MATH/CMSC 456 :: UPDATED COURSE INFO

Instructor: Gorjan Alagic (galagic@umd.edu); ATL 3102, office hours: by appointment

Textbook: *Introduction to Modern Cryptography*, Katz and Lindell;

Webpage: alagic.org/cmsc-456-cryptography-spring-2020/ (check for updates);

Piazza: piazza.com/umd/spring2020/cmsc456

ELMS: active, slides posted there, assignments will be as well.

Gradescope: active, access through ELMS.

Check these setups asap, and let me know if you run into issues!

TAs (Our spot: shared open area across from IRB 5234)

- Elijah Grubb (egrubb@cs.umd.edu) 11am-12pm TuTh (Iribe);
- Justin Hontz (jhontz@terpmail.umd.edu) 1pm-2pm MW (Iribe);

Additional help:

- Chen Bai (cbai1@terpmail.umd.edu) 3:30-5:30pm Tu (2115 ATL, starting Feb 4)
- Bibhusa Rawal (bibhusa@terpmail.umd.edu) 3:30-5:30pm Th (2115 ATL, starting Feb 6)

RECAP: LOGISTICS

Course plan (big picture)

- 8 lectures: symmetric-key crypto
- 4 lectures: RSA and Diffie-Hellman (Carl Miller); 2 lectures : secret sharing (Bill Gasarch);
- midterm;
- 10 lectures: public-key crypto II, advanced topics;
- final.

Grading: 40% homework, 30% midterm exam, 30% final exam

Homework(~ 10 sets): collaboration allowed, must write up your own, no late homework whatsoever (but lowest grade will be dropped); first set distributed 2nd week (ELMS → Gradescope.)

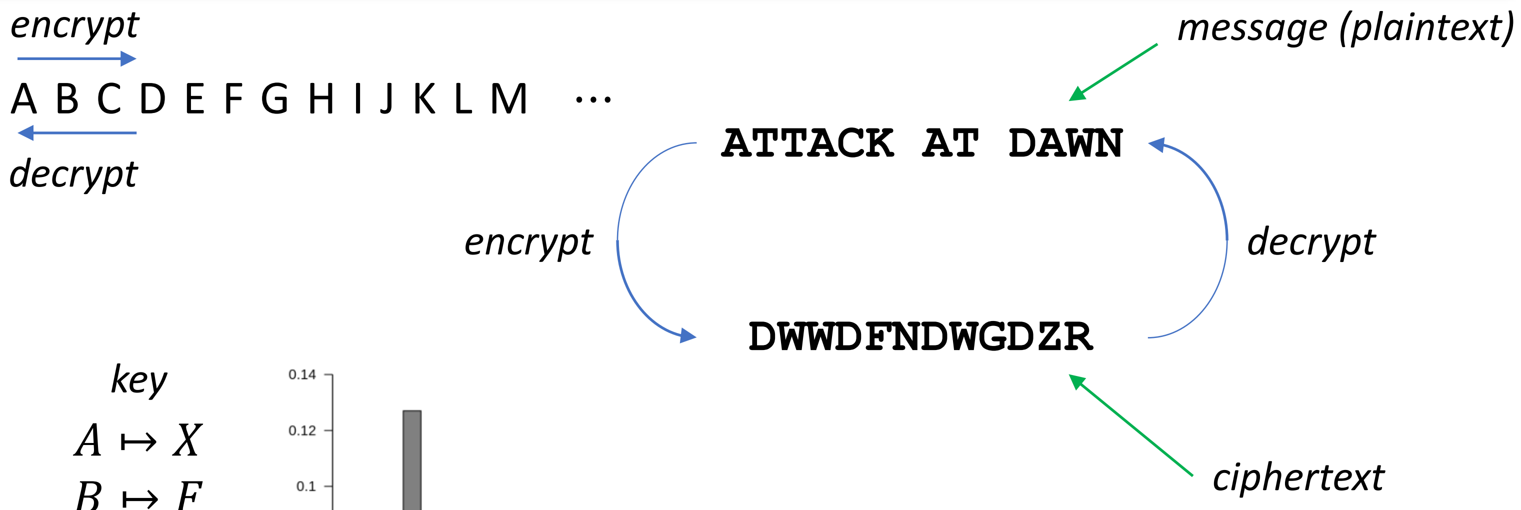
Exams:

- closed book/device, one two-sided page of notes;
- midterm March 31st ;
- final May 18th .

RECAP: HISTORICAL CIPHERS

Caesar cipher

- basic shift cipher;
- broken: brute-force keysearch.

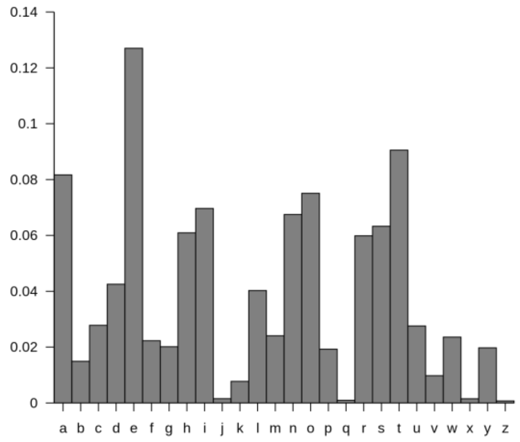


Substitution cipher

- permute alphabet instead of shifting;
- broken: frequency analysis.

key

$A \mapsto X$
$B \mapsto F$
$C \mapsto D$
$D \mapsto L$
$E \mapsto P$
…
…



Vigenére cipher

- "add" plaintext and repeated passphrase;
- broken: frequency analysis + brute-force key.

RECAP: MODERN CRYPTO

Why we do crypto this way?

- history was not kind to previous ciphers;
- from the 70s on: a much more rigorous approach;
- be as careful and formal as possible when describing the task, the setting, what it means to be "secure," the cryptosystem itself;
- when possible, try to establish security via rigorous reasoning (i.e., theorem-proving.)

The course is about: the above approach, in the theoretical setting:

"possible in principle … vs impossible, even in principle"

Some things we won't study:

- IT security
- real-world implementation details
- specific performance/security tradeoffs

