Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- . . . but it requires long keys

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- . . . but it requires long keys
- This is inevitable if we insist on perfect secrecy (recall that, in a perfectly secret scheme, $|\mathcal{K}| \geq |\mathcal{M}|$)

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- ... but it requires long keys
- This is inevitable if we insist on perfect secrecy (recall that, in a perfectly secret scheme, $|\mathcal{K}| \geq |\mathcal{M}|$)

What more is there to do?

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- . . . but it requires long keys

- This is inevitable if we insist on perfect secrecy (recall that, in a perfectly secret scheme, $|\mathcal{K}| \geq |\mathcal{M}|$)

What more is there to do?

- We would still really like to have "secure" schemes with short keys...
- We need to give up on perfect secrecy

☹

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- . . . but it requires long keys

- This is inevitable if we insist on perfect secrecy (recall that, in a perfectly secret scheme, $|\mathcal{K}| \geq |\mathcal{M}|$)

What more is there to do?

- We would still really like to have "secure" schemes with short keys...
- We need to give up on perfect secrecy

☹

Can we relax the security definition in a meaningful way?

Computational secrecy

What if the adversary is not computationally unbounded?

Computational secrecy

What if the adversary is not computationally unbounded?

Say that the adversary is only able to run algorithms for 2^{112} clock cycles...

- Cost of this computation: $\approx$ 10000 times the gross world product since $300\,000$BC
- Number of clock cycles of a supercomputer running since the Big-Bang

Computational secrecy

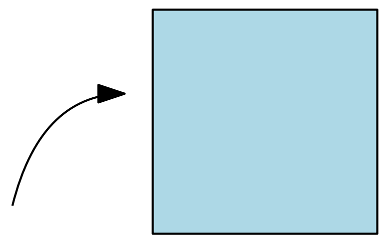
What if the adversary is not computationally unbounded?

Say that the adversary is only able to run algorithms for 2^{112} clock cycles...

- Cost of this computation: $\approx$ 10000 times the gross world product since $300\,000$BC
- Number of clock cycles of a supercomputer running since the Big-Bang

...and only manages to extract some information with probability 2^{-60}

- It is more likely that the next meteorite that hits Earth lands in this square



Computational secrecy

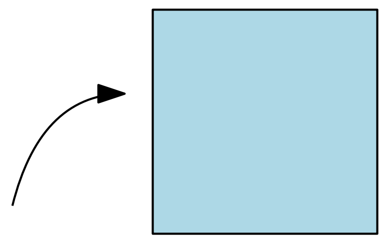
What if the adversary is not computationally unbounded?

Say that the adversary is only able to run algorithms for 2^{112} clock cycles...

- Cost of this computation: $\approx$ 10000 times the gross world product since $300\,000$BC
- Number of clock cycles of a supercomputer running since the Big-Bang

...and only manages to extract some information with probability 2^{-60}

- It is more likely that the next meteorite that hits Earth lands in this square



Do we need to be concerned?

Computational secrecy

We relax perfect secrecy in two ways:

- We allow secrecy to fail with some tiny probability
- We only restrict our attention to “efficient” attackers

Computational secrecy

Our starting point is the following (equivalent) definition of perfect secrecy:

Definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *with message space* $\mathcal{M}$ *is* **perfectly indistinguishable** *if for every* $\mathcal{A}$ *it holds:*

$$\Pr[\mathit{PrivK}^{\mathit{eav}}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$$

Computational secrecy

Our starting point is the following (equivalent) definition of perfect secrecy:

Definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *with message space* $\mathcal{M}$ *is* **perfectly indistinguishable** *if for every* $\mathcal{A}$ *it holds:*

$$\Pr[\mathit{PrivK}^{\mathit{eav}}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$$

We want to define a concept of **computational indistinguishability**

Computational secrecy

Our starting point is the following (equivalent) definition of perfect secrecy:

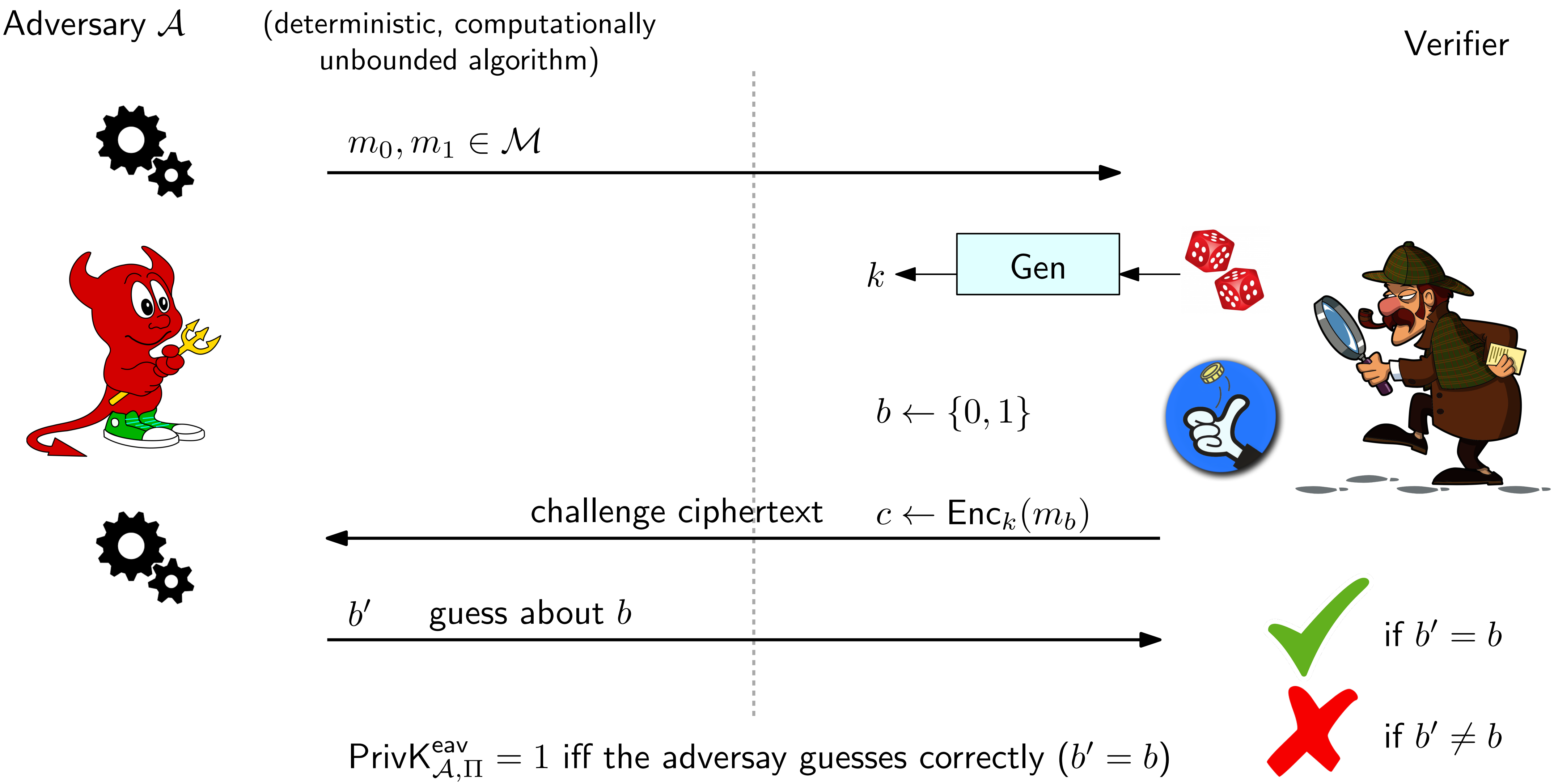
> **Definition**: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *with message space* $\mathcal{M}$ *is* **perfectly indistinguishable** *if for every* $\mathcal{A}$ *it holds:*
>
> $$\Pr[\mathit{PrivK}^{\mathit{eav}}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$$

We want to define a concept of **computational indistinguishability**

Two possible approaches:

- **Concrete**
- **Asymptotic**

Reminder: Perfect indistinguishability



Computational secrecy (concrete)

Candidate definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *is* (t, ε)*-indistinguishable if for every attacker* $\mathcal{A}$ *running in time at most* t*, it holds that:*

$$\Pr[\mathit{PrivK}^{\mathit{eav}}_{\mathcal{A},\Pi} = 1] \leq \frac{1}{2} + \varepsilon$$

Computational secrecy (concrete)

Candidate definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *is* (t, ε)*-indistinguishable if for every attacker* $\mathcal{A}$ *running in time at most* t*, it holds that:*

$$\Pr[\mathit{PrivK}^{eav}_{\mathcal{A},\Pi} = 1] \leq \frac{1}{2} + \varepsilon$$

Example: A $(2^{112}, 2^{-60})$-indistinguishable scheme remains secure against any adversary that runs for at most 2^{122} clock cycles (the adversary's advantage will be at most 2^{-60})

Computational secrecy (concrete)

Candidate definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *is* (t, ε)*-indistinguishable if for every attacker* $\mathcal{A}$ *running in time at most* t*, it holds that:*

$$\Pr[\mathit{PrivK}^{eav}_{\mathcal{A},\Pi} = 1] \leq \frac{1}{2} + \varepsilon$$

Example: A $(2^{112}, 2^{-60})$-indistinguishable scheme remains secure against any adversary that runs for at most 2^{122} clock cycles (the adversary's advantage will be at most 2^{-60})

Observation: $(\infty, 0)$-indistinguishability is equivalent to perfect indistinguishability

Problems with the concrete definition

- If a scheme is (t, ε)-indistinguishable, what can we say about $(t+1, \varepsilon)$?

Problems with the concrete definition

- If a scheme is (t, ε)-indistinguishable, what can we say about $(t+1, \varepsilon)$?

- What can we do in a clock cycle?

  The definition depends on the exact details of the computational model...

Problems with the concrete definition

- If a scheme is (t, ε)-indistinguishable, what can we say about $(t+1, \varepsilon)$?

- What can we do in a clock cycle?

  The definition depends on the exact details of the computational model...

- A scheme can be (t, ε)-indistinguishable for many choices of t and ε

  — How do we pick t?

Problems with the concrete definition

- If a scheme is (t, ε)-indistinguishable, what can we say about $(t+1, \varepsilon)$?

- What can we do in a clock cycle?

  The definition depends on the exact details of the computational model...

- A scheme can be (t, ε)-indistinguishable for many choices of t and ε

  — How do we pick t?

  — What if computers become faster?

Problems with the concrete definition

- If a scheme is (t, ε)-indistinguishable, what can we say about $(t+1, \varepsilon)$?

- What can we do in a clock cycle?

  The definition depends on the exact details of the computational model...

- A scheme can be (t, ε)-indistinguishable for many choices of t and ε

  — How do we pick t?

  — What if computers become faster?

  — We would like to have a scheme where users can adjust the security guarantees as desired

Problems with the concrete definition

- If a scheme is (t, ε)-indistinguishable, what can we say about $(t+1, \varepsilon)$?

- What can we do in a clock cycle?

  The definition depends on the exact details of the computational model...

- A scheme can be (t, ε)-indistinguishable for many choices of t and ε

  — How do we pick t?

