n^{c+1}}$  and  $\forall n > n_g, g(n) \leq \frac{1}{n^{c+1}}$ .

# Negligible Functions: Examples

## Problem

Let  $f(\cdot)$  and  $g(\cdot)$  be negligible functions. Show that  $f(n) + g(n)$  is negligible.

We need to show that  $\forall c, \exists n_0$ , such that  $\forall n > n_0, f(n) + g(n) \leq \frac{1}{n^c}$ .

- Since  $f(\cdot)$  and  $g(\cdot)$  are both negligible functions, we know that  $\exists n_f, n_g$  corresponding to  $c + 1$ , such that  $\forall n > n_f, f(n) \leq \frac{1}{n^{c+1}}$  and  $\forall n > n_g, g(n) \leq \frac{1}{n^{c+1}}$ .

For a given  $c$ , let  $n_0 = \max(n_f, n_g, 2)$ .  $\forall n > n_0$ :

$$\begin{aligned} f(n) + g(n) &\leq \frac{1}{n^{c+1}} + \frac{1}{n^{c+1}} \\ &\leq \frac{2}{n^{c+1}} \\ &\leq \frac{n}{n^{c+1}} \quad (\text{Since } n \geq n_0 \geq 2) \\ &\leq \frac{1}{n^c} \end{aligned}$$

# Negligible Functions: Examples

## Problem

Let  $\nu(\cdot)$  be a negligible function and  $p(\cdot)$  be a polynomial s.t.  $p(n) \geq 0$ ,  $\forall n > 0$ . Show that  $\nu(n) \cdot p(n)$  is negligible.

# Negligible Functions: Examples

## Problem

Let  $\nu(\cdot)$  be a negligible function and  $p(\cdot)$  be a polynomial s.t.  $p(n) \geq 0$ ,  $\forall n > 0$ . Show that  $\nu(n) \cdot p(n)$  is negligible.

We need to show that  $\forall c, \exists n_0$ , such that  $\forall n > n_0, \nu(n) \cdot p(n) \leq \frac{1}{n^c}$ .

# Negligible Functions: Examples

## Problem

Let  $\nu(\cdot)$  be a negligible function and  $p(\cdot)$  be a polynomial s.t.  $p(n) \geq 0$ ,  $\forall n > 0$ . Show that  $\nu(n) \cdot p(n)$  is negligible.

We need to show that  $\forall c, \exists n_0$ , such that  $\forall n > n_0, \nu(n) \cdot p(n) \leq \frac{1}{n^c}$ .

- Since  $p(\cdot)$  is a polynomial function, we know that  $\exists n_p, c_p$ , such that,  $\forall n > n_p, p(n) \leq n^{c_p}$ .



# Negligible Functions: Examples

## Problem

Let  $\nu(\cdot)$  be a negligible function and  $p(\cdot)$  be a polynomial s.t.  $p(n) \geq 0$ ,  $\forall n > 0$ . Show that  $\nu(n) \cdot p(n)$  is negligible.

We need to show that  $\forall c, \exists n_0$ , such that  $\forall n > n_0, \nu(n) \cdot p(n) \leq \frac{1}{n^c}$ .

- Since  $p(\cdot)$  is a polynomial function, we know that  $\exists n_p, c_p$ , such that,  $\forall n > n_p, p(n) \leq n^{c_p}$ .
- Since  $\nu(\cdot)$  is a negligible function, we know that  $\exists n_\nu$  corresponding to  $c + c_p$ , such that  $\forall n > n_\nu, \nu(n) \leq \frac{1}{n^{c+c_p}}$ .

# Negligible Functions: Examples

## Problem

Let  $\nu(\cdot)$  be a negligible function and  $p(\cdot)$  be a polynomial s.t.  $p(n) \geq 0$ ,  $\forall n > 0$ . Show that  $\nu(n) \cdot p(n)$  is negligible.

We need to show that  $\forall c, \exists n_0$ , such that  $\forall n > n_0, \nu(n) \cdot p(n) \leq \frac{1}{n^c}$ .