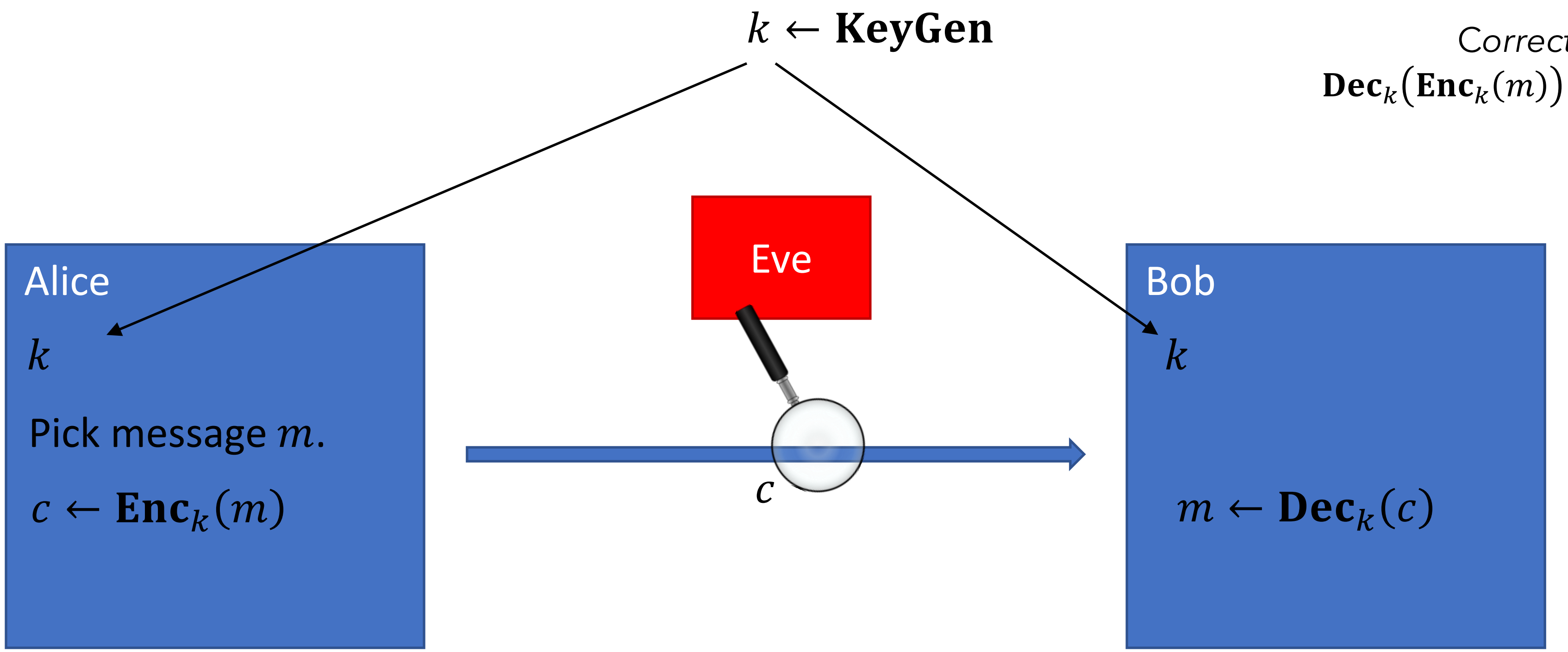
These are interesting things too, just not in scope.

RECAP: ENCRYPTION SCHEMES

Generic approach to encryption:

- generate key via some algorithm: $k \leftarrow \mathbf{KeyGen}$
- encrypt via some algorithm: $c \leftarrow \mathbf{Enc}_k(m)$
- decrypt via some algorithm: $m \leftarrow \mathbf{Dec}_k(c)$

The triple $(\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ is called an *encryption scheme*.



Correctness:
$\mathbf{Dec}_k(\mathbf{Enc}_k(m)) = m$ for all m.

RECAP: ENCRYPTION SCHEMES: ONE-TIME PAD

Examples: one-time pad (Vernam cipher, ~1882)

- *Key generation* : sample uniformly random $k \in \{0,1\}^n$
- *Encryption* : $\mathbf{Enc}_k(m) = m \oplus k$
- *Decryption* : $\mathbf{Dec}_k(c) = c \oplus k$;

(note 1: messages are interpreted as bitstrings.)

(note 2: key length = message length = ciphertext length = n.)

Bitwise XOR (+ mod 2):

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 1 = 0$$

Get very friendly and familiar with OTP: it will keep cropping up!

We proved that this is secure under one (and hence all) of our notions of **perfect secrecy.**

Basic proof idea:

- key is uniformly random;
- ciphertext is a "shift" of the key by some string (namely the plaintext);
- hence ciphertext is also uniformly random, for any plaintext;
- this fulfills one of the definitions of perfect secrecy.

Food for thought.
OTP key space is of size 2^n. If n is small (e.g., $2^8 = 256$), is brute-force key search possible?

RECAP: ENCRYPTION SCHEMES: SECRECY

Definition 1. (very informal) An encryption scheme is **semantically secret** if, for all choices of adversary $\boldsymbol{A}$, message m, "prior information" function g, and "target information" function f, the following property holds:

$$\Pr\left[f(m) \leftarrow \boldsymbol{A}\left(g(m), \mathbf{Enc}_k(m)\right)\right] = \Pr\left[f(m) \leftarrow \boldsymbol{A}\left(g(m)\right)\right].$$

Definition 2. An encryption scheme is **perfectly secret** if, for every plaintext distribution $\mathcal{M}$, every plaintext m, and every ciphertext c,

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

Definition 3. An encryption scheme is **perfectly secret** if, for every plaintext distribution $\mathcal{M}$, every plaintext pair m, m', and every ciphertext c,

$$\Pr_k[\mathbf{Enc}_k(m) = c] = \Pr_k[\mathbf{Enc}_k(m') = c]$$

Definition 4. An encryption scheme has **perfectly indistinguishable ciphertexts** if, for every adversary $\boldsymbol{A}$,

$$\Pr_k[\boldsymbol{A} \text{ wins the IND game}] = \frac{1}{2}.$$

Theorem 1. Definitions 1-4 are all equivalent.

II. (SIMPLE) ENCRYPTION (continued)

Reading: Ch.2 (p.25-40)

ONE-TIME PAD: THE SCHEME

One-time pad

- *Key generation* : sample uniformly random $k \in \{0,1\}^n$;
- *Encryption* : $\mathbf{Enc}_k(m) = m \oplus k$;
- *Decryption* : $\mathbf{Dec}_k(c) = c \oplus k$.

The OTP achieves perfect secrecy. Are there other schemes that do the job?

Shannon's Theorem. Let $(\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ be an encryption scheme that satisfies perfect secrecy. Let $\mathcal{M}, \mathcal{K}, \mathcal{C}$ denote the message, key, and ciphertext sets, respectively. Then $|\mathcal{K}| \geq |\mathcal{M}|$. Moreover, if $|\mathcal{K}| = |\mathcal{M}|$, then

1. $\mathbf{KeyGen}$ outputs a uniformly random key in $\mathcal{K}$, and
2. For every $m \in \mathcal{M}$ and every $c \in \mathcal{C}$, there exists a unique key $k \in \mathcal{K}$ such that $\mathbf{Enc}_k(m) = c$.

- what this means: basically only one way to build an encryption scheme that satisfies perfect secrecy;
- … and the one-time pad is it.

ONE-TIME PAD: IS IT REALLY "PERFECT"?

Recall: we had a bunch of assumptions.

- Alice and Bob can share a secret in advance;
- they have their own private spaces;
- Alice can send only one transmission, on a single channel;
- Eve (eavesdropper) can observe *everything* that is transmitted on that channel.
- *Eve cannot do anything else.*

