2550 Intro to cybersecurity

L5: Crypto: OWF, PRG

abhi shelat

Perfect secrecy

(Gen, Enc, Dec, $\mathcal{M}$, $\mathcal{K}$)

is said to be PERFECTLY SECRET if

for every pair m_1, m_2 of messages $m \in \mathcal{M}$, and every ciphertext, ciphertext c is equally likely to represent either m_1 or m_2 (if the key k is uniformly sampled from the keyset)

— Equivalent to "Shannon security"

Perfect secrecy

$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathcal{M}, \mathcal{K})$

is said to be PERFECTLY SECRET if

$$\forall m_1, m_2 \in \mathcal{M}, \forall c$$

$$\Pr[k \leftarrow \mathsf{Gen} \;:\; \mathsf{Enc}_k(m_1) = c]$$

$$=$$

$$\Pr[k \leftarrow \mathsf{Gen} \;:\; \mathsf{Enc}_k(m_2) = c]$$

One-time pad (Vernam 1917)

Key is too long

$$
\begin{aligned}
\mathcal{M} &= \{0,1\}^n \\
\mathcal{K} &= \{0,1\}^n \\
\mathsf{Gen} &= k = k_1k_2\ldots k_n \leftarrow \{0,1\}^n \\
Enc_k(m_1m_2\ldots m_n) &= c_1c_2\ldots c_n \text{ where } c_i = m_i \oplus k_i \\
Dec_k(c_1c_2\ldots c_n) &= m_1m_2\ldots m_n \text{ where } m_i = c_i \oplus k_i
\end{aligned}
$$

Uniform distribution on strings of len n

$K = \{0,1\}^n$

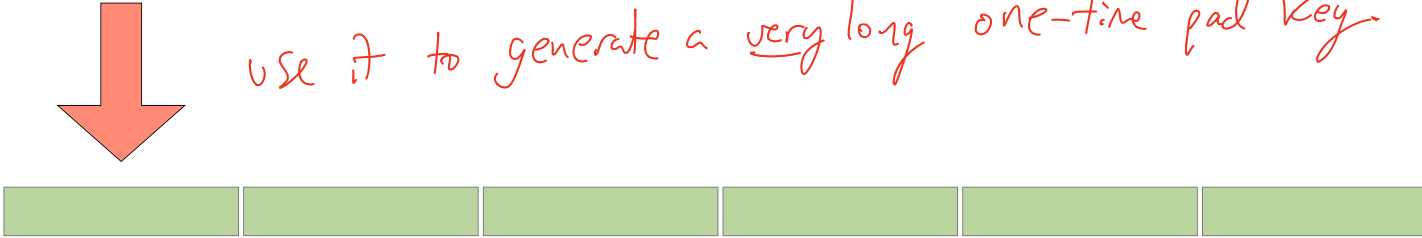
is sampled from $U_n = \{0,1\}^n$

randomly sampled binary strings of length n.

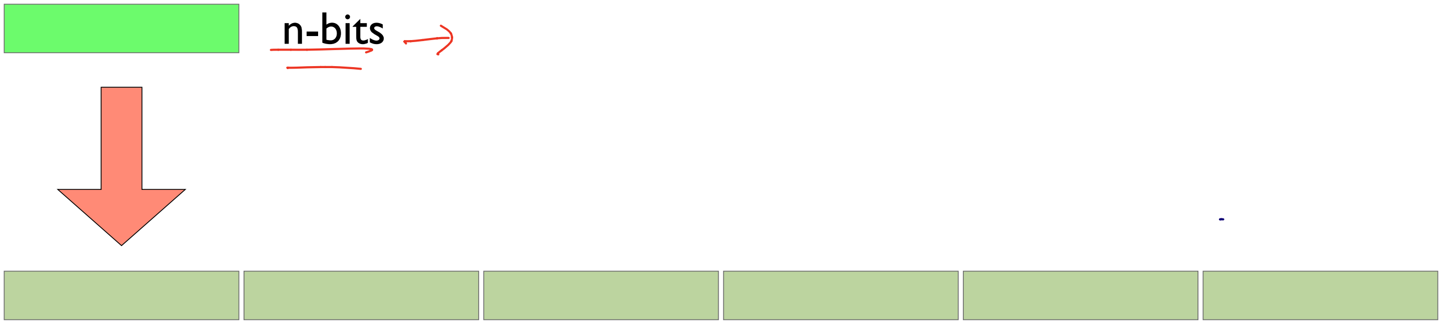
Goal:

pick a small key,
say 128 bits long

use it to generate a very long one-time pad key.



