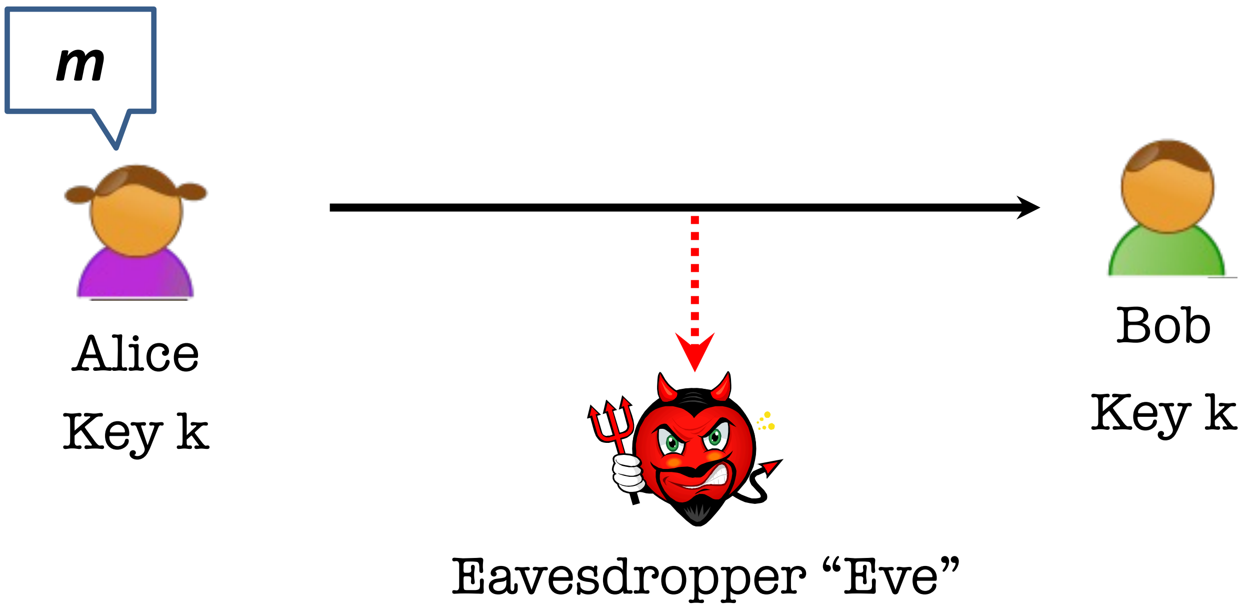
MIT 6.875/6.5620/18.425

Foundations of Cryptography
Lecture 2

Course website: ***https://mit6875.github.io/***

Lecture 1 Recap

Secure Communication



- Alice and Bob have a common key k

- Algorithms (Gen, Enc, Dec).
- Correctness: $Dec\big(k, Enc(k,m)\big) = m$.
- Security: Perfect Secrecy = Perfect Indistinguishability.

How to Define Security

Perfect secrecy: A Posteriori = A Priori

For all m, c: $\Pr[\mathcal{M} = m \mid E(\mathcal{K}, \mathcal{M}) = c] = \Pr[\mathcal{M} = m]$

Perfect indistinguishability:

For all m_0, m_1, c: $\Pr[E(\mathcal{K}, m_0) = c] = \Pr[E(\mathcal{K}, m_1) = c]$

The two definitions are equivalent!

Is there a perfectly secure scheme?

- **One-time Pad**: $E(k, m) = k \oplus m$
- **However**: Keys are as long as Messages
- **WORSE, Shannon's theorem**:
  for **any** perfectly secure scheme, |key|≥|message|.

Can we overcome Shannon's conundrum?

Perfect Indistinguishability: a Turing test

For all m_0, m_1, c: $\Pr[E(\mathcal{K}, m_0) = c] = \Pr[E(\mathcal{K}, m_1) = c]$

World 0:

k ← $\mathcal{K}$

$c = E(k, m_0)$

World 1:

k ← $\mathcal{K}$

$c = E(k, m_1)$

 is a **distinguisher**.