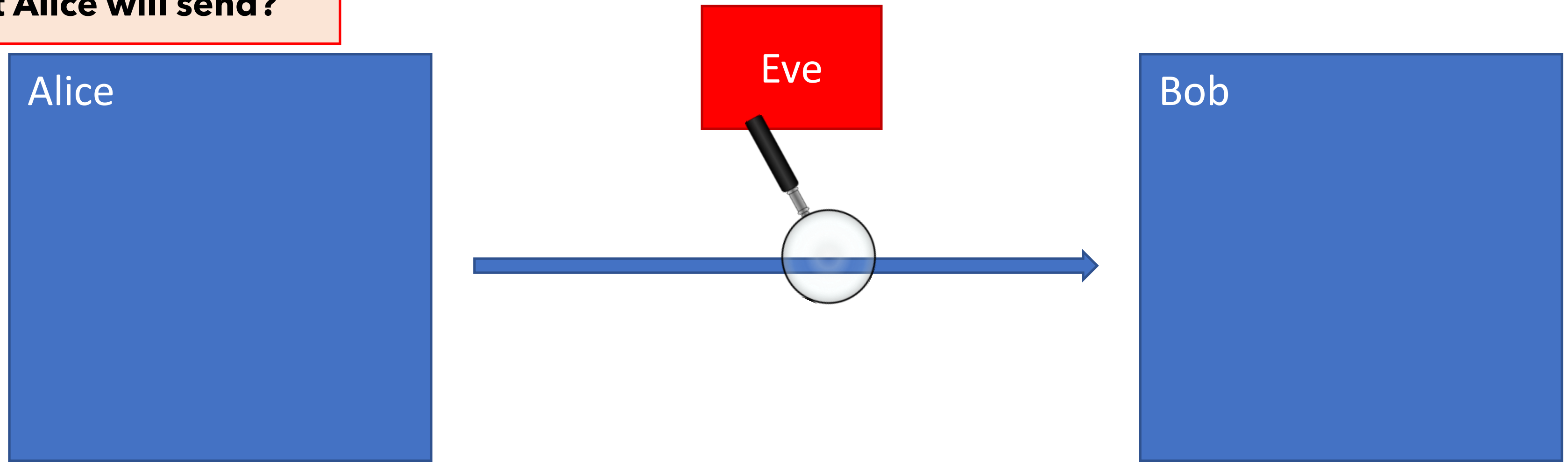
What if they can't?

What if Eve can look at Alice's screen?

What if they want to send multiple messages?

What if Eve can also change messages in transit?

What if Eve knows something about what Alice will send?



ONE-TIME PAD: IS IT REALLY "PERFECT"?

Consider: using OTP twice, i.e., to send $2n$ bits.

Shannon's theorem: for perfect secrecy, $|\mathcal{K}| = |\mathcal{M}|$. But here $|\mathcal{K}| = |\mathcal{M}|/2$. So not perfectly secret.

Some attack examples:

(1.) If Eve **can't** know any of the plaintexts:

- she observes two ciphertexts $c, c' \in \{0,1\}^n$;
- they were generated with same key: $c = m \oplus k$ and $c' = m' \oplus k$;
- bitwise, so $c_j = c_j{}'$ if and only if $m_j = m_j{}'$. Plaintext information is leaking! **SCHEME BROKEN**

(2.) If Eve **can** know one of the plaintexts:

- she is told m, and observes two ciphertexts $c, c' \in \{0,1\}^n$;
- now $c = m \oplus k$, so Eve computes $k = c \oplus m$;
- complete key recovery, and trivial to recover m'. **SCHEME BROKEN**

In fact, Shannon says you can't even use OTP to send $n+1$ bits securely!

Corollary. To encrypt a hard drive, you need another hard drive of equal size to store the decryption key.

ONE-TIME PAD: IS IT REALLY "PERFECT"?

Recall: we had a bunch of assumptions.

- Alice and Bob can share a secret in advance;
- they have their own private spaces;
- Alice can send only one transmission, on a single channel;
- Eve (eavesdropper) can observe *everything* that is transmitted on that channel.
- *Eve cannot do anything else.*

What if they can't?

What if Eve can look at Alice's screen?

What if they want to send multiple messages?

What if Eve can also change messages in transit?

What if Eve knows something about what Alice will send?

Later: we will see that the other relaxations are also a disaster for the OTP.

So what does this mean?

By Shannon's theorem, it means we **have to give up on something** in perfect secrecy.

III. COMPUTATIONALLY-SECURE ENCRYPTION

Reading: p.43-70

WHAT DO WE RELAX?

Shannon: if you want fancy features (like long messages) you have to give up something.

What can we give up?

Definition 4. An encryption scheme has **perfectly indistinguishable ciphertexts** if, for every adversary A,

$$\Pr_k[A \text{ wins the IND game}] = \frac{1}{2}\,.$$

- If the adversary can break our scheme, but it takes them 10 billion years, do we care?
- If the adversary can break our scheme, but only with probability 1 in 10^{100}, do we care?
- Probably not. Can we leverage that somehow? And get more out of crypto?

YES!

This "simple" change allows us to go from boring, almost useless crypto (OTP) …

… to amazing crypto whose limits we are still trying to understand!

COMPUTATIONAL CRYPTO: A PREVIEW

Let's postpone technical details for now.

What could this give us? Recall OTP:

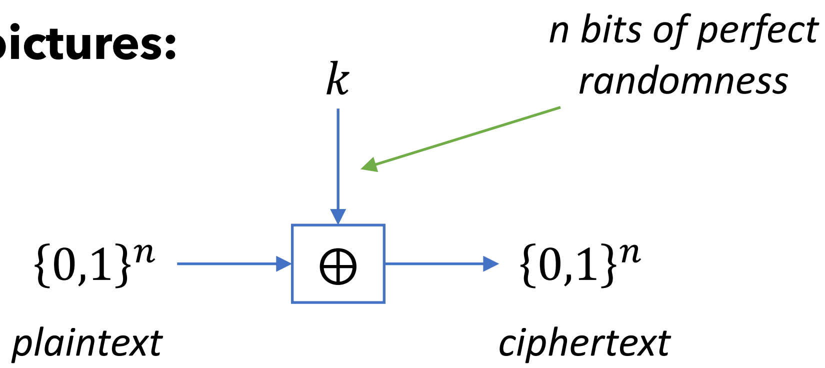
- *Key generation* : sample uniformly random $k \in \{0,1\}^n$;
- *Encryption* : $\mathbf{Enc}_k(m) = m \oplus k$;
- *Decryption* : $\mathbf{Dec}_k(c) = c \oplus k$.

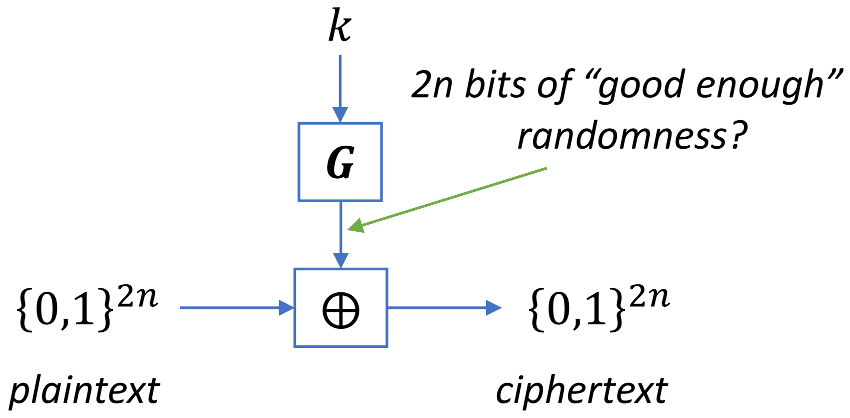
Remember from programming:
Random number generators: deterministic programs that turn a small *seed* into a much longer sequence of "random-looking" numbers. Suppose

$$\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{2n}$$

is such a generator.

In pictures:





Reasonable to hope: if no "feasible" algorithm can distinguish $\boldsymbol{G}(k)$ from random, then this scheme is secure against all "feasible" adversaries.

COMPUTATIONAL CRYPTO: A PREVIEW

Let's postpone technical details for now.

