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/ (slides, reading);

Piazza: piazza.com/umd/spring2020/cmsc456

ELMS: active, slides and reading 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: THE BIG IDEA

Shannon's theorem: if you want perfect secrecy, one-time pad is as good as it gets.

Limits of one-time pad:

- can only send one message;
- message cannot be longer than the shared key;

} *solution: computationally-secure encryption*

- what if you can't share key in advance?
- what if Eve is allowed to *change* messages?

} *later in the course*

- …

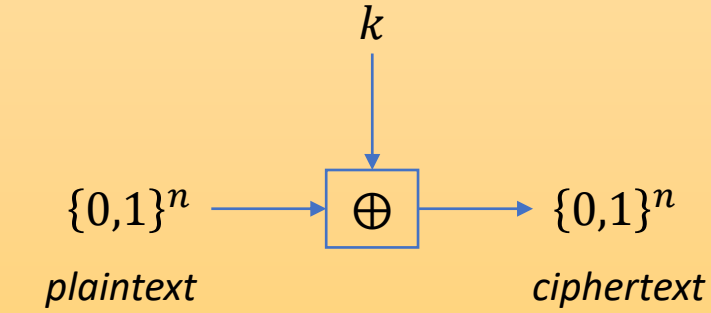
Obviously:

- the crypto we use every day does not suffer from these drawbacks;
- … (Shannon's theorem) it follows that we must relax perfect secrecy; What to do?
  1. *allow adversaries to succeed with exponentially-small probability (roughly);*
  2. *allow adversaries to succeed after exponential time (roughly);*

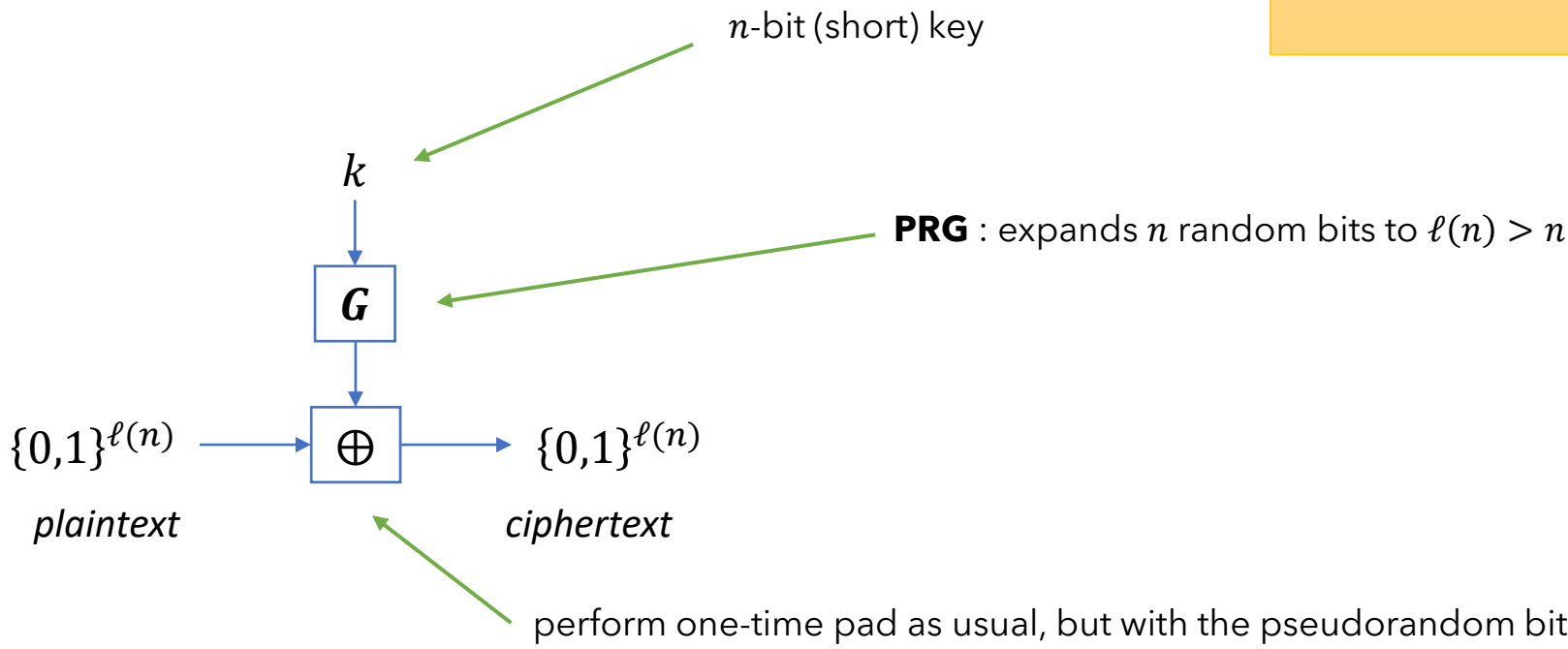
Claim: if done right, no "real" security loss, but huge gain in features!

RECAP: COMPUTATIONALLY-SECURE ENCRYPTION

One-time pad



Idea: use "good enough" randomness in OTP (instead of perfect.)



PRG : expands n random bits to $\ell(n) > n$ "pseudorandom" bits

perform one-time pad as usual, but with the pseudorandom bits

(To decrypt: reverse the arrows from ciphertext to plaintext.)

PRG scheme

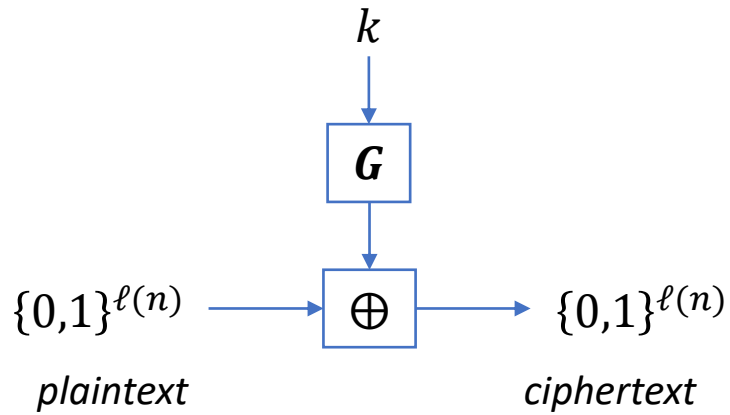
Key generation : sample $k \leftarrow \{0,1\}^n$;
Encryption : $\mathbf{Enc}_k(m) = m \oplus \boldsymbol{G}(k)$;
Decryption : $\mathbf{Dec}_k(c) = c \oplus \boldsymbol{G}(k)$.