- Since  $p(\cdot)$  is a polynomial function, we know that  $\exists n_p, c_p$ , such that,  $\forall n > n_p, p(n) \leq n^{c_p}$ .
- Since  $\nu(\cdot)$  is a negligible function, we know that  $\exists n_\nu$  corresponding to  $c + c_p$ , such that  $\forall n > n_\nu, \nu(n) \leq \frac{1}{n^{c+c_p}}$ .

For a given  $c$ , let  $n_0 = \max(n_\nu, n_p)$ .  $\forall n > n_0$ :

$$\begin{aligned}\nu(n) \cdot p(n) &\leq \frac{1}{n^{c+c_p}} \cdot n^{c_p} \\ &\leq \frac{1}{n^{c+c_p-c_p}} \\ &\leq \frac{1}{n^c}\end{aligned}$$

# Distributions & Ensembles

- Recall:  $X$  is a distribution over sample space  $\mathcal{S}$  if it assigns probability  $p_s$  to the element  $s \in \mathcal{S}$  s.t.  $\sum_s p_s = 1$

## Ensemble

A sequence  $\{X_n\}_{n \in \mathbb{N}}$  is called an ensemble if for each  $n \in \mathbb{N}$ ,  $X_n$  is a probability distribution over  $\{0, 1\}^*$ .

- Generally,  $X_n$  will be a distribution over the sample space  $\{0, 1\}^{\ell(n)}$  (where  $\ell(\cdot)$  is a polynomial)

# Computational Indistinguishability

- So far in the definitions of uniform ciphertext security and perfect security, we required the two distributions to be **identical**.

# Computational Indistinguishability

- So far in the definitions of uniform ciphertext security and perfect security, we required the two distributions to be **identical**.
- Going forward, since we (primarily) only care about security against efficient adversaries, our security definitions will not be quite as demanding.

# Computational Indistinguishability

- So far in the definitions of uniform ciphertext security and perfect security, we required the two distributions to be **identical**.
- Going forward, since we (primarily) only care about security against efficient adversaries, our security definitions will not be quite as demanding.
- In most cases, it would suffice for two distributions (say  $X$  and  $Y$ ) to “look alike” to *any* efficient test.

# Computational Indistinguishability

- So far in the definitions of uniform ciphertext security and perfect security, we required the two distributions to be **identical**.
- Going forward, since we (primarily) only care about security against efficient adversaries, our security definitions will not be quite as demanding.
- In most cases, it would suffice for two distributions (say  $X$  and  $Y$ ) to “look alike” to *any* efficient test.
- Efficient test = efficient computation = non-uniform PPT

# Computational Indistinguishability

- So far in the definitions of uniform ciphertext security and perfect security, we required the two distributions to be **identical**.
- Going forward, since we (primarily) only care about security against efficient adversaries, our security definitions will not be quite as demanding.
- In most cases, it would suffice for two distributions (say  $X$  and  $Y$ ) to “look alike” to *any* efficient test.
- Efficient test = efficient computation = non-uniform PPT
- If no non-uniform PPT adversary  $\mathcal{A}$  can tell them apart, we say that  $X$  and  $Y$  are **computationally indistinguishable**.



# Computational Indistinguishability

- So far in the definitions of uniform ciphertext security and perfect security, we required the two distributions to be **identical**.
- Going forward, since we (primarily) only care about security against efficient adversaries, our security definitions will not be quite as demanding.
- In most cases, it would suffice for two distributions (say  $X$  and  $Y$ ) to “look alike” to *any* efficient test.
- Efficient test = efficient computation = non-uniform PPT
- If no non-uniform PPT adversary  $\mathcal{A}$  can tell them apart, we say that  $X$  and  $Y$  are **computationally indistinguishable**.