  — What if computers become faster?

  — We would like to have a scheme where users can adjust the security guarantees as desired

Does not lead to a clean theory

Computational secrecy (asymptotic)

We only want to defend from *efficient* adversaries

Computational secrecy (asymptotic)

We only want to defend from *efficient* adversaries

- What does *efficient* mean?

Computational secrecy (asymptotic)

We only want to defend from *efficient* adversaries

- What does *efficient* mean?

  Usually, in complexity theory, efficient = polynomial-time

Computational secrecy (asymptotic)

We only want to defend from *efficient* adversaries

- What does *efficient* mean?

  Usually, in complexity theory, efficient = polynomial-time

- Polynomial with respect to what...?

Computational secrecy (asymptotic)

We only want to defend from *efficient* adversaries

- What does *efficient* mean?

  Usually, in complexity theory, efficient $=$ polynomial-time

- Polynomial with respect to what...?

  Introduce a new *security parameter* n

  — Allows to tune the security of the scheme (e.g., think of it as the key length)

  — Chosen by the honest parties (Alice and Bob)

  — Known by the adversary

Computational secrecy (asymptotic)

We only want to defend from *efficient* adversaries

- What does *efficient* mean?

  Usually, in complexity theory, efficient = polynomial-time

- Polynomial with respect to what...?

  Introduce a new *security parameter* n

  — Allows to tune the security of the scheme (e.g., think of it as the key length)

  — Chosen by the honest parties (Alice and Bob)

  — Known by the adversary

  Measure probabilities and running times as a function of n

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- We only restrict our attention to “efficient” attackers
- We allow secrecy to fail with some tiny probability

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- We only restrict our attention to “efficient” attackers ← polynomial running times
- We allow secrecy to fail with some tiny probability

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- We only restrict our attention to “efficient” attackers ⟵ polynomial running times
- We allow secrecy to fail with some tiny probability ⟵ probabilities that are *negligible* in n

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Equivalently:

- There exists a polynomial p such that $f(n) \leq p(n)$ for all $n > 0$

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Equivalently:

- There exists a polynomial p such that $f(n) \leq p(n)$ for all $n > 0$
- There exists N and c such that $f(n) \leq n^c$ for all $n \geq N$.

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Equivalently:

- There exists a polynomial p such that $f(n) \leq p(n)$ for all $n > 0$
- There exists N and c such that $f(n) \leq n^c$ for all $n \geq N$.

"$f(n)$ grows at most as fast as some polynomial in n"

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Equivalently:

- There exists a polynomial p such that $f(n) \leq p(n)$ for all $n > 0$
- There exists N and c such that $f(n) \leq n^c$ for all $n \geq N$.

"$f(n)$ grows at most as fast as <u>some</u> polynomial in n"

Small abuse: f is sometimes said to be polynomial

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Equivalently:

- There exists a polynomial p such that $f(n) \leq p(n)$ for all $n > 0$
- There exists N and c such that $f(n) \leq n^c$ for all $n \geq N$.

"$f(n)$ grows at most as fast as <u>some</u> polynomial in n"

Small abuse: f is sometimes said to be polynomial

A function η is **negligible** if, **for every** polynomial p, $\eta(n) = O(\frac{1}{p(n)})$.

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Equivalently:

- There exists a polynomial p such that $f(n) \leq p(n)$ for all $n > 0$
- There exists N and c such that $f(n) \leq n^c$ for all $n \geq N$.

"$f(n)$ grows at most as fast as some polynomial in n"

Small abuse: f is sometimes said to be polynomial

A function η is **negligible** if, **for every** polynomial p, $\eta(n) = O(\frac{1}{p(n)})$.

Equivalently:

- **For every** polynomial p, there exists $N \geq 1$ such that $\eta(n) \leq \frac{1}{p(n)}$ for all $n \geq N$

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Equivalently:

- There exists a polynomial p such that $f(n) \leq p(n)$ for all $n > 0$
- There exists N and c such that $f(n) \leq n^c$ for all $n \geq N$.

"$f(n)$ grows at most as fast as <u>some</u> polynomial in n"

Small abuse: f is sometimes said to be polynomial

A function η is **negligible** if, **for every** polynomial p, $\eta(n) = O(\frac{1}{p(n)})$.

Equivalently:

- **For every** polynomial p, there exists $N \geq 1$ such that $\eta(n) \leq \frac{1}{p(n)}$ for all $n \geq N$
- **For every** $c \geq 0$, there exists $N \geq 1$ such that $\eta(n) \leq \frac{1}{n^c}$ for all $n \geq N$

Definitions

A function $f : \mathbb{N}^+ \to \mathbb{R}^+$ is **polynomially bounded** if $f(n) = O(n^c)$ for some constant c.

Equivalently:

- There exists a polynomial p such that $f(n) \leq p(n)$ for all $n > 0$
- There exists N and c such that $f(n) \leq n^c$ for all $n \geq N$.

"$f(n)$ grows at most as fast as <u>some</u> polynomial in n"

Small abuse: f is sometimes said to be polynomial

A function η is **negligible** if, **for every** polynomial p, $\eta(n) = O(\frac{1}{p(n)})$.

Equivalently:

- **For every** polynomial p, there exists $N \geq 1$ such that $\eta(n) \leq \frac{1}{p(n)}$ for all $n \geq N$
- **For every** $c \geq 0$, there exists $N \geq 1$ such that $\eta(n) \leq \frac{1}{n^c}$ for all $n \geq N$

"$\eta(n)$ approaches 0 faster than the inverses of <u>all</u> polynomials in n"

Closure properties (poly + poly)

If $f(n)$ and $g(n)$ are polynomially bounded then $h(n) = f(n) + g(n)$ is polynomially bounded

Closure properties (poly + poly)

If $f(n)$ and $g(n)$ are polynomially bounded then $h(n) = f(n) + g(n)$ is polynomially bounded

- There is some N and some c such that $f(n) \leq n^c$ for all $n \geq N$
- There is some N' and some c' such that $g(n) \leq n^{c'}$ for all $n \geq N'$

Closure properties (poly + poly)

If $f(n)$ and $g(n)$ are polynomially bounded then $h(n) = f(n) + g(n)$ is polynomially bounded

- There is some N and some c such that $f(n) \leq n^c$ for all $n \geq N$
- There is some N' and some c' such that $g(n) \leq n^{c'}$ for all $n \geq N'$
- For all $n \geq \max\{N, N', 2\}$:

$$h(n) = f(n) + g(n) \leq n^c + n^{c'} \leq 2n^{\max\{c,c'\}} \leq n^{\max\{c,c'\}+1}$$

Closure properties (poly + poly)

If $f(n)$ and $g(n)$ are polynomially bounded then $h(n) = f(n) + g(n)$ is polynomially bounded

- There is some N and some c such that $f(n) \leq n^c$ for all $n \geq N$
- There is some N' and some c' such that $g(n) \leq n^{c'}$ for all $n \geq N'$
- For all $n \geq \max\{N, N', 2\}$:

$$h(n) = f(n) + g(n) \leq n^c + n^{c'} \leq 2n^{\max\{c,c'\}} \leq n^{\max\{c,c'\}+1}$$

The time spent calling two polynomially bounded subroutines (sequentially) is polynomially bounded

Closure properties (poly · poly)

If $f(n)$ and $g(n)$ are polynomially bounded then $h(n) = f(n) \cdot g(n)$ is polynomially bounded

Closure properties (poly $\cdot$ poly)

If $f(n)$ and $g(n)$ are polynomially bounded then $h(n) = f(n) \cdot g(n)$ is polynomially bounded

- There is some N and some c such that $f(n) \leq n^c$ for all $n \geq N$
- There is some N' and some c' such that $g(n) \leq n^{c'}$ for all $n \geq N'$

Closure properties (poly · poly)

If $f(n)$ and $g(n)$ are polynomially bounded then $h(n) = f(n) \cdot g(n)$ is polynomially bounded

- There is some N and some c such that $f(n) \leq n^c$ for all $n \geq N$
- There is some N' and some c' such that $g(n) \leq n^{c'}$ for all $n \geq N'$
- For all $n \geq \max\{N, N'\}$:

$$h(n) = f(n) \cdot g(n) \leq n^c \cdot n^{c'} \leq n^{c+c'}$$

Closure properties (poly · poly)

If $f(n)$ and $g(n)$ are polynomially bounded then $h(n) = f(n) \cdot g(n)$ is polynomially bounded

- There is some N and some c such that $f(n) \leq n^c$ for all $n \geq N$
- There is some N' and some c' such that $g(n) \leq n^{c'}$ for all $n \geq N'$
- For all $n \geq \max\{N, N'\}$:

$$h(n) = f(n) \cdot g(n) \leq n^c \cdot n^{c'} \leq n^{c+c'}$$

The time spent calling a polynomially bounded subroutine a polynomially bounded number of times is polynomially bounded

Closure properties (negligible $+$ negligible)

If $\eta(n)$ and $f(n)$ are negligible then $h(n) = \eta(n) + f(n)$ is negligible

Closure properties (negligible $+$ negligible)

If $\eta(n)$ and $f(n)$ are negligible then $h(n) = \eta(n) + f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$

Closure properties (negligible + negligible)

If $\eta(n)$ and $f(n)$ are negligible then $h(n) = \eta(n) + f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$
- There is some N' such that $\eta(n) \leq \frac{1}{n^{c+1}}$ for all $n \geq N'$
- There is some N'' such that $f(n) \leq \frac{1}{n^{c+1}}$ for all $n \geq N''$

Closure properties (negligible + negligible)

If $\eta(n)$ and $f(n)$ are negligible then $h(n) = \eta(n) + f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$
- There is some N' such that $\eta(n) \leq \frac{1}{n^{c+1}}$ for all $n \geq N'$
- There is some N'' such that $f(n) \leq \frac{1}{n^{c+1}}$ for all $n \geq N''$
- For all $n \geq \max\{N, N', 2\}$:

$$h(n) = \eta(n) + f(n) \leq \frac{2}{n^{c+1}} = \frac{2}{n} \cdot \frac{1}{n^c} \leq \frac{1}{n^c}$$

Closure properties (negligible + negligible)

If $\eta(n)$ and $f(n)$ are negligible then $h(n) = \eta(n) + f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$
- There is some N' such that $\eta(n) \leq \frac{1}{n^{c+1}}$ for all $n \geq N'$
- There is some N'' such that $f(n) \leq \frac{1}{n^{c+1}}$ for all $n \geq N''$
- For all $n \geq \max\{N, N', 2\}$:

$$h(n) = \eta(n) + f(n) \leq \frac{2}{n^{c+1}} = \frac{2}{n} \cdot \frac{1}{n^c} \leq \frac{1}{n^c}$$

The probability of failure of an algorithm that calls two subroutines that fail with negligible probability is negligible

Closure properties (negligible · poly)

If $\eta(n)$ is negligible and $f(n)$ is polynomially bounded then $h(n) = \eta(n) \cdot f(n)$ is negligible

Closure properties (negligible · poly)

If $\eta(n)$ is negligible and $f(n)$ is polynomially bounded then $h(n) = \eta(n) \cdot f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$

Closure properties (negligible · poly)