10^{10} * n-bits



10^{10} * n-bits

should appear to have been sampled from $\mathcal{U}_{10^{10}\cdot n}$

what security properties are needed for this to work?

$10^{10} * $ n-bits

should appear to be the same as a random string $\{0,1\}^{10^{10}n}$

$$U_{10^{10}n}$$

random

pseudo-random.

what does it mean for a process {X} to be pseudo-random?

No efficient algorithm can <u>distinguish</u>
between the output of this process $\underline{\{X_n\}_{n \in \mathbb{N}}}$
and truly random samples from U_n.

parameterized experiment

ensembles

$$\underbrace{\{X_n\}}_{}{}_{n \in \mathbb{N}}$$

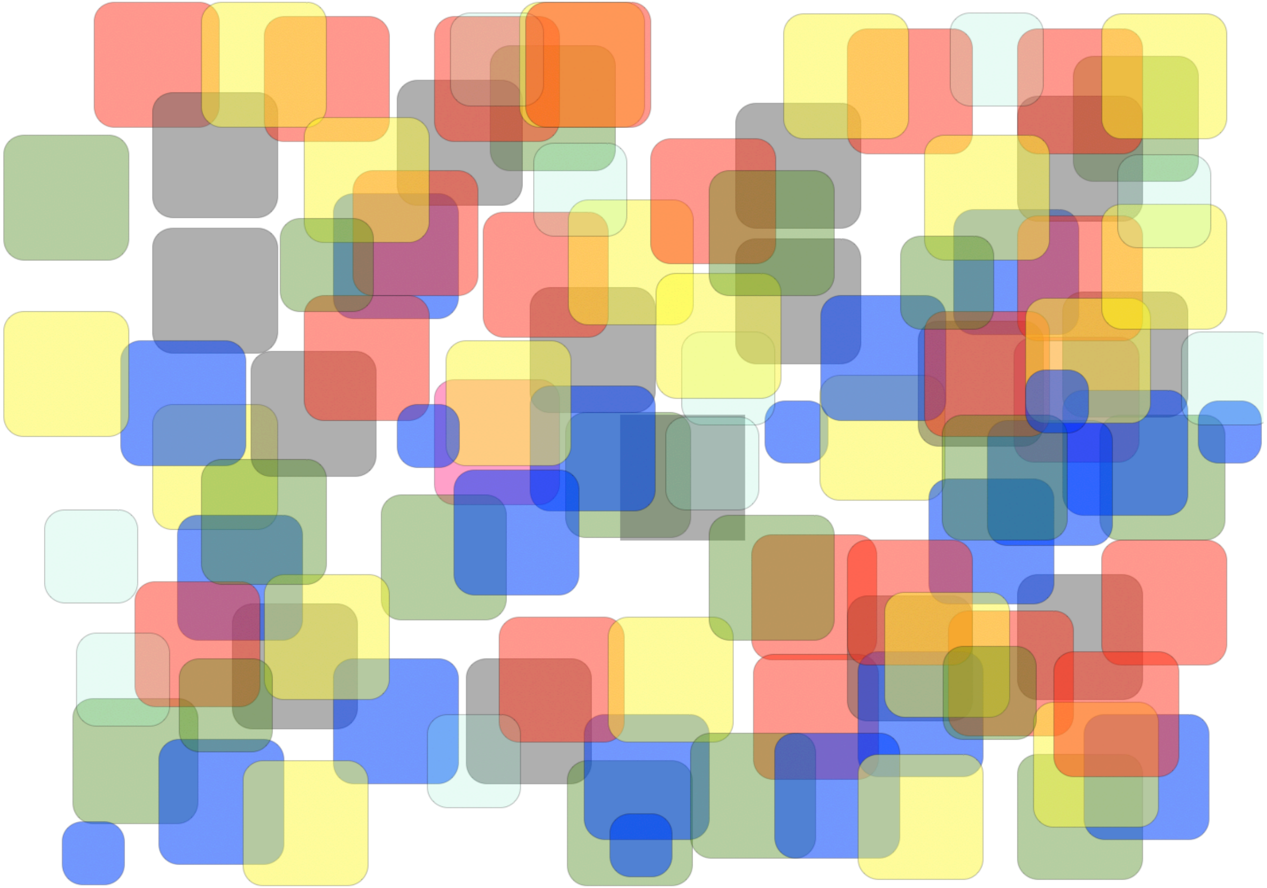
a sequence of probability distributions
where X_n is a distribution over strings of length n

