2550 Intro to cybersecurity

L9: Computational security, PRG

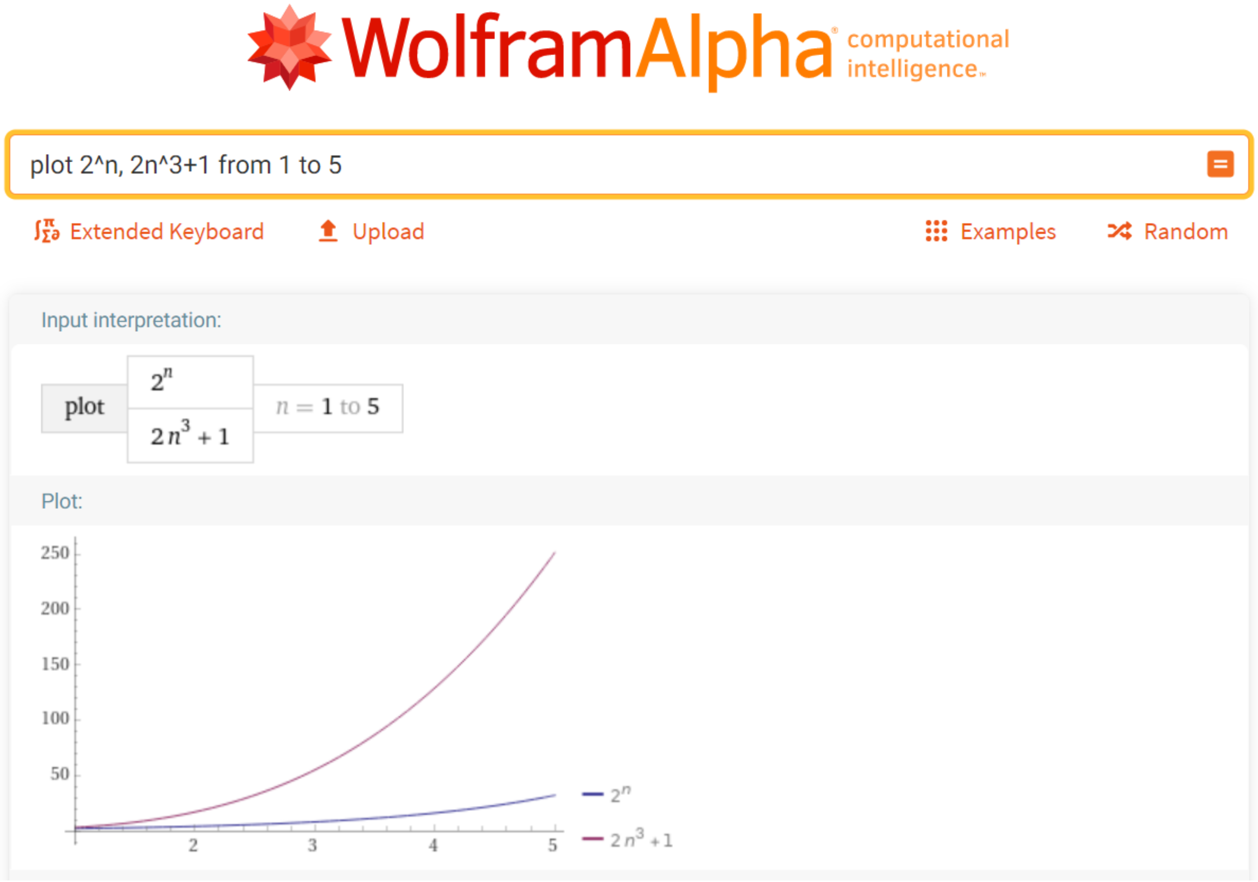
abhi shelat/Ran Cohen

Thanks to Gil Segev (HUJI) for sharing slides

Polynomial vs. Exponential

- Consider the functions $f(n) = 2n^3 + 1$ and $g(n) = 2^n$
- Which function is “bigger”?

n	$2n^3 + 1$	2^n
1	3	2
2	17	4
3	55	8
4	129	16
5	251	32

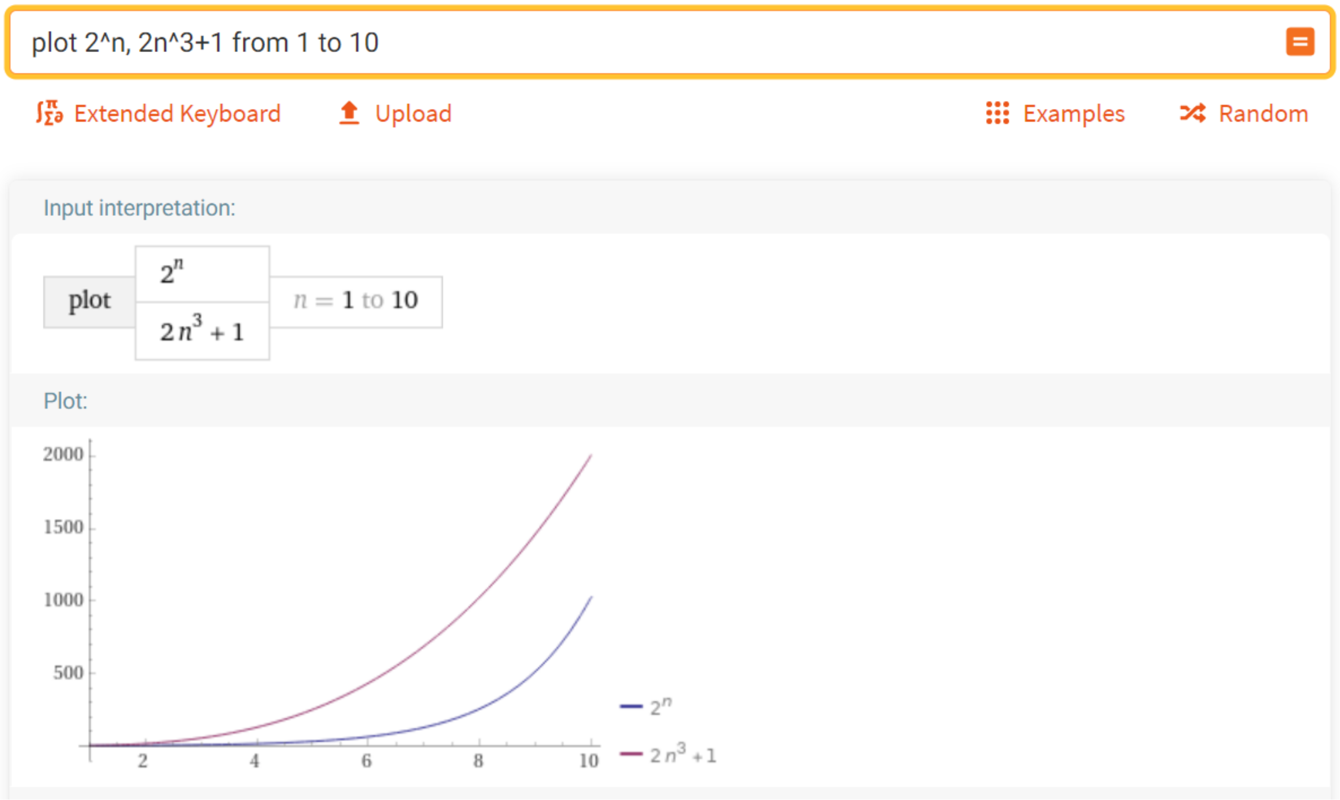


Polynomial vs. Exponential

- Consider the functions $f(n) = 2n^3 + 1$ and $g(n) = 2^n$
- Which function is "bigger"?