If $\eta(n)$ is negligible and $f(n)$ is polynomially bounded then $h(n) = \eta(n) \cdot f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$
- There is some N' and some c' such that $f(n) \leq n^{c'}$ for all $n \geq N'$

Closure properties (negligible · poly)

If $\eta(n)$ is negligible and $f(n)$ is polynomially bounded then $h(n) = \eta(n) \cdot f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$
- There is some N' and some c' such that $f(n) \leq n^{c'}$ for all $n \geq N'$
- Pick N such that $\eta(n) \leq \frac{1}{n^{c+c'}}$ for all $n \geq N$ **Such N exists, why?**

Closure properties (negligible · poly)

If $\eta(n)$ is negligible and $f(n)$ is polynomially bounded then $h(n) = \eta(n) \cdot f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$
- There is some N' and some c' such that $f(n) \leq n^{c'}$ for all $n \geq N'$
- Pick N such that $\eta(n) \leq \frac{1}{n^{c+c'}}$ for all $n \geq N$ **Such N exists, why?**
- For all $n \geq \max\{N, N'\}$:

$$h(n) = \eta(n) \cdot f(n) \leq \frac{1}{n^{c+c'}} \cdot n^{c'} = \frac{1}{n^c}$$

Closure properties (negligible · poly)

If $\eta(n)$ is negligible and $f(n)$ is polynomially bounded then $h(n) = \eta(n) \cdot f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$
- There is some N' and some c' such that $f(n) \leq n^{c'}$ for all $n \geq N'$
- Pick N such that $\eta(n) \leq \frac{1}{n^{c+c'}}$ for all $n \geq N$ **Such N exists, why?**
- For all $n \geq \max\{N, N'\}$:

$$h(n) = \eta(n) \cdot f(n) \leq \frac{1}{n^{c+c'}} \cdot n^{c'} = \frac{1}{n^c}$$

The probability of failure of an algorithm that makes a polynomially bounded number of calls to a subroutine that fails with negligible probability is negligible

Closure properties (negligible · poly)

If $\eta(n)$ is negligible and $f(n)$ is polynomially bounded then $h(n) = \eta(n) \cdot f(n)$ is negligible

- Pick any c, we show that there exists $N \geq 1$ such that $h(n) \leq \frac{1}{n^c}$ for all $n \geq N$
- There is some N' and some c' such that $f(n) \leq n^{c'}$ for all $n \geq N'$
- Pick N such that $\eta(n) \leq \frac{1}{n^{c+c'}}$ for all $n \geq N$ **Such N exists, why?**
- For all $n \geq \max\{N, N'\}$:

$$h(n) = \eta(n) \cdot f(n) \leq \frac{1}{n^{c+c'}} \cdot n^{c'} = \frac{1}{n^c}$$

The probability of failure of an algorithm that makes a polynomially bounded number of calls to a subroutine that fails with negligible probability is negligible

As a special case, the product of two negligible functions is negligible

Negligible and polynomially bounded functions

Which of the following functions are polynomially bounded? Which are negligible?

$n^2 + 4n - 2$	n^{100}	$n^3 + \cos(n)$	$n!$
$\frac{1}{n^{10}} + 2^{-n/2}$	2^n	3^{-n}	$\sqrt[3]{n} + \frac{1}{n}$
$n^{-n} \cdot (n^5 + n^2)$	$2^{\sqrt{n}}$	$\sqrt{n}$	$42 - \frac{1}{1+\log n}$
$4^{\sqrt{\log n}}$	n^{-5}	$2^{-\log n \cdot \log\log n}$	$(1 + \frac{1}{n})^n$

Negligible and polynomially bounded functions

Which of the following functions are polynomially bounded? Which are negligible?

$n^2 + 4n - 2$	n^{100}	$n^3 + \cos(n)$	$n!$
$\frac{1}{n^{10}} + 2^{-n/2}$	2^n	3^{-n}	$\sqrt[3]{n} + \frac{1}{n}$
$n^{-n} \cdot (n^5 + n^2)$	$2^{\sqrt{n}}$	$\sqrt{n}$	$42 - \frac{1}{1+\log n}$
$4^{\sqrt{\log n}}$	n^{-5}	$2^{-\log n \cdot \log\log n}$	$(1 + \frac{1}{n})^n$

Negligible and polynomially bounded functions

Which of the following functions are polynomially bounded? Which are negligible?

$n^2 + 4n - 2$	n^{100}	$n^3 + \cos(n)$	$n!$
$\frac{1}{n^{10}} + 2^{-n/2}$	2^n	3^{-n}	$\sqrt[3]{n} + \frac{1}{n}$
$n^{-n} \cdot (n^5 + n^2)$	$2^{\sqrt{n}}$	$\sqrt{n}$	$42 - \frac{1}{1+\log n}$
$4^{\sqrt{\log n}}$	n^{-5}	$2^{-\log n \cdot \log\log n}$	$(1 + \frac{1}{n})^n$

Private-key encryption schemes, redefined

Before defining computational secrecy, we need to redefine private-key encryption schemes to take into account the security parameter

Private-key encryption schemes, redefined

Before defining computational secrecy, we need to redefine private-key encryption schemes to take into account the security parameter

The default message space $\mathcal{M}$ is $\{0, 1\}^*$. A private-key encryption scheme consists of three algorithms:

Private-key encryption schemes, redefined

Before defining computational secrecy, we need to redefine private-key encryption schemes to take into account the security parameter

The default message space $\mathcal{M}$ is $\{0,1\}^*$. A private-key encryption scheme consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes 1^n (i.e., n written in unary) as input and outputs a *key* $k \in \mathcal{K}$. W.l.o.g. we assume that $|k| \geq n$. We write $k \leftarrow \mathsf{Gen}(1^n)$

Private-key encryption schemes, redefined

Before defining computational secrecy, we need to redefine private-key encryption schemes to take into account the security parameter

The default message space $\mathcal{M}$ is $\{0,1\}^*$. A private-key encryption scheme consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes 1^n (i.e., n written in unary) as input and outputs a *key* $k \in \mathcal{K}$. W.l.o.g. we assume that $|k| \geq n$. We write $k \leftarrow \mathsf{Gen}(1^n)$

Private-key encryption schemes, redefined

Before defining computational secrecy, we need to redefine private-key encryption schemes to take into account the security parameter

The default message space $\mathcal{M}$ is $\{0,1\}^*$. A private-key encryption scheme consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes 1^n (i.e., n written in unary) as input and outputs a *key* $k \in \mathcal{K}$. W.l.o.g. we assume that $|k| \geq n$. We write $k \leftarrow \text{Gen}(1^n)$

- Enc is a (possibly randomized) polynomial-time algorithm that takes as input a key $k \in \mathcal{K}$ and a *message* $m \in \mathcal{M}$ and outputs a *ciphertext* c.

Private-key encryption schemes, redefined

Before defining computational secrecy, we need to redefine private-key encryption schemes to take into account the security parameter

The default message space $\mathcal{M}$ is $\{0,1\}^*$. A private-key encryption scheme consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes 1^n (i.e., n written in unary) as input and outputs a *key* $k \in \mathcal{K}$. W.l.o.g. we assume that $|k| \geq n$. We write $k \leftarrow \mathsf{Gen}(1^n)$
- Enc is a (possibly randomized) polynomial-time algorithm that takes as input a key $k \in \mathcal{K}$ and a *message* $m \in \mathcal{M}$ and outputs a *ciphertext* c.

Private-key encryption schemes, redefined

Before defining computational secrecy, we need to redefine private-key encryption schemes to take into account the security parameter

The default message space $\mathcal{M}$ is $\{0,1\}^*$. A private-key encryption scheme consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes 1^n (i.e., n written in unary) as input and outputs a *key* $k \in \mathcal{K}$. W.l.o.g. we assume that $|k| \geq n$. We write $k \leftarrow \text{Gen}(1^n)$
- Enc is a (possibly randomized) polynomial-time algorithm that takes as input a key $k \in \mathcal{K}$ and a *message* $m \in \mathcal{M}$ and outputs a *ciphertext* c.
- Dec is a deterministic polynomial-time algorithm that takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$ or an error, denoted by $\perp$, if c cannot be obtained by encrypting m.

Private-key encryption schemes, redefined

Before defining computational secrecy, we need to redefine private-key encryption schemes to take into account the security parameter

The default message space $\mathcal{M}$ is $\{0,1\}^*$. A private-key encryption scheme consists of three algorithms:

- Gen is a randomized polynomial-time algorithm that takes 1^n (i.e., n written in unary) as input and outputs a *key* $k \in \mathcal{K}$. W.l.o.g. we assume that $|k| \geq n$. We write $k \leftarrow \mathsf{Gen}(1^n)$

- Enc is a (possibly randomized) polynomial-time algorithm that takes as input a key $k \in \mathcal{K}$ and a *message* $m \in \mathcal{M}$ and outputs a *ciphertext* c.

- Dec is a deterministic polynomial-time algorithm that takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs a message $m \in \mathcal{M}$ or an error, denoted by $\perp$, if c cannot be obtained by encrypting m.

If $M = \{0,1\}^{\ell(n)}$ then $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a **fixed-length** private-key encryption scheme (for messages of length $\ell(n)$)