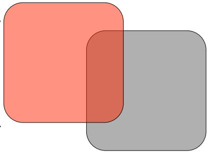
Computational Indistinguishability

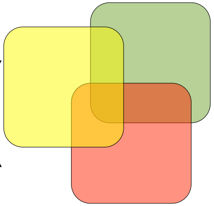
efficient

"let there be two parameterized experiments, *X* and *Y*. as the experiment size increases, no p.p.t. algorithm *D* succeeds in distinguishing *X* from *Y*."

what does it mean
for an algorithm D to
distinguish a sample?



D()= “evens”

D()= “odds”

Two ensembles are comp. indistinguishable

Two ensembles are comp. indistinguishable

$$\{X_n\}_{n \in N} \approx \{Y_n\}_{n \in N}$$ n station.

" for every efficient algorithm D, there exists a negligible function $\epsilon(\cdot)$ such that

$$\left| \Pr[t \leftarrow X_n : D(t) = 1] - \Pr[t \leftarrow Y_n : D(t) = 1] \right| \leq \epsilon(n)$$

Two ensembles are comp. indistinguishable

$$\{X_n\}_{n \in N} \approx \{Y_n\}_{n \in N}$$

if for all non-uniform p.p.t. alg D,
there exists a negligible function $\epsilon(n)$
such that for all n

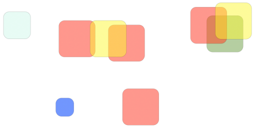
Two ensembles are comp. indistinguishable

$$\{X_n\}_{n\in N} \approx \{Y_n\}_{n\in N}$$

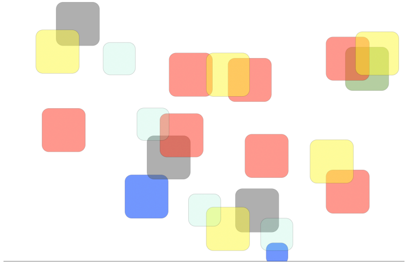
if for all non-uniform p.p.t. alg D,
there exists a negligible function $\epsilon(n)$
such that for all n

$$\left|\Pr\left[t \leftarrow X_n, D(t) = 1\right] - \Pr\left[t \leftarrow Y_n, D(t) = 1\right]\right| \leq \epsilon(n).$$

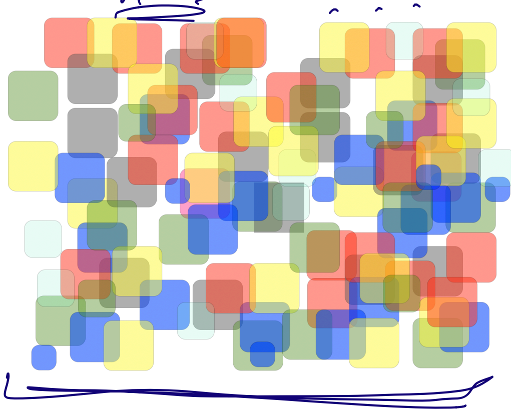
n = 1



n = 2



n = 3



what does it mean
for a process $\{X\}$ to be
pseudo-random?