For all EVE and all m_0, m_1: $\Pr[EVE(c) = 0 \mid k \leftarrow \mathcal{K}; c = E(k, m_0)]$
$= \Pr[EVE(c) = 0 \mid k \leftarrow \mathcal{K}; c = E(k, m_1)]$

Perfect Indistinguishability: a Turing test

For all m_0, m_1, c: $\Pr[E(\mathcal{K}, m_0) = c] = \Pr[E(\mathcal{K}, m_1) = c]$

World 0:

k $\leftarrow \mathcal{K}$

$c = E(k, m_0)$

World 1:

k $\leftarrow \mathcal{K}$

$c = E(k, m_1)$

 is a **distinguisher**.

For all EVE and all m_0, m_1: $\Pr[k \leftarrow \mathcal{K}; c = E(k, m_0): EVE(c) = 0]$
$= \Pr[k \leftarrow \mathcal{K}; c = E(k, m_1): EVE(c) = 0]$

Perfect Indistinguishability: a Turing test

For all m_0, m_1, c: $\Pr[E(\mathcal{K}, m_0) = c] = \Pr[E(\mathcal{K}, m_1) = c]$

World 0:

k $\leftarrow \mathcal{K}$

$c = E(k, m_0)$

World 1:

k $\leftarrow \mathcal{K}$

$c = E(k, m_1)$

 is a **distinguisher**.

For all EVE and all m_0, m_1:

$$\Pr[k \leftarrow \mathcal{K}; b \leftarrow \{0,1\};\ c = E(k, m_b):\ EVE(c) = b] = 1/2$$

The Key Idea:
Computationally Bounded Adversaries

The Axiom of ~~Modern Crypto~~ Life

Feasible Computation = Probabilistic polynomial-time*

(**p.p.t.** = Probabilistic polynomial-time)

(polynomial in a security parameter n)

So, Alice and Bob are **fixed** p.p.t. algorithms.

(e.g., run in time n^2)

Eve is **any** p.p.t. algorithm.

(e.g., run in time n^4, or n^100, or n^10000,…)

* in recent years, quantum polynomial-time

Computational Indistinguishability (take 1)

World 0:

k ← $\mathcal{K}$

$c = E(k, m_0)$

World 1:

k ← $\mathcal{K}$

$c = E(k, m_1)$

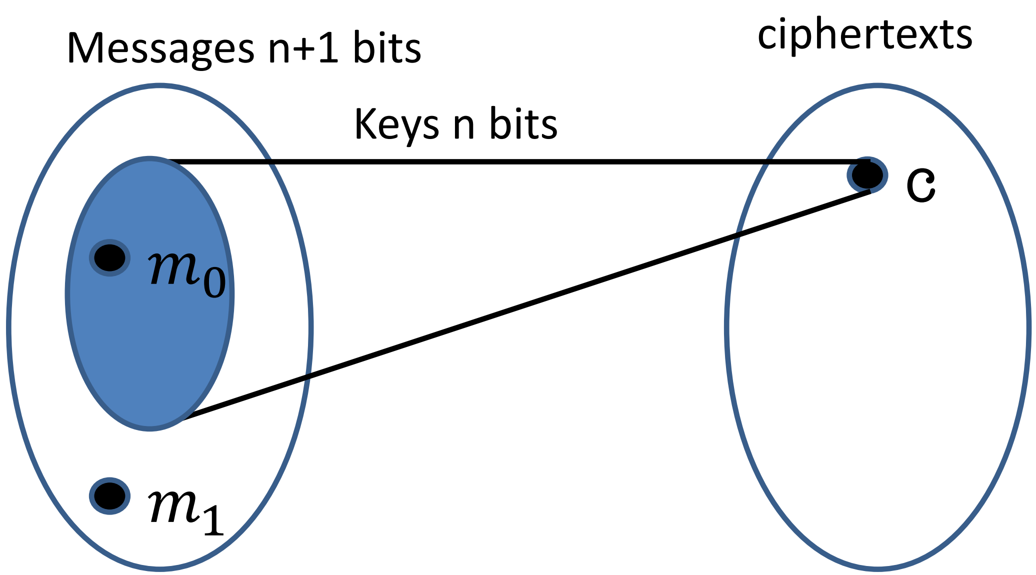
is a **p.p.t.** distinguisher.