The adversarial indistinguishability experiment, revisited

Adversary $\mathcal{A}$ — probabilistic polynomial-time algorithm with input 1^n

Verifier

$m_0, m_1 \in \mathcal{M}$

$(|m_0| = |m_1|)$

$k \leftarrow$ Gen $\leftarrow$

$b \leftarrow \{0,1\}$

challenge ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$

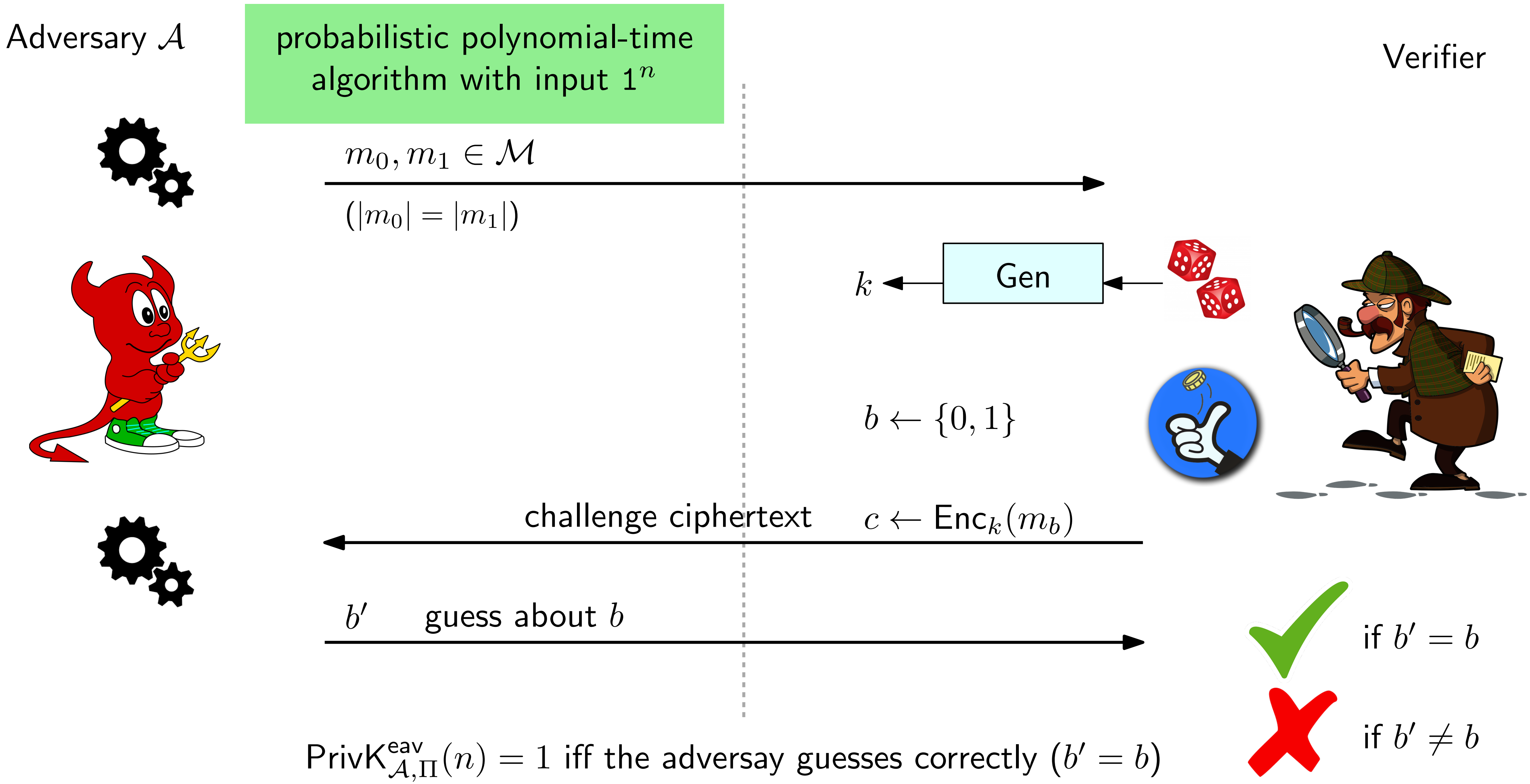
b' guess about b

if $b' = b$

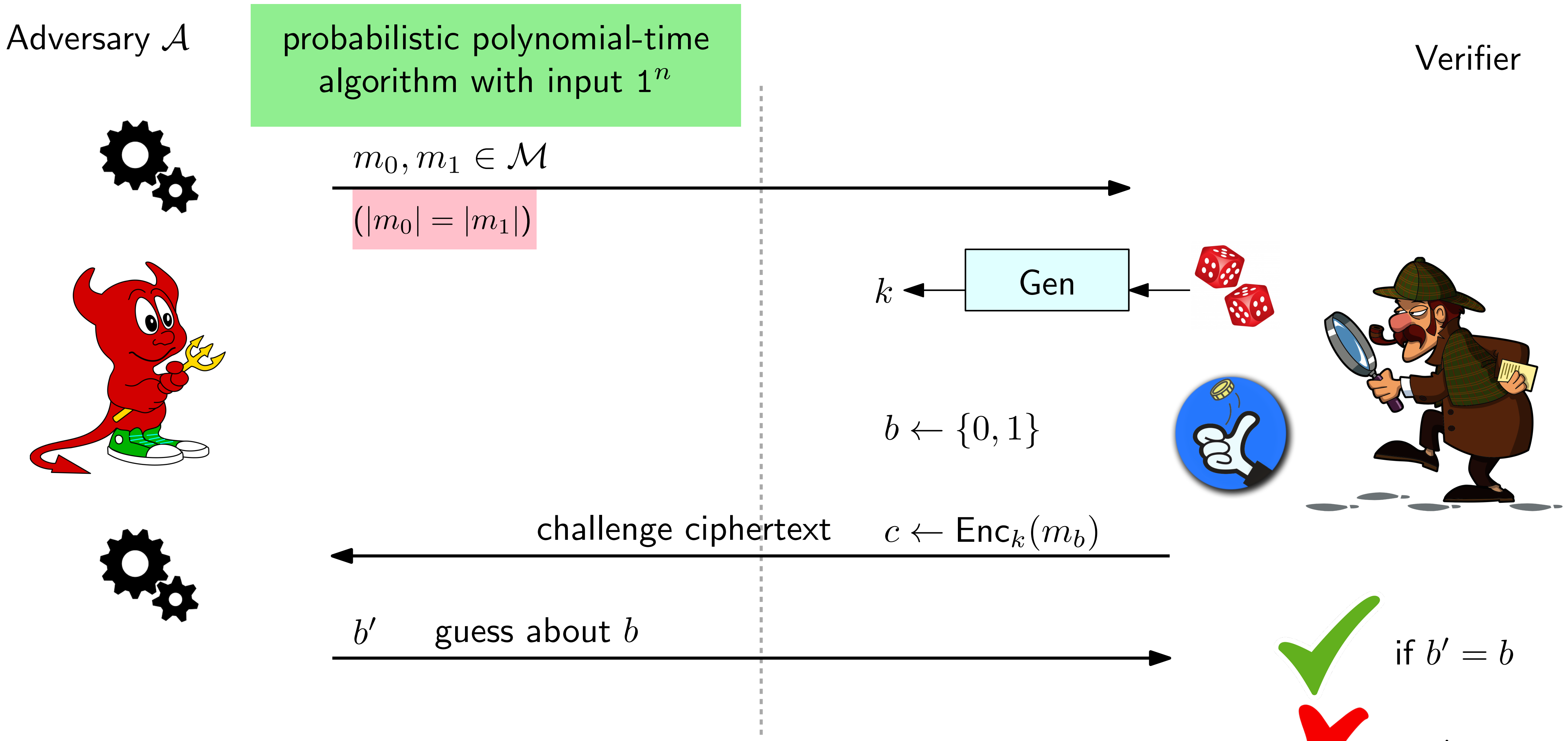
if $b' \neq b$

$\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1$ iff the adversay guesses correctly $(b' = b)$

The adversarial indistinguishability experiment, revisited



The adversarial indistinguishability experiment, revisited



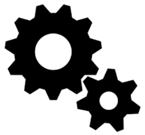
$\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1$ iff the adversay guesses correctly $(b' = b)$

The adversarial indistinguishability experiment, revisited

Adversary $\mathcal{A}$

probabilistic polynomial-time algorithm with input 1^n

Verifier



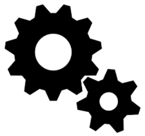
$m_0, m_1 \in \mathcal{M}$

$(|m_0| = |m_1|)$

$k \leftarrow$ Gen $\leftarrow$

$b \leftarrow \{0,1\}$

challenge ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$



b' guess about b

 if $b' = b$

 if $b' \neq b$

Notation includes the security parameter

$\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1$ iff the adversay guesses correctly $(b' = b)$

Computational indistinguishability (asymptotic)

Definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *has indistinguishable encryptions in the presence of an eavesdropper (is* **EAV-secure***) if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ε *such that:*

$$\Pr[\mathit{PrivK}^{eav}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Computational indistinguishability (asymptotic)

Definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *has indistinguishable encryptions in the presence of an eavesdropper (is* **EAV-secure***) if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ε *such that:*

$$\Pr[\mathit{PrivK}^{eav}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Computational indistinguishability (asymptotic)

Definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *has indistinguishable encryptions in the presence of an eavesdropper (is* **EAV-secure***) if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ε *such that:*

$$\Pr[\mathit{PrivK}^{eav}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Computational indistinguishability (asymptotic)

Definition: *A private key encryption scheme* $\Pi = (Gen, Enc, Dec)$ *has indistinguishable encryptions in the presence of an eavesdropper (is* **EAV-secure***) if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ε *such that:*

$$\Pr[\mathit{PrivK}^{eav}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Observation: perfect indistinguishability implies EAV-security

Example 1

Consider a scheme where:

- $\mathsf{Gen}(1^n)$ returns a key chosen uniformly at random in $\{0,1\}^n$
- The best possible adversary $\mathcal{A}$ performs a brute-force search over the key space
- If the running time of the adversary is $t(n)$ then:

$$\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + O\left(\frac{t(n)}{2^n}\right)$$

Example 1

Consider a scheme where:

- $\mathsf{Gen}(1^n)$ returns a key chosen uniformly at random in $\{0,1\}^n$
- The best possible adversary $\mathcal{A}$ performs a brute-force search over the key space
- If the running time of the adversary is $t(n)$ then:

$$\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + O\left(\frac{t(n)}{2^n}\right)$$

Is this scheme EAV-secure?

Example 1

Consider a scheme where:

- $\mathsf{Gen}(1^n)$ returns a key chosen uniformly at random in $\{0,1\}^n$
- The best possible adversary $\mathcal{A}$ performs a brute-force search over the key space
- If the running time of the adversary is $t(n)$ then:

$$\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + O\left(\frac{t(n)}{2^n}\right)$$

Is this scheme EAV-secure? Yes!

For all polynomial running times $t(n)$, all functions in $O\left(\frac{t(n)}{2^n}\right)$ are negligible

Example 2

Consider a scheme where:

- $\mathsf{Enc}_k(m)$ runs in $n^2 \cdot |m|$ steps
- Breaking the scheme requires 2^n steps

Example 2

Consider a scheme where:

- $\mathsf{Enc}_k(m)$ runs in $n^2 \cdot |m|$ steps
- Breaking the scheme requires 2^n steps

What happens when computers get four times faster?

Example 2

Consider a scheme where:

- $\mathsf{Enc}_k(m)$ runs in $n^2 \cdot |m|$ steps
- Breaking the scheme requires 2^n steps

What happens when computers get four times faster?

- Alice and Bob can decide to increase the security parameter from n to $2n$

Example 2

Consider a scheme where:

- $\mathsf{Enc}_k(m)$ runs in $n^2 \cdot |m|$ steps
- Breaking the scheme requires 2^n steps

What happens when computers get four times faster?

- Alice and Bob can decide to increase the security parameter from n to $2n$
- The number of steps of $\mathsf{Enc}_k(m)$ becomes $(2n)^2 \cdot |m| = 4n^2 \cdot |m|$, and the actual time spent stays the same

Example 2

Consider a scheme where:

- $\mathsf{Enc}_k(m)$ runs in $n^2 \cdot |m|$ steps
- Breaking the scheme requires 2^n steps

What happens when computers get four times faster?

- Alice and Bob can decide to increase the security parameter from n to $2n$
- The number of steps of $\mathsf{Enc}_k(m)$ becomes $(2n)^2 \cdot |m| = 4n^2 \cdot |m|$, and the actual time spent stays the same
- The number of steps required to break the scheme becomes 2^{2n}

Example 2

Consider a scheme where:

- $\mathsf{Enc}_k(m)$ runs in $n^2 \cdot |m|$ steps
- Breaking the scheme requires 2^n steps

What happens when computers get four times faster?