WolframAlpha computational intelligence.

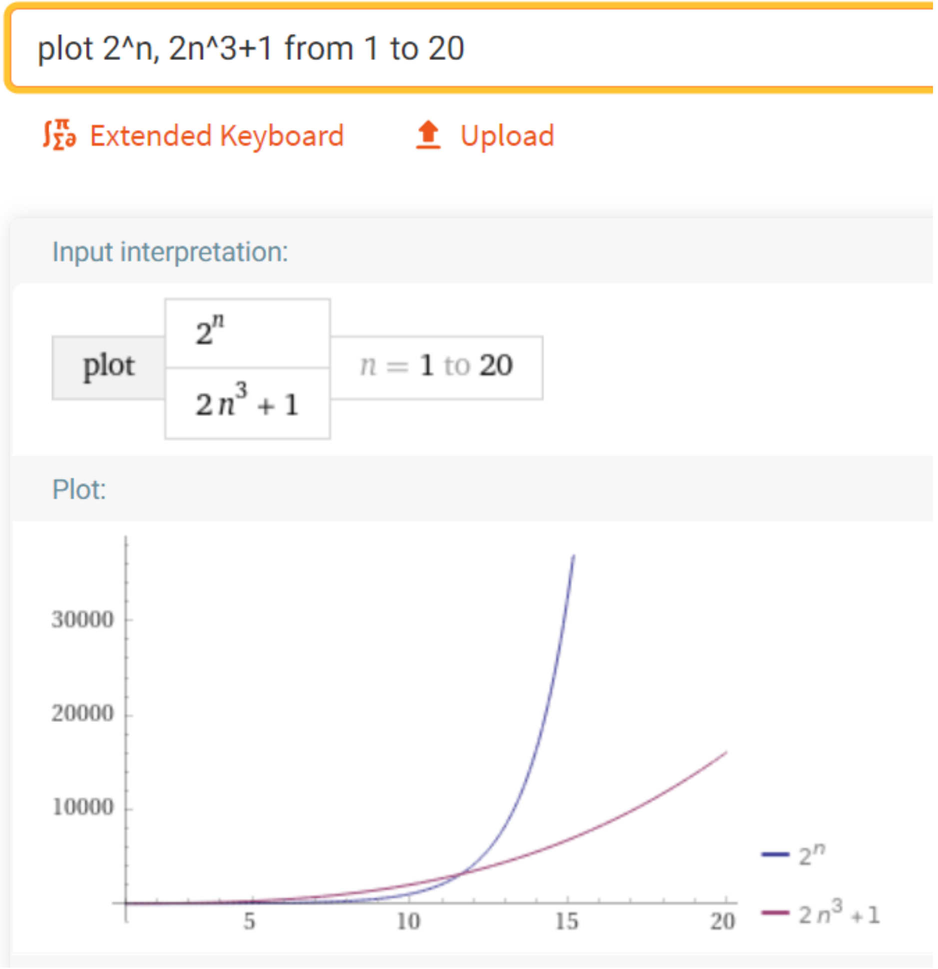
n	$2n^3 + 1$	2^n
1	3	2
2	17	4
3	55	8
4	129	16
5	251	32
6	433	64
7	687	128
8	1025	256
9	1459	512
10	2001	1024



Polynomial vs. Exponential

- Consider the functions $f(n) = 2n^3 + 1$ and $g(n) = 2^n$
- Which function is "bigger"?

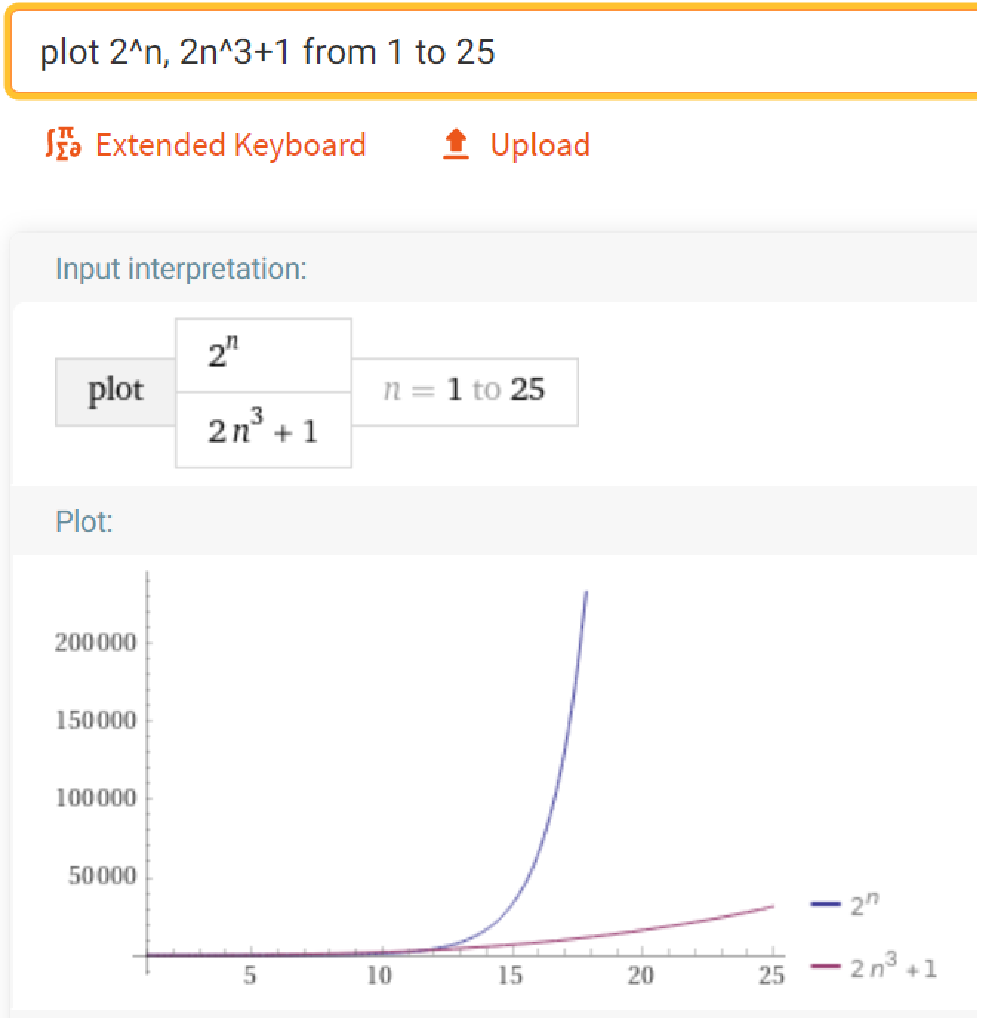
n	$2n^3 + 1$	2^n
11	2663	2048
12	3457	4096
13	4395	8192
14	5489	16384
20	16001	1,048,576
30	54001	1,073,741,824
35	85751	34,359,738,368