pseudo-random

An ensemble {X} is said to be

pseudo-random

pseudo-random :

if

$$\underbrace{\{X\}_{n\in\mathbb{N}}}_{} \approx \{U_n\}_{n\in\mathbb{N}}$$

{your generator is computationally indistinguishable from uniformly random strings.}

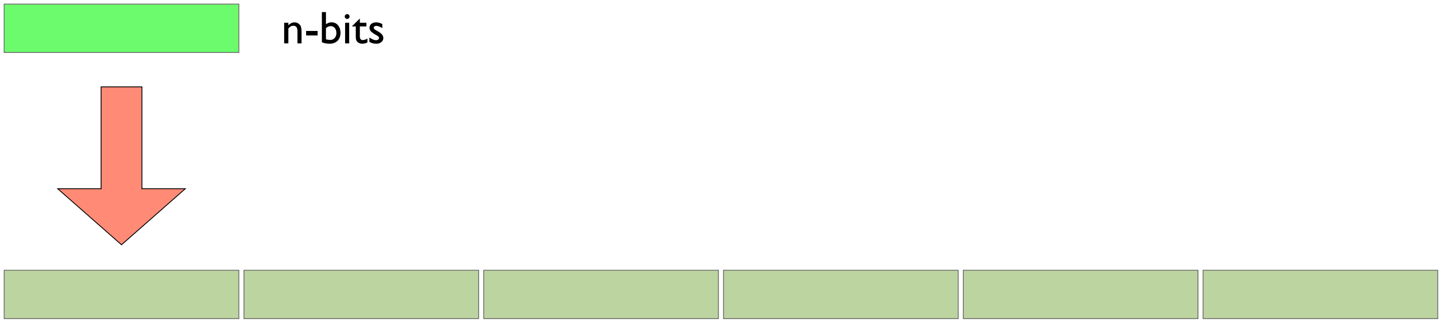
An ensemble {X} is said to be

pseudo-random

if

$$\{X\}_{n\in\mathbb{N}} \approx \{U_n\}_{n\in\mathbb{N}}$$

Original goal



n-bits

10^{10} * n-bits

Pseudo-random generator

A function $G : \{0,1\}^n \to \{0,1\}^m$

is a pseudo-random generator if

an efficient algorithm G that.

① expands its input. i.e. $|G(x)| > |x|$

② its output is pseudo-random, i.e.

its output is comp. indistinguishable from uniformly random strings of the same length.

Pseudo-random generator

A function $G : \{0,1\}^n \rightarrow \{0,1\}^m$

is a pseudo-random generator if

G can be computed in p.p.t.

$|G(x)| > \ell(|x|)$ for some $\ell(y) > y$

$\{x \leftarrow U_n \ : \ G(x)\}_{n\in\mathbb{N}}$ is pseudo-random

How can we build pseudo-random generators?

$$\underbrace{7 \cdot 7 \cdots 7}_{x \text{ times}}$$

$5^{5} \bmod 167$

$5^1 \cdot 5^2 = 25 \cdot 5^3 = 125 \cdot 5^4 = 625$

$501 = 167 \cdot 3$

$124 \cdot 5$

620

119

$$7^x \bmod p$$

easy

$$X \xrightarrow[\text{Easy}]{f(x)} Y$$

Y = 12345

$$(7, p, Y) \rightarrow x$$

such that $7^{x} \bmod p = Y$

discrete logarithm problem

Incredibly hard!

World record in discrete logarithms in GF(p)