- Alice and Bob can decide to increase the security parameter from n to $2n$
- The number of steps of $\mathsf{Enc}_k(m)$ becomes $(2n)^2 \cdot |m| = 4n^2 \cdot |m|$, and the actual time spent stays the same
- The number of steps required to break the scheme becomes 2^{2n}
- The time needed to break the scheme increases by a factor of 2^n and decreases by a factor of 4

Example 2

Consider a scheme where:

- $\mathsf{Enc}_k(m)$ runs in $n^2 \cdot |m|$ steps
- Breaking the scheme requires 2^n steps

What happens when computers get four times faster?

- Alice and Bob can decide to increase the security parameter from n to $2n$
- The number of steps of $\mathsf{Enc}_k(m)$ becomes $(2n)^2 \cdot |m| = 4n^2 \cdot |m|$, and the actual time spent stays the same
- The number of steps required to break the scheme becomes 2^{2n}
- The time needed to break the scheme increases by a factor of 2^n and decreases by a factor of 4
- Overall, the attack became $2^n/4 = 2^{n-2}$ times slower.

Example 2

Consider a scheme where:

- $\mathsf{Enc}_k(m)$ runs in $n^2 \cdot |m|$ steps
- Breaking the scheme requires 2^n steps

What happens when computers get four times faster?

- Alice and Bob can decide to increase the security parameter from n to $2n$
- The number of steps of $\mathsf{Enc}_k(m)$ becomes $(2n)^2$ ... tual time spent stays the same
- The number of ste ... eme becomes 2^{2n}
- The time needed to break the scheme increases by a factor of 2^n and decreases by a factor of 4
- Overall, the attack became $2^n/4 = 2^{n-2}$ times slower.

A increase in computing power resulted in a more difficult attack!

Example 3

Consider an adversary $\mathcal{A}$ that:

- Runs for n^3 minutes
- Breaks the scheme with probability $\min\{2^{40} \cdot 2^{-n}, 1\}$

Example 3

Consider an adversary $\mathcal{A}$ that:

- Runs for n^3 minutes
- Breaks the scheme with probability $\min\{2^{40} \cdot 2^{-n}, 1\}$

How large do we need to choose n?

n	48	64	128	256	512	1024
running time	2.5 months	6 months	4 years	32 years	255 years	2041 years
probability of success	1 in 256	$\approx$ 1 in 17 mil	$\approx$ 3 in 10^{26}	$\approx$ 3 in 10^{65}	$\approx$ 1 in 10^{142}	$\approx$ 2 in 10^{296}

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- We allow secrecy to fail with some tiny probability ← probabilities that are *negligible* in n
- We only restrict our attention to "efficient" attackers ← polynomial running times

Are **both** relaxations needed?

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- ~~We allow secrecy to fail with some tiny probability~~ ⟵ ~~probabilities that are *negligible* in n~~
- We only restrict our attention to "efficient" attackers ⟵ polynomial running times

Are **both** relaxations needed?

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- ~~We allow secrecy to fail with some tiny probability~~ $\longleftarrow$ probabilities that are *negligible* in n
- We only restrict our attention to "efficient" attackers $\longleftarrow$ polynomial running times

Are **both** relaxations needed?

- The discussion in the previous lecture shows that, as soon as we use short keys, there is an adversary that runs in polynomial-time and has some tiny advantage $\frac{\epsilon}{4}$

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- We allow secrecy to fail with some tiny probability $\longleftarrow$ probabilities that are *negligible* in n
- ~~We only restrict our attention to "efficient" attackers~~ $\longleftarrow$ ~~polynomial running times~~

Are **both** relaxations needed?

- The discussion in the previous lecture shows that, as soon as we use short keys, there is an adversary that runs in polynomial-time and has some tiny advantage $\frac{\epsilon}{4}$

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- We allow secrecy to fail with some tiny probability $\longleftarrow$ probabilities that are *negligible* in n
- ~~We only restrict our attention to "efficient" attackers~~ $\longleftarrow$ ~~polynomial running times~~

Are **both** relaxations needed?

- The discussion in the previous lecture shows that, as soon as we use short keys, there is an adversary that runs in polynomial-time and has some tiny advantage $\frac{\epsilon}{4}$
- We can always run a brute-force attack on the scheme. The discussion in the previous lecture shows that a computationally unbounded adversary has advantage at least $\frac{1}{8}$ for some pair of messages (when keys are at least one bit shorter than messages)

Computational secrecy (asymptotic)

We relax perfect secrecy in two ways:

- We allow secrecy to fail with some tiny probability ← probabilities that are *negligible* in n
- ~~We only restrict our attention to "efficient" attackers~~ ← ~~polynomial running times~~

Are **both** relaxations needed?

- The discussion in the previous lecture shows that, as soon as we use short keys, there is an adversary that runs in polynomial-time and has some tiny advantage $\frac{\epsilon}{4}$
- We can always run a brute-force attack on the scheme. The discussion in the previous lecture shows that a computationally unbounded adversary has advantage at least $\boxed{\frac{1}{8}}$ for some pair of messages (when keys are at least one bit shorter than messages)

Not negligible!

Leaking the length of the message

In general, encryption does not hide the plaintext length

- This is captured in the indistinguishably experiment by requiring $|m_0| = |m_1|$

Leaking the length of the message

In general, encryption does not hide the plaintext length

- This is captured in the indistinguishably experiment by requiring $|m_0| = |m_1|$

One should still be aware that leaking the plaintext length is. . .

- Inconsequential if the plaintext length is already public or is not sensitive
- Problematic in other cases!

Leaking the length of the message

In general, encryption does not hide the plaintext length

- This is captured in the indistinguishably experiment by requiring $|m_0| = |m_1|$

One should still be aware that leaking the plaintext length is...

- Inconsequential if the plaintext length is already public or is not sensitive
- Problematic in other cases!
  - Revealing the length of a $\texttt{yes}/\texttt{no}$ answer reveals the answer

Leaking the length of the message

In general, encryption does not hide the plaintext length

- This is captured in the indistinguishably experiment by requiring $|m_0| = |m_1|$

One should still be aware that leaking the plaintext length is...

- Inconsequential if the plaintext length is already public or is not sensitive
- Problematic in other cases!
  - Revealing the length of a `yes`/`no` answer reveals the answer
  - Revealing the number of (possibly binary) digits of a number can leak, e.g., the range of a salary

Leaking the length of the message

In general, encryption does not hide the plaintext length

- This is captured in the indistinguishably experiment by requiring $|m_0| = |m_1|$

One should still be aware that leaking the plaintext length is. . .

- Inconsequential if the plaintext length is already public or is not sensitive
- Problematic in other cases!
  - Revealing the length of a `yes`/`no` answer reveals the answer
  - Revealing the number of (possibly binary) digits of a number can leak, e.g., the range of a salary
  - Revealing the number of results of a search query leaks information on the popularity of the keyword

Leaking the length of the message

In general, encryption does not hide the plaintext length

- This is captured in the indistinguishably experiment by requiring $|m_0| = |m_1|$

One should still be aware that leaking the plaintext length is...

- Inconsequential if the plaintext length is already public or is not sensitive
- Problematic in other cases!
  - Revealing the length of a `yes`/`no` answer reveals the answer
  - Revealing the number of (possibly binary) digits of a number can leak, e.g., the range of a salary
  - Revealing the number of results of a search query leaks information on the popularity of the keyword
  - If the plaintext is compressed then encrypted, the ciphertext length leaks information about the amount of redundancy (entropy) of the plaintext

Leaking the length of the message

In general, encryption does not hide the plaintext length

- This is captured in the indistinguishably experiment by requiring $|m_0| = |m_1|$

One should still be aware that leaking the plaintext length is. . .

- Inconsequential if the plaintext length is already public or is not sensitive
- Problematic in other cases!
  - Revealing the length of a `yes`/`no` answer reveals the answer
  - Revealing the number of (possibly binary) digits of a number can leak, e.g., the range of a salary
  - Revealing the number of results of a search query leaks information on the popularity of the keyword
  - If the plaintext is compressed then encrypted, the ciphertext length leaks information about the amount of redundancy (entropy) of the plaintext

In Google maps, the map tiles are compressed and (essentially) static. The size of the ciphertext can be used to determine the viewed location

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- . . . but it requires long keys

- This is inevitable if we insist on perfect secrecy (recall that, in a perfectly secret scheme, $|\mathcal{K}| \geq |\mathcal{M}|$)

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- . . . but it requires long keys
- This is inevitable if we insist on perfect secrecy (recall that, in a perfectly secret scheme, $|\mathcal{K}| \geq |\mathcal{M}|$)

We have a security definition that allows for short keys and works against adversaries with polynomially bounded running times

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- . . . but it requires long keys
- This is inevitable if we insist on perfect secrecy (recall that, in a perfectly secret scheme, $|\mathcal{K}| \geq |\mathcal{M}|$)

We have a security definition that allows for short keys and works against adversaries with polynomially bounded running times

Is there a secure private-key encryption scheme (with short keys) according to this new definition?

Where do we stand?

- We have a perfectly secret encryption scheme (one-time pad)...
- ...but it requires long keys

- This is inevitable if we insist on perfect secrecy (recall that, in a perfectly secret scheme, $|\mathcal{K}| \geq |\mathcal{M}|$)

We have a security definition that allows for short keys and works against adversaries with polynomially bounded running times

Is there a secure private-key encryption scheme (with short keys) according to this new definition?

It depends...

Pseudorandom Number Generators

If pseudorandom number generators (PRGs) exist, then the answer is “yes”

Pseudorandom Number Generators

If pseudorandom number generators (PRGs) exist, then the answer is "yes"

- We don't know if PRGs exist

Pseudorandom Number Generators

If pseudorandom number generators (PRGs) exist, then the answer is “yes”

- We don’t know if PRGs exist

- If PRGs exist then P $\neq$ NP

Pseudorandom Number Generators

If pseudorandom number generators (PRGs) exist, then the answer is "yes"

- We don't know if PRGs exist
- If PRGs exist then $P \neq NP$

It is widely believed that $P \neq NP$, although this would not imply that PRGs exist...

Pseudorandom Number Generators

If pseudorandom number generators (PRGs) exist, then the answer is "yes"

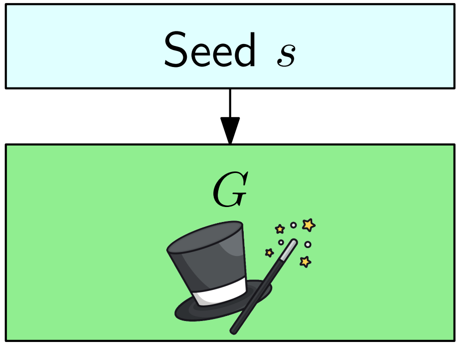
- We don't know if PRGs exist
- If PRGs exist then $\mathrm{P} \neq \mathrm{NP}$

It is widely believed that $\mathrm{P} \neq \mathrm{NP}$, although this would not imply that PRGs exist...