# Computational Indistinguishability

- So far in the definitions of uniform ciphertext security and perfect security, we required the two distributions to be **identical**.
- Going forward, since we (primarily) only care about security against efficient adversaries, our security definitions will not be quite as demanding.
- In most cases, it would suffice for two distributions (say  $X$  and  $Y$ ) to “look alike” to *any* efficient test.
- Efficient test = efficient computation = non-uniform PPT
- If no non-uniform PPT adversary  $\mathcal{A}$  can tell them apart, we say that  $X$  and  $Y$  are **computationally indistinguishable**.

How do we formalize this?

# Computational Indistinguishability



- $\mathcal{A}$ 's output can be encoded using just one bit:  
 $1 = \text{"from } X\text{"}$  and  $0 = \text{"from } Y\text{"}$

# Computational Indistinguishability



- $\mathcal{A}$ 's output can be encoded using just one bit:  
 $1 = \text{"from } X\text{"}$  and  $0 = \text{"from } Y\text{"}$
- We want  $\mathcal{A}$  to output 1, with "almost similar" probability in both the above scenarios.

# Computational Indistinguishability



- $\mathcal{A}$ 's output can be encoded using just one bit:  
 $1 = \text{"from } X\text{"}$  and  $0 = \text{"from } Y\text{"}$
- We want  $\mathcal{A}$  to output 1, with "almost similar" probability in both the above scenarios.

# Computational Indistinguishability



- $\mathcal{A}$ 's output can be encoded using just one bit:  
1 = "from  $X$ " and 0 = "from  $Y$ "
- We want  $\mathcal{A}$  to output 1, with "almost similar" probability in both the above scenarios.

$$\Pr [x \leftarrow X; \mathcal{A}(1^n, x) = 1] \approx \Pr [y \leftarrow Y; \mathcal{A}(1^n, y) = 1] \implies$$

$$\left| \Pr [x \leftarrow X; \mathcal{A}(1^n, x) = 1] - \Pr [y \leftarrow Y; \mathcal{A}(1^n, y) = 1] \right| \leq \nu(n).$$

# Computationally Indistinguishability: Definition

## Definition (Computationally Indistinguishability)

Two ensembles of probability distributions  $X = \{X_n\}_{n \in \mathbb{N}}$  and  $Y = \{Y_n\}_{n \in \mathbb{N}}$  are said to be **computationally indistinguishable** if for every non-uniform PPT  $\mathcal{A}$  there exists a negligible function  $\nu(\cdot)$  s.t.:

$$\left| \Pr [x \leftarrow X_n; \mathcal{A}(1^n, x) = 1] - \Pr [y \leftarrow Y_n; \mathcal{A}(1^n, y) = 1] \right| \leq \nu(n).$$

# Computationally Indistinguishability: Definition

## Definition (Computationally Indistinguishability)

Two ensembles of probability distributions  $X = \{X_n\}_{n \in \mathbb{N}}$  and  $Y = \{Y_n\}_{n \in \mathbb{N}}$  are said to be **computationally indistinguishable** if for every non-uniform PPT  $\mathcal{A}$  there exists a negligible function  $\nu(\cdot)$  s.t.:

$$\left| \Pr [x \leftarrow X_n; \mathcal{A}(1^n, x) = 1] - \Pr [y \leftarrow Y_n; \mathcal{A}(1^n, y) = 1] \right| \leq \nu(n).$$

- The quantity

$\left| \Pr [x \leftarrow X_n; D(1^n, x) = 1] - \Pr [y \leftarrow Y_n; D(1^n, y) = 1] \right|$  is called the **advantage** or bias of  $\mathcal{A}$  in distinguishing  $X$  and  $Y$ .