For all **p.p.t.** EVE and all m_0, m_1:

$$\Pr[k \leftarrow \mathcal{K}; b \leftarrow \{0,1\}; c = E(k, m_b): EVE(c) = b] = 1/2$$

Still subject to Shannon's impossibility!

Still subject to Shannon's impossibility!



Set of messages consistent with c = {D(k,c): all k}

Consider Eve that picks a random key k and

outputs 0 if D(k,c) = m_0 **w.p $\geq 1/2^n$**

outputs 1 if D(k,c) = m_1 **w.p = 0**

and a random bit if neither holds.

Bottomline: Pr[EVE succeeds] $\geq$ 1/2 + $1/2^n$

New Notion: Negligible Functions

Functions that grow slower than 1/p(n) for any polynomial p.

Definition: A function $\mu: \mathbb{N} \rightarrow \mathbb{R}$ is **negligible** if
for every polynomial function p,
there exists an n_0 s.t.
for all $n > n_0$:

μ**(n) < 1/p(n)**

Key property: Events that occur with negligible probability look ***to poly-time algorithms*** like they ***never*** occur.

New Notion: Negligible Functions

Functions that grow slower than 1/p(n) for any polynomial p.

Definition: A function $\mu: \mathbb{N} \rightarrow \mathbb{R}$ is **negligible** if
for every polynomial function p,
there exists an n_0 s.t.
for all $n > n_0$:

μ**(n) < 1/p(n)**

Question: Let $\mu(n) = 1/n^{\log n}$. Is μ negligible?

New Notion: Negligible Functions

Functions that grow slower than 1/p(n) for any polynomial p.

Definition: A function $\mu: \mathbb{N} \rightarrow \mathbb{R}$ is **negligible** if

for every polynomial function p,

there exists an n_0 s.t.

for all $n > n_0$:

μ(n) < 1/p(n)

Question: Let $\mu(n) = 1/n^{100}$ if n is prime and $\mu(n) = 1/2^n$ otherwise. Is μ negligible?

New Notion: Negligible Functions

Functions that grow slower than 1/p(n) for any polynomial p.

Definition: A function $\mu: \mathbb{N} \rightarrow \mathbb{R}$ is **negligible** if
for every polynomial function p,
there exists an n_0 s.t.
for all $n > n_0$:

μ(n) < 1/p(n)

Question (PS1) Let $\mu(n)$ be a negligible function and q(n) a polynomial function. Is $\mu(n)q(n)$ a negligible function?

Security Parameter: n *(sometimes λ)*

Definition: A function $\mu: \mathbb{N} \rightarrow \mathbb{R}$ is **negligible** if
for every polynomial function p,
there exists an n_0 s.t.
for all $n > n_0$:

μ(n) < 1/p(n)

- Runtimes & success probabilities are measured as a function of n.
- ***Want***: Honest parties run in time (fixed) polynomial in n.
- ***Allow***: Adversaries to run in time (arbitrary) polynomial in n,
- ***Require***: adversaries to have success probability negligible in n.

Computational Indistinguishability (take 2)

World 0:

k ← $\mathcal{K}$

$c = E(k, m_0)$

World 1:

k ← $\mathcal{K}$

$c = E(k, m_1)$

is a distinguisher.

For all **p.p.t.** EVE, **there is a negligible function μ**

s.t. for all m_0, m_1:

$$\Pr[k \leftarrow K; b \leftarrow \{0,1\};\ c = E(k, m_b):\ EVE(c) = b] \leq \frac{1}{2} + \mu(n)$$

Our First Crypto Tool: Pseudorandom Generators (PRG)

Pseudo-random Generators

Informally: **Deterministic** Programs that stretch a "truly random" seed into a (much) longer sequence of **"seemingly random"** bits.