Pragmatic approach: assume that PRGs exist (and hope for the best) 🤞

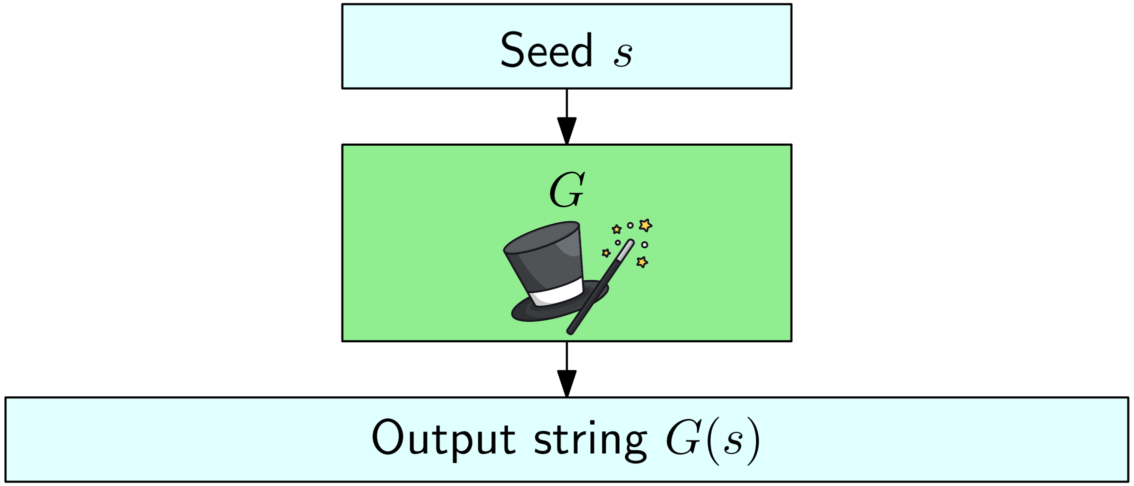
Pseudorandom Number Generators (informal)

A pseudorandom number generator is a deterministic polynomial-time algorithm that takes a binary string s (**seed**) chosen uniformly at random...



Pseudorandom Number Generators (informal)

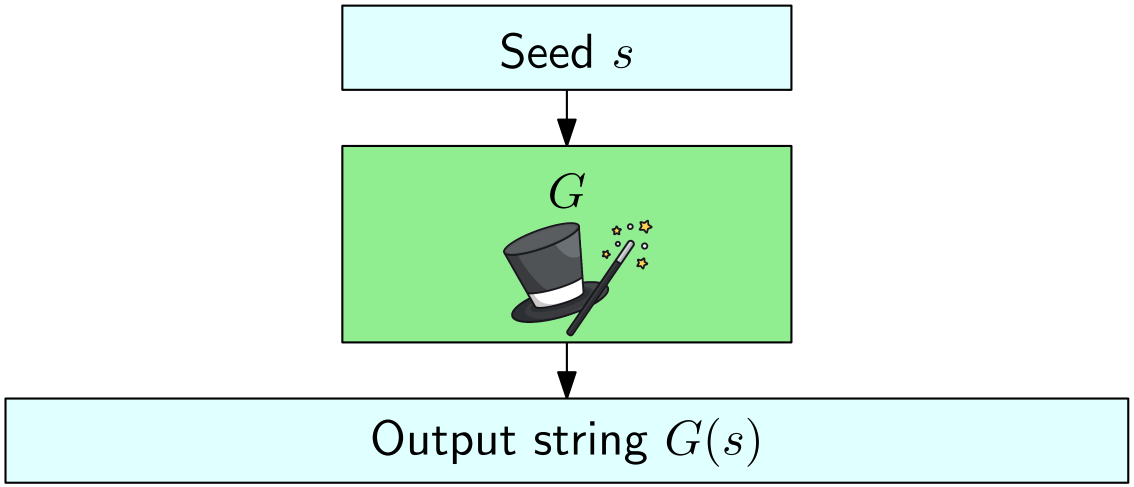
A pseudorandom number generator is a deterministic polynomial-time algorithm that takes a binary string s (**seed**) chosen uniformly at random...



And outputs a pseudorandom string $G(s)$ such that $|G(s)| > |s|$

Pseudorandom Number Generators (informal)

A pseudorandom number generator is a deterministic polynomial-time algorithm that takes a binary string s (**seed**) chosen uniformly at random...

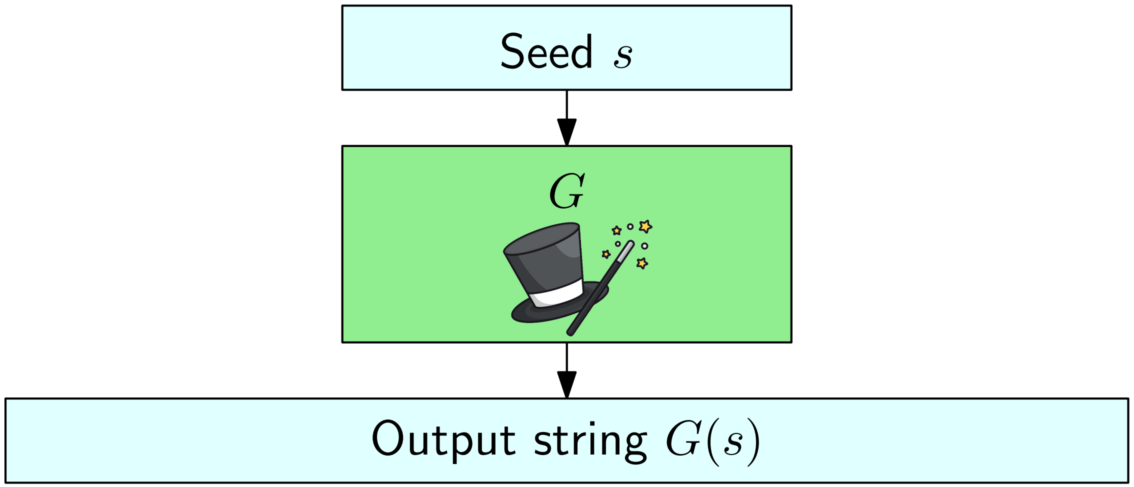


And outputs a pseudorandom string $G(s)$ such that $|G(s)| > |s|$

Intuition: G transforms a small number of "true random bits" into many "random looking" bits

Pseudorandom Number Generators (informal)

A pseudorandom number generator is a deterministic polynomial-time algorithm that takes a binary string s (**seed**) chosen uniformly at random. . .



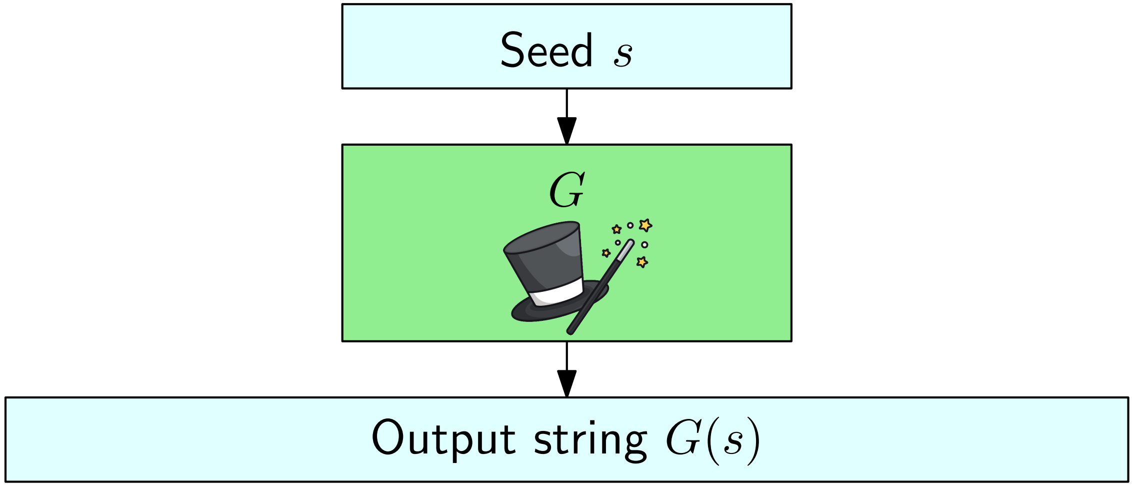
And outputs a pseudorandom string $G(s)$ such that $|G(s)| > |s|$

Intuition: G transforms a small number of "true random bits" into many "random looking" bits

Intuition: The output of a PRG should "look random". How do we formalize this?

Pseudorandom Number Generators (informal)

A pseudorandom number generator is a deterministic polynomial-time algorithm that takes a binary string s (**seed**) chosen uniformly at random...



And outputs a pseudorandom string $G(s)$ such that $|G(s)| > |s|$

Intuition: G transforms a small number of "true random bits" into many "random looking" bits

Intuition: The output of a PRG should "look random". How do we formalize this?

Randomness

Which of the following binary strings is random?

Which of the following binary strings is uniform?

0101010101010101 1001011011101001 0000000011111111

1001001001001001 0000000000000000

Randomness

Which of the following binary strings is random?

Which of the following binary strings is uniform?

0101010101010101 1001011011101001 0000000011111111

1001001001001001 0000000000000000

These questions are meaningless...

Randomness

Which of the following binary strings is random?

Which of the following binary strings is uniform?

0101010101010101 1001011011101001 0000000011111111

1001001001001001 0000000000000000

These questions are meaningless...

- Randomness is captured by probability distributions
- Uniformity is a property of distributions (not binary strings)
- The uniform distribution over a set X assigns probability $\frac{1}{|X|}$ to every element in X

Randomness

Which of the following binary strings is random?

Which of the following binary strings is uniform?

0101010101010101 1001011011101001 0000000011111111

1001001001001001 0000000000000000

These questions are meaningless...

- Randomness is captured by probability distributions
- Uniformity is a property of distributions (not binary strings)
- The uniform distribution over a set X assigns probability $\frac{1}{|X|}$ to every element in X

Informally, we sometimes say that x is "random / uniform" to mean that it was sampled from a random/uniform distribution. . .

Randomness

Which of the following binary strings is random?

Which of the following binary strings is uniform?

0101010101010101 1001011011101001 0000000011111111

1001001001001001 0000000000000000

These questions are meaningless...

- Randomness is captured by probability distributions
- Uniformity is a property of distributions (not binary strings)
- The uniform distribution over a set X assigns probability $\frac{1}{|X|}$ to every element in X

Informally, we sometimes say that x is "random / uniform" to mean that it was sampled from a random/uniform distribution...

...and that x is "pseudorandom" if it is the output of a PRG

Pseudorandomness

Historically, a candidate PRG was considered good if its outputs were able to pass a collection of statistical tests (that would be satisfied by “truly random” strings)

Pseudorandomness

Historically, a candidate PRG was considered good if its outputs were able to pass a collection of statistical tests (that would be satisfied by "truly random" strings)

Examples:

- Is the first bit of the output 1 with probability $\approx \frac{1}{2}$?

Pseudorandomness

Historically, a candidate PRG was considered good if its outputs were able to pass a collection of statistical tests (that would be satisfied by "truly random" strings)

Examples:

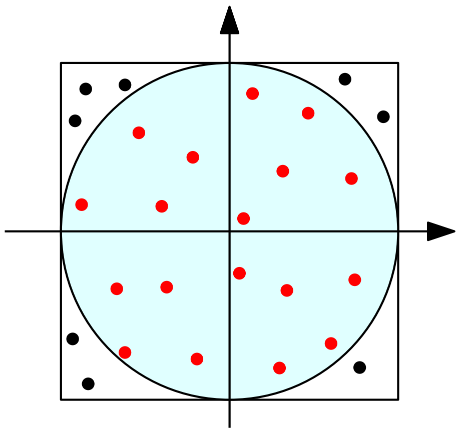
- Is the first bit of the output 1 with probability $\approx \frac{1}{2}$?
- Is the parity of any subset of bits 1 with probability $\approx \frac{1}{2}$?