Polynomial vs. Exponential

- Consider the functions $f(n) = 2n^3 + 1$ and $g(n) = 2^n$
- Which function is "bigger"?

n	$2n^3 + 1$	2^n
11	2663	2048
12	3457	4096
13	4395	8192
14	5489	16384
20	16001	1,048,576
30	54001	1,073,741,824
35	85751	34,359,738,368

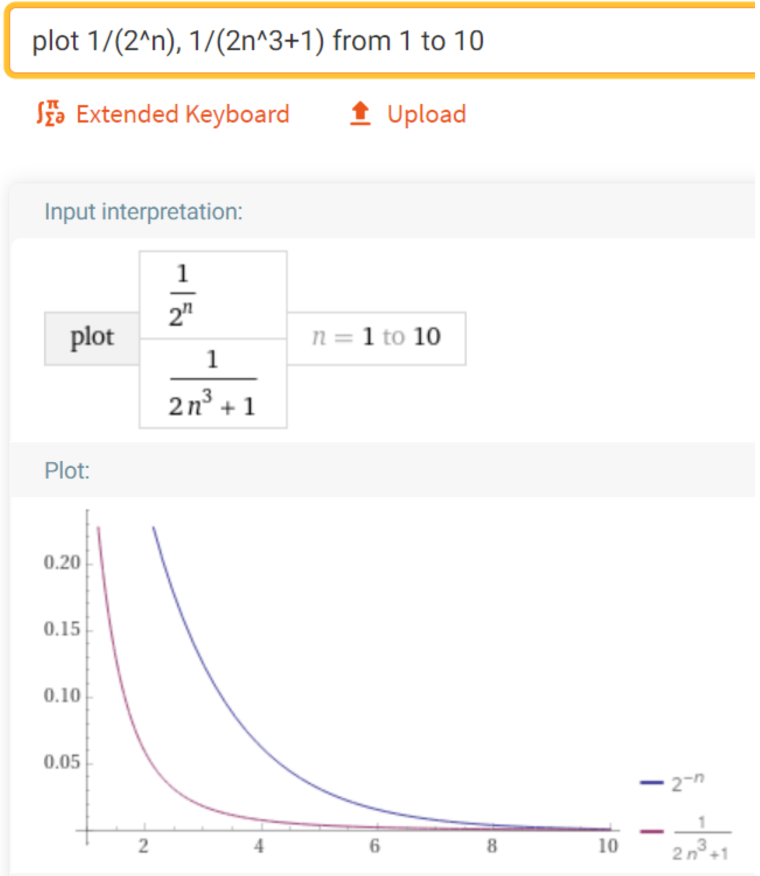


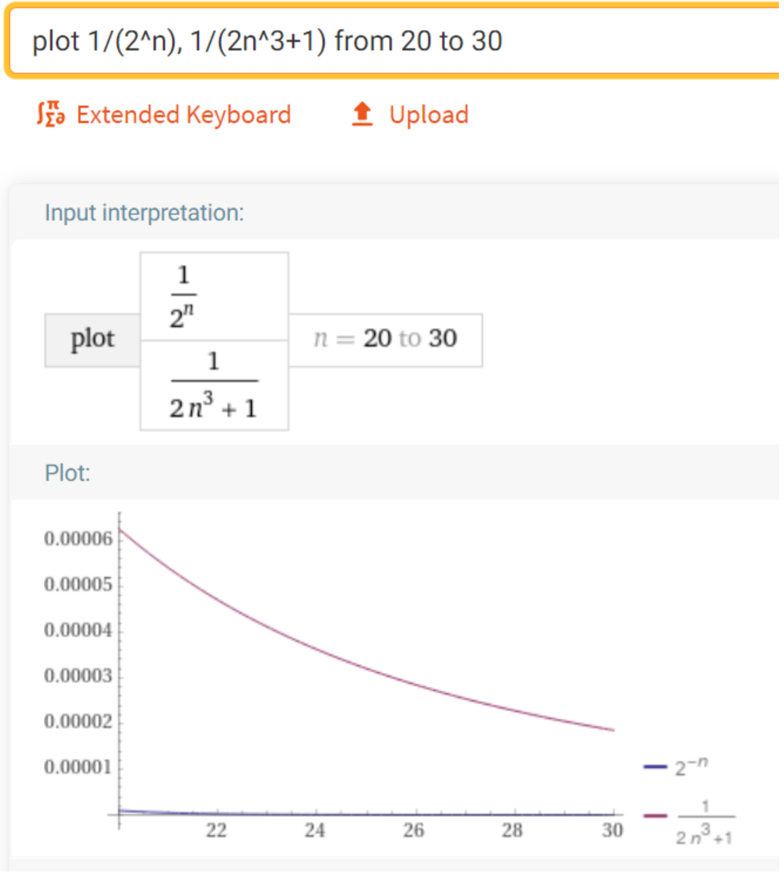
Polynomial vs. Exponential

- A polynomial function (over the integers) is of the form
$$f(n) = \sum_{i=0}^{d} a_i n^i = a_d n^d + a_{d-1} n^{d-1} \ldots a_1 n + a_0$$
where d is constant and $a_0, \ldots, a_d$ are integers
- For example: $n^2 + 5$, $2n^{1000000} + n^{1000} + 50n^{10}$
- A function f is dominated by a polynomial function if there exists a constant d such that for sufficiently large n's $f(n) < n^d$
(formally, there exists N such that for all $n > N$ it holds that $f(n) < n^d$)
- By abuse of language we sometime call such f also a polynomial, e.g. $n^5 + \log(n)$
- A function f is (dominated by) an exponential function if for sufficiently large n's $f(n) < c^{p(n)}$ for a constant c and a polynomial $p(\cdot)$
- For example: 2^n , $2^{(n^2)}$, 100000^n

Polynomial vs. Exponential

- Consider the functions $f(n) = \frac{1}{2n^3+1}$ and $g(n) = \frac{1}{2^n}$
- Which function is "smaller"?





Polynomial vs. Exponential

- A function is negligible if it approaches 0 faster than any inverse polynomial
- **Definition:** A function $f: \mathbb{N} \to \mathbb{R}$ is a negligible function if for any positive polynomial $p(\cdot)$ there exists N such that for all $n > N$ it holds that

$$f(n) < \frac{1}{p(n)}$$

- For example: 2^{-n}, $2^{-\sqrt{n}}$ and $2^{-\log^2(n)}$ are negligible functions
- $1/2$, $1/\log^2(n)$ and $1/n^5$ are non-negligible functions

Last Lecture

- Symmetric-key encryption
- Perfect secrecy

$$\Pr[M = m \mid C = c] = \Pr[M = m]$$

- Limitations of perfect secrecy
  - Considers security for only a single message
  - The key must be as long as the message

Can we guarantee "security" while avoiding these limitations?

