COS433/Math 473: Cryptography

Mark Zhandry

Princeton University

Spring 2020

Reminders

HW1 Due Feb 20^{th}
HW2 Due Feb 27^{th}

PR1 Due March 10^{th}

Previously on COS 433…

Theorem: No stateless *randomized* encryption scheme can have perfect security for multiple messages

Security Parameter **λ**

Additional input to system, dictates “security level”

Key, message, ciphertext size all **polynomial** in **λ**

Probability of adversary success is **negligible** in **λ**

Defining Encryption Again

Syntax:

- Key space $\mathbf{K_\lambda}$
- Message space $\mathbf{M_\lambda}$
- Ciphertext space $\mathbf{C_\lambda}$
- **Enc:** $\mathbf{K_\lambda \times M_\lambda \rightarrow C_\lambda}$ (potentially randomized)
- **Dec:** $\mathbf{K_\lambda \times C_\lambda \rightarrow M_\lambda}$

Correctness:

- $\mathbf{|k|=\log|K_\lambda|,\ |m|=\log|M_\lambda|,\ |c|=\log|C_\lambda|}$ polynomial in $\boldsymbol{\lambda}$
- For all $\boldsymbol{\lambda}$, $\mathbf{k \in K_\lambda,\ m \in M_\lambda}$,

$$\mathbf{Pr[\ Pr[Dec(k,\ Enc(k,m)\ ) = m\ ] = 1}$$

Statistical Distance

Given two distributions D_1, D_2 over a set X, define

$$\Delta(D_1,D_2) = ½\Sigma_x \mid Pr[D_1=x] - Pr[D_2=x] \mid$$

Observations:

$$0 \leq \Delta(D_1,D_2) \leq 1$$

$$\Delta(D_1,D_2) = 0 \iff D_1 \stackrel{d}{=} D_2$$

$$\Delta(D_1,D_2) \leq \Delta(D_1,D_3) + \Delta(D_3,D_2)$$

(**Δ** is a metric)

Another View of Statistical Distance

Theorem: $\Delta(D_1,D_2) \geq \varepsilon$ iff $\exists$ (potentially randomized) A s.t.

$$| \Pr[A(D_1) = 1] - \Pr[A(D_2) = 1] | \geq \varepsilon$$

Terminology: for any A,

$$| \Pr[A(D_1) = 1] - \Pr[A(D_2) = 1] |$$

is called the "advantage" of A in distinguishing D_1 and D_2

Statistical Security (Asymptotic)

Definition: A scheme **(Enc,Dec)** has **statistical secrecy for d messages** if $\exists$ negligible **ε** such that $\forall$ two sequences $(m_0^{(i)})_{i\in[d]}$, $(m_1^{(i)})_{i\in[d]}$ $\in$ M_λ^d,

$$\Delta\left[\left(Enc(K_\lambda, m_0^{(i)})\right)_{i\in[d]}, \left(Enc(K_\lambda, m_1^{(i)})\right)_{i\in[d]} \right] < \varepsilon(\lambda)$$

We will call such a scheme **d**-time statistically secure

Limits of Statistical Security

Theorem: Suppose **(Enc,Dec)** has plaintext space $\mathbf{M} = \{0,1\}^n$ and key space $\mathbf{K} = \{0,1\}^t$. Moreover, assume it is **(d, 0.4999)**-secure. Then:

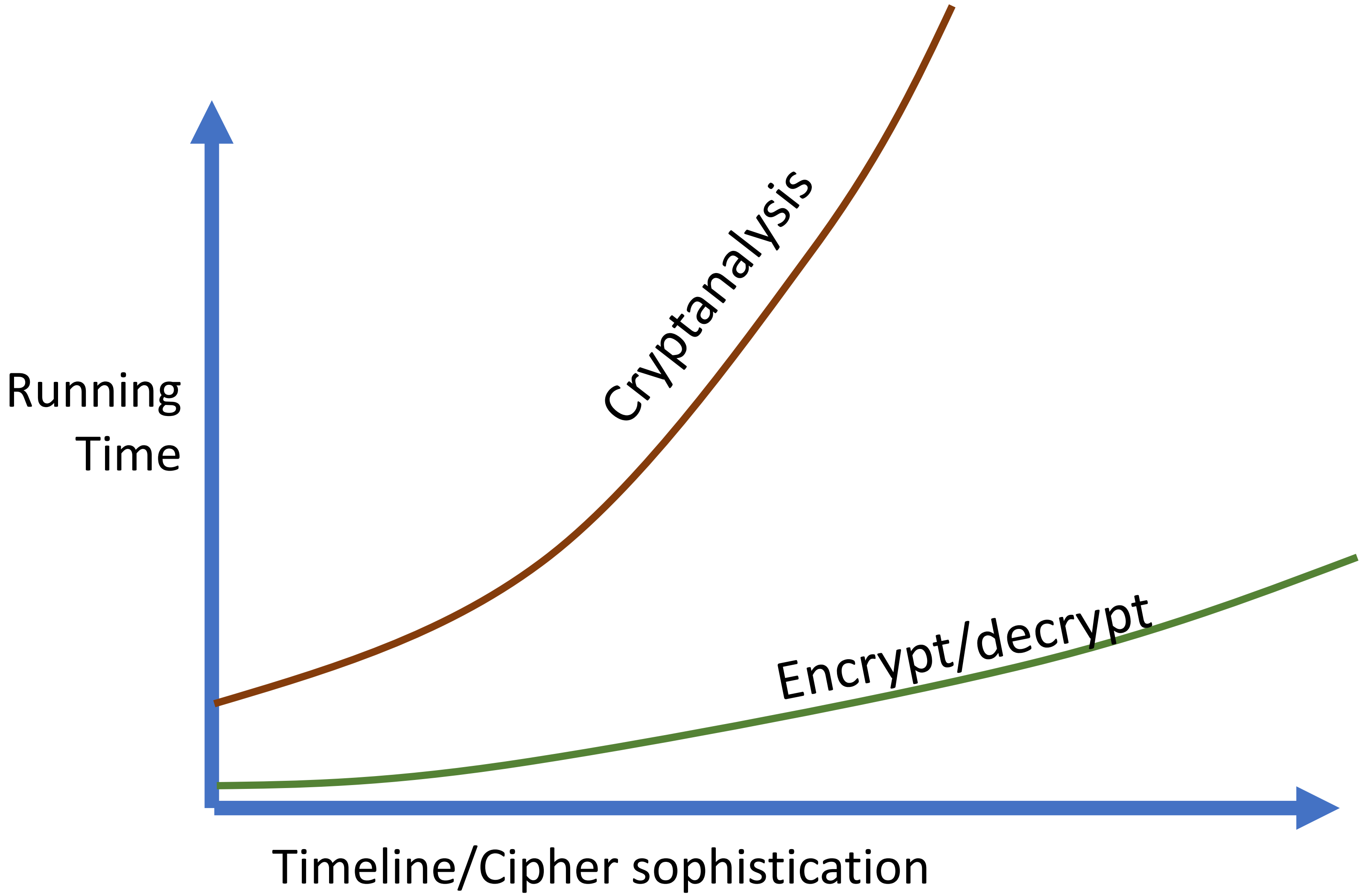
$$t \geq d\ n$$

In other words, the key must be at least as long as the total length of all messages encrypted

Takeaway

If you don’t want to physically exchange keys frequently, you cannot obtain statistical security

So, now what?


Running Time
Cryptanalysis
Encrypt/decrypt
Timeline/Cipher sophistication

Computational Security

We are ok if adversary takes a really long time

Only considered attack for adversaries that don’t take too long

Today: Continuation of Computational Security

Brute Force Attacks

Simply try every key until find right one

If keys have length **λ**, $\mathbf{2}^{\lambda}$ is upper bound on attack

Applicable when easy to check if key is correct

- In case of perfect/statistical security, not possible

Crypto and P vs NP

What if P = NP?

From this point forward, almost all crypto we will see depends on computational assumptions

Defining Encryption Yet Again

Syntax:

- Key space $\mathbf{K_\lambda}$
- Message space $\mathbf{M_\lambda}$
- Ciphertext space $\mathbf{C_\lambda}$
- $\mathbf{Enc:\ K_\lambda \times M_\lambda \rightarrow C_\lambda}$ (potentially randomized)
- $\mathbf{Dec:\ K_\lambda \times C_\lambda \rightarrow M_\lambda}$

Correctness:

- $\mathbf{|k|=|K_\lambda|,\ |m|=|M_\lambda|,\ |c|=|C_\lambda|}$ polynomial in $\boldsymbol{\lambda}$
- **Enc, Dec** running time polynomial in $\boldsymbol{\lambda}$
- For all $\boldsymbol{\lambda}$, $\mathbf{k \in K_\lambda,\ m \in M_\lambda}$,