How to define "seemingly random"?

Can such a G exist?

How to **Define** a Strong Pseudo Random Number Generator?

Def 1 [Indistinguishability]

"No polynomial-time algorithm can distinguish b[illegible]tween the output of a PRG on a random seed vs. a truly rando[illegible]g"

= "as good as" a truly random string for [illegible]tical purposes.

Def 2 [Next-bit Unpredictab[illegible]

"No polynomial-time alg[illegible] can predict the (i+1)th bit of the output of a PRG give[illegible]rst i bits, better than chance"

Def 3 [Incon[illegible]ssibility]

"No polynomial-time algorithm can compress the output of the PRG into a shorter string"

ALL THREE DEFS EQUIVALENT!

PRG Def 1: Indistinguishability

Definition [Indistinguishability]:

A deterministic polynomial-time computable function $G: \{0,1\}^n \rightarrow \{0,1\}^m$ is a PRG if:

(a) It is expanding: $m > n$ and

(b) for every PPT algorithm D (called a distinguisher or a statistical test) if there is a negligible function μ such that:

$$| \Pr[\, \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{Un})) = \mathbf{1} \,] - \Pr[\, \boldsymbol{D}(\boldsymbol{U_m}) = \mathbf{1} \,] | = \boldsymbol{\mu}(\boldsymbol{n})$$

Notation: U_n (resp. U_m) denotes the random distribution on n-bit (resp. m-bit) strings; m is shorthand for m(n).

PRG Def 1: Indistinguishability

WORLD 1:
The Pseudorandom World

$y \leftarrow G(U_n)$

WORLD 2:
The Truly Random World

$y \leftarrow U_m$

PPT Distinguisher gets y but cannot tell which world she is in

Why is this a good definition

Good for all Applications:

As long as we can find truly random seeds, can replace true randomness by the output of PRG(seed) in ANY (polynomial-time) application.

If the application behaves differently, then it constitutes a (polynomial-time) statistical test between PRG(seed) and a truly random string.

PRG $\Longrightarrow$ Overcoming Shannon's Conundrum

(or, How to Encrypt n+1 bits using an n-bit key)

$Gen(1^n)$: Generate a random n-bit key k.

$Enc(k, m)$ where m is an $(\mathrm{n}+1)$**-bit** message:

Expand k into a (n+1)-bit pseudorandom string $\mathrm{k}' = G(\mathrm{k})$

One-time pad with k': ciphertext is $k' \oplus m$

$Dec(k, c)$ outputs $\mathrm{G}(k) \oplus c$

Correctness:

$Dec(k, c)$ outputs $\mathrm{G}(k) \oplus c = \mathrm{G}(k) \oplus G(k) \oplus \mathrm{m} = \mathrm{m}$

PRG $\Longrightarrow$ Overcoming Shannon's Conundrum

Security: **your first reduction!**

Suppose for contradiction that there is a p.p.t. EVE, a polynomial function p and m_0, m_1 $s.t.$

$$\Pr[k \leftarrow \mathcal{K}; b \leftarrow \{0,1\};\ c = E(k, m_b):\ EVE(c) = b] \geq \frac{1}{2} + 1/p(n)$$

PRG $\Longrightarrow$ Overcoming Shannon's Conundrum

Security: your first reduction!

Suppose for contradiction that there is a p.p.t. EVE, a polynomial function p and m_0, m_1 $s.t.$

$$\rho = \Pr[k \leftarrow \{0,1\}^n\ ; b \leftarrow \{0,1\};\ c = G(k) \oplus m_b:\ EVE(c) = b] \geq \frac{1}{2} + 1/p(n)$$

$$\text{Let } \rho' = \Pr[k' \leftarrow \{0,1\}^{n+1}\ ; b \leftarrow \{0,1\};\ c = k' \oplus m_b:\ EVE(c) = b] = \frac{1}{2}$$

This will give us a distinguisher EVE' for G, contradicting the assumption that G is a pseudorandom generator. QED.