RECAP: COMPUTATIONALLY-SECURE ENCRYPTION

Idea: use "good enough" randomness in OTP (instead of perfect.)

How to make this formal?

1. define pseudorandomness;
2. define notion of "efficient" and "inefficient";
3. define a relaxed notion of security;
4. prove that this scheme works.



RECAP: EFFICIENT VS INEFFICIENT

Efficient algorithms: probabilistic, polynomial-time (PPT.)

- running time: at most polynomial in the input size;

AND

- success probability: at least inverse-polynomial in input size;

Think: "Achieves a noticeable success probability, in a reasonable amount of time."

Inefficient algorithms:

- running time: superpolynomial in input size (i.e., exponential or nearly so.);

OR

- success probability: at most inverse-superpolynomial (i.e., negligibile.)

Think: "can achieve noticeable success probability only by spending an UNreasonable amount of time."

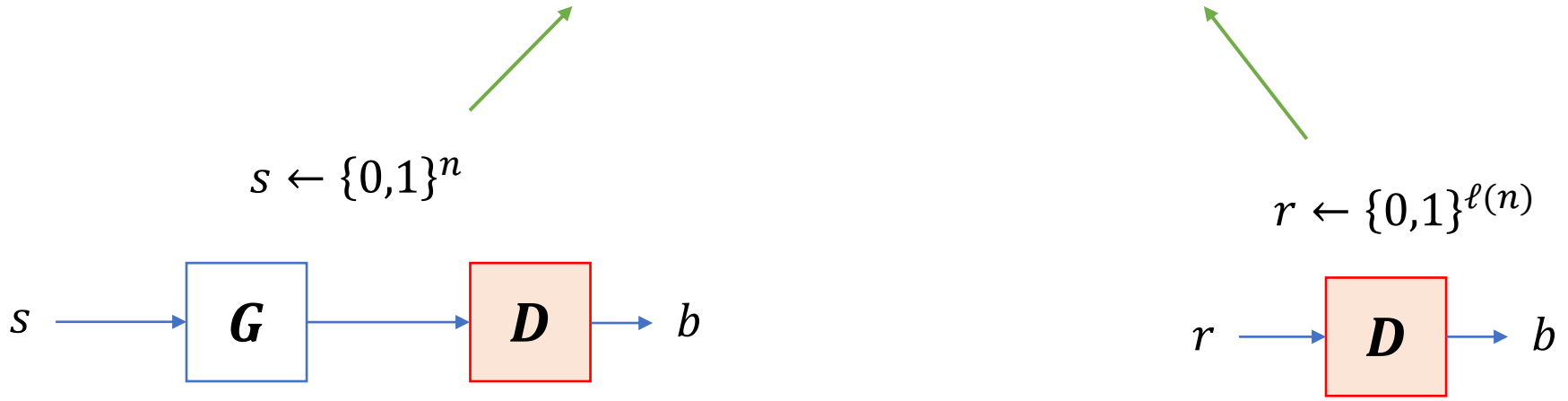
RECAP: PRGs

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 $\boldsymbol{D}$,

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

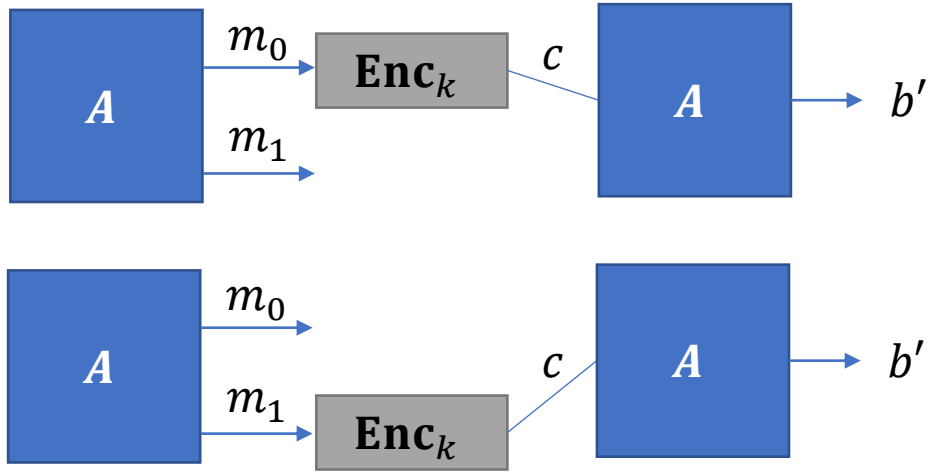


RECAP: "IND" SECRECY

Indistinguishability experiment (IND).

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

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



Definition. An encryption scheme $(\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ 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

The PRG scheme is secure.

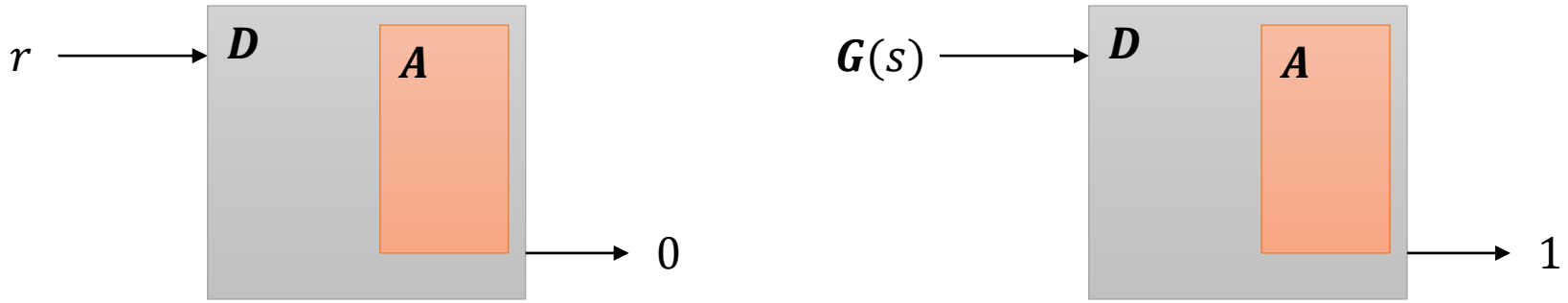
Theorem. The PRG scheme has indistinguishable ciphertexts.