This Week: Computational Security

What is "computational" security?

- The information is all there: $\mathrm{Enc}_k(m)$ may completely determine k and m
- It should be computationally infeasible to retrieve any useful information

E.g., 2000 years using current technology

Two realistic relaxations compared to last week:

1. Security is preserved only against computationally bounded adversaries
2. Allow such adversaries to succeed with some small probability

Small enough such that will essentially never happen

The Concrete Approach

"A scheme is (t, ϵ)-secure if every adversary running for time at most t succeeds in breaking the scheme with probability at most ϵ"

Sample parameters

- $t = 2^{60}$
  (order of the number of seconds since the big bang)
- $\epsilon = 2^{-30}$
  (expected to occur once every 100 years)
- $\epsilon = 2^{-60}$
  (expected to occur once every 100 billion years)
- Very important in practice, may be tailored to specific technology
- In general, hard to analyze
- Not always clear what's can we say if the adversary runs for time $2t$ or $t/2$

The Asymptotic Approach

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

Definition:
An algorithm A runs in **probabilistic polynomial-time** if there exists a polynomial $p(\cdot)$ such that, for any input $x \in \{0,1\}^*$ and random tape $r \in \{0,1\}^*$, the computation of $A(x; r)$ terminates within $p(|x|)$ steps.

The security parameter

- Gen takes as input the security parameter 1^n and outputs $k \in \mathcal{K}_n$
- Keys produced by $\mathrm{Gen}(1^n)$ should provide security against adversaries whose running time is polynomial in n (increasing n provides better security)
- $\mathcal{K} = \bigcup_{n\in\mathbb{N}} \mathcal{K}_n$, $\mathcal{M} = \bigcup_{n\in\mathbb{N}} \mathcal{M}_n$, $\mathcal{C} = \bigcup_{n\in\mathbb{N}} \mathcal{C}_n$

Why These Choices?

- **"Efficient": Probabilistic polynomial time (PPT)**
- **"Negligible": Smaller than any inverse polynomial**

Intuitively well-behaved under composition:

- $\mathrm{poly}(n) \times \mathrm{poly}(n) = \mathrm{poly}(n)$
  Polynomially many invocations of a PPT algorithm is still a PPT algorithm
- $\mathrm{poly}(n) \times \mathrm{negligible}(n) = \mathrm{negligible}(n)$
  Polynomially many invocations of a PPT algorithm that succeeds with a negligible probability is an algorithm that succeeds with a negligible probability overall

Outline

- **Security notion: Indistinguishable encryptions**
- **Basic primitive: Pseudorandom generator (PRG)**
- **PRG-based one-time pad**
- **Stream ciphers**

Indistinguishable Encryptions

The most basic notion of security for symmetric-key encryption

- Encryptions of any two messages should be indistinguishable
- Adversary still observes only a single ciphertext

$$\mathrm{Enc}_k(m_0) \approx \mathrm{Enc}_k(m_1)$$

Seems weaker compared to perfect secrecy

- Perfectly-secure encryption reveals no information
- Intuitively, what security does indistinguishable encryptions provide?

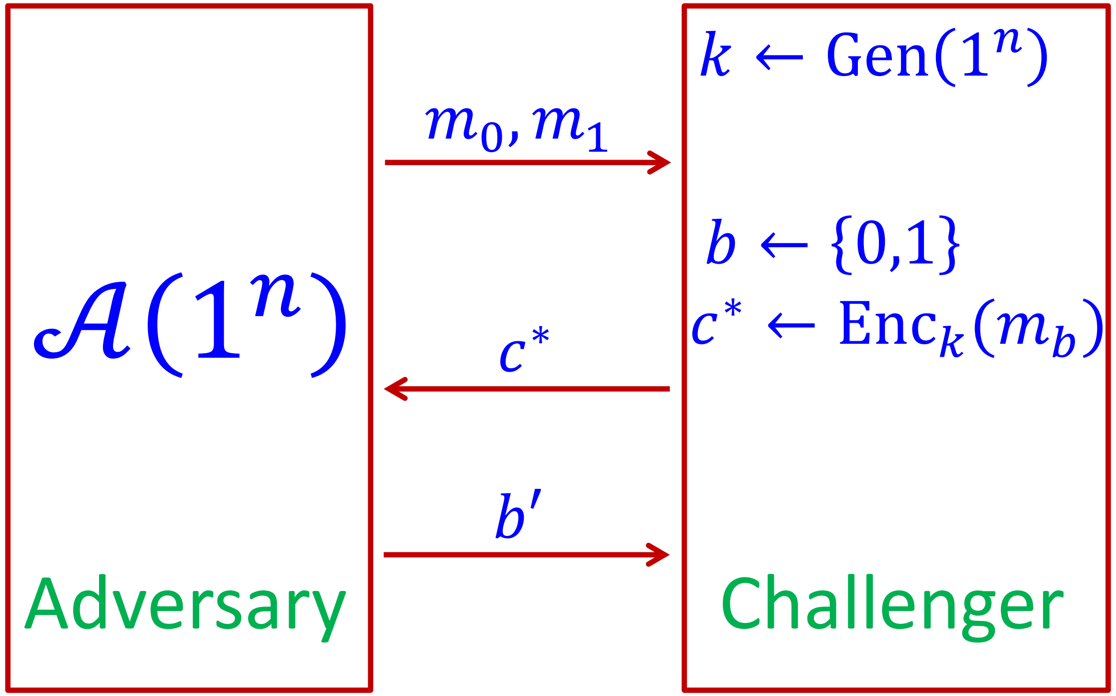
Indistinguishable Encryptions

Given $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ and an adversary $\mathcal{A}$, consider the experiment $\mathrm{IND}_{\Pi,\mathcal{A}}(n)$:

Definition:
Π has **indistinguishable encryptions** if for every PPT adversary $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that

$$\Pr\left[\mathrm{IND}_{\Pi,\mathcal{A}}(n) = 1\right] \leq \frac{1}{2} + \nu(n)$$

where the probability is taken over the random coins used by $\mathcal{A}$ and by the experiment



$$\mathrm{IND}_{\Pi,\mathcal{A}}(n) = \begin{cases} 1, & \text{if } b' = b \\ 0, & \text{otherwise} \end{cases}$$

$^*\ m_0, m_1 \in \mathcal{M}_n$

Semantic Security

- Semantic security [Goldwasser-Micali '82]:
  "Whatever" can be computed efficiently given the ciphertext,
  can essentially be computed efficiently without the ciphertext

Theorem:
Π is **semantically secure** if and only if it has **indistinguishable encryptions**

Why do we need both notions?