PRG $\Longrightarrow$ Overcoming Shannon's Conundrum

Distinguisher EVE' for G.

Get as input a string y, run EVE(y$\oplus m_b$) for a random b, and let EVE's output be b'. Output "PRG" if b=b' and "RANDOM" otherwise.

$$\Pr[EVE' \, outputs \text{ "}PRG\text{" } | \, y \; is \; pseudorandom]$$
$$= \rho \geq \frac{1}{2} + 1/p(n)$$

$$\Pr[EVE' \, outputs \text{ "}PRG\text{" } | \, y \; is \; random] = \rho' = \frac{1}{2}$$

Therefore, $\Pr[EVE' \, outputs \text{ "}PRG\text{" } | \, y \; is \; pseudorandom] -$
$$\Pr[EVE' \, outputs \text{ "}PRG\text{" } | \, y \; is \; random]$$
$$\geq 1/p(n)$$

$\blacksquare$

PRG $\Longrightarrow$ Overcoming Shannon's Conundrum

(or, How to Encrypt n+1 bits using an n-bit key)

$\boldsymbol{Q1}$: Do PRGs exist?

(Exercise: If P=NP, PRGs do not exist.)

$\boldsymbol{Q2}$: How do we encrypt longer messages or many messages with a fixed key?

(**Length extension**: If there is a PRG that stretches by one bit, there is one that stretches by polynomially many bits)

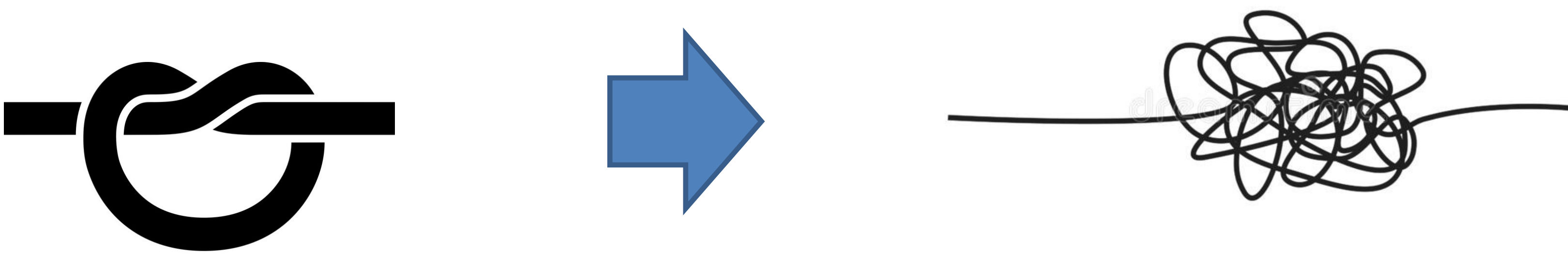
(**Pseudorandom functions**: PRGs with exponentially large stretch and "random access" to the output.)

$\boldsymbol{Q1}$: Do PRGs exist?

Constructing PRGs: Two Methodologies

The Practical Methodology

1. Start from a design framework
(e.g. “appropriately chosen functions composed appropriately many times look random”)

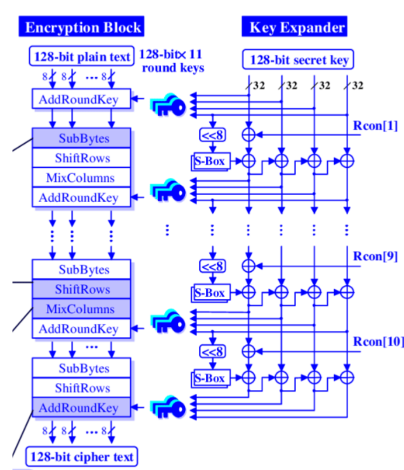


Constructing PRGs: Two Methodologies

The Practical Methodology

1. Start from a design framework
(e.g. “appropriately chosen functions composed appropriately many times look random”)

2. Come up with a candidate construction



Rijndael
(now the Advanced Encryption Standard)