Proof.

Proof by contradiction: "if PRG scheme is broken, then the underlying PRG is broken."

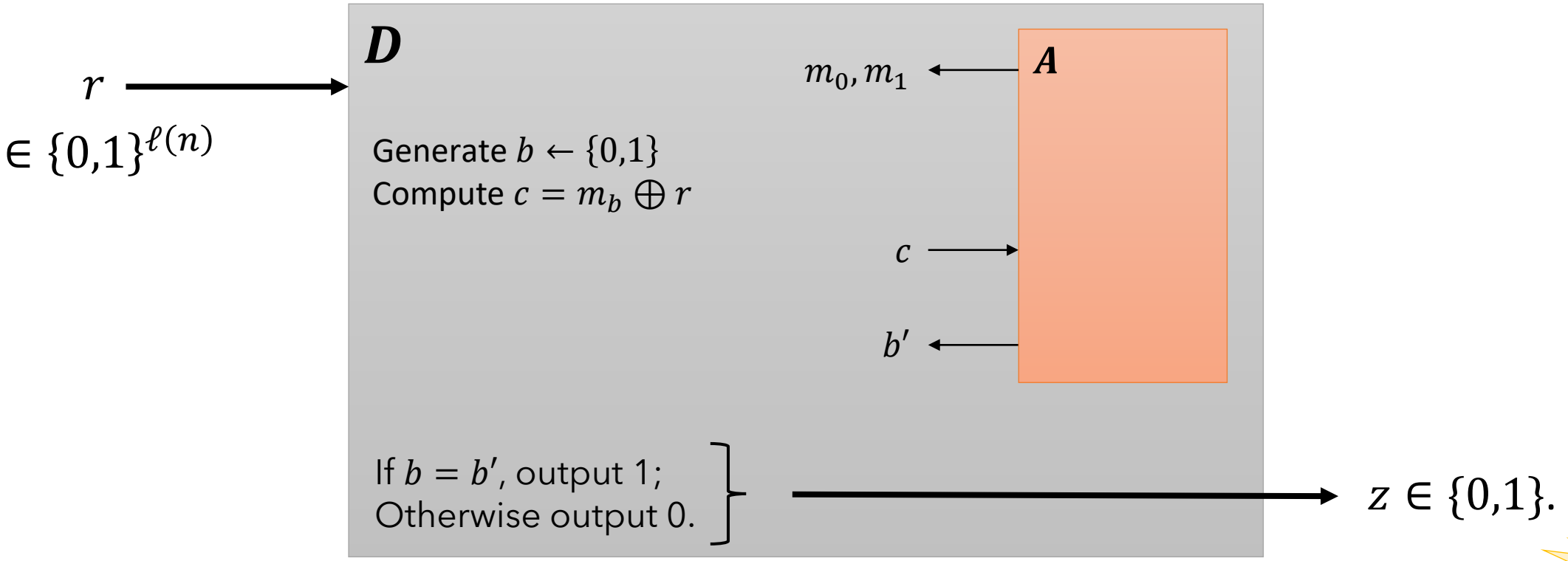
More concretely:

- Let $\boldsymbol{G}$ be a PRG, and let $\Pi(\boldsymbol{G})$ be the PRG scheme using $\boldsymbol{G}$;
- Given a PPT $\boldsymbol{A}$ that wins IND game against $\Pi(\boldsymbol{G})$ …
- … we build a PPT $\boldsymbol{D}$ who distinguishes the output of $\boldsymbol{G}$ from random:



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.

$\boldsymbol{A}$ will LOSE: OTP perfect!

$\boldsymbol{A}$ 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 $\Pr[z = 1] = 1/2 + 1/p(n)$.

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

□

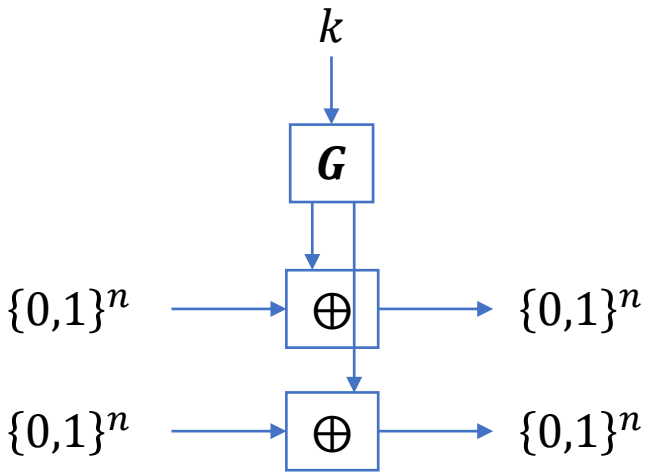
IV. PSEUDORANDOM FUNCTIONS

Reading: p.71-95

WHAT DID PRG ENCRYPTION GET US?

PRGs enable:

- *fixed-length* encryption with poly-size messages;
- with stateful schemes, allows multiple messages, up to a total $\ell(n)$ bits.