- Semantic security explains "what security means"
- Indistinguishability of encryptions is "easier to work with"

Turing Award '12

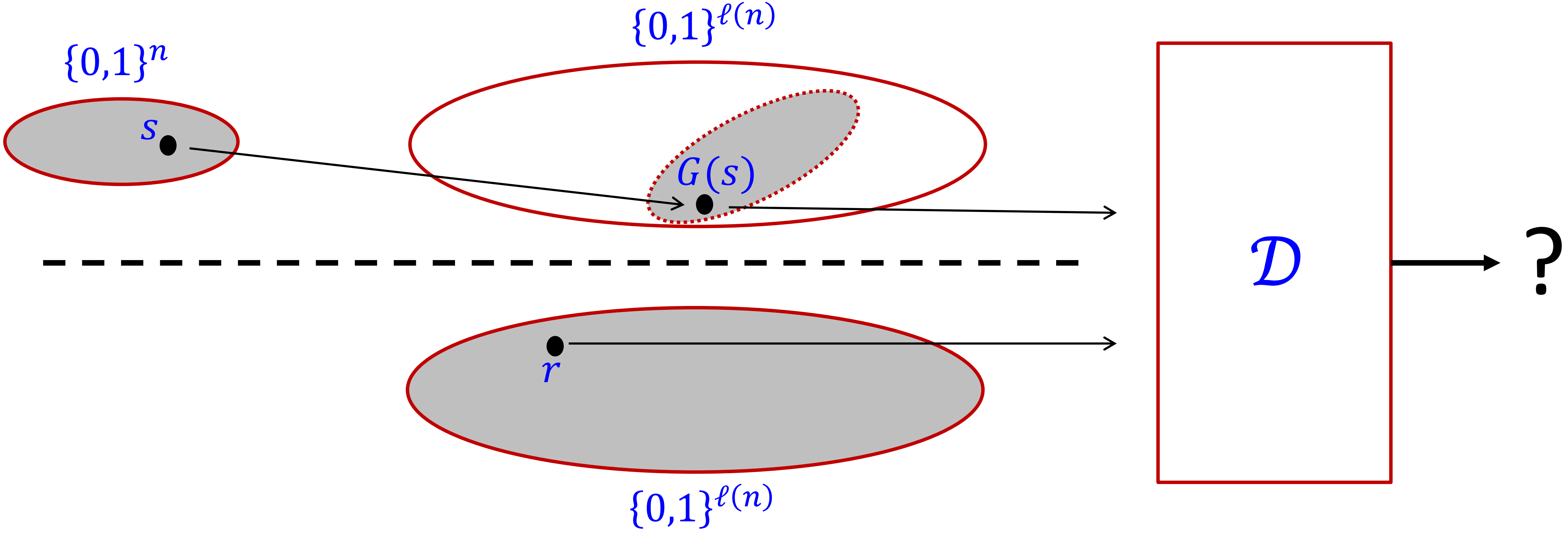
Outline

- **Security notion: Indistinguishable encryptions**
- **Basic primitive: Pseudorandom generator (PRG)**
- **PRG-based one-time pad**
- **Stream ciphers**

Pseudorandom Generators (PRGs)

Goal: Expand a short random seed into a long "random-looking" value

- $G: \{0,1\}^* \to \{0,1\}^*$
- "Random looking" = "indistinguishable" from the uniform distribution

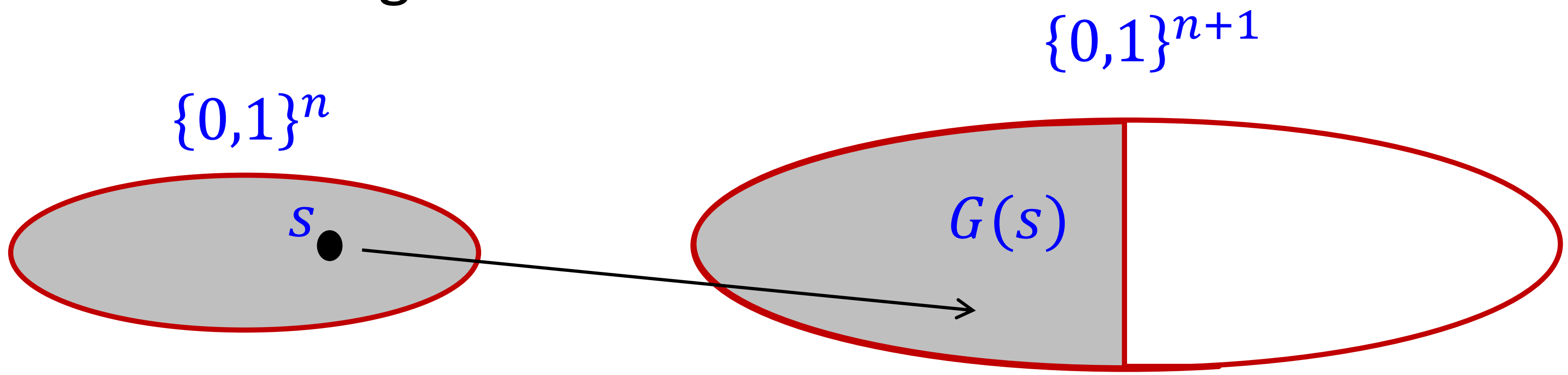


Pseudorandom Generators (PRGs)

Consider an expansion of 1 bit, i.e., $G: \{0,1\}^n \rightarrow \{0,1\}^{n+1}$

Half of $\{0,1\}^{n+1}$ is not in the image of G

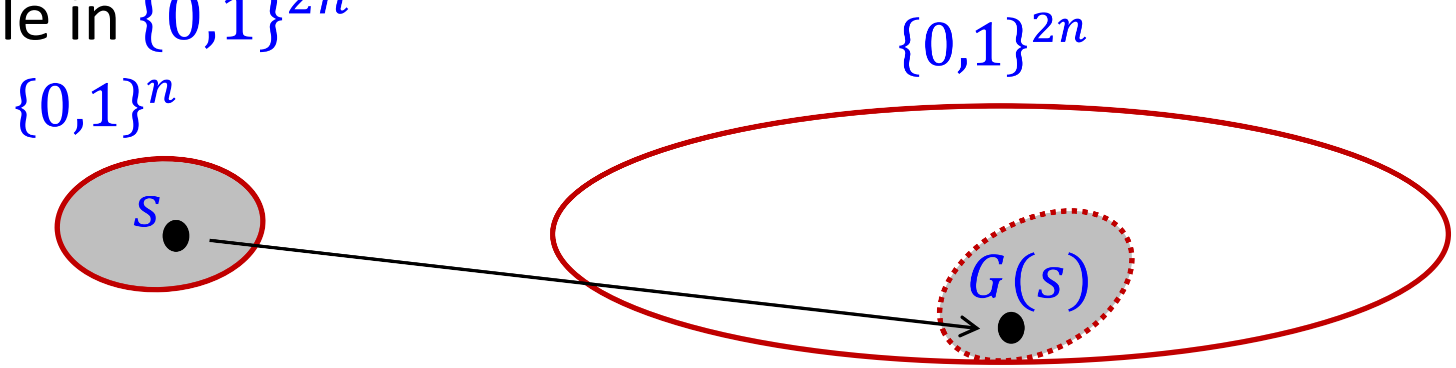
$$\frac{|\{0,1\}^n|}{|\{0,1\}^{n+1}|} = \frac{1}{2}$$