$$\mathbf{Pr[\ Pr[Dec(k,\ Enc(k,m)\ ) = m\ ] = 1}$$

Defining Security

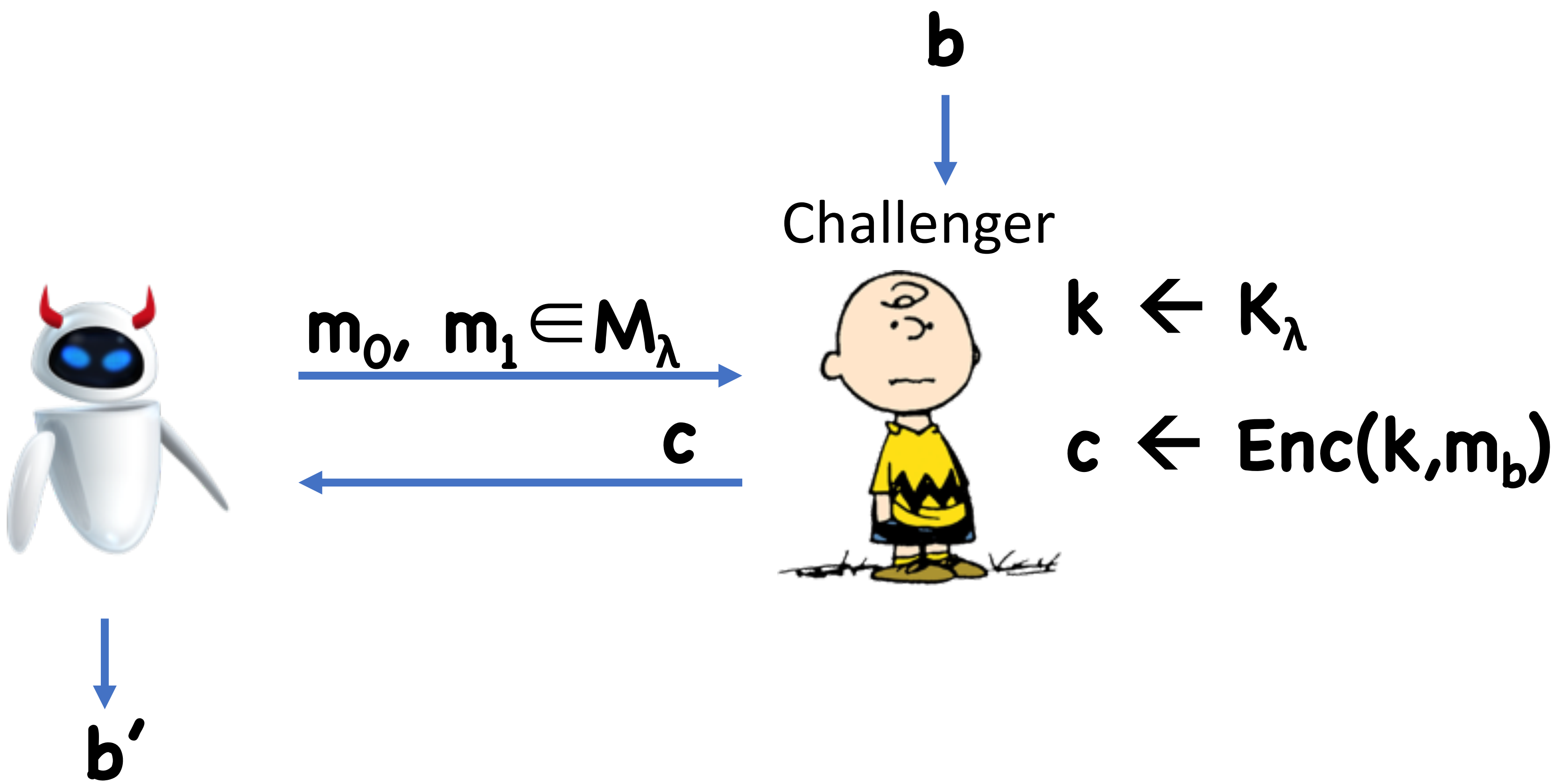
Consider an attacker as a probabilistic efficient algorithm

Attacker gets to choose the messages

All attacker has to do is distinguish them

Security Experiment/Game

(One-time setting)



$\text{IND-Exp}_b($  **$,\lambda)$**

Security Definition

(One-time setting, concrete)

Definition: (Enc, Dec) has **(t,ε)-ciphertext indistinguishability** if, for all  running in time at most **t**

| $\Pr[1 \leftarrow \text{IND-Exp}_0($  $)]$

$- \Pr[1 \leftarrow \text{IND-Exp}_1($  $)]$ | $\le \varepsilon$

Security Definition

(One-time setting, asymptotic)

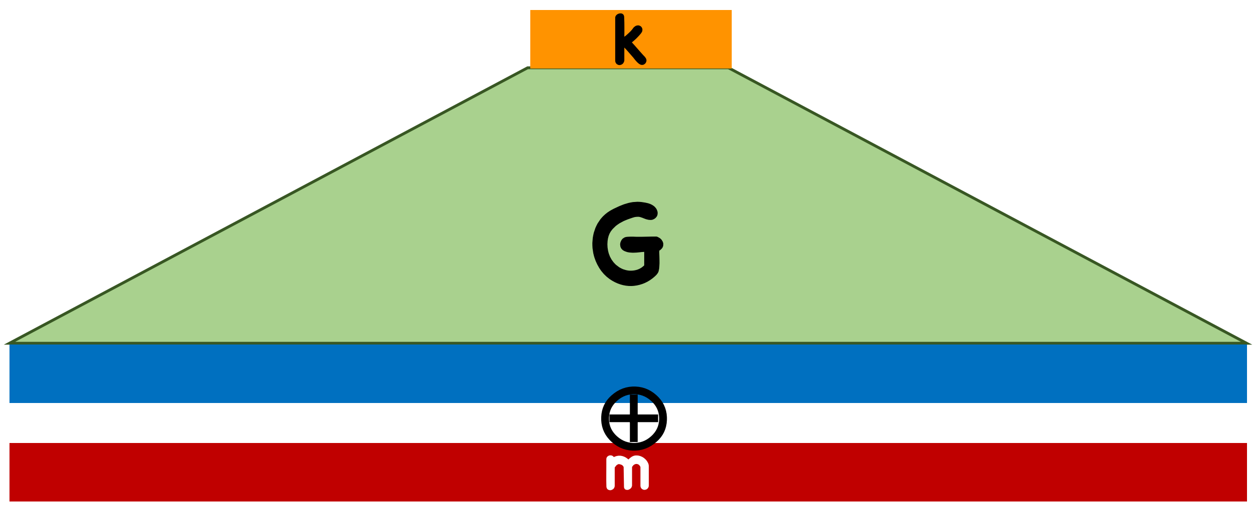
Definition: (Enc, Dec) has **ciphertext indistinguishability** if, for all  running in polynomial time, $\exists$ negligible ε s.t.

| Pr[1←IND-Exp$_0$( **,λ)]**

− Pr[1←IND-Exp$_1$( **,λ)] | ≤ ε(λ)**

Construction with $|k| << |m|$

Idea: use OTP, but have key generated by some expanding procedure **G**



What do we want out of **G**?

Defining Pseudorandom Generator (PRG)

Syntax:

- Seed space $\mathbf{S}_\lambda$
- Output space $\mathbf{X}_\lambda$
- $\mathbf{G}$: $\mathbf{S}_\lambda \rightarrow \mathbf{X}_\lambda$ (deterministic)

Correctness:

- $|s|=\log|S_\lambda|$, $|x|=\log|X_\lambda|$ polynomial in $\boldsymbol{\lambda}$,
- $|X_\lambda| > 2\times|S_\lambda|$
- Running time of $\mathbf{G}$ polynomial in $\boldsymbol{\lambda}$

Security of PRGs

Definition: $\mathbf{G{:}S_\lambda \rightarrow X_\lambda}$ is a **secure pseudorandom generator** (PRG) if:

- For all  running in polynomial time, $\exists$ negl **ε,**

$\Big|$ **Pr[**  $\mathbf{(G(s))=1 : s \leftarrow S_\lambda]}$

$\quad$ **– Pr[**  $\mathbf{(x)=1 : x \leftarrow X_\lambda]} \Big| \leq \boldsymbol{\varepsilon(\lambda)}$

Secure PRG → Ciphertext Indistinguishability

$K_\lambda = S_\lambda$
$M_\lambda = X_\lambda$ (assumed to be $\{0,1\}^n$)
$C_\lambda = X_\lambda$

$Enc(k,m) = PRG(k) \oplus m$
$Dec(k,c) = PRG(k) \oplus c$

Security?

Intuitively, security is obvious:

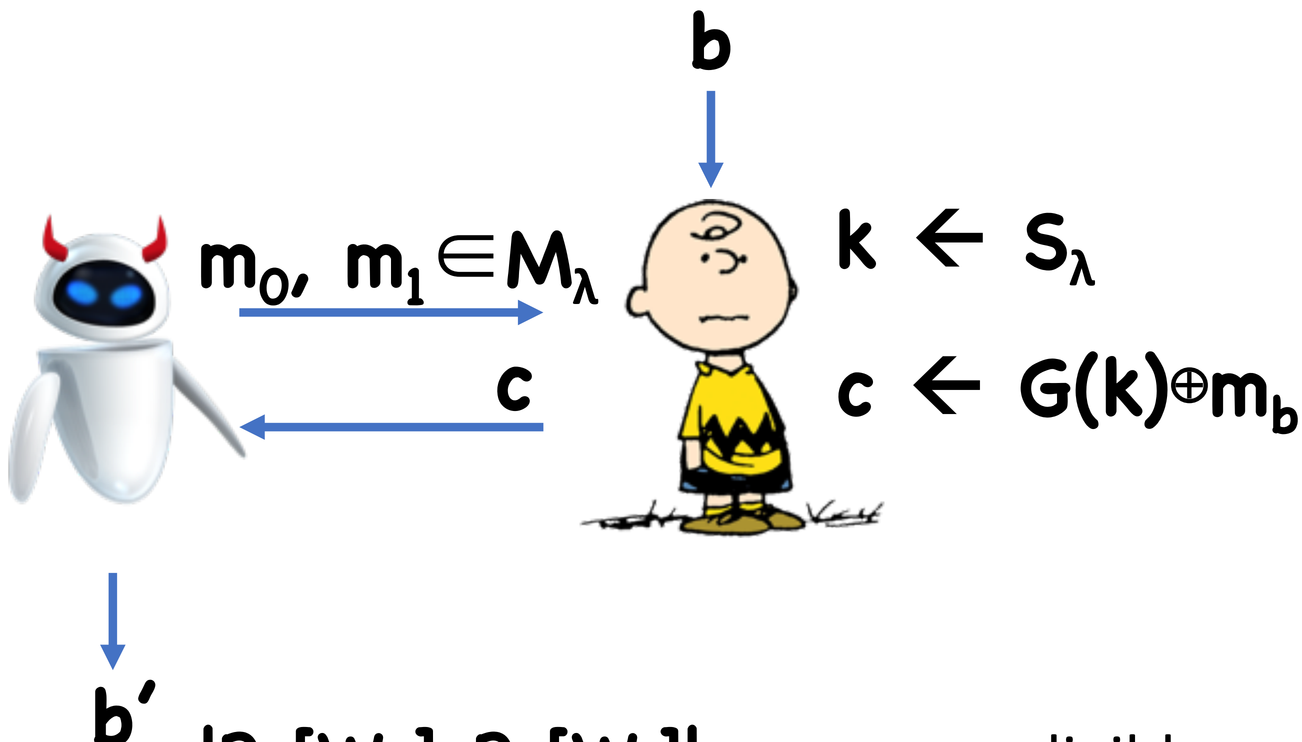
- **PRG(k)** ”looks” random, so should completely hide **m**
- However, formalizing this argument is non-trivial.