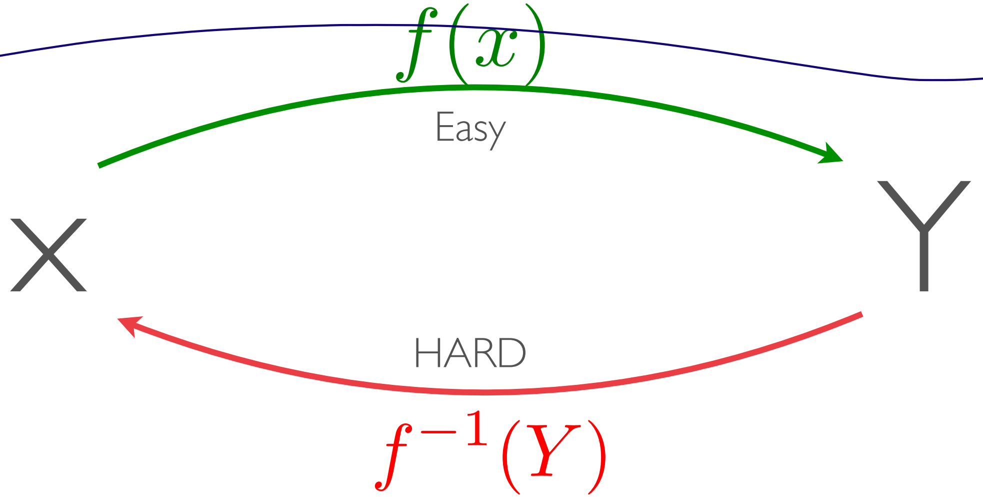
232 digits (768 bits)

6600yrs of CPU time

Intel Xeon E5-2660 at 2.2 GHz

2016

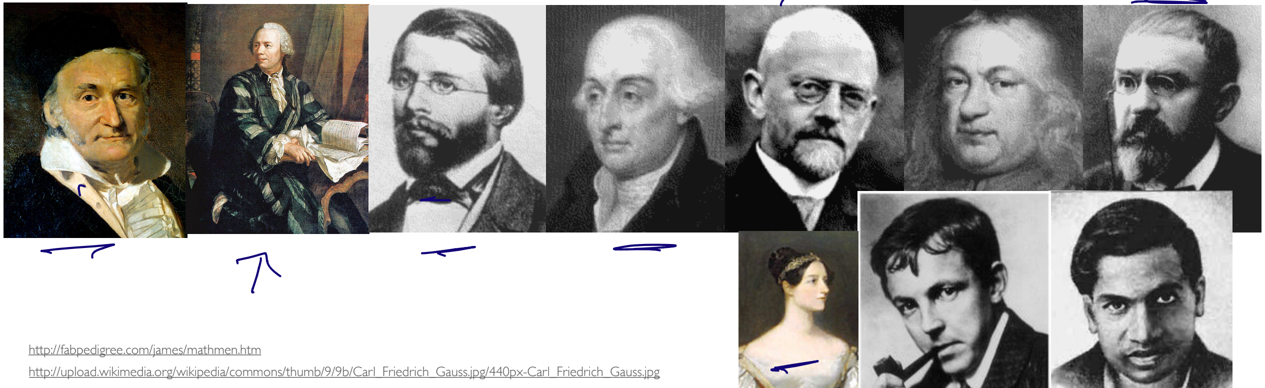
Thorsten Kleinjung, Claus Diem, Arjen K. Lenstra, Christine Priplata, and Colin Stahlke



compute an x such that $7^x = Y \bmod p$

One-way function. (one-way permutation)

$$\begin{tikzcd}[column sep=huge]
X \arrow[r, bend left, "f(x)", "\text{Easy}"'] & Y \arrow[l, bend left, "\text{HARD}"', "f^{-1}(Y)"]
\end{tikzcd}$$



http://fabpedigree.com/james/mathmen.htm

http://upload.wikimedia.org/wikipedia/commons/thumb/9/9b/Carl_Friedrich_Gauss.jpg/440px-Carl_Friedrich_Gauss.jpg

We have a one-way function

Blum-Micali Pseudo-random generator

PRG(s): parameters: $g, p \rightarrow$ n-bit modulus.

(1) compute $y = g^s \bmod p$.