# Computationally Indistinguishability: Definition

## Definition (Computationally Indistinguishability)

Two ensembles of probability distributions  $X = \{X_n\}_{n \in \mathbb{N}}$  and  $Y = \{Y_n\}_{n \in \mathbb{N}}$  are said to be **computationally indistinguishable** if for every non-uniform PPT  $\mathcal{A}$  there exists a negligible function  $\nu(\cdot)$  s.t.:

$$\left| \Pr [x \leftarrow X_n; \mathcal{A}(1^n, x) = 1] - \Pr [y \leftarrow Y_n; \mathcal{A}(1^n, y) = 1] \right| \leq \nu(n).$$

- The quantity  $\left| \Pr [x \leftarrow X_n; D(1^n, x) = 1] - \Pr [y \leftarrow Y_n; D(1^n, y) = 1] \right|$  is called the **advantage** or bias of  $\mathcal{A}$  in distinguishing  $X$  and  $Y$ .
- Therefore,  $X$  and  $Y$  are computationally indistinguishable if all non-uniform PPT  $\mathcal{A}$  have negligible advantage in distinguishing them.

# Properties of Computational Indistinguishability

- Notation:  $\{X_n\} \approx_c \{Y_n\}$  means computational indistinguishability

# Properties of Computational Indistinguishability

- Notation:  $\{X_n\} \approx_c \{Y_n\}$  means computational indistinguishability
- **Closure**: If we apply an efficient operation on  $X$  and  $Y$ , they remain computationally indistinguishable. That is,  $\forall$  non-uniform PPT  $M$

$$\{X_n\} \approx_c \{Y_n\} \implies \{M(X_n)\} \approx_c \{M(Y_n)\}$$

# Properties of Computational Indistinguishability

- Notation:  $\{X_n\} \approx_c \{Y_n\}$  means computational indistinguishability
- **Closure**: If we apply an efficient operation on  $X$  and  $Y$ , they remain computationally indistinguishable. That is,  $\forall$  non-uniform PPT  $M$

$$\{X_n\} \approx_c \{Y_n\} \implies \{M(X_n)\} \approx_c \{M(Y_n)\}$$

# Properties of Computational Indistinguishability

- Notation:  $\{X_n\} \approx_c \{Y_n\}$  means computational indistinguishability
- **Closure**: If we apply an efficient operation on  $X$  and  $Y$ , they remain computationally indistinguishable. That is,  $\forall$  non-uniform PPT  $M$

$$\{X_n\} \approx_c \{Y_n\} \implies \{M(X_n)\} \approx_c \{M(Y_n)\}$$

Proof Idea: If not,  $\mathcal{A}$  can use  $M$  to tell them apart!

# Properties of Computational Indistinguishability

- Notation:  $\{X_n\} \approx_c \{Y_n\}$  means computational indistinguishability
- **Closure**: If we apply an efficient operation on  $X$  and  $Y$ , they remain computationally indistinguishable. That is,  $\forall$  non-uniform PPT  $M$

$$\{X_n\} \approx_c \{Y_n\} \implies \{M(X_n)\} \approx_c \{M(Y_n)\}$$

Proof Idea: If not,  $\mathcal{A}$  can use  $M$  to tell them apart!

- **Transitivity**: If  $X, Y$  are computationally indistinguishable, and  $Y, Z$  are computationally indistinguishable; then  $X, Z$  are also computationally indistinguishable.

# Generalizing Transitivity: Hybrid Lemma

## Lemma (Hybrid Lemma)

*Let  $X^1, \dots, X^m$  be distribution ensembles for  $m = \text{poly}(n)$ . If for every  $i \in [m - 1]$ ,  $X^i$  and  $X^{i+1}$  are computationally indistinguishable, then  $X^1$  and  $X^m$  are computationally indistinguishable.*

# Generalizing Transitivity: Hybrid Lemma

## Lemma (Hybrid Lemma)

*Let  $X^1, \dots, X^m$  be distribution ensembles for  $m = \text{poly}(n)$ . If for every  $i \in [m - 1]$ ,  $X^i$  and  $X^{i+1}$  are computationally indistinguishable, then  $X^1$  and  $X^m$  are computationally indistinguishable.*

This is the hybrid technique, started more generally, in the computational setting.