Consider an expansion of n bits, i.e., $G: \{0,1\}^n \rightarrow \{0,1\}^{2n}$

The image of G is negligible in $\{0,1\}^{2n}$

$$\frac{|\{0,1\}^n|}{|\{0,1\}^{2n}|} = \frac{2^n}{2^{2n}} = 2^{-n}$$



Pseudorandom Generators (PRGs)

Definition (PRG):
Let $G: \{0,1\}^* \rightarrow \{0,1\}^*$ be a poly-time computable function and let $\ell(\cdot)$ be a polynomial such that for any input $s \in \{0,1\}^n$ we have $G(s) \in \{0,1\}^{\ell(n)}$. Then, G is a **pseudorandom generator** if the following two conditions hold:

- **Expansion:** $\ell(n) > n$
- **Pseudorandomness:** For every PPT "distinguisher" $\mathcal{D}$ there exists a negligible function $\nu(\cdot)$ such that

$$\left| \Pr_{s \leftarrow \{0,1\}^n}\left[\mathcal{D}\big(G(s)\big) = 1\right] - \Pr_{r \leftarrow \{0,1\}^{\ell(n)}}\left[\mathcal{D}(r) = 1\right] \right| \leq \nu(n)$$

- The notation $x \leftarrow \{0,1\}^m$ denotes that x is sampled from the uniform distribution over $\{0,1\}^m$ (each value is obtained with probability $1/2^m$)

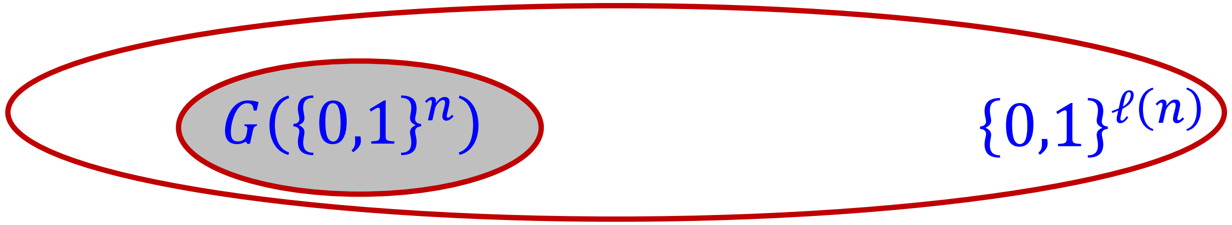
Pseudorandom Generators (PRGs)

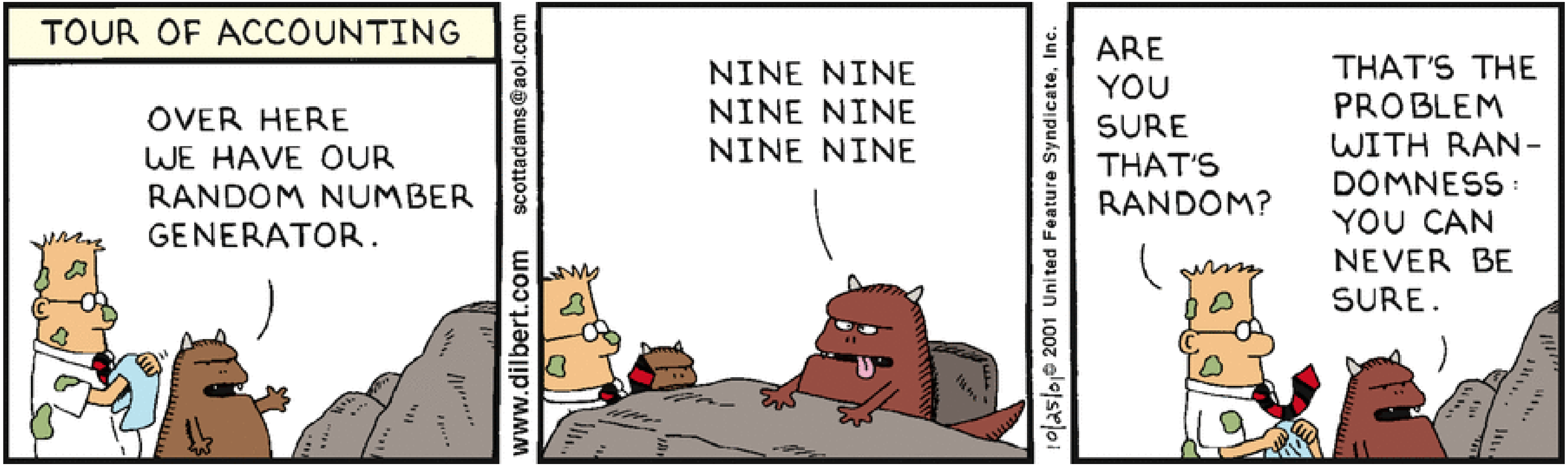
Definition (PRG):
Let $G: \{0,1\}^* \to \{0,1\}^*$ be a poly-time computable function and let $\ell(\cdot)$ be a polynomial such that for any input $s \in \{0,1\}^n$ we have $G(s) \in \{0,1\}^{\ell(n)}$. Then, G is a **pseudorandom generator** if the following two conditions hold:

- **Expansion:** $\ell(n) > n$
- **Pseudorandomness:** For every PPT "distinguisher" $\mathcal{D}$ there exists a negligible function $\nu(\cdot)$ such that

$$\left| \Pr_{s \leftarrow \{0,1\}^n}\left[\mathcal{D}\big(G(s)\big) = 1\right] - \Pr_{r \leftarrow \{0,1\}^{\ell(n)}}\left[\mathcal{D}(r) = 1\right] \right| \leq \nu(n)$$

"$G(s)$ is as good as random"





http://dilbert.com/strip/2001-10-25

Do PRGs Exist?

If so, then how difficult is it to construct a PRG?

Let's gain some intuition: Can you propose PRG candidates?

Recall the two properties:

- **Expansion:** $|G(s)| > |s|$
- **Pseudorandomness:** For every PPT $\mathcal{D}$ there exists a negligible $\nu(\cdot)$ such that

$$\left| \Pr_{s \leftarrow \{0,1\}^n}\left[\mathcal{D}\big(G(s)\big) = 1\right] - \Pr_{r \leftarrow \{0,1\}^{\ell(n)}}\left[\mathcal{D}(r) = 1\right] \right| \leq \nu(n)$$

Let's Try

- Consider the following candidates that expand a seed $s = s_1 \cdots s_n \in \{0,1\}^n$ by a single bit:

$G(s) = s_1 \cdots s_n 0$	Is it distinguishable from a truly random string $r_1 \cdots r_n r_{n+1}$?	YES
$G(s) = s_1 \cdots s_n s_1$	Is it distinguishable from a truly random string $r_1 \cdots r_n r_{n+1}$?	YES
$G(s) = s_1 \cdots s_n z$ where $z = s_1 \oplus \cdots \oplus s_n$	Is it distinguishable from a truly random string $r_1 \cdots r_n r_{n+1}$?	YES