Solution: reductions

- Assume toward contradiction an adversary for the encryption scheme, derive an adversary for the PRG

Security

Assume towards contradiction that there is a  and non-negligible **ε** such that



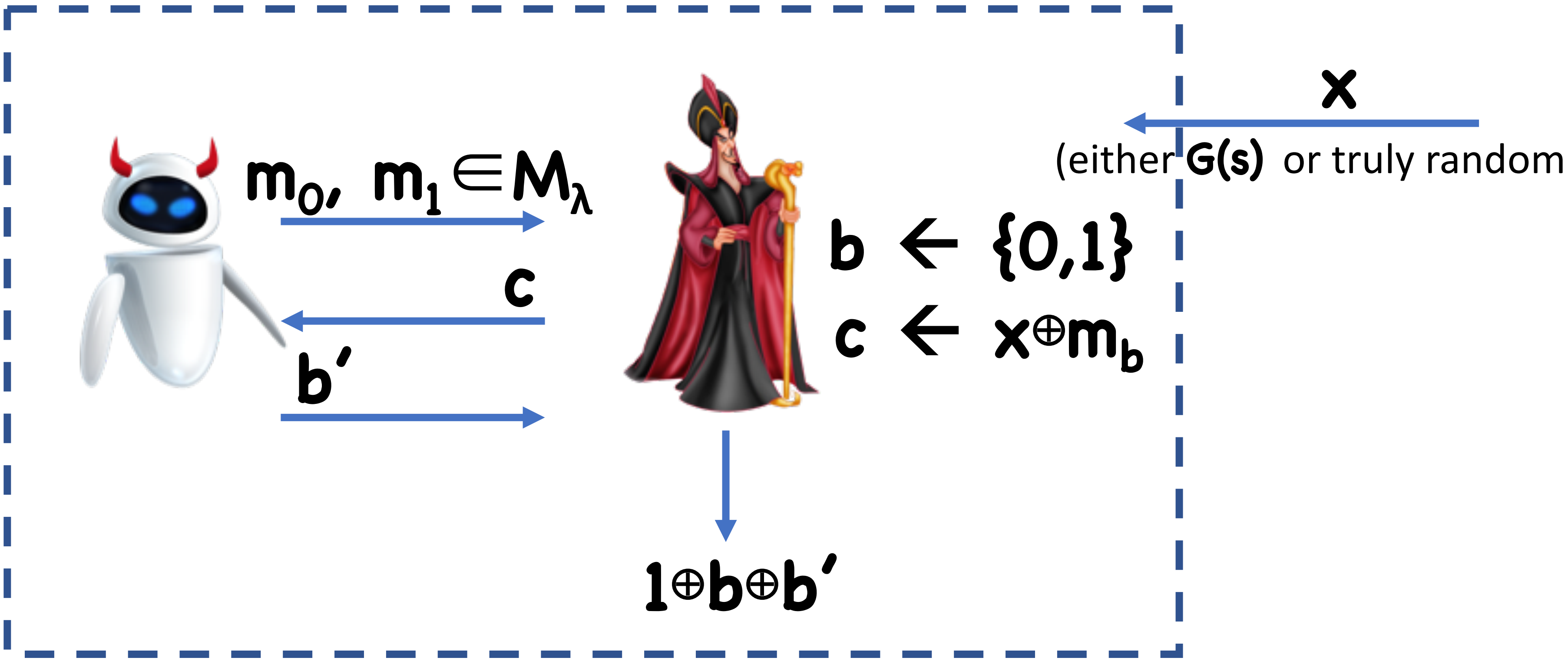
$|Pr[W_0] - Pr[W_1]| \geq \varepsilon$, non-negligible

W_b: $b' = 1$ in IND-Exp_b

Security

Use  to build .  will run  as a subroutine,

and pretend to be 



Security

Case 1: $\mathbf{x = PRG(s)}$ for a random seed $\mathbf{s}$

-  "sees" $\mathbf{IND\text{-}Exp_b}$ for a random bit $\mathbf{b}$

$m_0, m_1 \in M_\lambda$ →

← c

↓ b'

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

$s \leftarrow S_\lambda$

$c \leftarrow PRG(s) \oplus m_b$

Security

Case 1: $\mathbf{x = PRG(s)}$ for a random seed $\mathbf{s}$

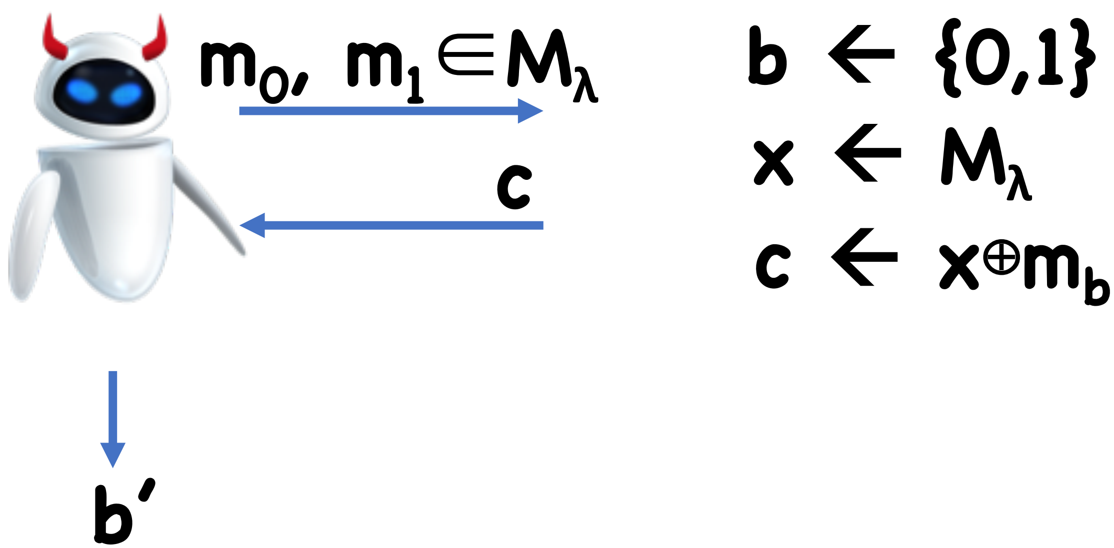
-  "sees" $\mathbf{IND\text{-}Exp_b}$ for a random bit $\mathbf{b}$
- $\mathbf{Pr[1 \oplus b \oplus b' = 1] = Pr[b = b']}$

$$\begin{aligned} &= \tfrac{1}{2}\,\Pr[b'=1 \mid b=1\,] \\ &\qquad + \tfrac{1}{2}\,(1 - \Pr[b'=1 \mid b=0]) \\ &= \tfrac{1}{2}(1 + \Pr[W_0] - \Pr[W_1]) \\ &= \tfrac{1}{2}(\,1 \pm \varepsilon\,) \end{aligned}$$

Security

Case 2: **x** is truly random

-  "sees" OTP encryption



Security

Case 2: $\mathbf{x}$ is truly random

-  "sees" OTP encryption
- Therefore $\Pr[b'=1 \mid b=0] = \Pr[b'=1 \mid b=1]$
- $\Pr[1 \oplus b \oplus b'=1] = \Pr[b=b']$

$$= \tfrac{1}{2} \Pr[b'=1 \mid b=1]$$

$$+ \tfrac{1}{2} (1 - \Pr[b'=1 \mid b=0])$$

$$= \tfrac{1}{2}$$

Security

Putting it together:

- $\Pr[$  $(G(s))=1 : s \leftarrow \{0,1\}^{\lambda}] = \frac{1}{2}(1 \pm \varepsilon(\lambda))$

- $\Pr[$  $(x)=1 : x \leftarrow \{0,1\}^{n}] = \frac{1}{2}$

- Absolute Difference: $\frac{1}{2}\varepsilon$, $\Rightarrow$ Contradiction!

Security

Thm: If **G** is a secure PRG, then **(Enc,Dec)** is has ciphertext indistinguishability

An Alternate Proof: Hybrids

Idea: define sequence of "hybrid" experiments "between" $\textbf{IND-Exp}_0$ and $\textbf{IND-Exp}_1$

In each hybrid, make small change from previous hybrid

Hopefully, each small change is undetectable

Using triangle inequality, overall change from $\textbf{IND-Exp}_0$ and $\textbf{IND-Exp}_1$ is undetectable

An Alternate Proof: Hybrids

Hybrid 0: IND-Exp$_0$

$m_0, m_1 \in M_\lambda$ →

← c

↓ b'

$k \leftarrow S_\lambda$

$c \leftarrow G(k) \oplus m_0$

An Alternate Proof: Hybrids

Hybrid 1:

$m_0, m_1 \in M_\lambda$ →

$x \leftarrow M_\lambda$

← c

$c \leftarrow x \oplus m_0$

↓

b'

An Alternate Proof: Hybrids

Hybrid 2:

$m_0, m_1 \in M_\lambda$ →

← c

↓ b'

$x \leftarrow M_\lambda$

$c \leftarrow x \oplus m_1$

An Alternate Proof: Hybrids

Hybrid 3: IND-Exp$_1$

$m_0, m_1 \in M_\lambda$ →

← c

↓ b'