$[\, s_0 \mid \; \mid \; \mid \; \mid \; \cdots \mid s_n \,]$ string s

$\downarrow$ compute

$[\, y_0 \mid \; \cdots \; \mid y_n \,]$ string y

(2) Output the string $\underbrace{y \,\|\, \left(s \overset{??}{<} \frac{p}{2}\right)}_{(\text{length } n+1)}$

$\rightarrow$ whether s is less than $p/2$ $\{0,1\}$

Blum-Micali Pseudo-random generator

PRG(s):

Input: s

1. Compute

$g^s \bmod p$

this is the out

2. Output all of the red bits and the first bit of s.

Blum-Micali Pseudo-random generator

$$f_{g,p}(x) = g^x \bmod p$$

$$b(x) = \begin{cases} 1 & x < (p-1)/2 \\ 0 & \text{o.w.} \end{cases}$$

$$G(s) = b(g^s \bmod p) \,|\, b(g^{g^s} \bmod p) \cdots$$

hard-core predicate.

Why is this secure?

Blum-Micali they show in a mathematical proof that "predicting <u>any</u> bit of the PRG output given only the prefix is as hard as solving the discrete log problem!!"

Why is this secure?

In this particular case, Blum-Micali *prove* that predicting the “next bit” of the output of this PRG is as hard as solving the hard problem from before: The discrete logarithm problem

But this PRG only expands 1 bit!

Let $G(s) : \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ be a prg.

$$X_0 \leftarrow s$$

$$X_i \,|\, b_i \leftarrow G(X_{i-1})$$

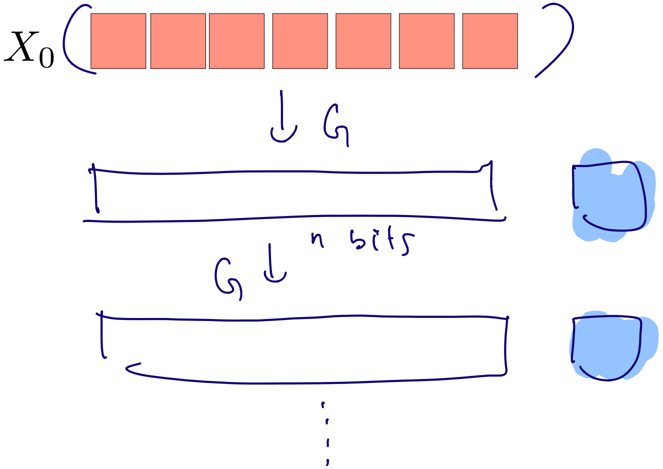
Output $b_1 b_2 \ldots b_{\ell(n)}$

Let $G(s) : \{0,1\}^n \to \{0,1\}^{n+1}$ be a prg.

$$X_0 \leftarrow s$$

$$X_i \mid b_i \leftarrow G(X_{i-1})$$

Output $b_1 b_2 \ldots b_{\ell(n)}$

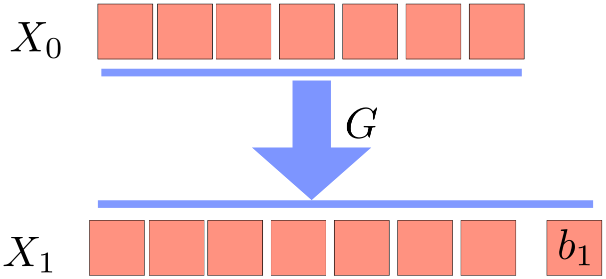


Let $G(s) : \{0,1\}^n \rightarrow \{0,1\}^{n+1}$ be a prg.

$$X_0 \leftarrow s$$

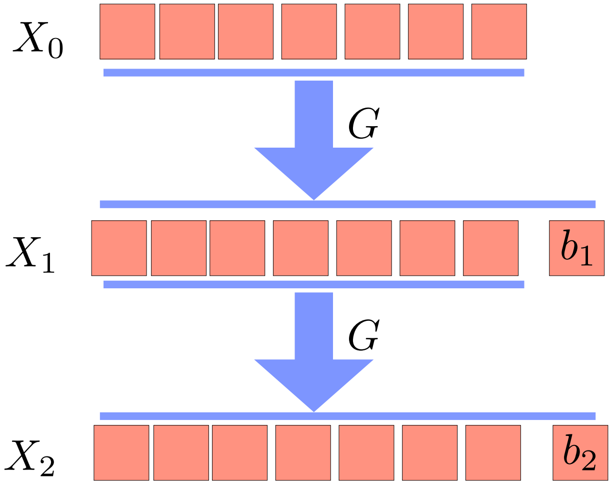
$$X_i \,|\, b_i \leftarrow G(X_{i-1})$$

Output $b_1 b_2 \dots b_{\ell(n)}$



Let $G(s) : \{0,1\}^n \to \{0,1\}^{n+1}$ be a prg.