- The existence of any PRG implies $P \neq NP$
- Constructions are known based on various computational assumptions

A Useful Fact

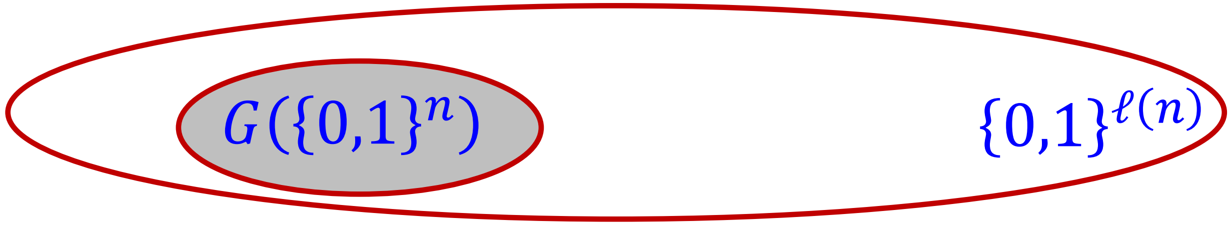
All efficiently testable statistical properties of the uniform distribution are preserved by the output of any PRG.

For example:

If G is a PRG then there exists a negligible function $\nu(\cdot)$ such that

$$\Pr_{s \leftarrow \{0,1\}^n}[\text{fraction of } 1'\text{s in } G(s) < 1/4] \leq \nu(n)$$

"$G(s)$ is as good as random"



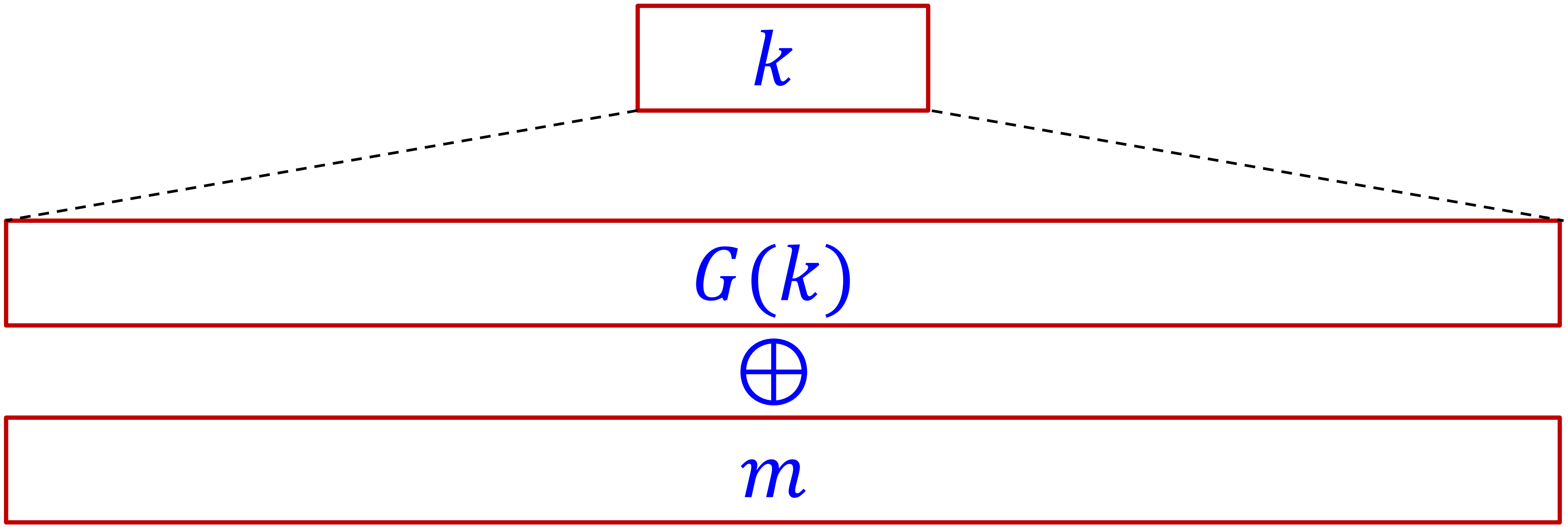
Outline

- **Security notion: Indistinguishable encryptions**
- **Basic primitive: Pseudorandom generator (PRG)**
- **PRG-based one-time pad**
- **Stream ciphers**

One-Time Pad Using a PRG

- Let G be a PRG with expansion $\ell(n)$
- $\mathcal{K}_n = \{0,1\}^n$ but $\mathcal{M}_n = \mathcal{C}_n = \{0,1\}^{\ell(n)}$
- $\mathrm{Gen}(1^n)$ samples $k \leftarrow \{0,1\}^n$
- $\mathrm{Enc}_k(m) = m \oplus G(k)$ & $\mathrm{Dec}_k(c) = c \oplus G(k)$

$\ell(n) = 2n$:
$|\mathcal{K}_n| = 2^n \ll 2^{2n} = |\mathcal{M}_n|$



One-Time Pad Using a PRG

- Let G be a PRG with expansion $\ell(n)$
- $\mathcal{K}_n = \{0,1\}^n$ but $\mathcal{M}_n = \mathcal{C}_n = \{0,1\}^{\ell(n)}$
- $\mathrm{Gen}(1^n)$ samples $k \leftarrow \{0,1\}^n$
- $\mathrm{Enc}_k(m) = m \oplus G(k)$ & $\mathrm{Dec}_k(c) = c \oplus G(k)$

$\ell(n) = 2n$:
$|\mathcal{K}_n| = 2^n \ll 2^{2n} = |\mathcal{M}_n|$

Theorem:
If G is a PRG, then the scheme has indistinguishable encryptions.

Paradigm: Proof by reduction

- Given an adversary $\mathcal{A}$ for the encryption scheme, construct a distinguisher $\mathcal{D}$ for the PRG
- $\mathcal{D}$ internally emulates $\mathcal{A}$
- $\mathcal{D}$'s efficiency and advantage are polynomially related to $\mathcal{A}$'s

One-Time Pad Using a PRG

- Let G be a PRG with expansion $\ell(n)$
- $\mathcal{K}_n = \{0,1\}^n$ but $\mathcal{M}_n = \mathcal{C}_n = \{0,1\}^{\ell(n)}$
- $\mathrm{Gen}(1^n)$ samples $k \leftarrow \{0,1\}^n$
- $\mathrm{Enc}_k(m) = m \oplus G(k)$ & $\mathrm{Dec}_k(c) = c \oplus G(k)$

