

COS433/Math 473: Cryptography

Mark Zhandry

Princeton University

Spring 2017

Announcements

Homework 2 due tomorrow

Grades and comments should be visible on
Blackboard

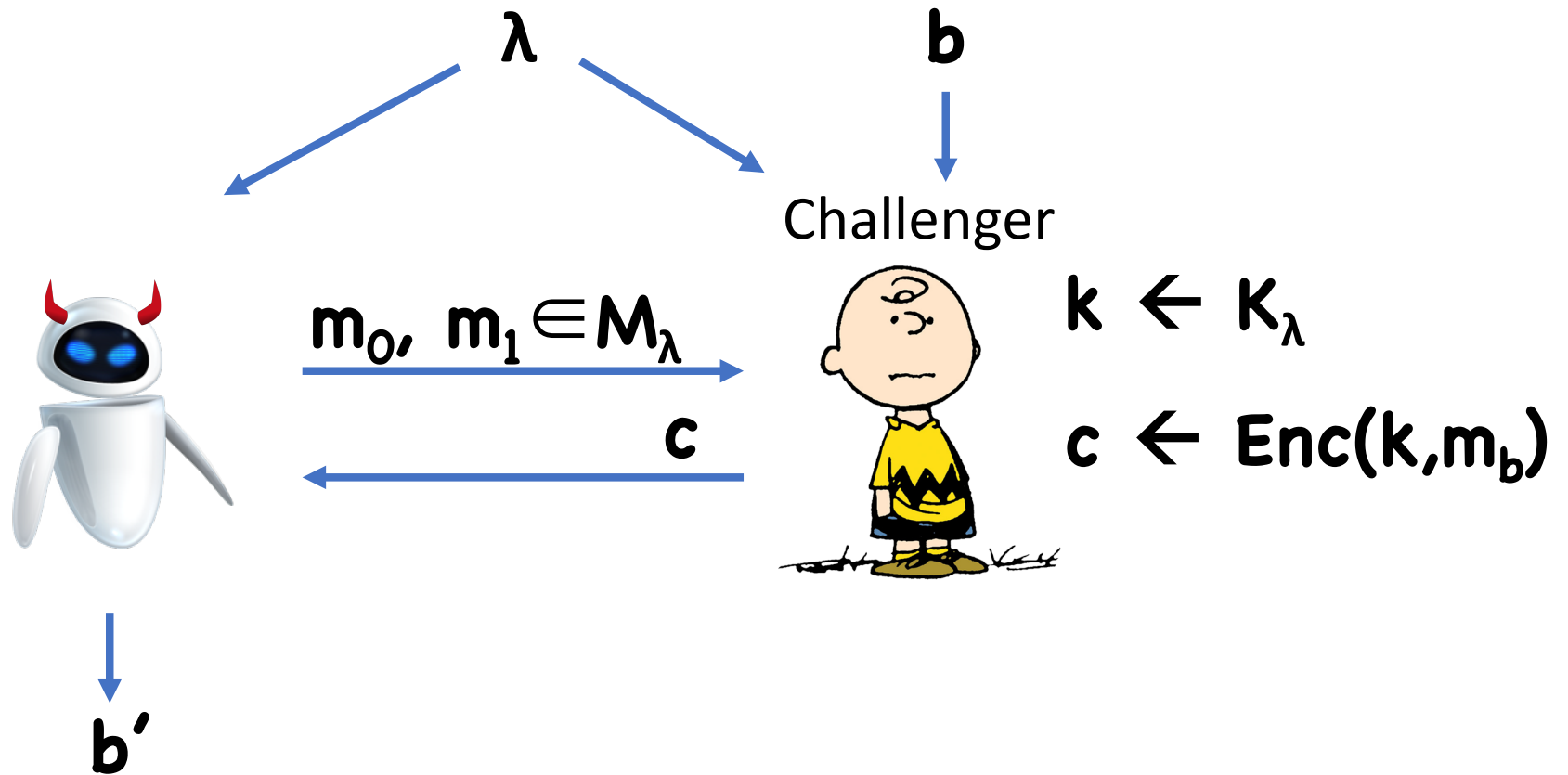
Last Time

$|key| \geq |total\ information\ encrypted|$ is necessary
for statistical security

Computational Security


PRGs

Encryption Security Experiment



$IND-Exp_b(\text{robot}, \lambda)$

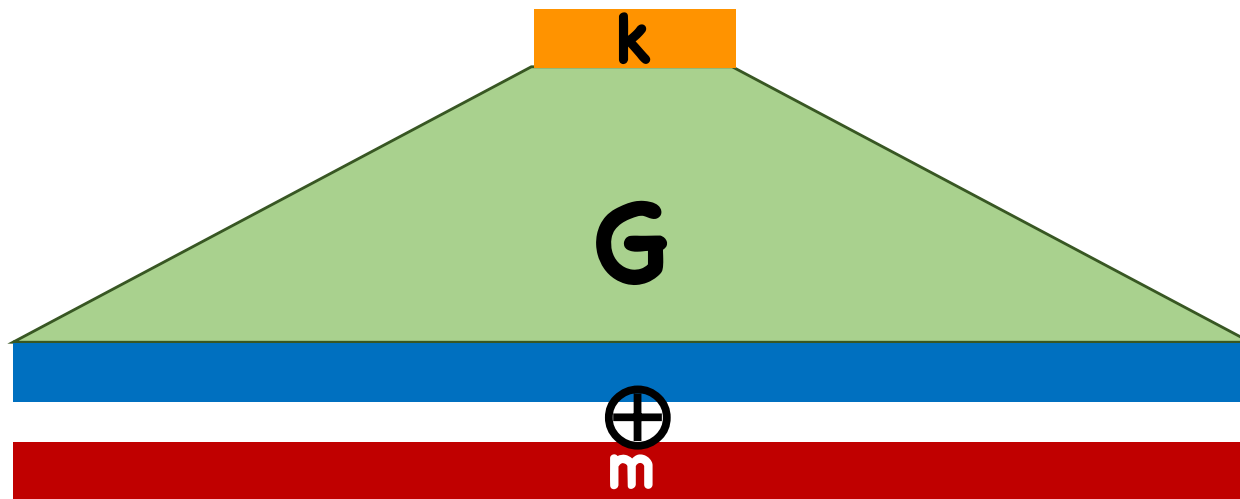
Encryption Security Definition

Definition: (Enc, Dec) has **ciphertext indistinguishability** if, for all probabilistic polynomial time (PPT) , there exists a negligible function ϵ such that