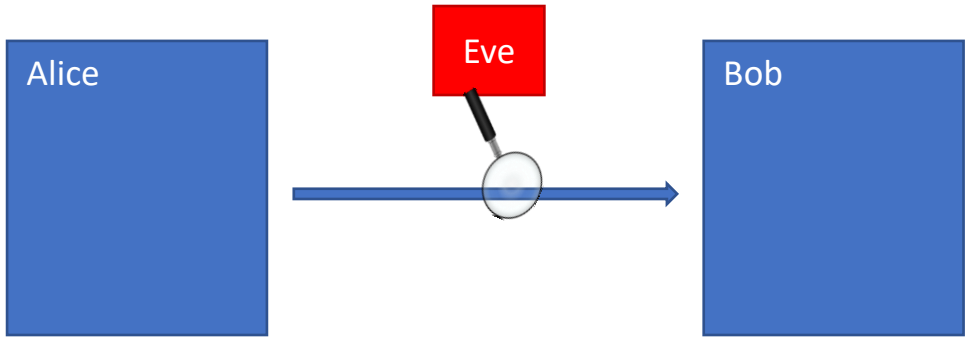


- but how do you decrypt? What if the ciphertexts arrive out of order?
- and what if you don't want to keep state? (a potential attack avenue.)
- and what if you want to send *arbitrarily many* messages?

MORE POWERFUL ATTACKS

So far...

- our model still grants adversary very little power;
- they are only a passive observer;
- in real world, they can do much more!



For example: they can *interrogate systems*.

- try to connect to some authorized system;
- guess passwords and see what happens;
- send transmissions and see if they decrypt to something;
- use real world power over parties to get them to send encrypted messages.

How do we capture things like this in our framework? ***Oracle algorithms.***

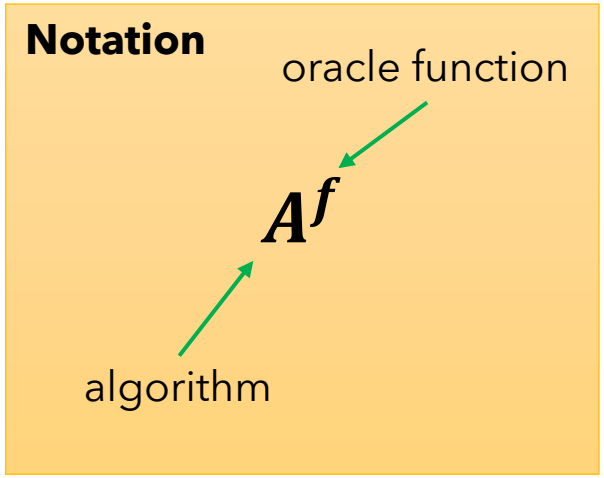
ORACLE ALGORITHMS

Oracle algorithm:

- same as a regular algorithm, but can "invoke" a special subroutine;
- this subroutine simply evaluates some function, and has no other effect;
- the subroutine behaves like a "black box" or an "oracle";
- it costs the algorithm only *one timestep* to query the oracle.

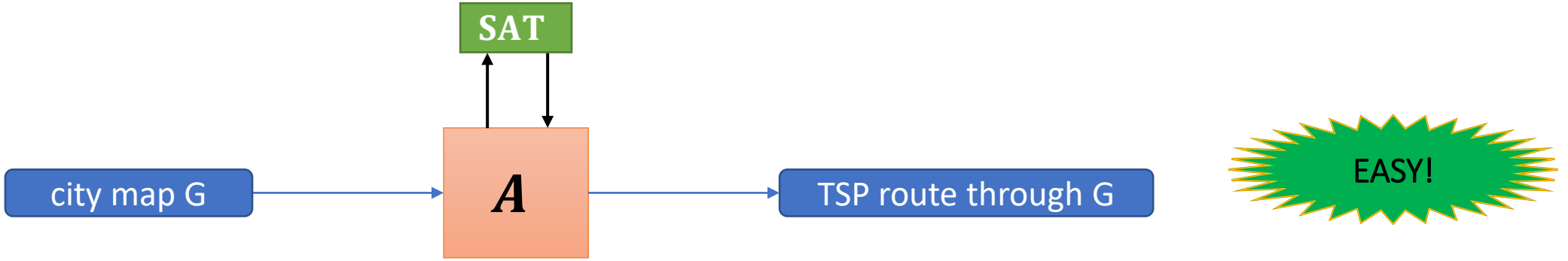
Think: invoking a compiled library method/function when programming.

Notation



Example:

- recall the class NP and polynomial-time reductions;
- there is a poly-time algorithm which, given a SAT oracle, solves the Traveling Salesman Problem;



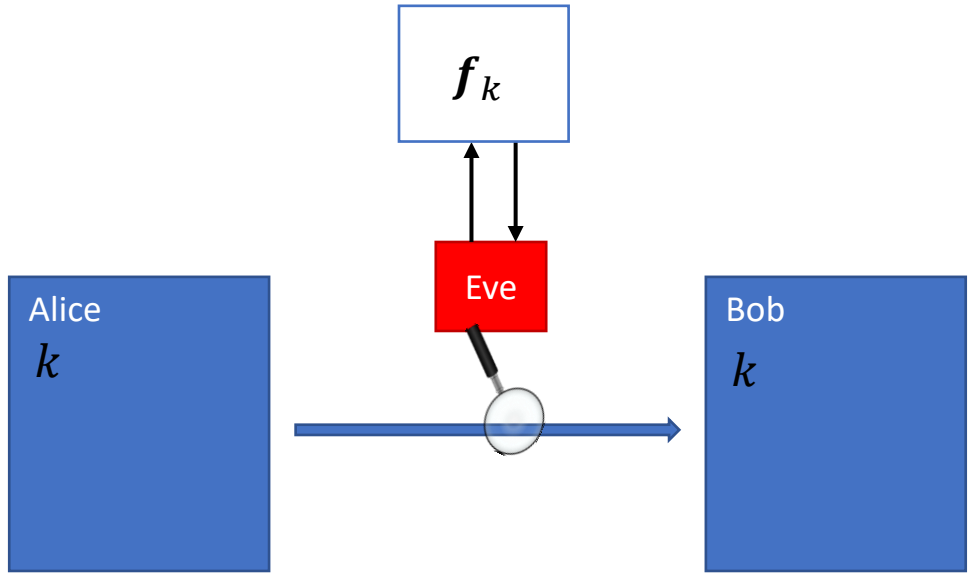
MORE POWERFUL ATTACKS

Oracle algorithms for us:

- allow adversaries to "query" cryptosystem in various ways;
- they can use whatever they learn to try to devise a better attack.