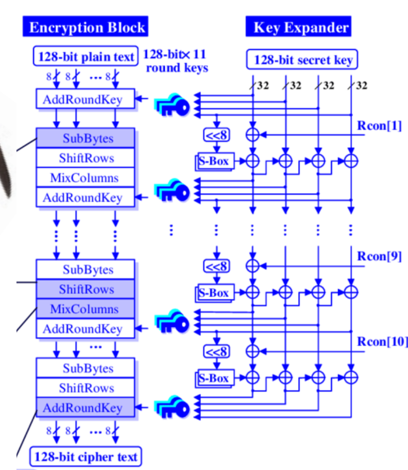
Constructing PRGs: Two Methodologies

<u>The Practical Methodology</u>

1. Start from a design framework
(e.g. "appropriately chosen functions composed appropriately many times look random")

2. Come up with a candidate construction

3. Do extensive cryptanalysis.

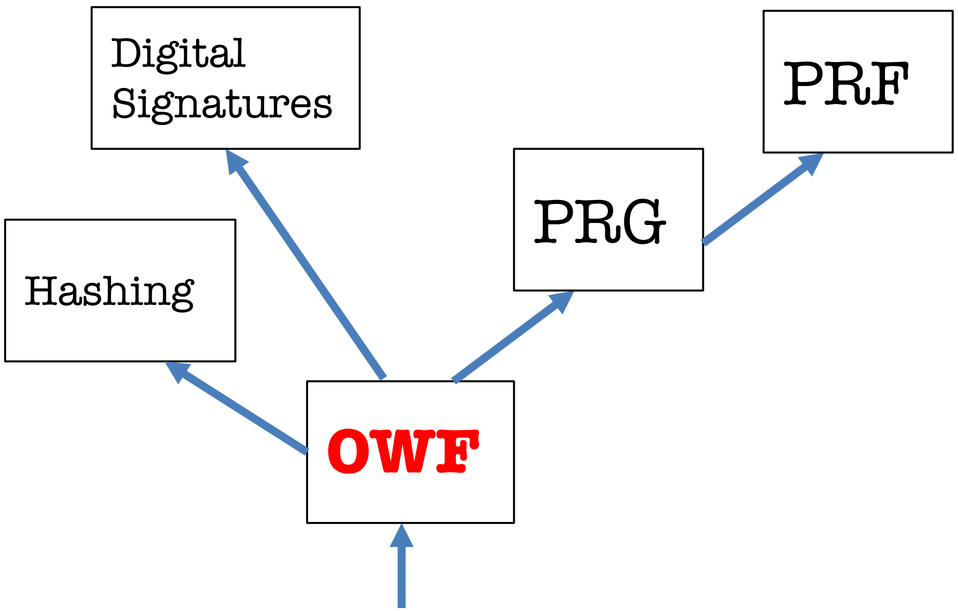


Constructing PRGs: Two Methodologies

The Foundational Methodology (much of this course)

Reduce to simpler primitives.

"Science wins either way" –Silvio Micali



well-studied, average-case hard, problems

Constructing PRGs: Two Methodologies

The Foundational Methodology (much of this course)

A PRG Candidate from the average-case hardness of Subset-sum:

G($a_1, \ldots, a_n, x_1, \ldots, x_n$) = ($a_1, \ldots, a_n, \sum_{i=1}^{n} x_i a_i$ mod 2^{n+1})

where a_i are random (n+1)-bit numbers, and x_i are random bits.

Beautiful Function:

If G is a one-way function, then G is a PRG.

If lattice problems are hard on the worst-case, G is a PRG.