Pseudorandomness

Historically, a candidate PRG was considered good if its outputs were able to pass a collection of statistical tests (that would be satisfied by "truly random" strings)

Examples:

- Is the first bit of the output 1 with probability $\approx \frac{1}{2}$?
- Is the parity of any subset of bits 1 with probability $\approx \frac{1}{2}$?
- If I interpret the string as a series of points in a square of side 2 centered in the origin, is the fraction of points within the circle of radius 1 centered in the origin $\approx \pi/4$?

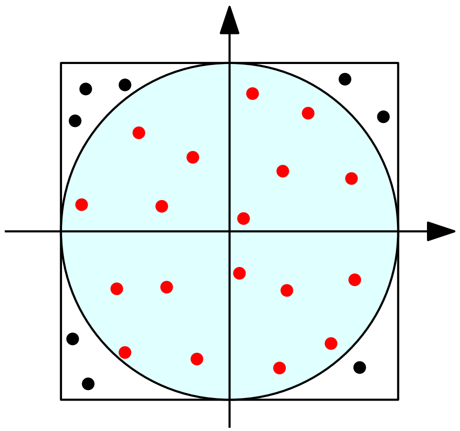


Pseudorandomness

Historically, a candidate PRG was considered good if its outputs were able to pass a collection of statistical tests (that would be satisfied by "truly random" strings)

Examples:

- Is the first bit of the output 1 with probability $\approx \frac{1}{2}$?
- Is the parity of any subset of bits 1 with probability $\approx \frac{1}{2}$?
- If I interpret the string as a series of points in a square of side 2 centered in the origin, is the fraction of points within the circle of radius 1 centered in the origin $\approx \pi/4$?



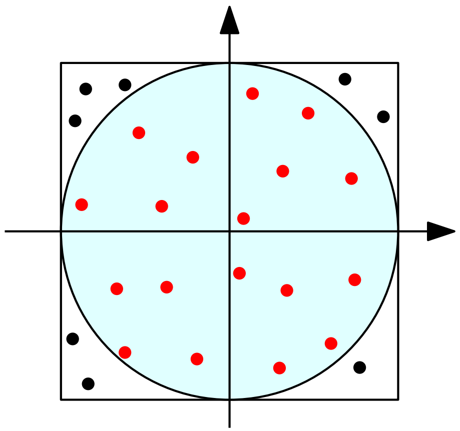
What if somebody comes up with a new, clever statistical test we did not think of before?

Pseudorandomness

Historically, a candidate PRG was considered good if its outputs were able to pass a collection of statistical tests (that would be satisfied by "truly random" strings)

Examples:

- Is the first bit of the output 1 with probability $\approx \frac{1}{2}$?
- Is the parity of any subset of bits 1 with probability $\approx \frac{1}{2}$?
- If I interpret the string as a series of points in a square of side 2 centered in the origin, is the fraction of points within the circle of radius 1 centered in the origin $\approx \pi/4$?



What if somebody comes up with a new, clever statistical test we did not think of before?

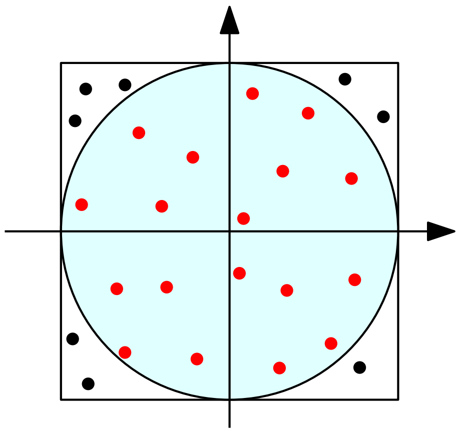
We would like a PRG to pass **all conceivable** statistical tests!

Pseudorandomness

Historically, a candidate PRG was considered good if its outputs were able to pass a collection of statistical tests (that would be satisfied by "truly random" strings)

Examples:

- Is the first bit of the output 1 with probability $\approx \frac{1}{2}$?
- Is the parity of any subset of bits 1 with probability $\approx \frac{1}{2}$?
- If I interpret the string as a series of points in a square of side 2 centered in the origin, is the fraction of points within the circle of radius 1 centered in the origin $\approx \pi/4$?



What if somebody comes up with a new, clever statistical test we did not think of before?

We would like a PRG to pass **all conceivable** statistical tests!

Is this even possible?

Pseudorandomness

- Let $n = |s|$ and consider a PRG that outputs ℓ bits. (recall that $\ell > n$)

Pseudorandomness

- Let $n = |s|$ and consider a PRG that outputs ℓ bits. (recall that $\ell > n$)

- Since G is deterministic, there are only 2^n possible inputs $x \implies$ at most 2^n possible outputs $G(s)$

Pseudorandomness

- Let $n = |s|$ and consider a PRG that outputs ℓ bits. (recall that $\ell > n$)

- Since G is deterministic, there are only 2^n possible inputs $x \implies$ at most 2^n possible outputs $G(s)$

- There are 2^ℓ binary strings with ℓ bits

$$2^\ell = 2^{\ell - n} \cdot 2^n \geq 2 \cdot 2^n$$

Pseudorandomness

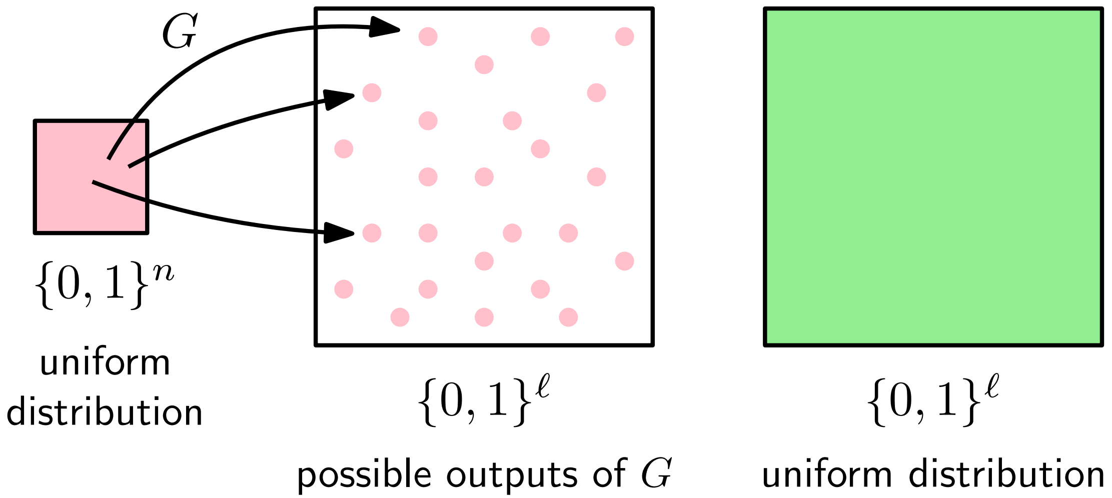
- Let $n = |s|$ and consider a PRG that outputs ℓ bits. (recall that $\ell > n$)

- Since G is deterministic, there are only 2^n possible inputs $x \implies$ at most 2^n possible outputs $G(s)$

- There are 2^ℓ binary strings with ℓ bits

$$2^\ell = 2^{\ell-n} \cdot 2^n \geq 2 \cdot 2^n$$

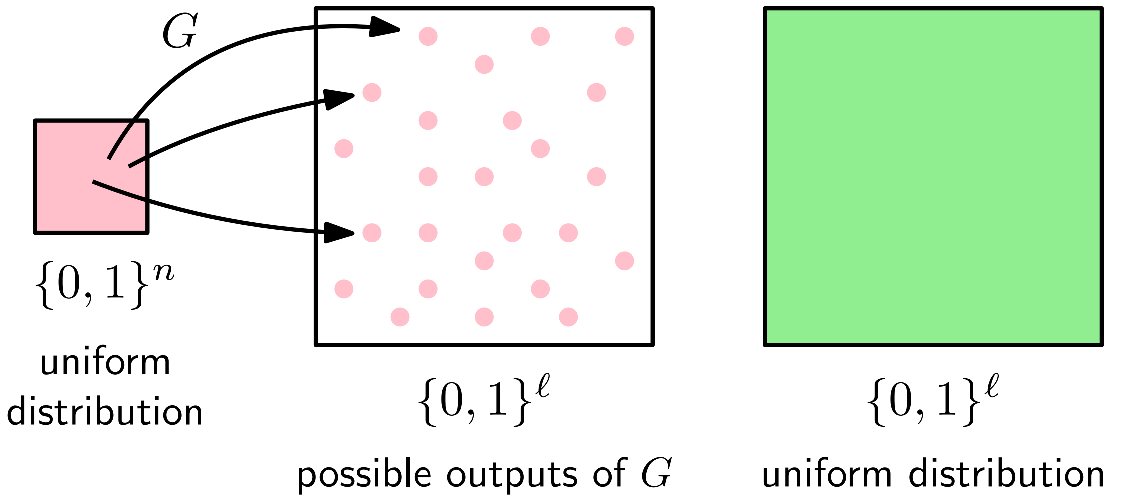
- At least half of the ℓ-bit strings (actually a $\frac{2^{\ell-n}-1}{2^{\ell-n}}$-fraction) can never be output by G !



Pseudorandomness

G will never pass the following statistical test (for some n):

- Look at a "sufficiently many" output/random strings
- If there are more than 2^n distinct strings, the test is passed
- Otherwise, the test is failed

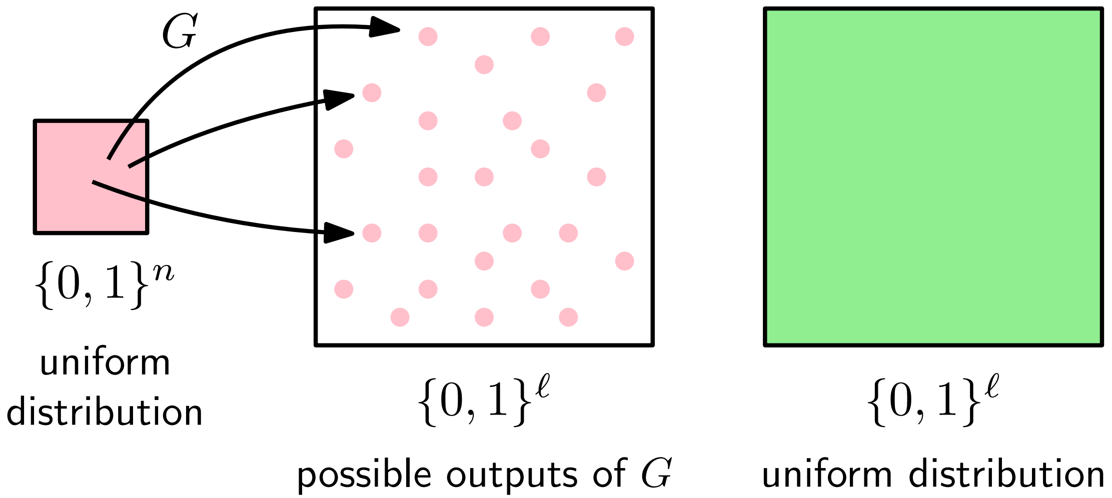


Pseudorandomness

G will never pass the following statistical test (for some n):

- Look at a "sufficiently many" output/random strings
- If there are more than 2^n distinct strings, the test is passed
- Otherwise, the test is failed

Observation: This is not an *efficient* test.