$$X_0 \leftarrow s$$
$$X_i \mid b_i \leftarrow G(X_{i-1})$$
Output $b_1 b_2 \dots b_{\ell(n)}$



Example

Lets use g=5 and p=167.
These values are too small to be secure, but illustrate the scheme.

Pick the seed: $\boxed{s=11}$

BM generator(11):

first bit

① $g^{11} \bmod 167 = 5^{11} \bmod 167 = \underline{164}$

$11 < \frac{167}{2}$. Yes $\longrightarrow$ $\boxed{1}$

② $g^{164} = 5^{164} \bmod 167 = \underline{147}$

$164 < \frac{167}{2} = $ No $\longrightarrow$ $\boxed{0}$

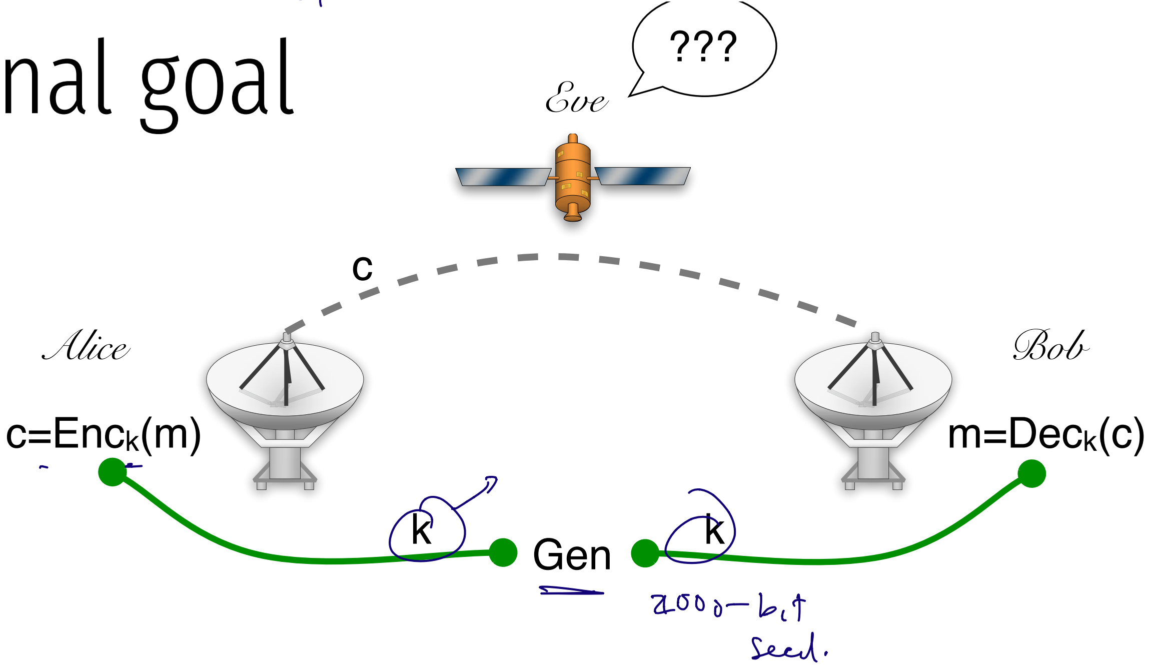
③ $5^{147} \bmod 167 = 51$

$147 < \frac{167}{2} = $ No $\longrightarrow$ $\boxed{0}$

④ $5^{51} \bmod 167 = 161$

$51 < 167/2 = $ $\longrightarrow$ $\boxed{1}$

Original goal



"for any pair of messages m_1,m_2,
Eve cannot tell whether $c = Enc_k(m_i)$."