$k \leftarrow S_\lambda$

$c \leftarrow G(k) \oplus m_1$

An Alternate Proof: Hybrids

| Pr[b'=1 : IND-Exp$_0$]-Pr[b'=1 : IND-Exp$_1$] |

= | Pr[b'=1 : Hyb 0]-Pr[b'=1 : Hyb 3] |

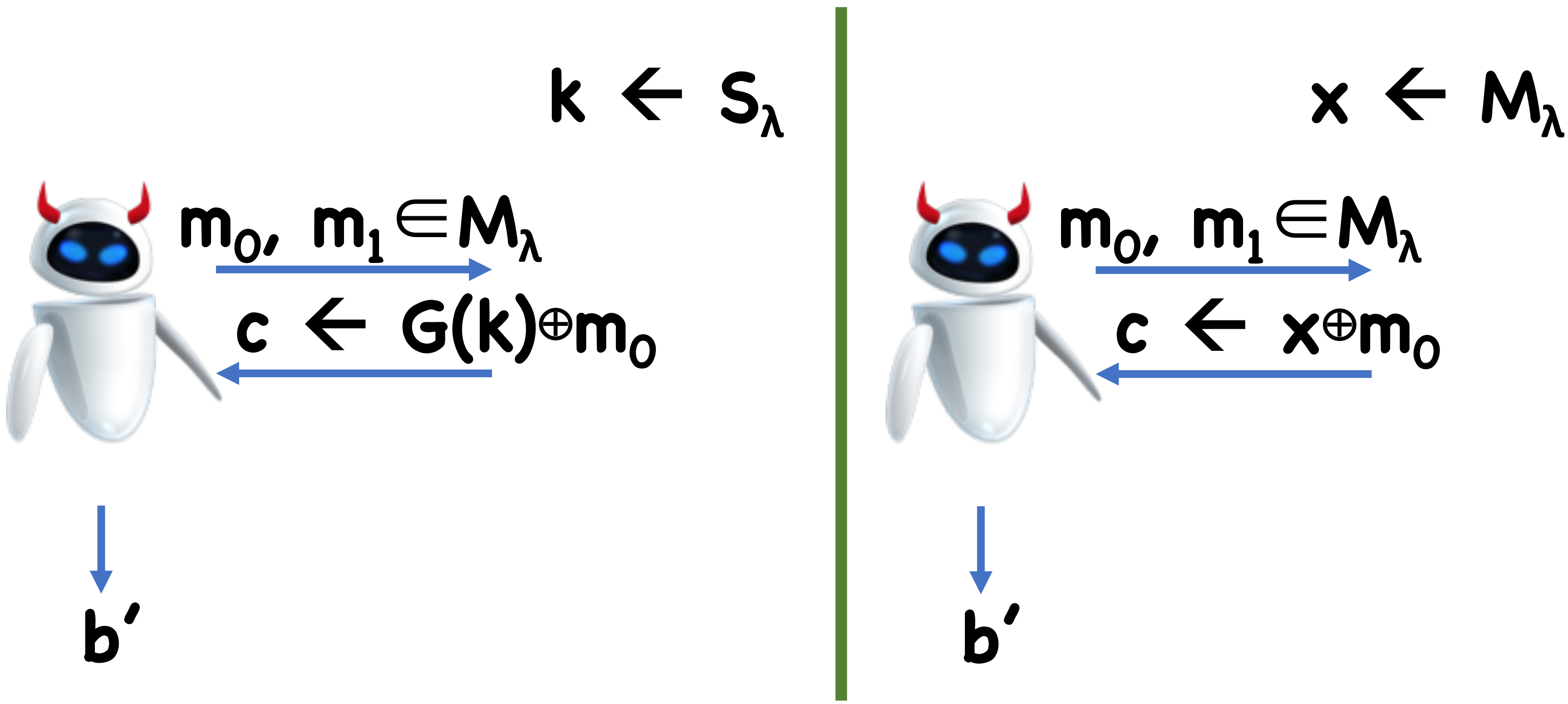
≤ | Pr[b'=1 : Hyb 0]-Pr[b'=1 : Hyb 1] |
+ | Pr[b'=1 : Hyb 1]-Pr[b'=1 : Hyb 2] |
+ | Pr[b'=1 : Hyb 2]-Pr[b'=1 : Hyb 3] |

If **|Pr[b'=1:IND-Exp$_0$]-Pr[b'=1:IND-Exp$_1$]|≥ε,**
Then for some **i=0,1,2,**

|Pr[b'=1:Hyb i]-Pr[b'=1:Hyb i+1]| ≥ ε/3

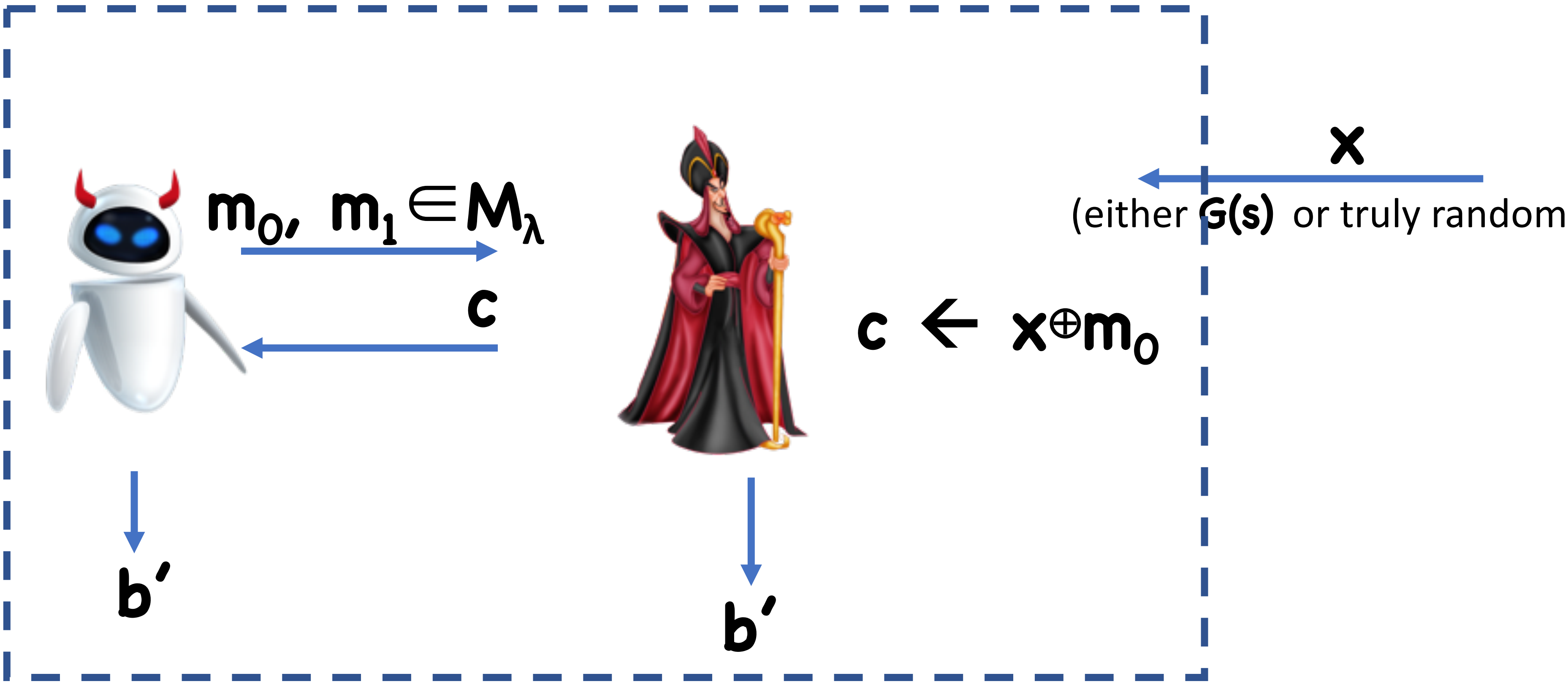
An Alternate Proof: Hybrids

Suppose  distinguishes **Hybrid 0** from **Hybrid 1** with advantage **ε/3**



An Alternate Proof: Hybrids

Suppose  distinguishes **Hybrid 0** from **Hybrid 1** with advantage **ε/3** $\Rightarrow$ Construct 



An Alternate Proof: Hybrids

Suppose  distinguishes **Hybrid 0** from **Hybrid 1** with advantage **ε/3** ⇒ Construct 

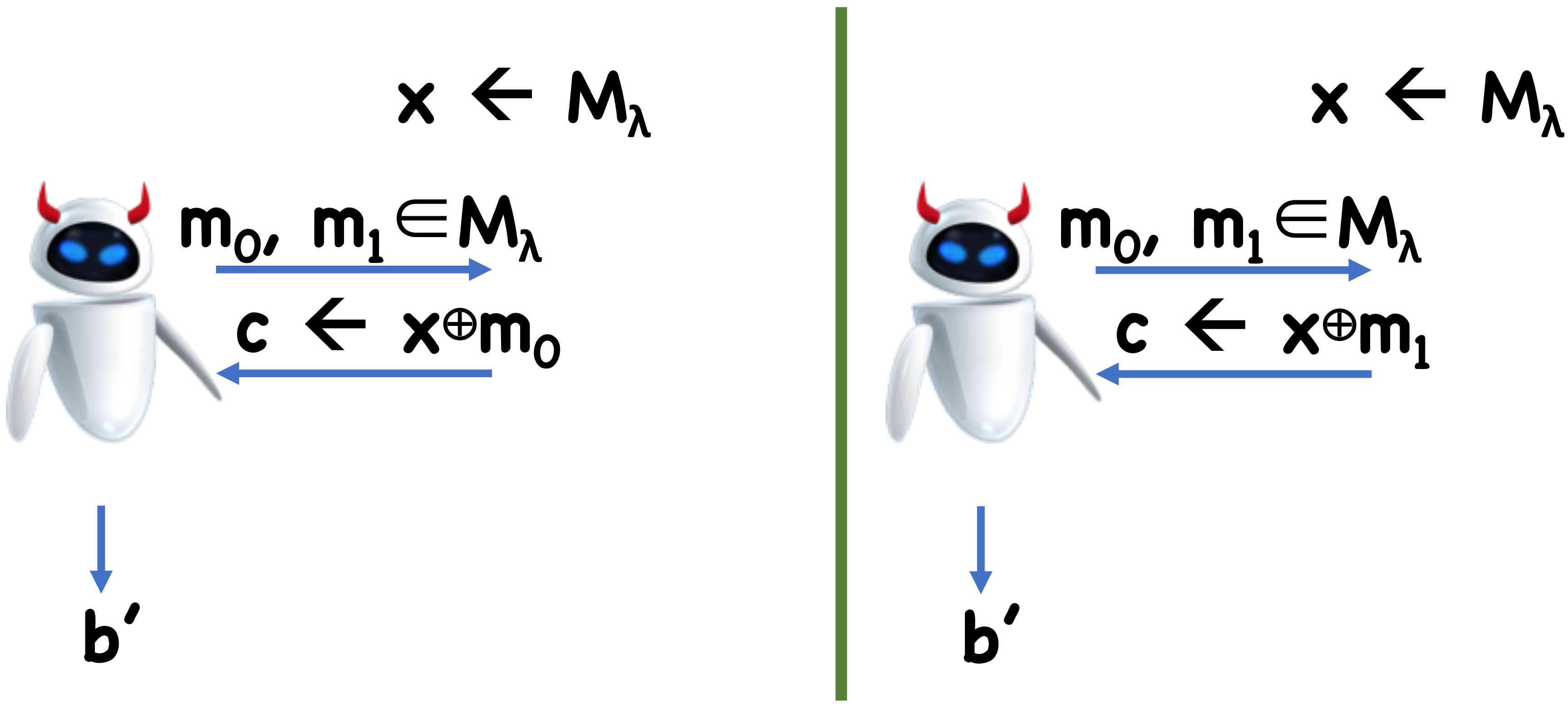
If  is given **G(s)** for a random **s**,  sees **Hybrid 0**

If  is given x for a random **x**,  sees **Hybrid 1**

Therefore, advantage of  is equal to advantage of  which is at least **ε/3** ⇒ Contradiction!