Pseudorandom Generators and (T)CS

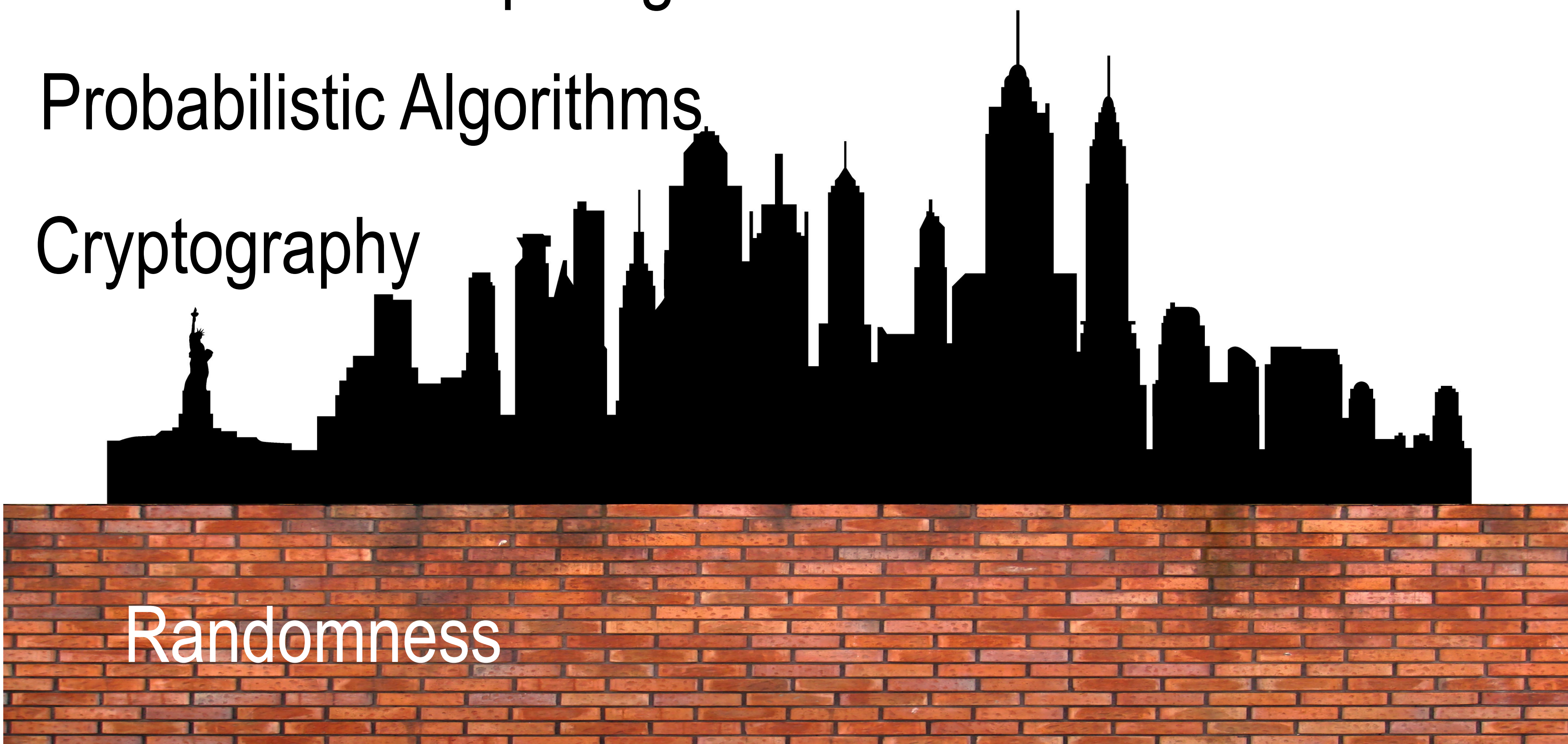
Randomness is a Fundamental Resource

Simulation/Sampling/MCMC

Distributed Computing

Probabilistic Algorithms

Cryptography



Where do we get random bits from?

1) Specialized Hardware: e.g., Transistor noise.

2) User Input: Every time random number used, user is queried.