review

private key encryption

Gen Enc Dec $\mathcal{M}$ $\mathcal{K}$

3 algorithms 2 sets

Gen (key generation)

$$k \leftarrow \mathsf{Gen}(1^n) \text{ s.t. } k \in \mathcal{K}$$

Enc (encryption)

$$c \leftarrow \mathsf{Enc}_k(m) \text{ for } k \in \mathcal{K}, m \in \mathcal{M}$$

Dec (decryption)

$$\forall m \in \mathcal{M}, k \in \mathcal{K}$$

$$\Pr[\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m] = 1$$

perfect secrecy

$$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathcal{M}, \mathcal{K})$$

is said to be perfectly secret if

$$\forall m_1, m_2 \in \mathcal{M} \text{ s.t. } |m_1| = |m_2|, \forall c$$

$$\Pr[k \leftarrow \mathsf{Gen} \;:\; \mathsf{Enc}_k(m_1) = c]$$

$$=$$

$$\Pr[k \leftarrow \mathsf{Gen} \;:\; \mathsf{Enc}_k(m_2) = c]$$

perfect secrecy

$$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathcal{M}, \mathcal{K})$$

is said to be perfectly secret if

$$\forall m_1, m_2 \in \mathcal{M} \text{ s.t. } |m_1| = |m_2|, \forall c$$

$$=$$

perfect secrecy

$$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathcal{M}, \mathcal{K})$$

is said to be perfectly secret if

$$\forall m_1, m_2 \in \mathcal{M} \text{ s.t. } |m_1| = |m_2|, \forall c$$

$$\{k \leftarrow \text{Gen}(1^n) \ : \ Enc_k(m_1)\}$$

$$=$$

$$\{k \leftarrow \text{Gen}(1^n) \ : \ Enc_k(m_2)\}$$

secure encryption

(For one message)

Def:

computational secrecy

$$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathcal{M}, \mathcal{K})$$

is said to be computationally secure if

$$\forall m_1, m_2 \in \mathcal{M} \text{ s.t. } |m_1| = |m_2|, \forall c$$

$$\{k \leftarrow \mathrm{Gen}(1^n) \ : \ Enc_k(m_1)\}$$

$$\approx$$ computationally indistinguishable

$=$

$$\{k \leftarrow \mathrm{Gen}(1^n) \ : \ Enc_k(m_2)\}$$

An encryption scheme

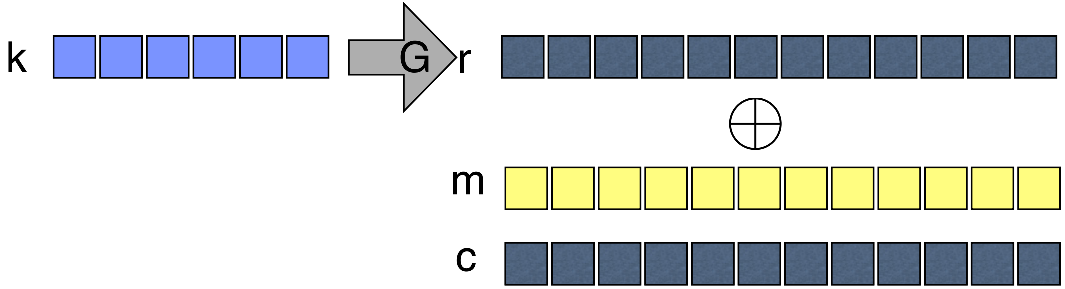
An encryption scheme

Gen(1^n) $\quad k \leftarrow U_{n/2}$ (key generation)

Enc$_k$(m) $\quad r \leftarrow G(k) \quad |r| = n$ (encryption)

output $m \oplus r$

Dec$_k$(c) (decryption)



What are the pros/cons of this scheme?

+ (1) short key. msg can be very long.

− (2) one modular exponentiation per bit of msg

− (3) traded perfect security for computational security

! cybersecurity: Evaluate the tradeoffs