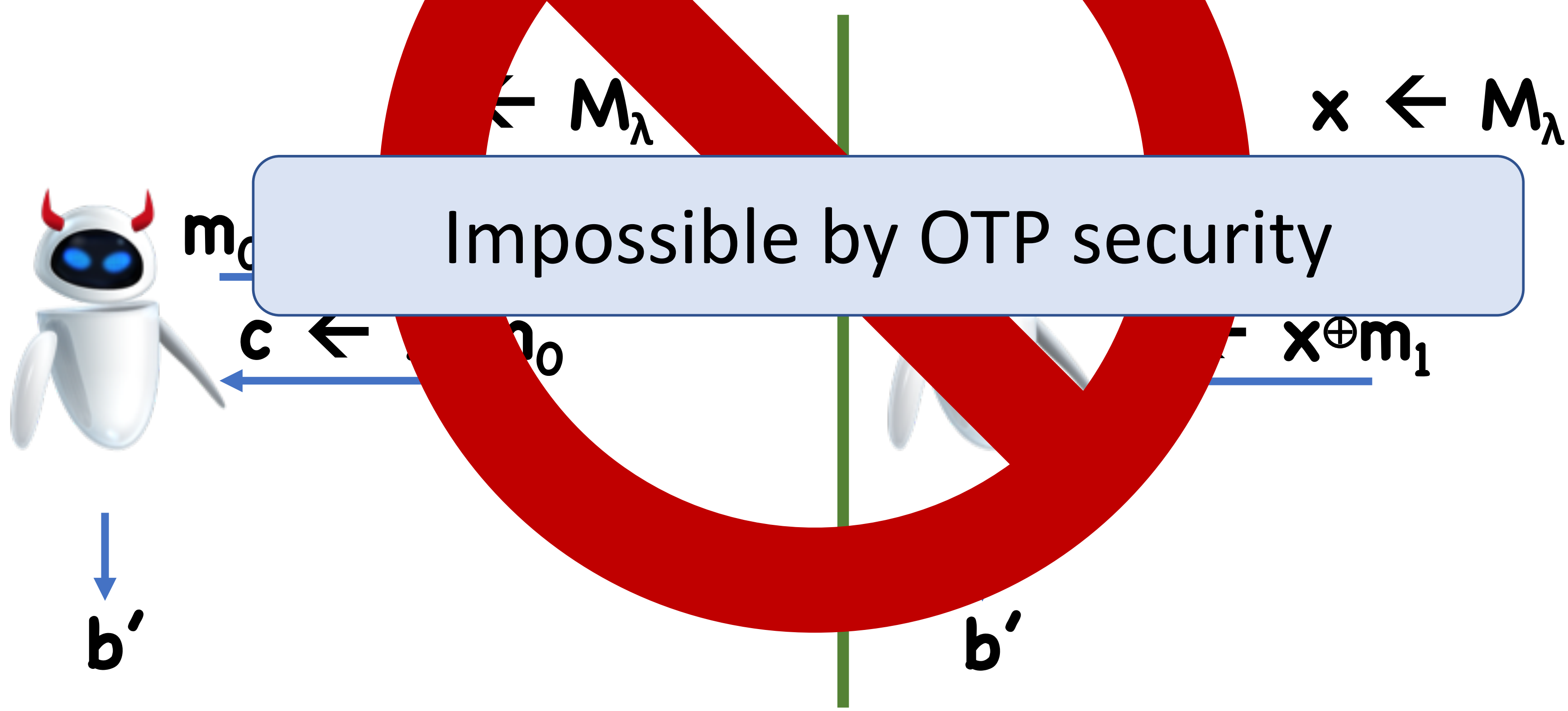
An Alternate Proof: Hybrids

Suppose  distinguishes **Hybrid 1** from **Hybrid 2** with advantage **ε/3**



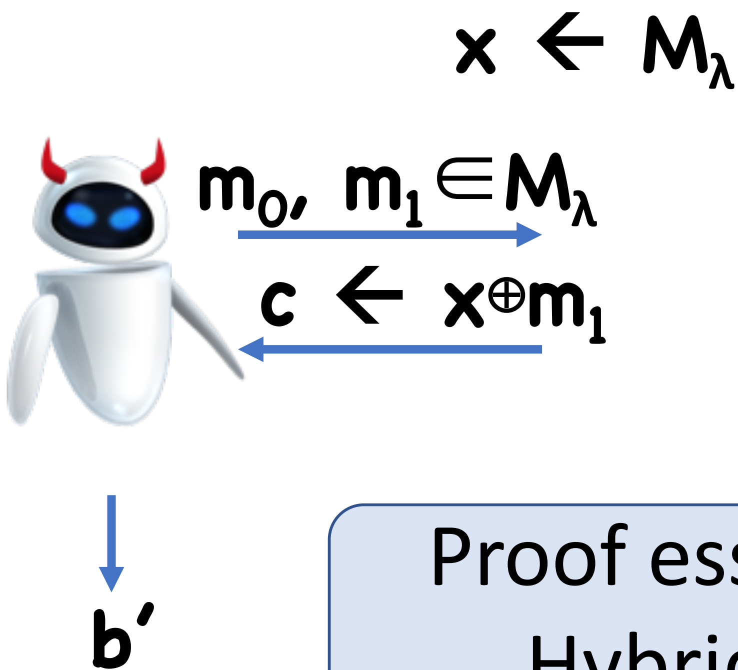
An Alternate Proof: Hybrids

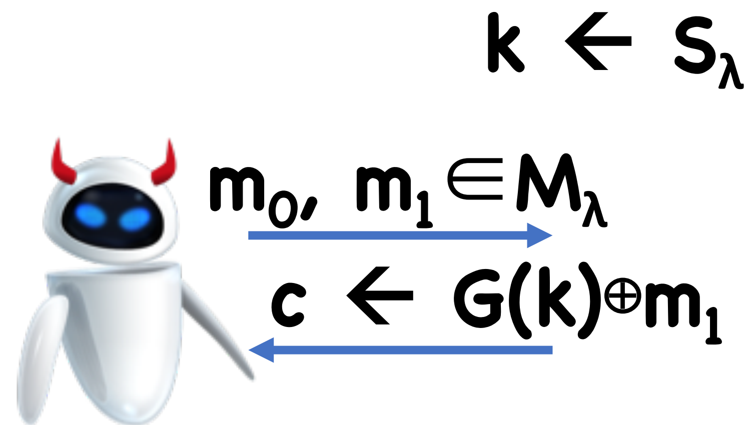
Suppose  disti… …1 from **Hybrid 2** with advantag… **…/3**



An Alternate Proof: Hybrids

Suppose  distinguishes **Hybrid 2** from **Hybrid 3** with advantage **ε/3**





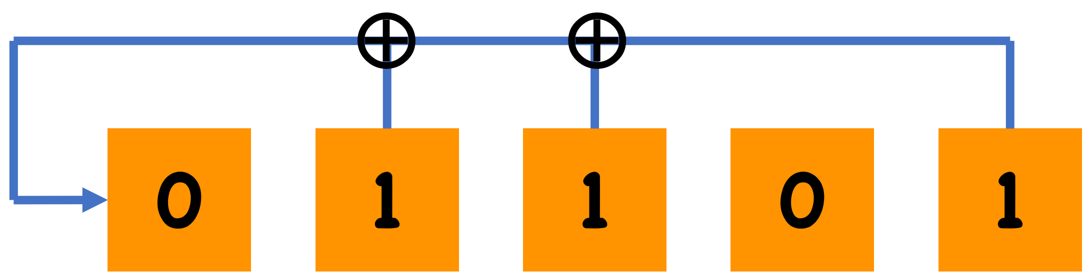
Proof essentially identical to Hybrid 0/Hybrid 1 case

How do we build PRGs?

Linear Feedback Shift Registers

In each step,

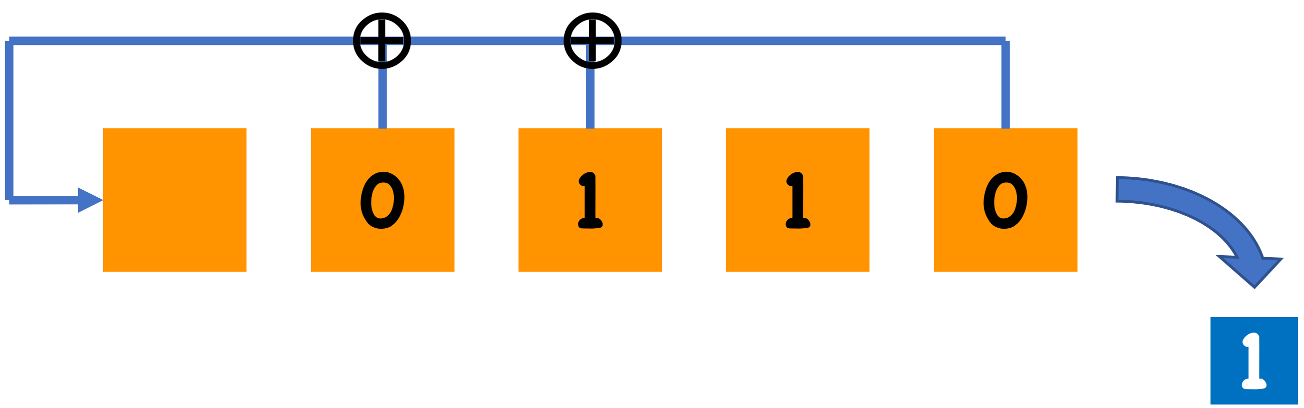
- Last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