What could this give us? Recall OTP:

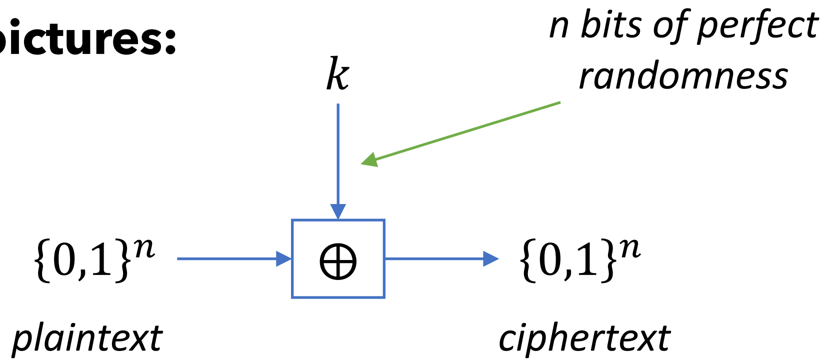
- *Key generation* : sample uniformly random $k \in \{0,1\}^n$;
- *Encryption* : $\mathbf{Enc}_k(m) = m \oplus k$;
- *Decryption* : $\mathbf{Dec}_k(c) = c \oplus k$.

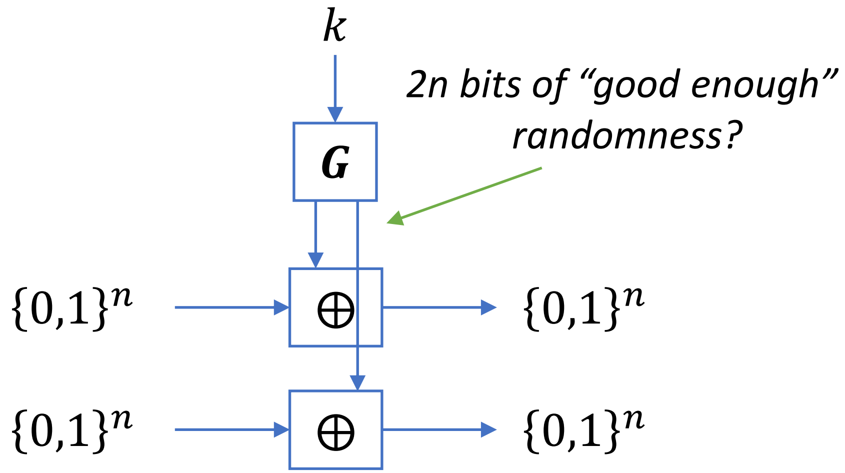
Remember from programming:
Random number generators: deterministic programs that turn a small *seed* into a much longer sequence of "random-looking" numbers. Suppose

$$\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{2n}$$

is such a generator.

In pictures:





Reasonable to hope: if no "feasible" algorithm can distinguish $\boldsymbol{G}(k)$ from random, then this scheme is secure against all "feasible" adversaries.

COMPUTATIONAL CRYPTO: CHALLENGES

Let's postpone technical details for now.

What could this give us? Recall OTP:

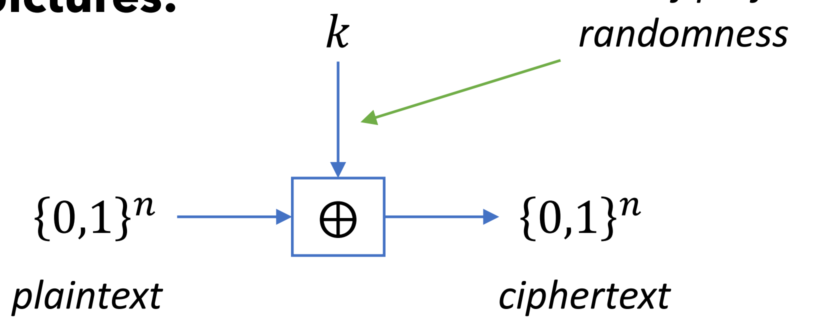
- *Key generation* : sample uniformly random $k \in \{0,1\}^n$;
- *Encryption* : $\mathbf{Enc}_k(m) = m \oplus k$;
- *Decryption* : $\mathbf{Dec}_k(c) = c \oplus k$.

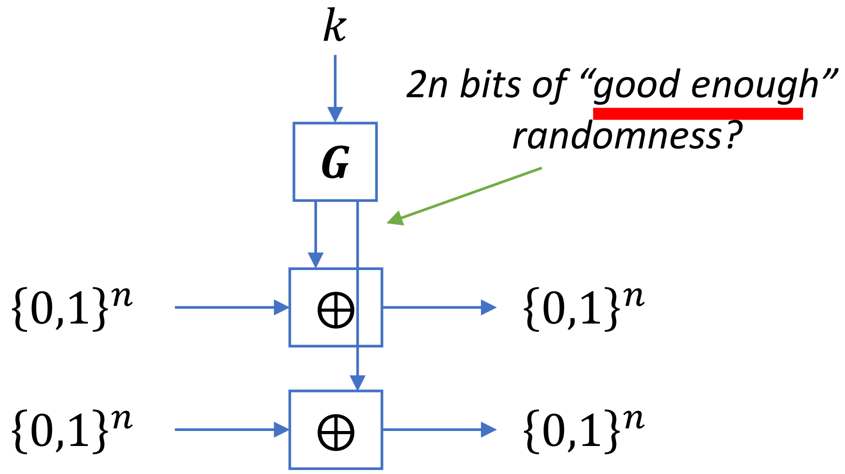
Remember from programming:
Random number generators: deterministic programs that turn a small *seed* into a much longer sequence of "random-looking" numbers.
Suppose

$$\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{2n}$$

is such a generator.

In pictures:





Reasonable to hope: if no "feasible" algorithm can distinguish $\boldsymbol{G}(k)$ from random, then this scheme is secure against all "feasible" adversaries.

Could we prove this?

COMPUTATIONAL CRYPTO: CHALLENGES

This intuition seems sound. How can we formalize it?

1. Notions to define:

- "random-looking"
- "good-enough" randomness
- "feasible" vs "infeasible" algorithms ←
- "secure" encryption (can't be same as perfect secrecy, we gave up on that.)

2. Stuff to construct:

- a function which produces "good enough" randomness against "feasible" algorithms

3. Theorems we have to prove:

- the construction in the previous slide is secure.

EFFICIENT vs INEFFICIENT ALGORITHMS

What should "feasible" (or efficient) mean?

- lots of natural choices, but…
- we are interested in "possible in principle" vs "not possible, even in principle;"
- we want the theory to be *simple* and *easy to work with*;
- in particular, we don't want to worry about details of the computational model.

To address all of these issues, we will take an approach similar to that of complexity theory.

EFFICIENT vs INEFFICIENT ALGORITHMS

What should "feasible" (or efficient) mean?

- running time measured *as a function of input size* (e.g., searching a list of size n takes time n; generating a list of all possible pairs takes time n^2.)
- work asymptotically: we care about the large-n limit, not what happens for, e.g., $n = 20$;
- randomness: all algorithms are assumed to have access to as many uniformly random coins as needed;
- **efficient** will mean that the running time is **polynomial** in the size of the input.

A bit more carefully:

Definition. An algorithm $\boldsymbol{A}$ is **efficient** if there exists a polynomial $p: \mathbb{N} \rightarrow \mathbb{N}$ and a positive integer N such that for all $n > N$ and all $x \in \{0,1\}^n$, the running time of $\boldsymbol{A}$ on input x is at most $p(n)$.

We will often use the shorthand **PPT** meaning Probabilistic, Polynomial-Time algorithm.

EFFICIENT vs INEFFICIENT ALGORITHMS

What about "infeasible"?