Why give away more power?

- model real situations more accurately;
- eliminate unnecessary weaknesses in system;
- explore ultimate limits of what crypto can do.



MORE POWERFUL ATTACKS

Oracle algorithms for us:

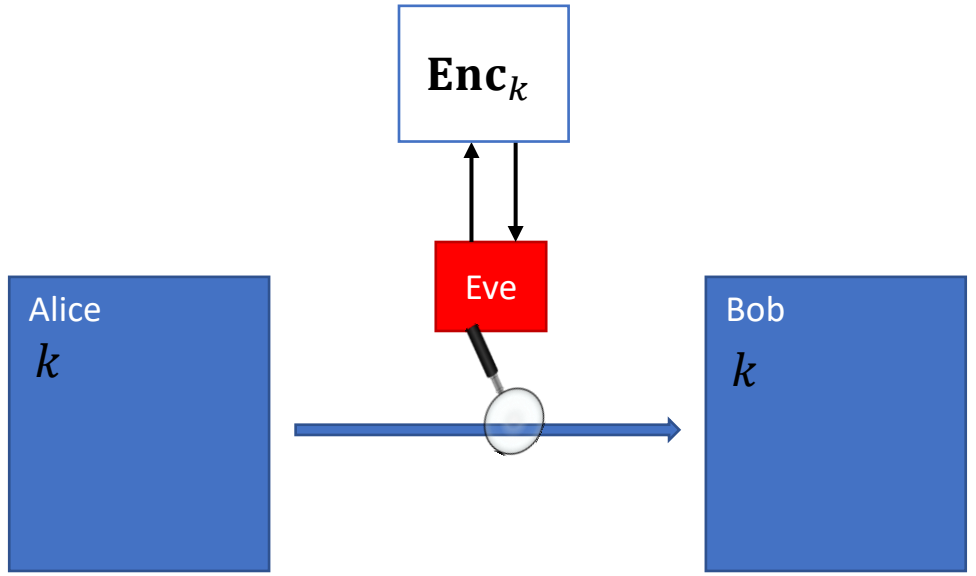
- allow adversaries to "query" cryptosystem in various ways;
- they can use whatever they learn to try to devise a better attack.

Why give away more power?

- model real situations more accurately;
- eliminate unnecessary weaknesses in system;
- explore ultimate limits of what crypto can do.

For example:

- give Eve access to encryption!
- can we still have secrecy?



PSEUDORANDOM FUNCTIONS

A more powerful primitive: pseudorandom *functions*.

PRG

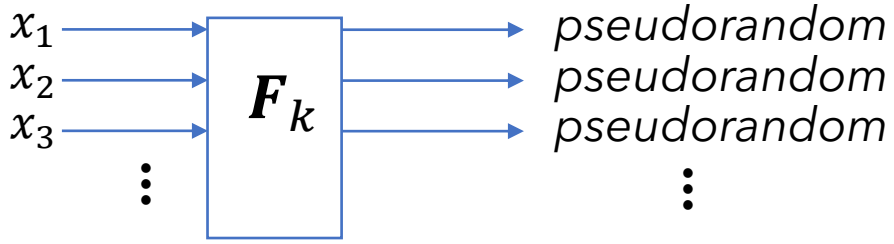
- public algorithm $\boldsymbol{G}$;
- if you plug in a random string…
- … you get back a longer, pseudorandom string.

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

PRF

- keyed algorithm $\boldsymbol{F}$ (kind of like encryption);
- if you plug in *any* string..
- … you get back a pseudorandom string.

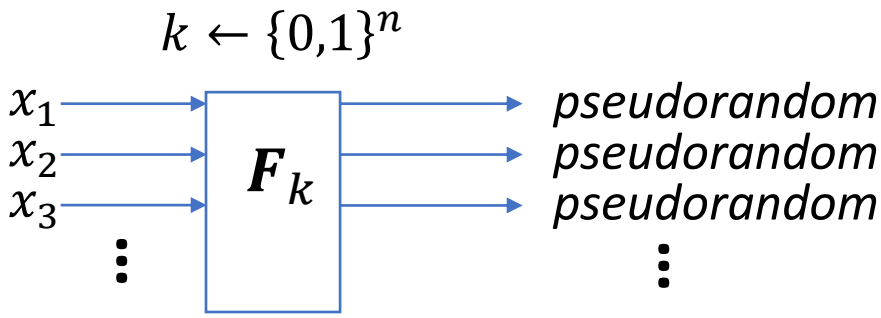
$k \leftarrow \{0,1\}^n$



- query anywhere, as many times as you want…
- … and still the output looks random!

PSEUDORANDOM FUNCTIONS

A more powerful primitive: pseudorandom *functions*.



It's a function family:

key:
- choose uniformly at random
- keep secret!

$$F: \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^\ell$$

input: choose any way you want

output: will look pseudorandom

Given a key $k \in \{0,1\}^n$, we get a function like this: $F_k: \{0,1\}^m \rightarrow \{0,1\}^\ell$.

PSEUDORANDOM FUNCTIONS

What does a PRF do?