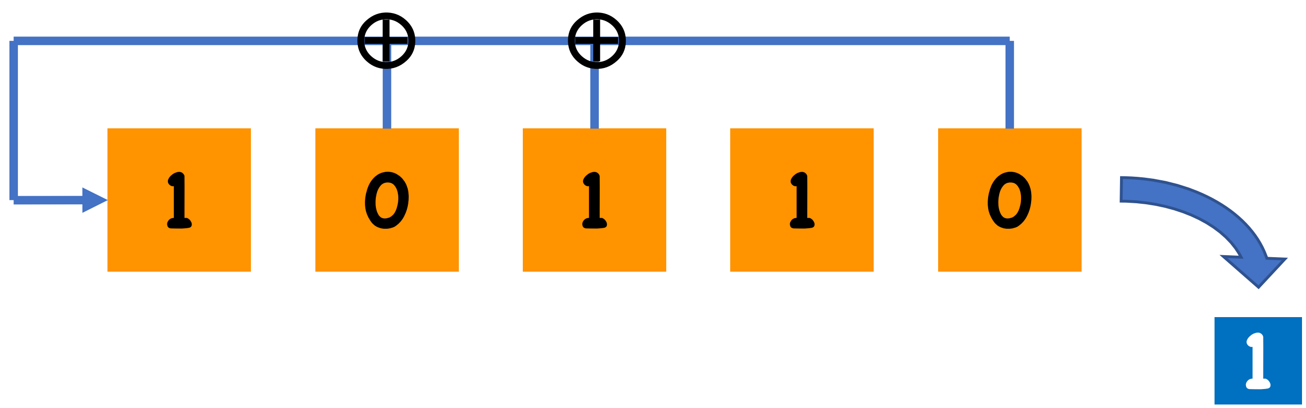
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

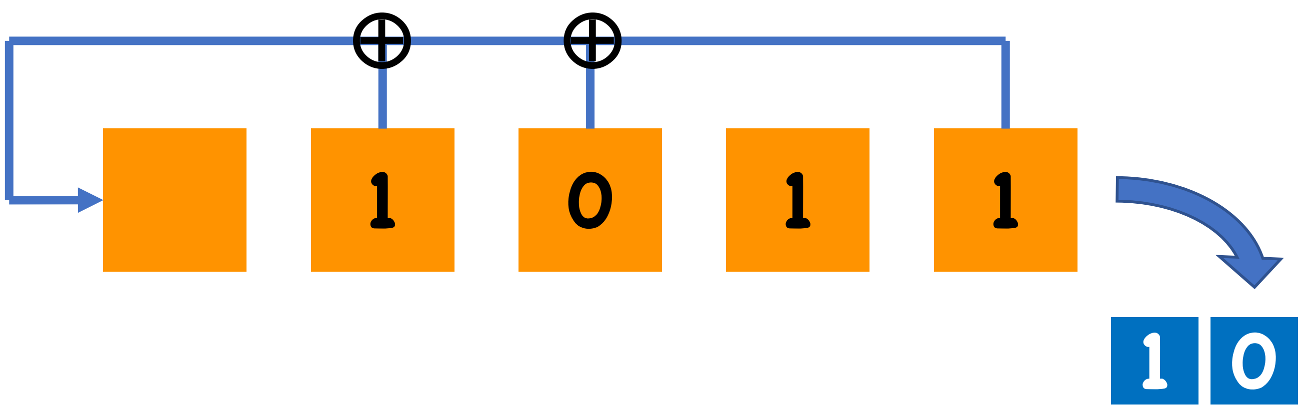
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

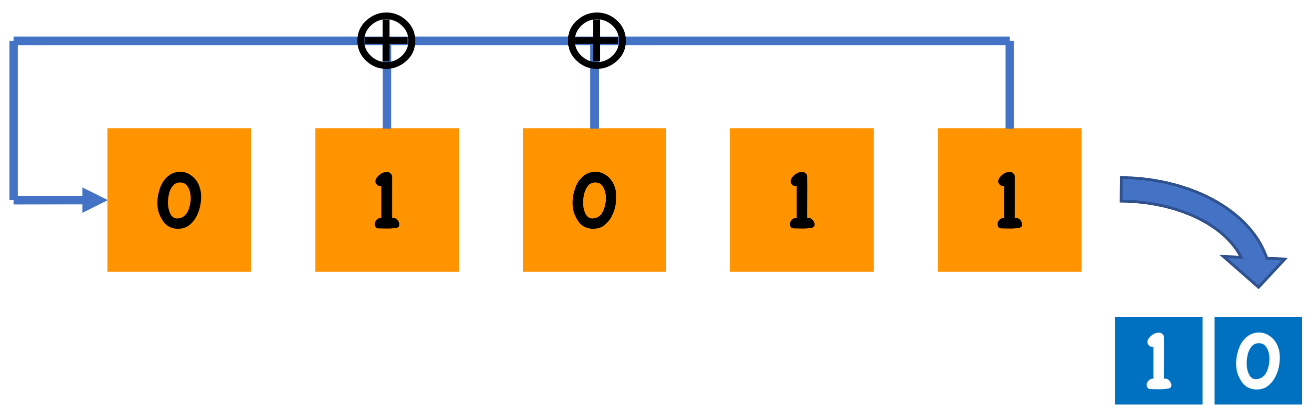
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

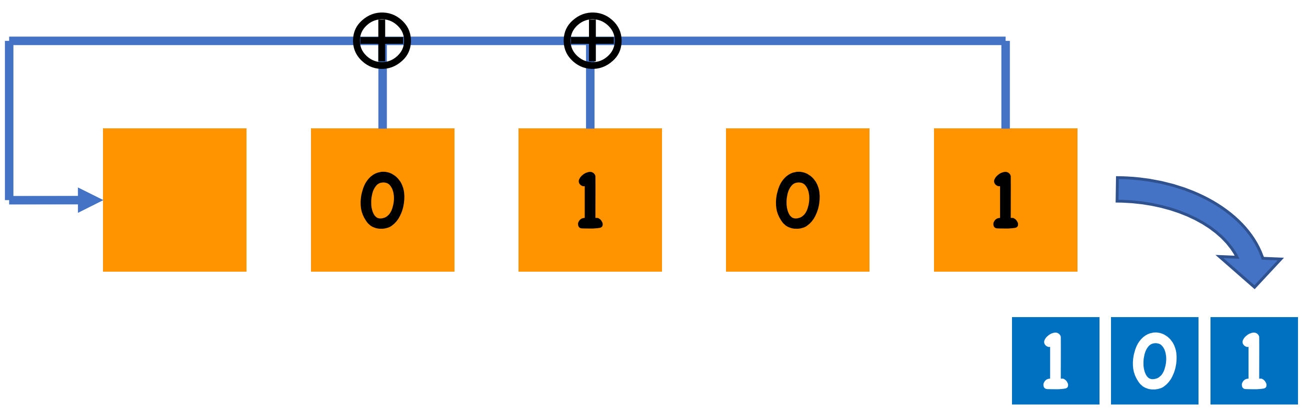
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

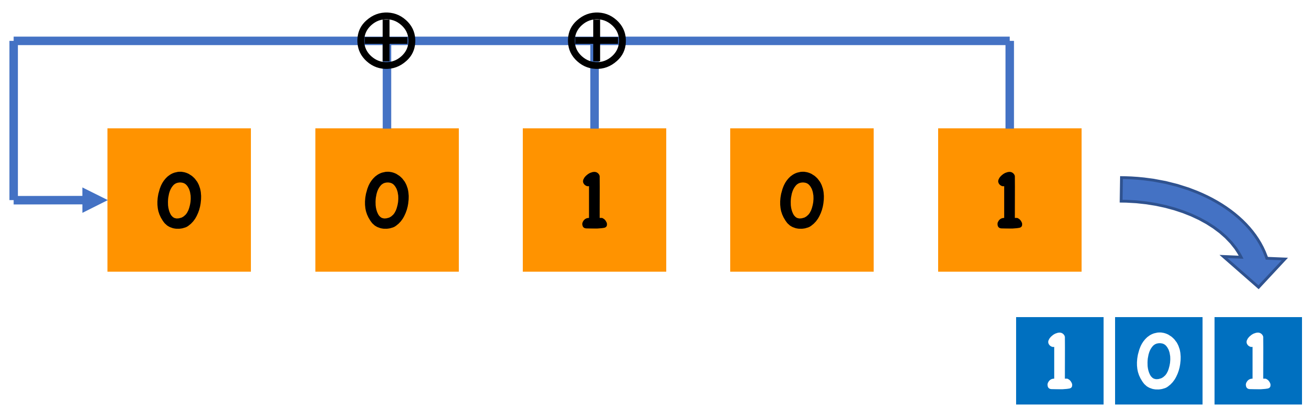
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

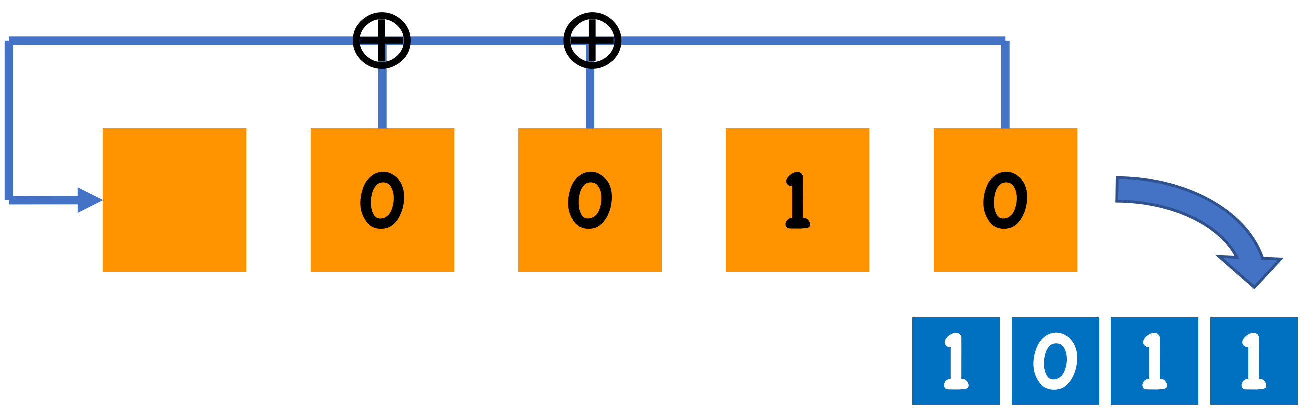
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