- just the negation of "feasible"!
- concretely: the running time is *larger than every polynomial*
  - i.e., bigger than n^{100} or even $\mathrm{n}^{10^{100}}$;
  - for example, exponential (e.g., 2^n) or more;
  - but not necessarily exponential: consider $2^{\sqrt{n}}$ or $n^{\log(n)}$;
  - we use the term **superpolynomial**.

What about success probability?

Similar approach: asymptotic, polynomial versus superpolynomial.

- efficient : success probability $1/p(n)$ for some polynomial p.
- inefficient : success probability smaller than $1/p(n)$ for all polynomials p.

also called **negligible** and written $\mathrm{negl}(n)$.

EFFICIENT vs INEFFICIENT ALGORITHMS

Recall: algorithms can often be repeated to amplify success probability;

Our notions are “stable” under this sort of amplification;

In particular:

- consider some random experiment (e.g., an adversary attacks some cryptosystem.)
- suppose some event $\boldsymbol{E}$ (e.g., system is broken) occurs with negligible probability;
- now repeat the experiment $p(n)$ times for **any** polynomial p;
- what is the probability that $\boldsymbol{E}$ occurs in **at least one** of the experiments?

Exercise: it’s still negligible.

COMPUTATIONAL CRYPTO: CHALLENGES

This intuition seems sound. How can we formalize it?

1. Notions to define:

- "random-looking"
- "good-enough" randomness
- "feasible" vs "infeasible" algorithms
- "secure" encryption (can't be same as perfect secrecy, we gave up on that.)

2. Stuff to construct:

- a function which produces "good enough" randomness against "feasible" algorithms

3. Theorems we have to prove:

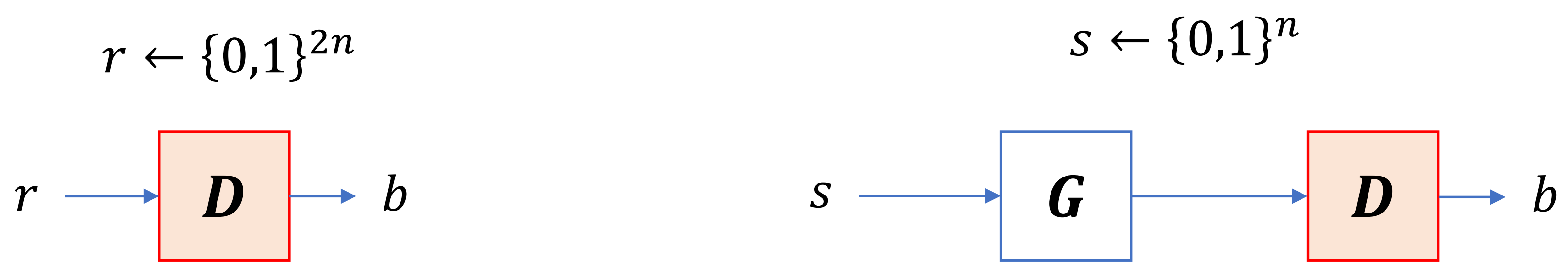
- the construction in the previous slide is secure.

Cryptographic pseudorandomness

We're not happy with garden-variety random number generators.

We need something much stronger. We need *indistinguishability from perfectly random*.

Let $\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{2n}$. Pick some algorithm $\boldsymbol{D}$. Consider these two experiments:



Crucial: $\boldsymbol{s}$ is sampled uniformly at random! (Otherwise, $\boldsymbol{G}(s)$ could simply be a fixed string!)

Want: *there is no efficient algorithm for* $\boldsymbol{D}$ that can distinguish these two experiments.

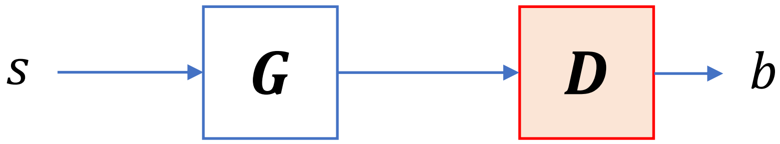
Cryptographic pseudorandomness

Definition. A **pseudorandom generator** is a deterministic, polynomial-time algorithm $\boldsymbol{G}$ satisfying the following:

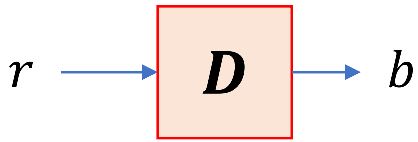
1. (expansion) $\boldsymbol{G}: \{0,1\}^n \rightarrow \{0,1\}^{\ell(n)}$ for some fixed polynomial ℓ satisfying $\ell(n) > n$ for all n.
2. (pseudorandomness) for every PPT algorithm $\boldsymbol{D}$,

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

$s \leftarrow \{0,1\}^n$



$r \leftarrow \{0,1\}^{\ell(n)}$



PSEUDORANDOM GENERATORS (PRGs)

How to break any PRG (in two easy steps).

Step 1: look up the PRG spec online;

Step 2: run the algorithm $\boldsymbol{D}$ below.

input: $r \in \{0,1\}^{\ell(n)}$

- try every possible $s \in \{0,1\}^n$;
- if you find one such that $\boldsymbol{G}(s) = r$, return $\mathbf{1}$.
- if not, return $\mathbf{0}$.

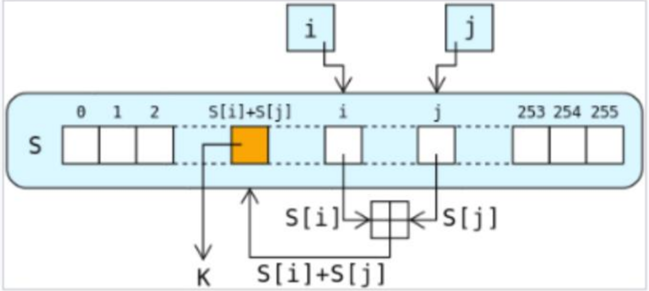
Pseudo-random generation algorithm (PRGA) [edit]

For as many iterations as are needed, the PRGA modifies the state and outpu

- increments i
- looks up the ith element of S, S[i], and adds that to j
- exchanges the values of S[i] and S[j] then uses the sum S[i] + S[j] value K below)
- then bitwise exclusive ORed (XORed) with the next byte of the message t