key input output

$$F: \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$$

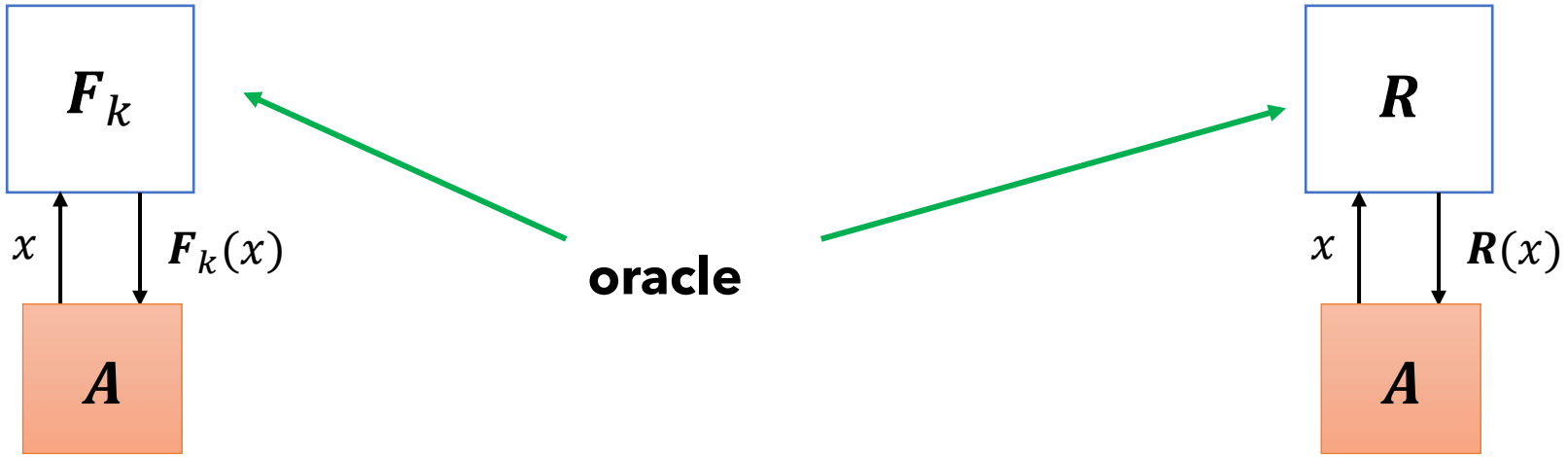
"Real" experiment:

- pick a key $k \leftarrow \{0,1\}^n$;
- "put function F_k in a box";
- give box to an adversary;
- adversary cannot open box…
- … but can plug in any input, and get output.

VS

"Ideal" experiment:

- pick a *completely random function* R;
- "put function R in a box";
- give box to an adversary;
- adversary cannot open box…
- … but can plug in any input, and get output.



PSEUDORANDOM FUNCTIONS

Formal definition: "oracle adversaries can't tell it apart from totally random"

Definition. A **pseudorandom function** is a deterministic, polynomial-time-computable function

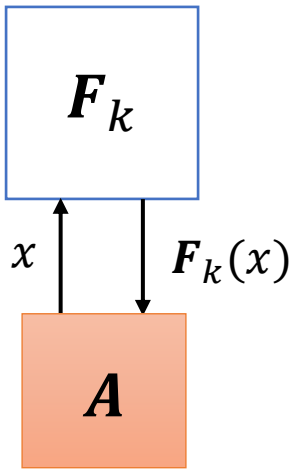
$$\boldsymbol{F}: \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$$

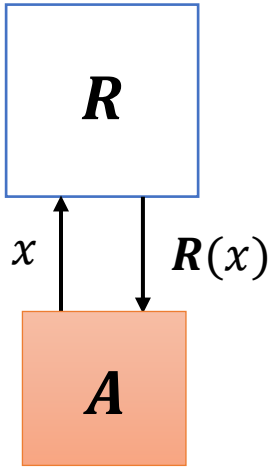
such that, for every PPT algorithm $\boldsymbol{A}$,

$$\left|\Pr_{k \leftarrow \{0,1\}^n}\left[\boldsymbol{A}^{\boldsymbol{F}_k}(1^n) = 1\right] - \Pr_{\boldsymbol{R} \leftarrow \mathcal{F}_{m,\ell}}\left[\boldsymbol{A}^{\boldsymbol{R}}(1^n) = 1\right]\right| \leq \mathrm{negl}(n).$$

uniformly random key

uniformly random function from m bits to ℓ bits





PRFs ARE AWESOME!

Output of PRF looks random!

- it's indistinguishable from uniformly random;
- we know from OTP that uniformly random ciphertexts are good;
- so let's build encryption!
- to encrypt, just apply the PRF: $\mathbf{Enc}_k(m) = \boldsymbol{F}_k(m)$.

Easy to check: adversary can't distinguish output from random.

Does this work?

PRFs vs PRGs

PRFs vs PRGs.

Can you build a PRG from a PRF?

Easy:

- Let $\boldsymbol{F}: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a PRF;
- Build a $\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{2n}$ like this:
- $\boldsymbol{G}(s) \coloneqq \boldsymbol{F}_s(00\cdots00)||\boldsymbol{F}_s(00\cdots01)$
- Easy to extend to arbitrary-length $\boldsymbol{G}$.

Note:

- in PRG setting, seed is uniformly random…
- so we can use it for the key.

So are PRFs stronger than PRGs?

PRFs vs PRGs

Can you build a PRF from a PRG?

Let $\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG, and define:

$\boldsymbol{G_0}: \{0,1\}^n \to \{0,1\}^n$ by $\boldsymbol{G_0}(x) = \boldsymbol{G}(x)|_1^n$ "apply $\boldsymbol{G}$, take left half"

$\boldsymbol{G_1}: \{0,1\}^n \to \{0,1\}^n$ by $\boldsymbol{G_0}(x) = \boldsymbol{G}(x)|_{n+1}^{2n}$ "apply $\boldsymbol{G}$, take right half"

Define $\boldsymbol{F}_k: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ by

$$\boldsymbol{F}_k(x) = G_{x_n}(G_{x_n}(\cdots(G_{x_1}(G_{x_0}(k))\cdots)$$

Theorem (Goldwasser, Goldreich, Micali '88): $\boldsymbol{F}_k$ is a pseudorandom function.

PRFs vs PRGs

Can you build a PRF from a PRG?