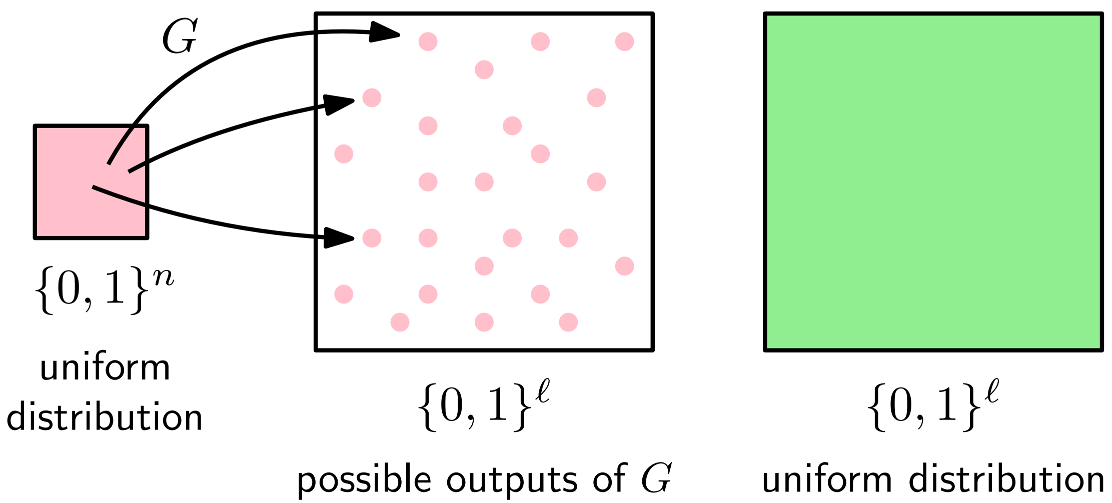
Pseudorandomness

G will never pass the following statistical test (for some n):

- Look at a "sufficiently many" output/random strings
- If there are more than 2^n distinct strings, the test is passed
- Otherwise, the test is failed

Observation: This is not an *efficient* test.

Idea: If adversaries are polynomially bounded, we only need to pass statistical tests that run in polynomial time



Pseudorandom Number Generators (formal)

Let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the output $G(s)$ is a string of length $\ell(n)$

Pseudorandom Number Generators (formal)

Let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the output $G(s)$ is a string of length $\ell(n)$

Expansion factor of G

Pseudorandom Number Generators (formal)

Let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the output $G(s)$ is a string of length $\ell(n)$

Expansion factor of G

G is a **pseudorandom generator (PRG)** if the following conditions hold:

Pseudorandom Number Generators (formal)

Let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the output $G(s)$ is a string of length $\ell(n)$

Expansion factor of G

G is a **pseudorandom generator (PRG)** if the following conditions hold:

- **Expansion:** For every $n \geq 1$, $\ell(n) > n$

Pseudorandom Number Generators (formal)

Let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the output $G(s)$ is a string of length $\ell(n)$

Expansion factor of G

G is a **pseudorandom generator (PRG)** if the following conditions hold:

- **Expansion:** For every $n \geq 1$, $\ell(n) > n$
- **Pseudorandomness:** For any probabilistic polynomial-time algorithm D, there is a negligible function η such that

$$\left| \Pr[D(\,G(s)\,) = 1] - \Pr[D(r) = 1] \right| \leq \eta(n)$$

where s is a uniform random variable in $\{0,1\}^n$ and r is a uniform random variable in $\{0,1\}^{\ell(n)}$

Pseudorandom Number Generators (formal)

Let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the output $G(s)$ is a string of length $\ell(n)$

Expansion factor of G

G is a **pseudorandom generator (PRG)** if the following conditions hold:

- **Expansion:** For every $n \geq 1$, $\ell(n) > n$
- **Pseudorandomness:** For any probabilistic polynomial-time algorithm D, there is a negligible function η such that

Probability over the randomness of D and the choice of s

$$\left| \Pr[D(\,G(s)\,) = 1] - \Pr[D(r) = 1] \right| \leq \eta(n)$$

where s is a uniform random variable in $\{0,1\}^n$ and r is a uniform random variable in $\{0,1\}^{\ell(n)}$

Pseudorandom Number Generators (formal)

Let G be a deterministic polynomial-time algorithm such that for any n and any input $s \in \{0,1\}^n$, the output $G(s)$ is a string of length $\ell(n)$

Expansion factor of G

G is a **pseudorandom generator (PRG)** if the following conditions hold:

- **Expansion:** For every $n \geq 1$, $\ell(n) > n$
- **Pseudorandomness:** For any probabilistic polynomial-time algorithm D, there is a negligible function η such that

Probability over the randomness of D and the choice of s

$$\Big| \Pr[D(\,G(s)\,) = 1] - \Pr[D(r) = 1] \Big| \leq \eta(n)$$

Probability over the randomness of D and the choice of r

where s is a uniform random variable in $\{0,1\}^n$ and r is a uniform random variable in $\{0,1\}^{\ell(n)}$

Examples

Consider a polynomial-time algorithm G that outputs $G(s) = \underbrace{000\ldots0}_{\ell(n)>|s|}$

Is it a PRG?

Examples

Consider a polynomial-time algorithm G that outputs $G(s) = \underbrace{000\ldots0}_{\ell(n) > |s|}$

Is it a PRG? Intuition: No, because the output does not "look random"

Formal proof?

Examples

Consider a polynomial-time algorithm G that outputs $G(s) = \underbrace{\texttt{000...0}}_{\ell(n)>|s|}$

Is it a PRG? Intuition: No, because the output does not "look random"

Formal proof?

We need to come up with a distinguisher $D(w)$ that guesses whether w comes from the output of $G(s)$ or it is chosen u.a.r. from $\{0,1\}^{\ell(n)}$

Examples

Consider a polynomial-time algorithm G that outputs $G(s) = \underbrace{\texttt{000...0}}_{\ell(n)>|s|}$

Is it a PRG? Intuition: No, because the output does not "look random"

Formal proof?

We need to come up with a distinguisher $D(w)$ that guesses whether w comes from the output of $G(s)$ or it is chosen u.a.r. from $\{0,1\}^{\ell(n)}$

Distinguisher $\mathcal{D}(w)$:

- If $w = \texttt{000...0}$:
  - Output 1 (guess that w "is pseudorandom")
- Otherwise output 0 (guess that w "is truly random")

Examples

Consider a polynomial-time algorithm G that outputs $G(s) = \underbrace{000\ldots0}_{\ell(n) > |s|}$

Is it a PRG? Intuition: No, because the output does not "look random"

Formal proof?

We need to come up with a distinguisher $D(w)$ that guesses whether w comes from the output of $G(s)$ or it is chosen u.a.r. from $\{0,1\}^{\ell(n)}$

Distinguisher $\mathcal{D}(w)$:

- If $w = 000\ldots0$:
  - Output 1 (guess that w "is pseudorandom")
- Otherwise output 0 (guess that w "is truly random")

- $\Pr[D(\,G(s)\,) = 1] = 1$
- $\Pr[D(r) = 1] = \frac{1}{2^{\ell(n)}}$

Examples

Consider a polynomial-time algorithm G that outputs $G(s) = \underbrace{\texttt{000\ldots0}}_{\ell(n) > |s|}$

Is it a PRG?

Intuition: No, because the output does not "look random"

Formal proof?

We need to come up with a distinguisher $D(w)$ that guesses whether w comes from the output of $G(s)$ or it is chosen u.a.r. from $\{0,1\}^{\ell(n)}$

Distinguisher $\mathcal{D}(w)$:

- If $w = \texttt{000\ldots0}$:
  - Output 1 (guess that w "is pseudorandom")
- Otherwise output 0 (guess that w "is truly random")

- $\Pr[D(\,G(s)\,) = 1] = 1$
- $\Pr[D(r) = 1] = \frac{1}{2^{\ell(n)}}$

$\left|1 - \frac{1}{2^{\ell(n)}}\right|$ is not negligible

Why are PRGs useful?

As far as polynomial-time algorithms are concerned, the output of $G(s)$ with a random seed s is indistinguishable (up to some negligible probability) from a random string r

Seed $s \in \{0,1\}^n$

$\downarrow$

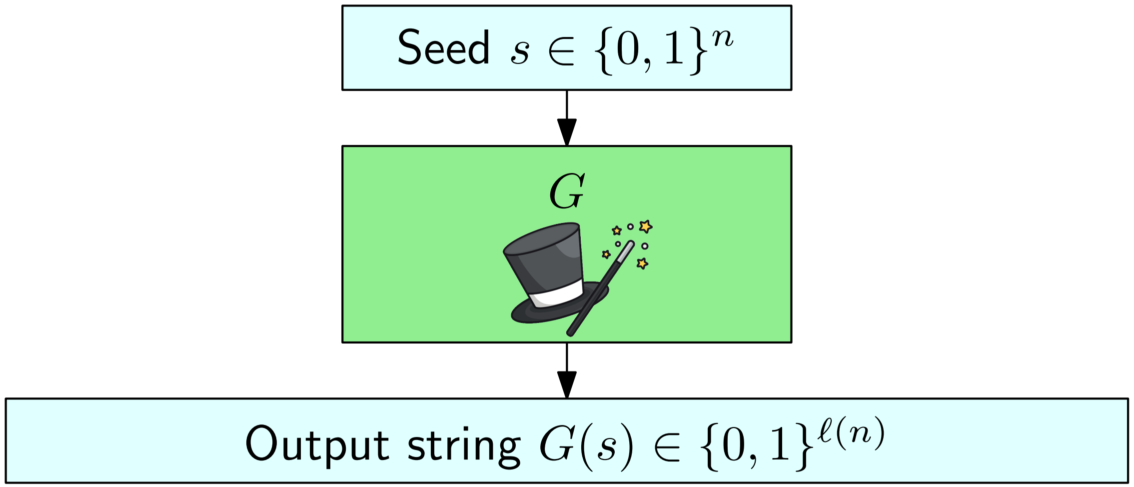
G

$\downarrow$

Output string $G(s) \in \{0,1\}^{\ell(n)}$

Why are PRGs useful?

As far as polynomial-time algorithms are concerned, the output of $G(s)$ with a random seed s is indistinguishable (up to some negligible probability) from a random string r



If we have a randomized polynomial-time algorithm that uses $\ell(n)$ random bits, and we replace those random bits with the output of $G(s)$, the resulting (randomized) algorithm "behaves the same" except for a negligible probability