Each element of S is swapped with another element at least once every 256 i

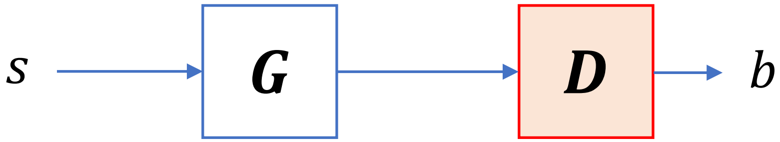
```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```



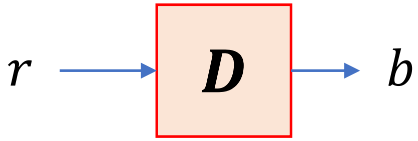
The lookup stage of RC4. The output byte is selected by looking up the values of S[i] and S[j], adding them together modulo 256, and then using the sum as an index into S; S(S[i] + S[j]) is used as a byte of the key stream, K.

$\boldsymbol{D}$ always outputs 1.

$s \leftarrow \{0,1\}^n$



$r \leftarrow \{0,1\}^{\ell(n)}$



$\boldsymbol{D}$ outputs 0 except with probability $2^{n-\ell(n)}$.

PSEUDORANDOM GENERATORS (PRGs)

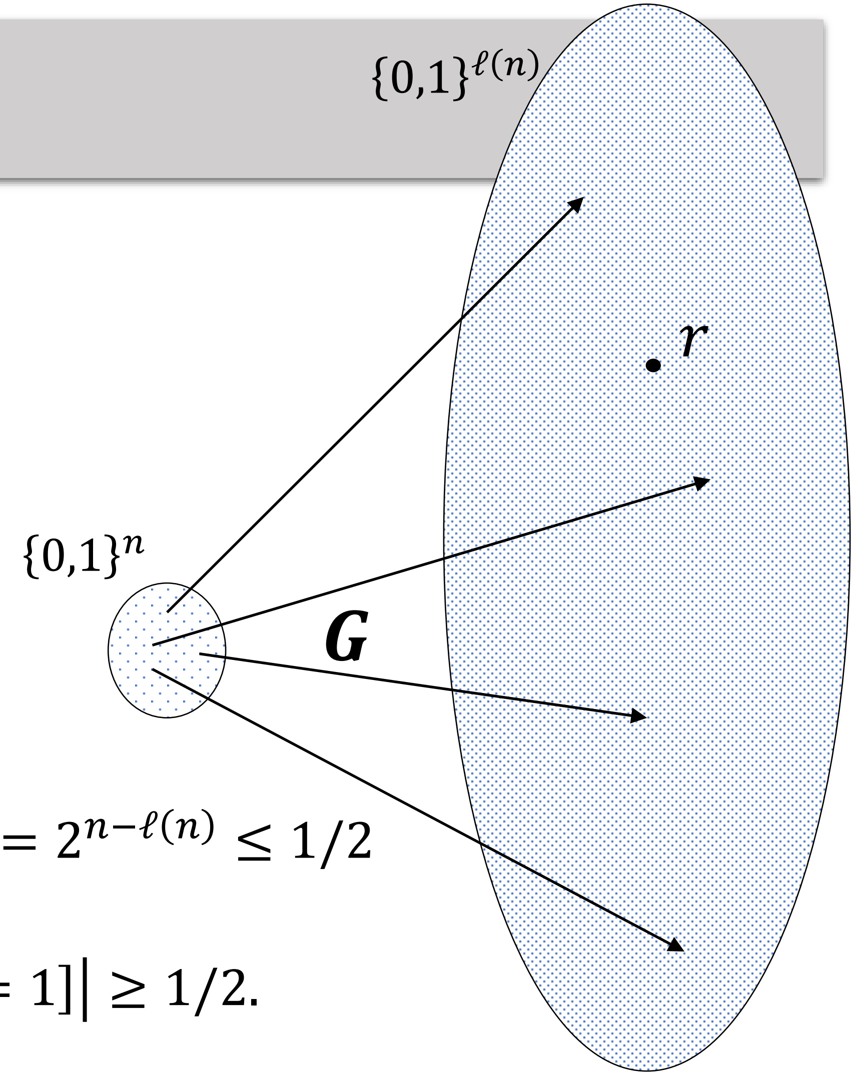
How to break any PRG (in two easy steps).

Step 1: look up the PRG spec online;

Step 2: run the algorithm $\boldsymbol{D}$ below.

input: $r \in \{0,1\}^{\ell(n)}$

- try every possible $s \in \{0,1\}^n$;
- if you find one such that $\boldsymbol{G}(s) = r$, return 1.
- if not, return $\mathbf{0}$.

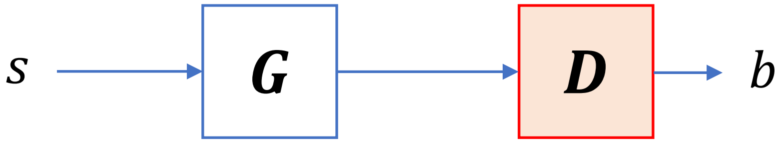


$$\Pr[r \in \boldsymbol{G}(\{0,1\}^n)] \leq \frac{\{0,1\}^n}{\{0,1\}^{\ell(n)}} = 2^{n-\ell(n)} \leq 1/2$$

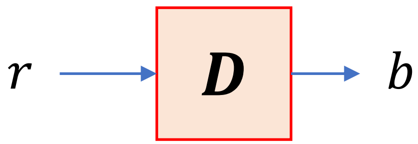
$$\Rightarrow \left|\Pr\left[\boldsymbol{D}\left(\boldsymbol{G}(s)\right) = 1\right] - \Pr[\boldsymbol{D}(r) = 1]\right| \geq 1/2.$$

$\boldsymbol{D}$ always outputs 1.

$s \leftarrow \{0,1\}^n$



$r \leftarrow \{0,1\}^{\ell(n)}$



$\boldsymbol{D}$ outputs 0 except with probability $2^{n-\ell(n)}$.

PSEUDORANDOM GENERATORS (PRGs)

How to construct PRGs.

It's an art form. Lots of constructions do exist.

For example: is this a **PRG**?

input: $s \in \{0,1\}^n$

- compute $b = s_1 \oplus s_2 \oplus \cdots \oplus s_n$
- output $s||b \in \{0,1\}^{n+1}$.

If yes, why? If no, how would you break it?

How about this? Is this a **PRG**?

```
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

(input is 256 bytes; output length arbitrary.)

Or this one?

```
while GeneratingOutput:
    i := i + w
    j := k + S[j + S[i]]
    k := k + i + S[j]
    swap values of S[i] and S[j]
    output z := S[j + S[i + S[z + k]]]
endwhile
```

(arithmetic still mod 256)

COMPUTATIONAL CRYPTO: CHALLENGES

This intuition seems sound. How can we formalize it?

1. Notions to define:

- "random-looking"
- "good-enough" randomness
- "feasible" vs "infeasible" algorithms
- "secure" encryption (can't be same as perfect secrecy, we gave up on that.)

2. Stuff to construct:

- a function which produces "good enough" randomness against "feasible" algorithms

3. Theorems we have to prove:

- the construction in the previous slide is secure.

RECALL: PERFECT SECRECY

Definition 1. (very informal) An encryption scheme is **semantically secret** if, for all choices of adversary $\boldsymbol{A}$, message m, "prior information" function g, and "target information" function f, the following property holds:

$$\Pr\left[f(m) \leftarrow \boldsymbol{A}\left(g(m), \mathbf{Enc}_{\boldsymbol{k}}(m)\right)\right] = \Pr\left[f(m) \leftarrow \boldsymbol{A}\left(g(m)\right)\right].$$

Definition 2. An encryption scheme is **perfectly secret** if, for every plaintext distribution $\mathcal{M}$, every plaintext m, and every ciphertext c,