Let $\boldsymbol{G}: \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG, and define:

$\boldsymbol{G}_0: \{0,1\}^n \to \{0,1\}^n$ by $\boldsymbol{G}_0(x) = \boldsymbol{G}(x)|_1^n$

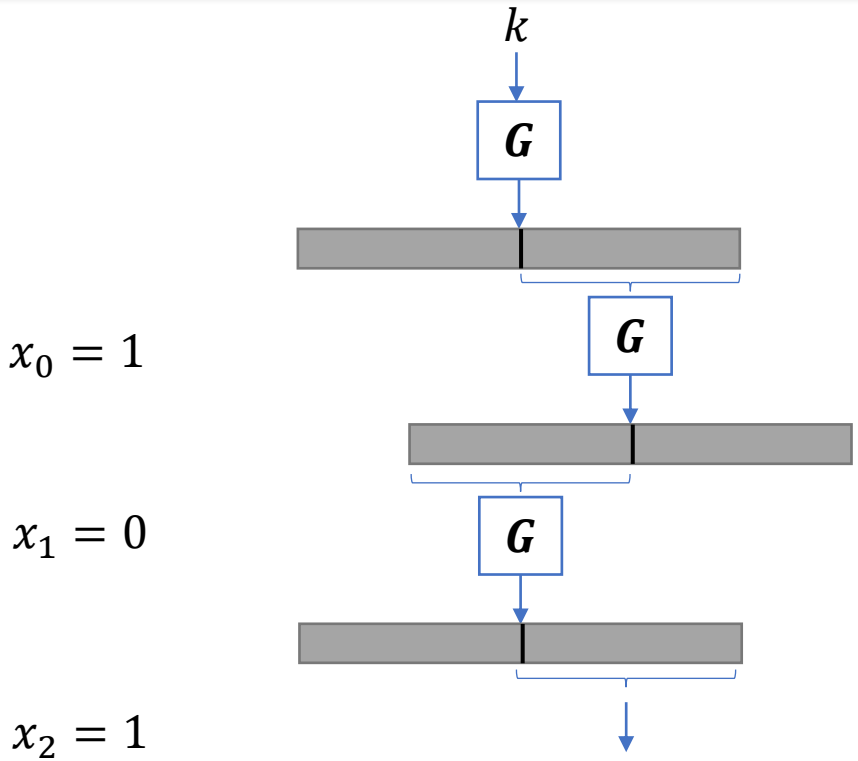
$\boldsymbol{G}_1: \{0,1\}^n \to \{0,1\}^n$ by $\boldsymbol{G}_0(x) = \boldsymbol{G}(x)|_{n+1}^{2n}$

"GGM PRF"

$$\boldsymbol{F}_k(x) = G_{x_n}(G_{x_n}(\cdots(G_{x_1}(G_{x_0}(k)))\cdots)$$

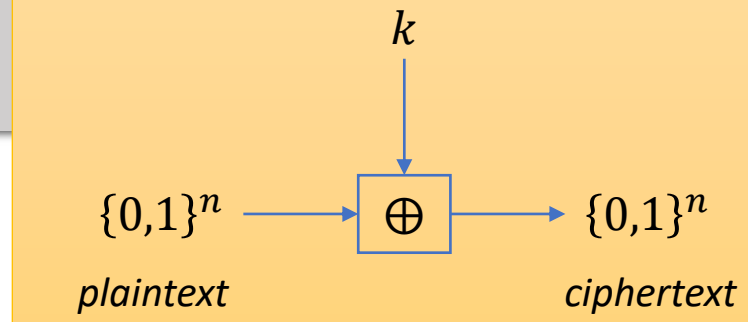
Example:

- suppose n=3
- compute $\boldsymbol{F}_k(101)$.



PRF ENCRYPTION

One-time pad

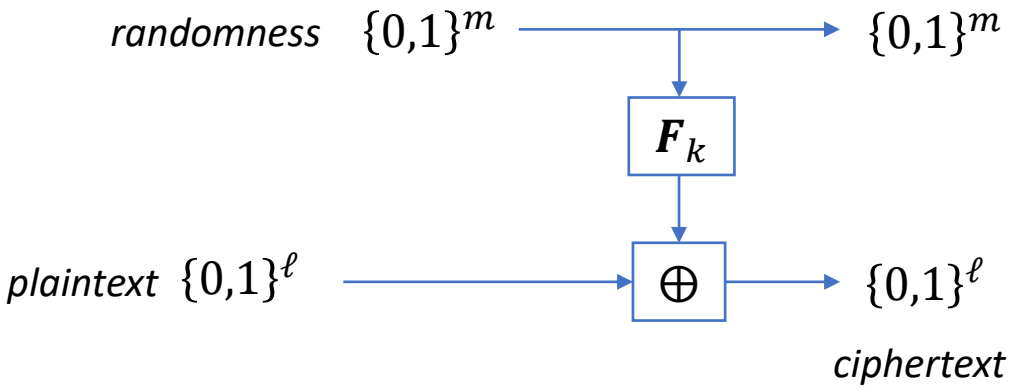


What's a PRF good for?

Lots of things! Like really powerful encryption:

Construction (PRF encryption). Let $\boldsymbol{F}: \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ be a PRF. Define a scheme:

- **KeyGen**: sample a PRF key $k \leftarrow \{0,1\}^n$;
- **Enc**: on input a message $m \in \{0,1\}^\ell$, sample $r \leftarrow \{0,1\}^m$ and output $(r, \boldsymbol{F}_k(r) \oplus m)$;
- **Dec**: on input a ciphertext (r, c), output $c \oplus \boldsymbol{F}_k(r)$.



Some properties

- at its core, there's still OTP
- can send arbitrarily-many messages!
- encryption is now a *randomized* algorithm

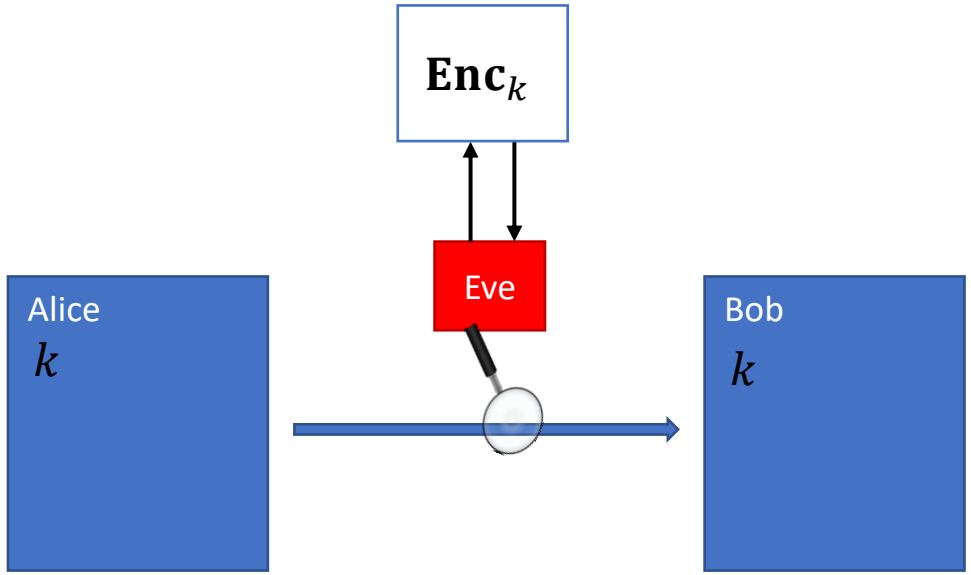
CPA ATTACKS

What about security?