$\ell(n) = 2n$:
$|\mathcal{K}_n| = 2^n \ll 2^{2n} = |\mathcal{M}_n|$

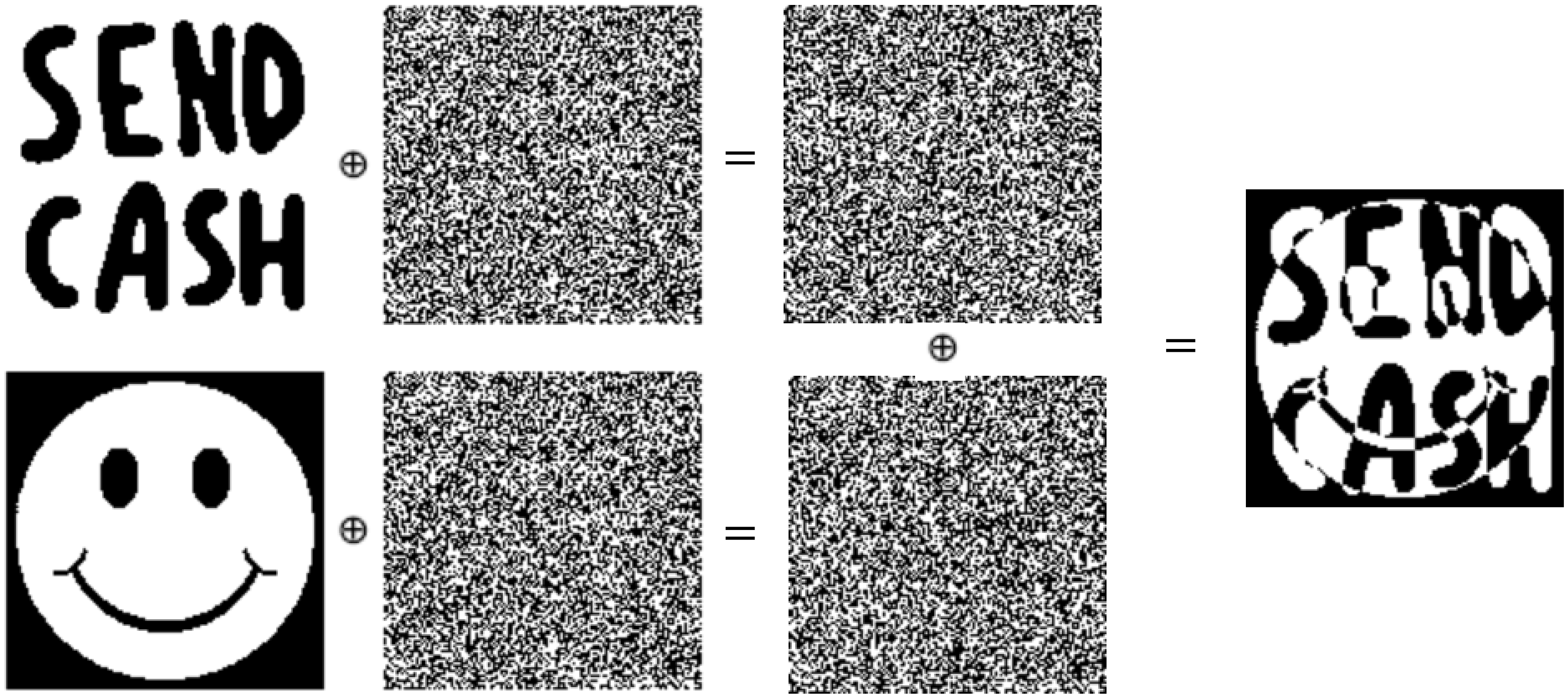
Theorem:
If G is a PRG, then the scheme has indistinguishable encryptions.

Significant progress but still only "one-time" security…

$$\mathrm{Enc}_k(m_1) \oplus \mathrm{Enc}_k(m_2) = m_1 \oplus G(k) \oplus m_2 \oplus G(k) = m_1 \oplus m_2$$

Key-Reuse attack

- MS Word/Excel 2002 used the same key when saving changes to the same document
- Illustration from https://cryptosmith.com/2008/05/31/stream-reuse/



Outline

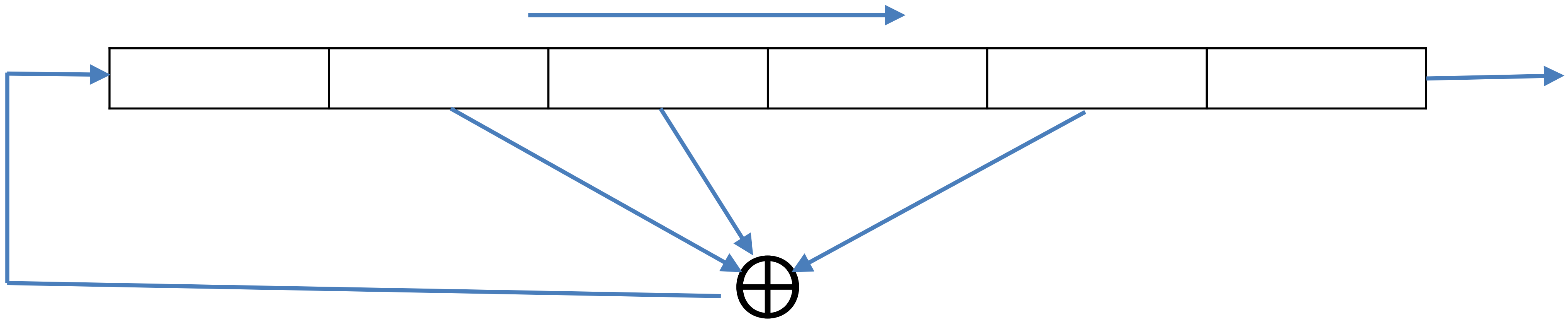
- **Security notion: Indistinguishable encryptions**
- **Basic primitive: Pseudorandom generator (PRG)**
- **PRG-based one-time pad**
- **Stream ciphers**

RC4

- Designed in 1987 by Ron Rivest (Rivest Cipher)
- Extremely fast, extremely simple, ideal for software
- Variable length key 40-2048 bits
- Generates blocks of 256 bytes (2048 bits)
- Very popular, used in many standards SSL/TLS, WEP, WPA
- Jan 2013: in a survey of 16 billions TLS connection ~50% protected using RC4
- Many known weaknesses:
  - 2001 Mantin-Shamir: $\Pr[2^{nd}\ byte = 0] = 2/256$
  - 2002 Mironov: 1^{st} byte has biased away from 0
  - 2011 Maitra et al.: bias in blocks 3-255
  - Quick solution – throw away first 512 bytes
  - 2013 AlFardan et al.: analyzed output from 2^{45} independent 128-bit RC4 keys
    found many new biases
    plaintext recovery attack against TLS

LFSR

- Linear feedback shift register
- Very useful for hardware-based design
- the initial state of the register is the seed
- In every round the cells are shifted to the right (the last cell is the output), the first cell becomes the XOR of certain locations



Used for:

- DVD encryption (CSS)
- GSM encryption (A5/1 and A5/2)
- Bluetooth (E0)

All broken

Salsa20

- Designed in 2005 by Dan Bernstein
- Part of the eSTREAM project
- Seed is 128/256 bits
- Uses additional nonce of 64 bits
- Can be used to encrypt up to 2^{70} bits
- In 2008 Bernstein designed ChaCha based on similar principles as Salsa, but with better diffusion
- 2014: Google replaced RC4 with ChaCha20 for TLS