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

Definition 3. An encryption scheme is **perfectly secret** if, for every plaintext distribution $\mathcal{M}$, every plaintext pair m, m', and every ciphertext c,

$$\Pr_k[\mathbf{Enc}_k(m) = c] = \Pr_k[\mathbf{Enc}_k(m') = c]$$

Definition 4. An encryption scheme has **perfectly indistinguishable ciphertexts** if, for every adversary $\boldsymbol{A}$,

$$\Pr_k[\boldsymbol{A} \text{ wins the IND game}] = \frac{1}{2}.$$

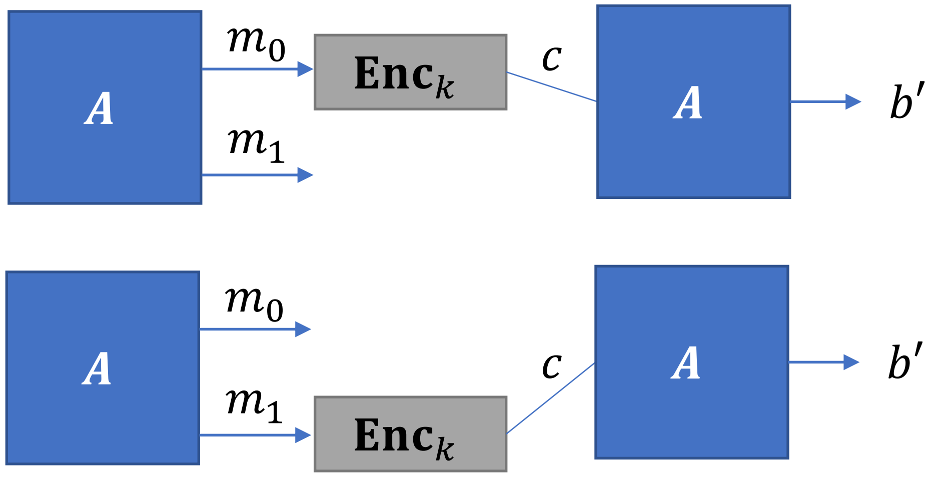
Theorem 1. Definitions 1-4 are all equivalent.

INDISTINGUISHABILITY OF CIPHERTEXTS

Indistinguishability experiment (IND).

1. $\boldsymbol{A}$ outputs two messages m_0, m_1 with $|m_0| = |m_1|$;
2. We sample a key $k \leftarrow \textbf{KeyGen}$, and a coin $b \leftarrow \{0,1\}$;
   then we give $\boldsymbol{A}$ the ciphertext $c \leftarrow \textbf{Enc}_k(m_b)$;
3. $\boldsymbol{A}$ outputs a bit b'.

We say $\boldsymbol{A}$ wins if $b = b'$**.**



Definition. An encryption scheme $(\textbf{KeyGen}, \textbf{Enc}, \textbf{Dec})$ has **indistinguishable ciphertexts** if, for every PPT adversary $\boldsymbol{A}$,

$$\Pr[\boldsymbol{A} \text{ wins IND}] \leq \frac{1}{2} + \mathrm{negl}(n).$$

COMPUTATIONALLY-SECURE ENCRYPTION

Construction.

Let $\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a pseudorandom generator.

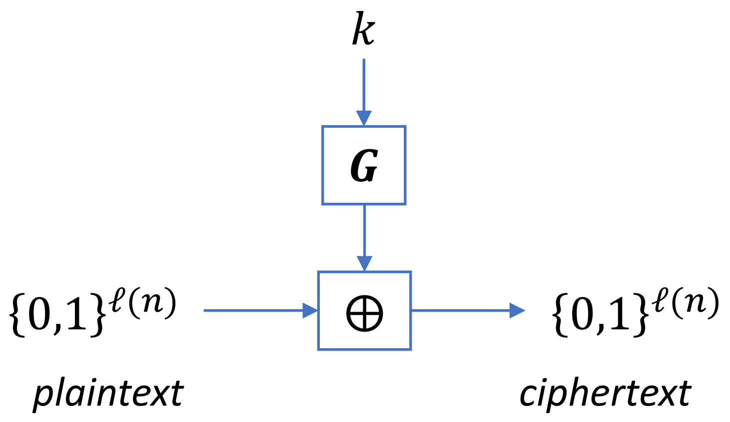
Define an encryption scheme (for $\ell(n)$-bit messages) as follows:

Construction: PRG scheme.

Key generation : sample $k \leftarrow \{0,1\}^n$;

Encryption : $\mathbf{Enc}_k(m) = m \oplus \boldsymbol{G}(k)$;

Decryption : $\mathbf{Dec}_k(c) = c \oplus \boldsymbol{G}(k)$.



- clearly, $\ell(n)$ can be much larger than n;
- so we can't hope for perfect secrecy (Shannon's theorem);
- can we have **IND** (indistinguishability of ciphertexts)?

COMPUTATIONAL CRYPTO: CHALLENGES

This intuition seems sound. How can we formalize it?

1. Notions to define:

- "random-looking" ✓
- "good-enough" randomness ✓
- "feasible" vs "infeasible" algorithms ✓
- "secure" encryption (can't be same as perfect secrecy, we gave up on that.) ✓

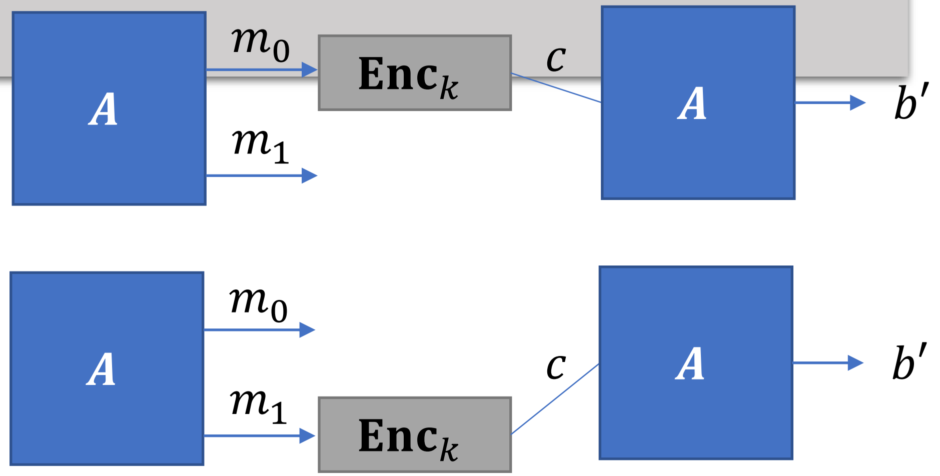
2. Stuff to construct:

- a function which produces "good enough" randomness against "feasible" algorithms ✓

3. Theorems we have to prove:

- the construction in the previous slide is secure. ←

PRG ENCRYPTION SECURITY PROOF



Claim: PRG encryption has indistinguishable ciphertexts.

- How to prove this?
- What's our only leverage? The assumption that $\boldsymbol{G}$ is a PRG;
- So let's try proof by contradiction:

"If there's an attacker $\boldsymbol{A}$ that can win the IND game,

then there's an ~~attacker~~ $\boldsymbol{D}$ against $\boldsymbol{G}$."

distinguisher

- called a "reductionist proof" or "proof by reduction."
- used ***a lot*** in crypto: learn it, get used to it!