It turns out, we get a big upgrade there too.

Chosen plaintext attacks (CPA):

- at anytime during their attack…
- … adversary can ask for *an encryption of any message!*
- "chosen plaintext";



Examples.

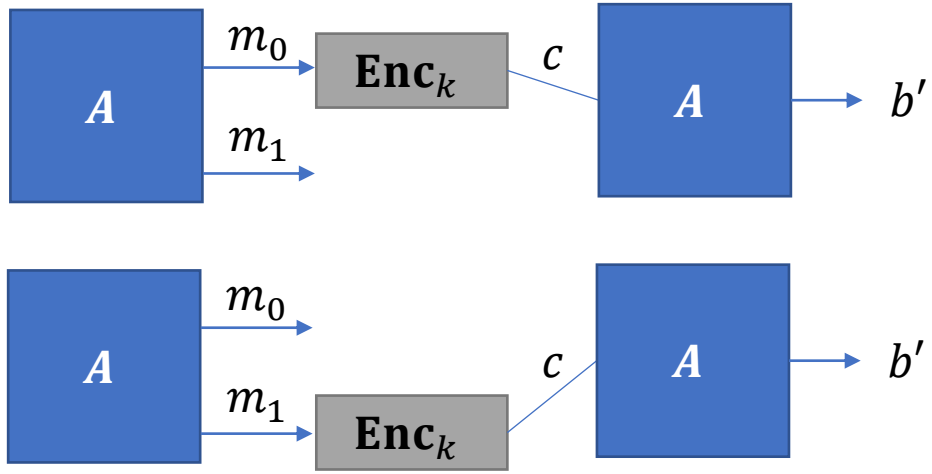
- against OTP: query $\mathbf{Enc}_k(0^n) = 0^n \oplus k = k$. *Complete key recovery with one query!*
- against PRG scheme: can only encrypt a limited number of times…
- … so adversary can just use them all up!

RECALL: "IND" SECRECY

Indistinguishability experiment (IND).

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

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



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

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

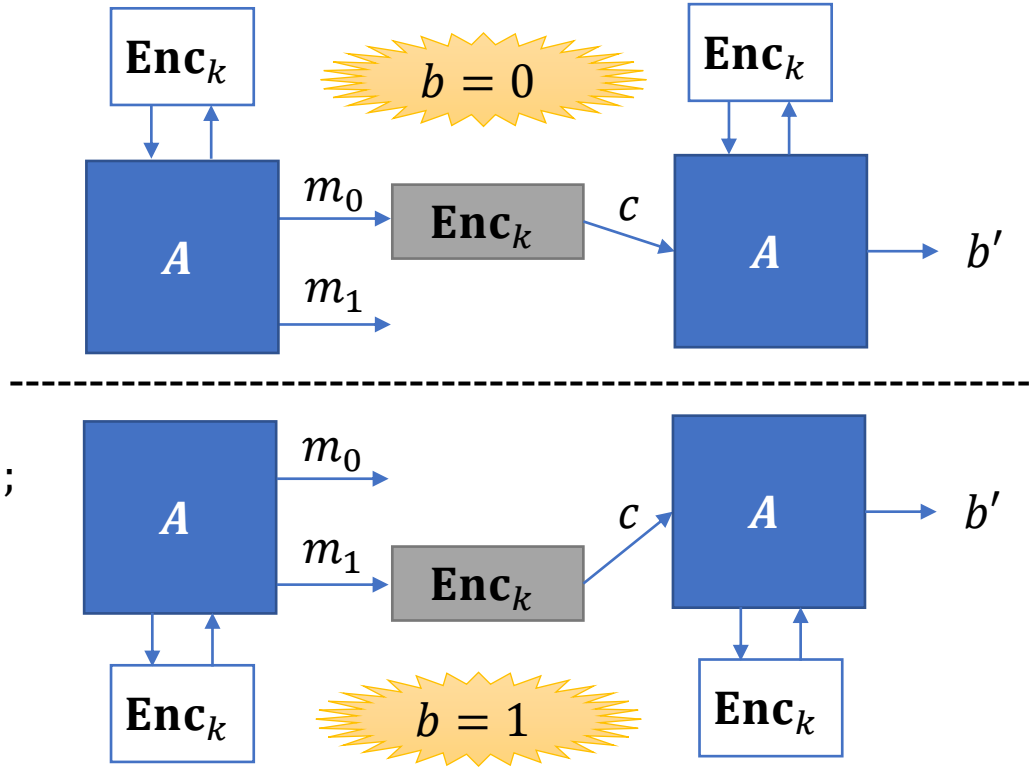
IND-CPA

Indistinguishability under Chosen Plaintext Attack.

INDCPA experiment:

1. Sample a key $k \leftarrow \mathbf{KeyGen}$;
2. Give adversary oracle access to $\mathbf{Enc}_k$;
3. $\boldsymbol{A}^{\mathbf{Enc}_k}$ outputs two messages m_0, m_1 with $|m_0| = |m_1|$;
4. Sample a coin $b \leftarrow \{0,1\}$; give $\boldsymbol{A}$ ciphertext $c \leftarrow \mathbf{Enc}_k(m_b)$;
5. $\boldsymbol{A}^{\mathbf{Enc}_k}$ outputs a bit b'.

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



Definition. An encryption scheme $(\mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$ is **IND-CPA** if, for every PPT adversary $\boldsymbol{A}$,

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

End of Lecture 3.