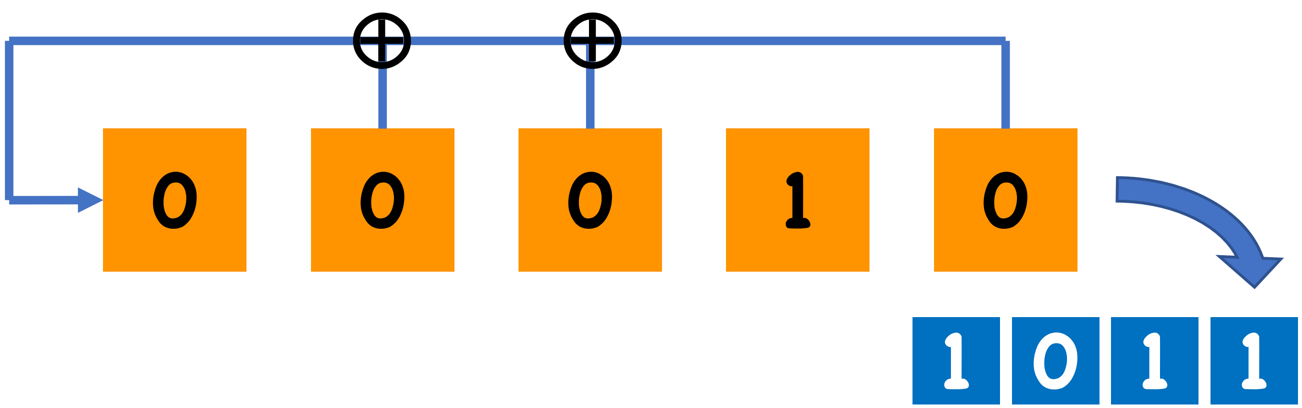
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



…

Linear Feedback Shift Registers

Are LFSR’s secure PRGs?

Linear Feedback Shift Registers

Are LFSR's secure PRGs?

No!

First **n** bits of output = initial state

Write $\mathbf{x} = \mathbf{x_1,\ldots,x_n, x'}$

Initialize LFSB to have state $\mathbf{x_1,\ldots,x_n}$

Run LFSB for $\mathbf{|x|}$ steps, obtaining $\mathbf{y}$

Check if $\mathbf{y = x}$

PRGs should be Unpredictable

More generally, it should be hard, given some bits of output, to predict subsequent bits

Definition: $G:S_\lambda \rightarrow \{0,1\}^{n(\lambda)}$ is **unpredictable** if, for all polynomial time  and any **$p=p(\lambda)$**, $\exists$ negligible **ε** such that:

$$\left| \Pr[G(s)_{p+1} \leftarrow \text{[image 2]}(G(s)_{[1,p]})] - \frac{1}{2} \right| \leq \varepsilon(\lambda)$$

PRGs should be Unpredictable

More generally, it should be hard, given some bits of output, to predict subsequent bits

Theorem: G is **unpredictable** iff it is **pseudorandom**

Proof

Pseudorandomness → Unpredictability

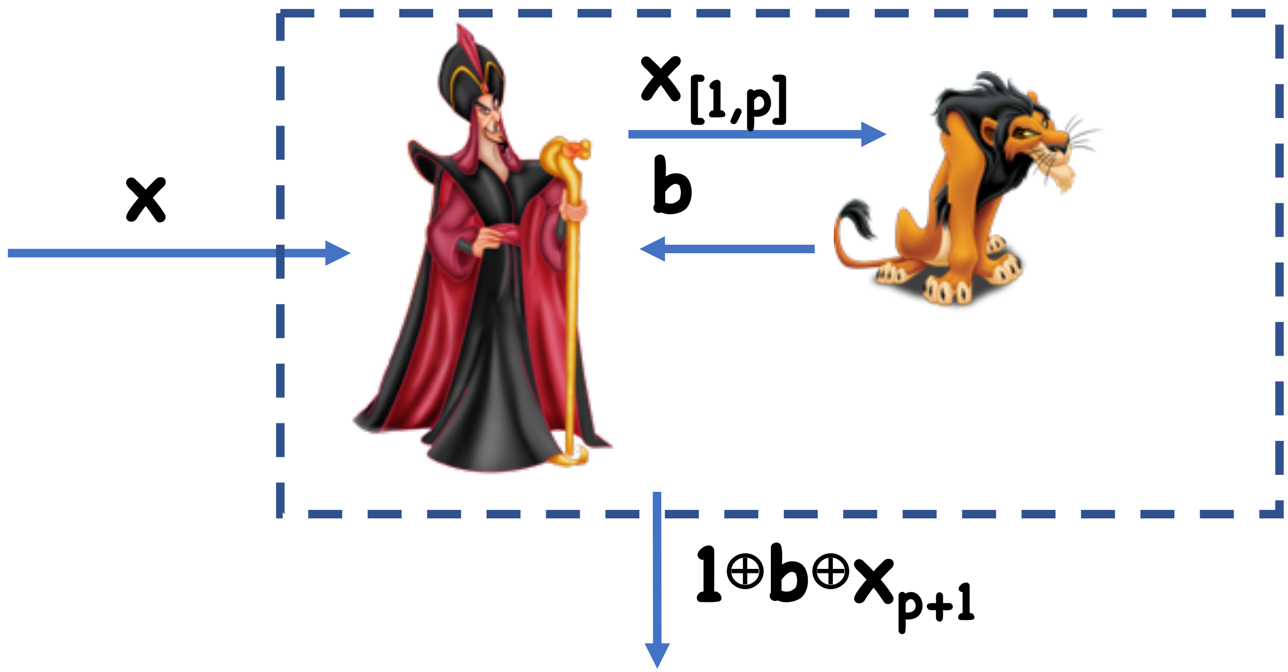
Assume towards contradiction  s.t.

$\left| \Pr[G(s)_{p+1} \leftarrow$  $(G(s)_{[1,p]})\ ] - \frac{1}{2} \right| > \varepsilon$

Proof

Pseudorandomness → Unpredictability

Construct 



Proof

Pseudorandomness → Unpredictability

Analysis:

- If $\mathbf{x}$ is random, $\Pr[1 \oplus b \oplus x_{p+1} = 1] = ½$
- If $\mathbf{x}$ is pseudorandom,

$$\Pr[1 \oplus b \oplus x_{p+1} = 1]$$

$= \Pr[G(s)_{p+1} \leftarrow$  $(G(s)_{[1,p]})\ ]$

$> (½ + \varepsilon) \text{ or } < (½ - \varepsilon)$

Proof

Unpredictability → Pseudorandomness

Assume towards contradiction  s.t.

| Pr[ $(G(s))=1 : s \leftarrow \{0,1\}^\lambda$]

− Pr[ $(x)=1 : x \leftarrow \{0,1\}^t$] | $> \varepsilon$

Proof

Unpredictability → Pseudorandomness

Hybrids:

$\mathbf{H_i}$: $\mathbf{x_{[1,i]} \leftarrow G(s), x_{[i+1,t]} \leftarrow \{0,1\}^{t-i}}$

$\mathbf{H_0}$: truly random $\mathbf{x}$

$\mathbf{H_t}$: pseudorandom $\mathbf{t}$

Proof

Unpredictability → Pseudorandomness

Hybrids:

H_i: $x_{[1,i]} \leftarrow G(s)$, $x_{[i+1,t]} \leftarrow \{0,1\}^{t-i}$

| Pr[ $(x)=1 : x \leftarrow H_s$]

− Pr[ $(x)=1 : x \leftarrow H_0$] | $> \varepsilon$

Let q_i = Pr[ $(x)=1 : x \leftarrow H_i$]

Proof

Unpredictability → Pseudorandomness

Hybrids:

H_i: $x_{[1,i]} \leftarrow G(s)$, $x_{[i+1,t]} \leftarrow \{0,1\}^{t-i}$

$$| q_t - q_0 | > \varepsilon$$

Let $q_i = \Pr[$  $(x)=1 : x \leftarrow H_i]$

Proof

Unpredictability → Pseudorandomness

Hybrids:

H_i: $x_{[1,i]} \leftarrow G(s)$, $x_{[i+1,t]} \leftarrow \{0,1\}^{t-i}$

By triangle inequality, there must exist an i s.t.