~~3) Quantumness (not for much of this class)~~

Usually biased, but can "extract" unbiased bits assuming the source has "some structure and enough entropy"

[randomness extraction: von Neumann,…]

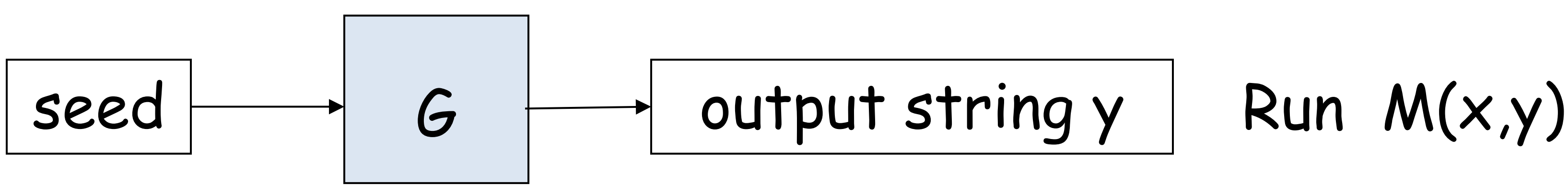
BUT: True randomness is an expensive commodity.

Application of PRGs: De-randomization

- Recall: L $\in$ **BPP** implies $\exists$ poly-time algorithm M

$$x \in L \Longrightarrow \Pr_{coins\ y}[M(x,y) \text{ accepts}] > 2/3$$
$$x \notin L \Longrightarrow \Pr_{coins\ y}[M(x,y) \text{ accepts}] < 1/3$$

- Use a **PRG** to generate the *m* random bits y:



Application of PRGs: De-randomization

Theorem: if PRGs exist, then $BPP \subseteq \cap_{\varepsilon>0} TIME\left(2^{m^{\varepsilon}}\right)$.

(in English) if PRGs exist, then every randomized poly-time algorithm can be simulated in **deterministic** sub-exponential time.

Proof Sketch: use PRG that expands from $n = m^{\varepsilon}$ bits to m bits.

$$x \in L \Longrightarrow \Pr_{seed\ y}[M(x, G(y))\ \text{accepts}] > \frac{2}{3} - \mu(n)$$

$$x \notin L \Longrightarrow \Pr_{seed\ y}[M(x, G(y))\ \text{accepts}] < \frac{1}{3} + \mu(n)$$

Why? If the above is not true, M is a distinguisher for the PRG!

Note: M is a (known, fixed, fixed poly-time) distinguisher.

Application of PRGs: De-randomization

Theorem: if PRGs exist, then $BPP \subseteq \cap_{\varepsilon>0} TIME\left(2^{m^{\varepsilon}}\right)$.

(in English) if PRGs exist, then every randomized poly-time algorithm can be simulated in **deterministic** sub-exponential time.

Proof Sketch: use PRG that expands from $n = m^{\varepsilon}$ bits to m bits.

$x \in L \Longrightarrow \#seed\ y$: $M(x, G(y))$ accepts $> 0.65 * 2^{n} = 0.65 * 2^{m^{\varepsilon}}$

$x \notin L \Longrightarrow \#seed\ y$: $M(x, G(y))$ accepts $< 0.35 * 2^{n} = 0.35 * 2^{m^{\varepsilon}}$

Here is the deterministic algorithm: enumerate over all seeds y and run $M\big(x, G(y)\big)$. If #accepts $> 0.65 * 2^{m^{\varepsilon}}$, accept else reject.

Application of PRGs: De-randomization

Theorem: if "exponentially secure" PRGs exist, then $BPP = P$.

Proof Sketch:

Use a PRG that expands from $n = O(\log m)$ bits to m bits that are indistinguishable not just by $\mathrm{poly}(n)$-time algorithms but also by $2^{c_1 n} = m^{c_2}$- time algorithms.

The previous proof goes through *mutatis mutandis*, using crucially the fact that the randomized algorithm (adversary for us) runs in fixed polynomial-time.

Next Lecture:

$\boldsymbol{Q2}$: How do we encrypt longer messages or many messages with a fixed key?

1. PRG length extension,
2. Pseudorandom functions (PRF) and PRG $\Longrightarrow$ PRF