# Generalizing Transitivity: Hybrid Lemma

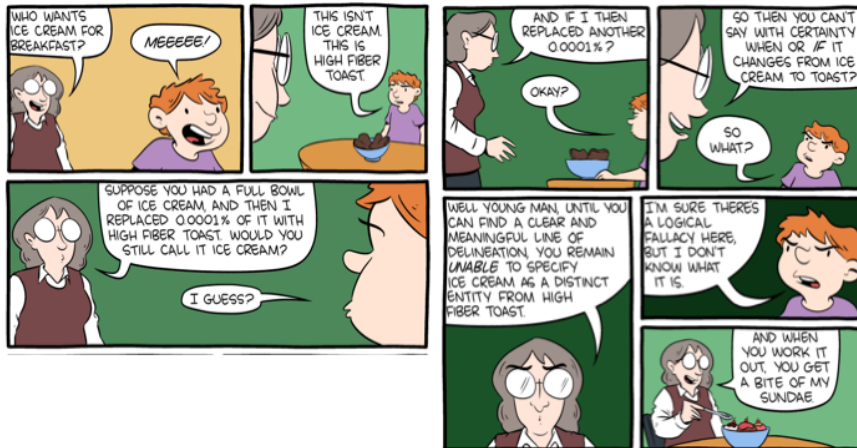
## Lemma (Hybrid Lemma)

*Let  $X^1, \dots, X^m$  be distribution ensembles for  $m = \text{poly}(n)$ . If for every  $i \in [m - 1]$ ,  $X^i$  and  $X^{i+1}$  are computationally indistinguishable, then  $X^1$  and  $X^m$  are computationally indistinguishable.*

This is the hybrid technique, started more generally, in the computational setting.

Used in most crypto proofs!

# Hybrid Lemma



smbc-comics.com

# Looking Back at OTP

## One-Time Pad

- $\text{KeyGen}(1^n) := k \xleftarrow{\$} \{0, 1\}^n$
- $\text{Enc}(k, m) := c = k \oplus m$
- $\text{Dec}(k, c) := m = k \oplus c$

Uniform ciphertext security and perfect security of OTP crucially depends on the following features:

- ① A key is as long as a message.
- ② A key is sampled uniformly at random.

# Looking Back at OTP

## One-Time Pad

- $\text{KeyGen}(1^n) := k \xleftarrow{\$} \{0, 1\}^n$
- $\text{Enc}(k, m) := c = k \oplus m$
- $\text{Dec}(k, c) := m = k \oplus c$

Uniform ciphertext security and perfect security of OTP crucially depends on the following features:

- ① A key is as long as a message.
- ② A key is sampled uniformly at random.

These features of OTP are also its key drawbacks! Why?

# Drawbacks of OTP

- **Length of the key is as much as that of the message.**
  - The key somehow needs to be exchanged between the sender and recipient in advance.
  - Longer the key, more inefficient the key exchange process will be.

# Drawbacks of OTP

- **Length of the key is as much as that of the message.**
  - The key somehow needs to be exchanged between the sender and recipient in advance.
  - Longer the key, more inefficient the key exchange process will be.
- **The key is sampled uniformly at random.**
  - Common sources of randomness:
    - key-strokes
    - mouse movement
    - power consumption
    - ...
  - These processes can only produce so much true randomness

# Drawbacks of OTP

- **Length of the key is as much as that of the message.**
  - The key somehow needs to be exchanged between the sender and recipient in advance.
  - Longer the key, more inefficient the key exchange process will be.
- **The key is sampled uniformly at random.**
  - Common sources of randomness:
    - key-strokes
    - mouse movement
    - power consumption
    - ...
  - These processes can only produce so much true randomness

Can we **expand** few random bits into many random “looking” bits?