$$| q_i - q_{i-1} | > \varepsilon/t$$

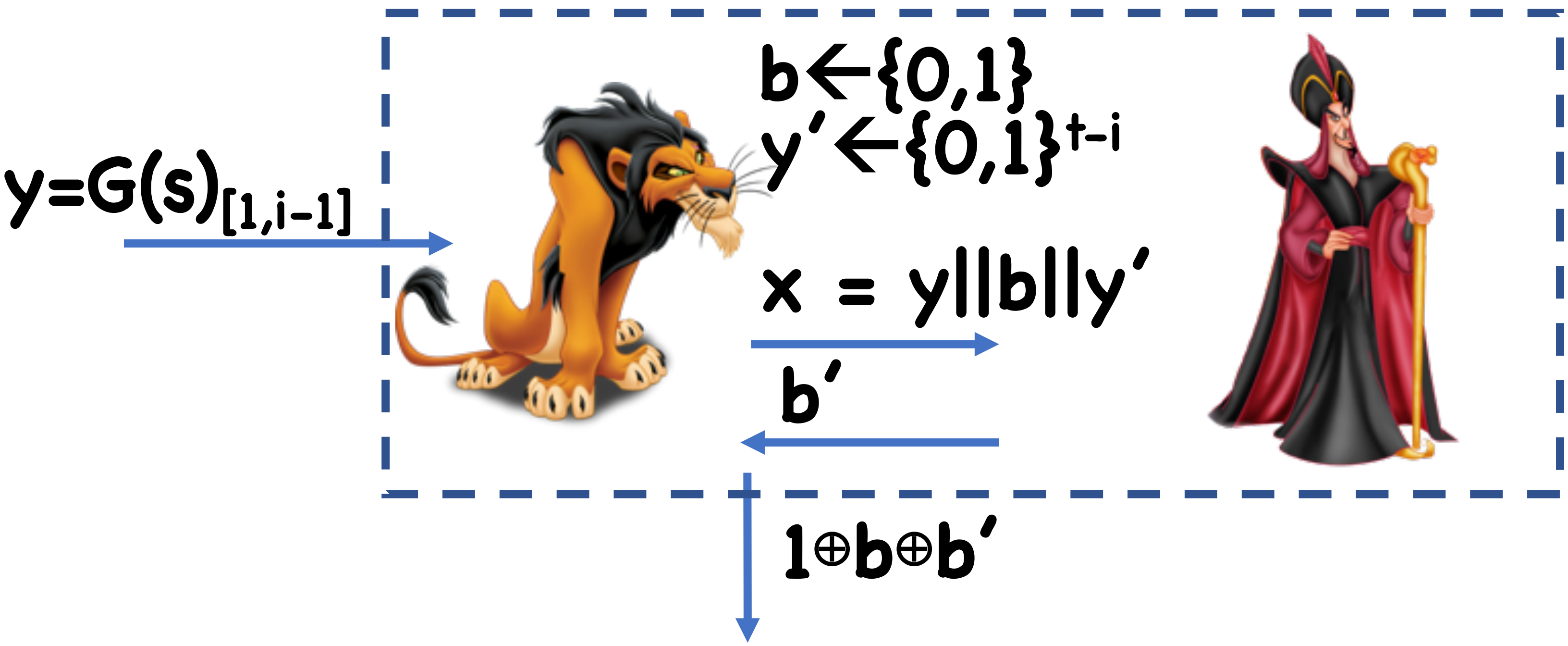
Can assume wlog that

$$q_i - q_{i-1} > \varepsilon/t$$

Proof

Unpredictability → Pseudorandomness

Construct 



Proof

Unpredictability → Pseudorandomness

Analysis:

- If $b = G(s)_i$, then  sees H_i

  $\Rightarrow$  outputs **1** with probability q_i

  $\Rightarrow$  outputs $b=G(s)_i$ with probability q_i

Proof

Unpredictability → Pseudorandomness

Analysis:

- If $\mathbf{b} = 1 \oplus \mathbf{G(s)}_i$, then

  Define q_i' as $\Pr[$  outputs $1]$

  $\frac{1}{2}(q_i' + q_i) = q_{i-1} \Rightarrow q_i' = 2q_{i-1} - q_i$

  $\Rightarrow$  outputs $\mathbf{G(s)}_{[1,i]}$ with probability

  $1 - q_i' = 1 + q_i - 2q_{i-1}$

Proof

Unpredictability → Pseudorandomness

Analysis:

- **Pr[** outputs $\mathbf{G(s)_i}$**]**

$$= \tfrac{1}{2}\,(q_i) + \tfrac{1}{2}\,(1 + q_i - 2q_{i-1})$$

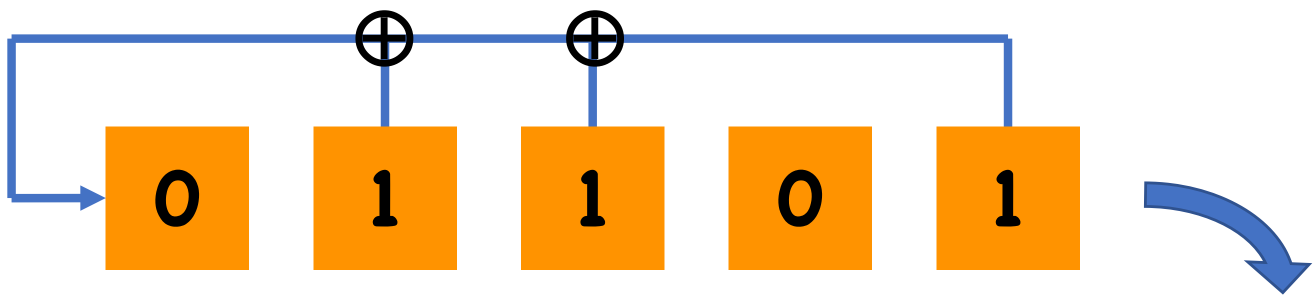
$$= \tfrac{1}{2} + q_i - q_{i-1}$$

$$> \tfrac{1}{2} + \varepsilon/t$$

Any ideas?

Linearity

LFSR's are linear:



$$\text{state}' = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \bullet \text{state}$$

$$\text{output} = (0\ 0\ 0\ 0\ 1) \bullet \text{state}$$

Linearity

LFSR's are linear:

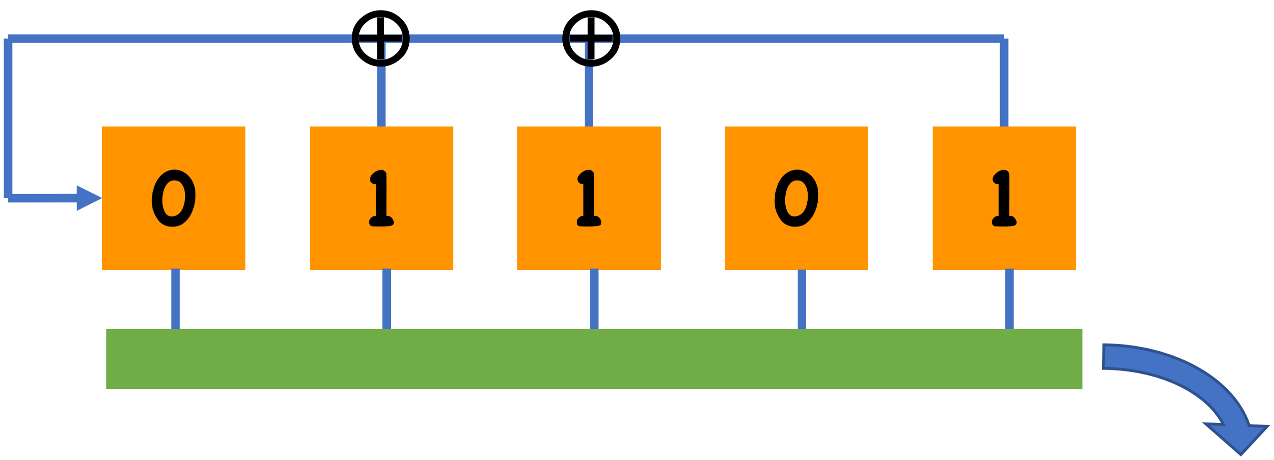
- Each output bit is a linear function of the initial state (that is, $\mathbf{G(s) = A \bullet s \ (mod\ 2)}$)

Any linear $\mathbf{G}$ <u>cannot</u> be a PRG

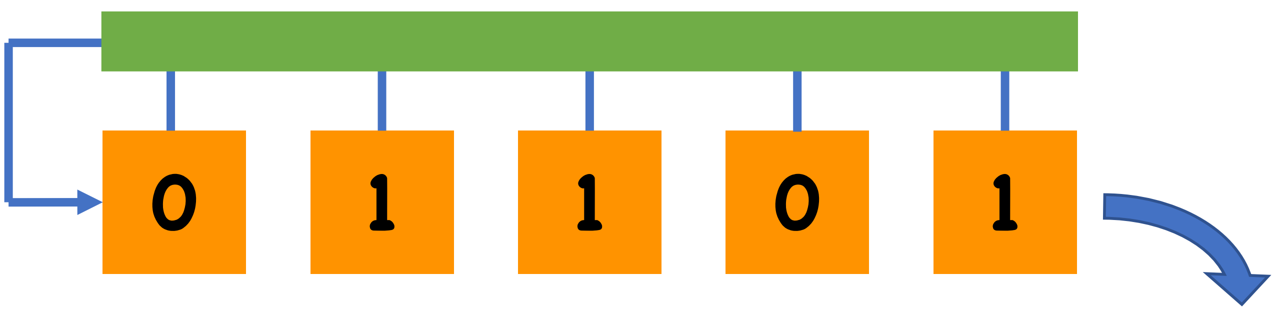
- Can check if $\mathbf{x}$ is in column-span of $\mathbf{A}$ using linear algebra

Introducing Non-linearity

Non-linearity in the output:



Non-linear feedback:



LFSR period

Period = number of bits before state repeats

After one period, output sequence repeats

Therefore, should have extremely long period

- Ideally almost 2^λ
- Possible to design LFSR's with period $2^\lambda-1$

Hardware vs Software

PRGs based on LFSR's are very fast in hardware

Unfortunately, not easily amenable to software

RC4

Fast software based PRG

Resisted attack for several years

No longer considered secure, but still widely used

RC4

State = permutation on **[256]** plus two integers
- Permutation stored as **256**-byte array **S**

Init(16-byte **k)**:
- For **i=0,…,255**
  S[i] = i
- **j = 0**
- **For i=0,…,255**
  j = j + S[i] + k[i mod 16] (mod 256)
  Swap **S[i]** and **S[j]**
- Output **(S,0,0)**

RC4

GetBits(S,i,j):

- **i++ (mod 256)**
- **j+= S[i] (mod 256)**
- Swap **S[i]** and **S[j]**
- **t = S[i] + S[j] (mod 256)**
- Output **(S,i,j), S[t]**

New state　　　Next output byte

Insecurity of RC4

Second byte of output is slightly biased towards 0

- **Pr[**second byte **= 0^8] ≈ 2/256**
- Should be **1/256**

Means RC4 is not secure according to our definition

-  outputs **1** iff second byte is equal to **0^8**
- Advantage: **≈ 1/256**

Not a serious attack in practice, but demonstrates some structural weakness

Insecurity of RC4

Possible to extend attack to actually recover the input **k** in some use cases

- The seed is set to **(IV, k)** for some initial value **IV**
- Encrypt messages as **RC4(IV,k)**$\oplus$**m**
- Also give **IV** to attacker
- Cannot show security assuming RC4 is a PRG

Can be used to completely break WEP encryption standard

Summary

Stream ciphers = secure encryption for arbitrary length, number of messages
(though we did not completely prove it)

However, implementation difficulties due to having to maintaining state

Reminders

HW1 Due Feb 20th
HW2 Due Feb 27th

PR1 Due March 10th