PRG ENCRYPTION SECURITY PROOF: RECALL PRG

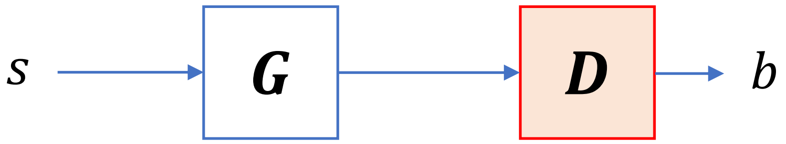
Cryptographic pseudorandomness

Definition. A **pseudorandom generator** is a deterministic, polynomial-time algorithm $\boldsymbol{G}$ satisfying the following:

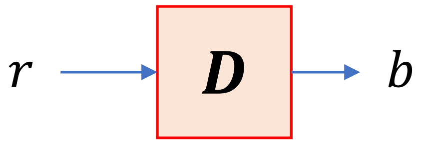
1. (expansion) $\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{\ell(n)}$ for some fixed polynomial ℓ satisfying $\ell(n) > n$ for all n.
2. (pseudorandomness) for every PPT algorithm $\boldsymbol{D}$,

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

$s \leftarrow \{0,1\}^n$



$r \leftarrow \{0,1\}^{\ell(n)}$

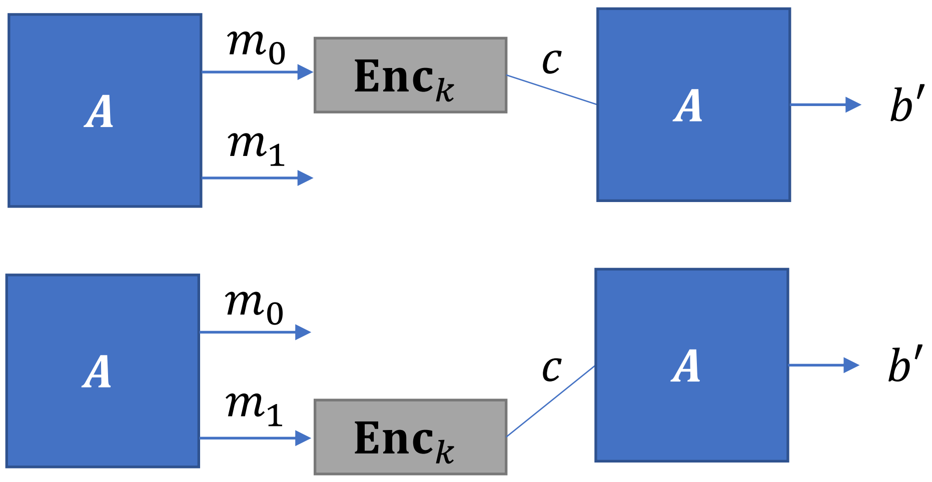


PRG ENCRYPTION SECURITY PROOF: RECALL IND

Indistinguishability experiment (IND).

1. $\boldsymbol{A}$ outputs two messages m_0, m_1 with $|m_0| = |m_1|$;
2. We sample a key $k \leftarrow \textbf{KeyGen}$, and a coin $b \leftarrow \{0,1\}$;
   then we give $\boldsymbol{A}$ the ciphertext $c \leftarrow \textbf{Enc}_k(m_b)$;
3. $\boldsymbol{A}$ outputs a bit b'.

We say $\boldsymbol{A}$ wins if $b = b'$.

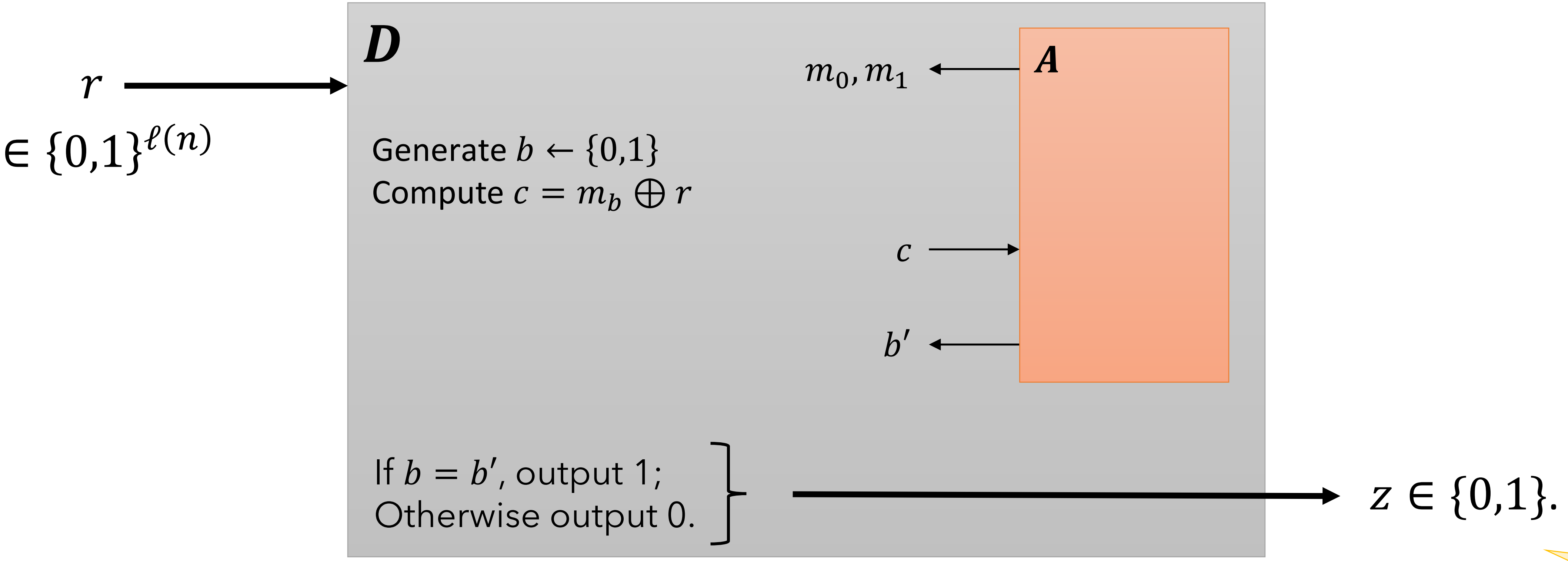


Definition. An encryption scheme $(\textbf{KeyGen}, \textbf{Enc}, \textbf{Dec})$ has **indistinguishable ciphertexts** if, for every PPT adversary $\boldsymbol{A}$,

$$\Pr[\boldsymbol{A} \text{ wins IND}] \leq \frac{1}{2} + \text{negl}(n).$$

PRG ENCRYPTION SECURITY PROOF

"If there's an attacker $\boldsymbol{A}$ that can win the IND game, then there's a distinguisher $\boldsymbol{D}$ against $\boldsymbol{G}$."



Key facts:

1. if r is uniformly random, $\boldsymbol{A}$ is playing the IND game against the one-time pad.
2. if r is $\boldsymbol{G}(s)$, $\boldsymbol{A}$ is playing the IND game against the PRG scheme.

A will LOSE: OTP perfect!

A will WIN: by assumption!

PRG ENCRYPTION: SECURITY PROOF

Let's analyze $\boldsymbol{D}$.

Two cases:

(1.) r is uniformly random in $\{0,1\}^{\ell(n)}$.

- Then $\boldsymbol{D}$ is an exact simulation of this IND game:
- $\boldsymbol{A}$ plays against the one-time pad with keylength $\ell(n)$;
- by perfect secrecy of OTP, $\boldsymbol{A}$ loses: $\Pr[b = b'] = 1/2$;
- it follows that $\Pr[z = 1] = 1/2$.

(2.) $r = \boldsymbol{G}(s)$ for uniformly random $s \in \{0,1\}^n$.

- Then $\boldsymbol{D}$ is an exact simulation of this IND game:
- $\boldsymbol{A}$ plays against the PRG scheme with PRG $\boldsymbol{G}$;
- by assumption, $\boldsymbol{A}$ wins noticeably, i.e. $\Pr[b = b'] \geq 1/2 + 1/p(n)$ for some polynomial p;
- it follows that $\Pr[z = 1] = 1/2 + 1/p(n)$.

$$\left|\Pr\left[\boldsymbol{D}\left(\boldsymbol{G}(s)\right) = 1\right] - \Pr[\boldsymbol{D}(r) = 1]\right| = \left|\left(\frac{1}{2} + \frac{1}{p(n)}\right) - \frac{1}{2}\right| = \frac{1}{p(n)}.$$

□