- Suppose you have  $n$  uniformly random bits:  $x = x_1 \| \dots \| x_n$



# Pseudorandomness

- Suppose you have  $n$  uniformly random bits:  $x = x_1 \| \dots \| x_n$
- Find a **deterministic** (polynomial-time) algorithm  $G$  such that:

# Pseudorandomness

- Suppose you have  $n$  uniformly random bits:  $x = x_1 \| \dots \| x_n$
- Find a **deterministic** (polynomial-time) algorithm  $G$  such that:
  - $G(x)$  outputs (say)  $n + 1$  bits:  $y = y_1 \| \dots \| y_{n+1}$

# Pseudorandomness

- Suppose you have  $n$  uniformly random bits:  $x = x_1 \| \dots \| x_n$
- Find a **deterministic** (polynomial-time) algorithm  $G$  such that:
  - $G(x)$  outputs (say)  $n + 1$  bits:  $y = y_1 \| \dots \| y_{n+1}$
  - $y$  looks “as good as” a truly random string  $r = r_1 \| \dots \| r_{n+1}$

# Pseudorandomness

- Suppose you have  $n$  uniformly random bits:  $x = x_1 \| \dots \| x_n$
- Find a **deterministic** (polynomial-time) algorithm  $G$  such that:
  - $G(x)$  outputs (say)  $n + 1$  bits:  $y = y_1 \| \dots \| y_{n+1}$
  - $y$  looks “as good as” a truly random string  $r = r_1 \| \dots \| r_{n+1}$
- That is, the following distributions are computationally indistinguishable.

$$\left\{ x \stackrel{\$}{\leftarrow} \{0, 1\}^n : G(x) \right\} \quad \text{and} \quad \left\{ r \stackrel{\$}{\leftarrow} \{0, 1\}^{n+1} : r \right\}$$

# Pseudorandomness

- Suppose you have  $n$  uniformly random bits:  $x = x_1 \| \dots \| x_n$
- Find a **deterministic** (polynomial-time) algorithm  $G$  such that:
  - $G(x)$  outputs (say)  $n + 1$  bits:  $y = y_1 \| \dots \| y_{n+1}$
  - $y$  looks “as good as” a truly random string  $r = r_1 \| \dots \| r_{n+1}$
- That is, the following distributions are computationally indistinguishable.

$$\left\{ x \stackrel{\$}{\leftarrow} \{0, 1\}^n : G(x) \right\} \quad \text{and} \quad \left\{ r \stackrel{\$}{\leftarrow} \{0, 1\}^{n+1} : r \right\}$$

- $G : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$  is called a **pseudorandom generator** (PRG) and its output is a **pseudorandom** string.

# Pseudorandom Generators (PRG)

## Definition (Pseudorandom Generator)

A deterministic algorithm  $G$  is called a pseudorandom generator (PRG) if:

- $G$  can be computed in polynomial time
- $|G(x)| > |x|$
- $\left\{x \stackrel{\$}{\leftarrow} \{0, 1\}^n; G(x)\right\} \approx_c \left\{U_{\ell(n)}\right\}$  where  $\ell(n) = |G(0^n)|$

The **stretch** of  $G$  is defined as:  $|G(x)| - |x|$

# Pseudorandom OTP

- We can now use a PRG to modify one-time pads as follows.

## Pseudorandom One-Time Pad

Let  $n$  be the security parameter and  $\ell(\cdot)$  be a polynomial. Let  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell(n)}$  be a PRG, and let the message space and ciphertext space be  $\{0, 1\}^{\ell(n)}$ .

- $\text{KeyGen}(1^n) := k \xleftarrow{\$} \{0, 1\}^n$
- $\text{Enc}(k, m) := c = G(k) \oplus m$
- $\text{Dec}(k, c) := m = G(k) \oplus c$

# Pseudorandom OTP

- We can now use a PRG to modify one-time pads as follows.

## Pseudorandom One-Time Pad

Let  $n$  be the security parameter and  $\ell(\cdot)$  be a polynomial. Let  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{\ell(n)}$  be a PRG, and let the message space and ciphertext space be  $\{0, 1\}^{\ell(n)}$ .

- $\text{KeyGen}(1^n) := k \xleftarrow{\$} \{0, 1\}^n$
- $\text{Enc}(k, m) := c = G(k) \oplus m$
- $\text{Dec}(k, c) := m = G(k) \oplus c$

Does this encryption scheme satisfy our original definitions of one-time uniform ciphertext security or one-time perfect security?