$$\left| \Pr[1 \leftarrow \text{IND-Exp}_0(\text{robot}, \lambda)] - \Pr[1 \leftarrow \text{IND-Exp}_1(\text{robot}, \lambda)] \right| \leq \epsilon(\lambda)$$


Construction with $|k| \ll |m|$


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




Pseudorandom Generators

Definition: $G:\{0,1\}^* \rightarrow \{0,1\}^*$ is a **secure pseudorandom generator (PRG)** if:

- G is computable in polynomial time
- G applied to λ bit strings produces strings of length $t(\lambda) > \lambda$
- G is deterministic
- For all **PPT** , \exists **negl** ϵ such that:

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

$$\left. - \Pr[\text{ (x)=1 : x \leftarrow \{0,1\}^{t(\lambda)}] \right| \leq \epsilon(\lambda)$$

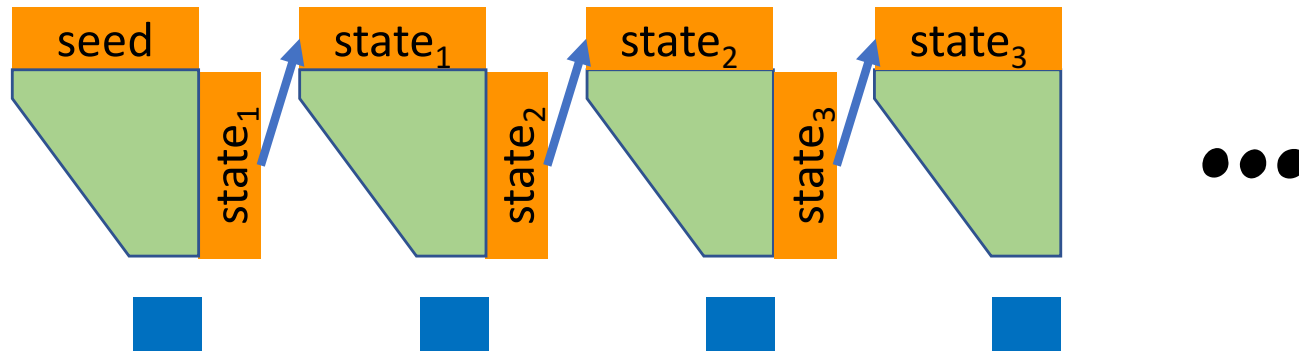
This Time

Stream Ciphers

Design of PRGs

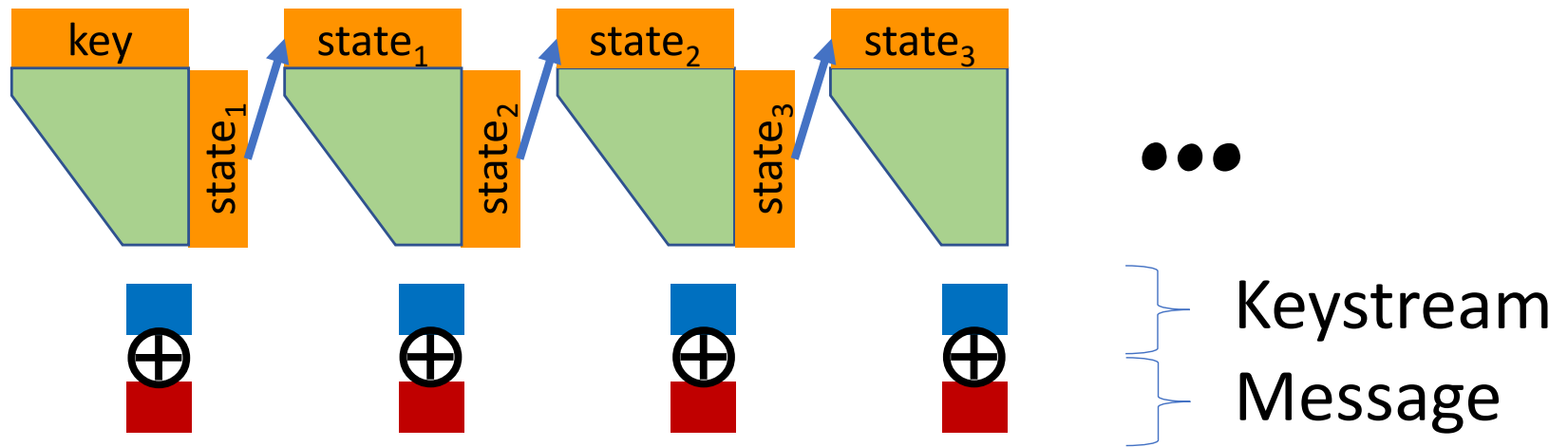
Pseudorandom Generators

PRGs usually allow for streaming arbitrarily long sequences of random bits



Stream Ciphers

Use “streaming” PRG to encrypt messages



Stream Ciphers

Use “streaming” PRG to encrypt messages

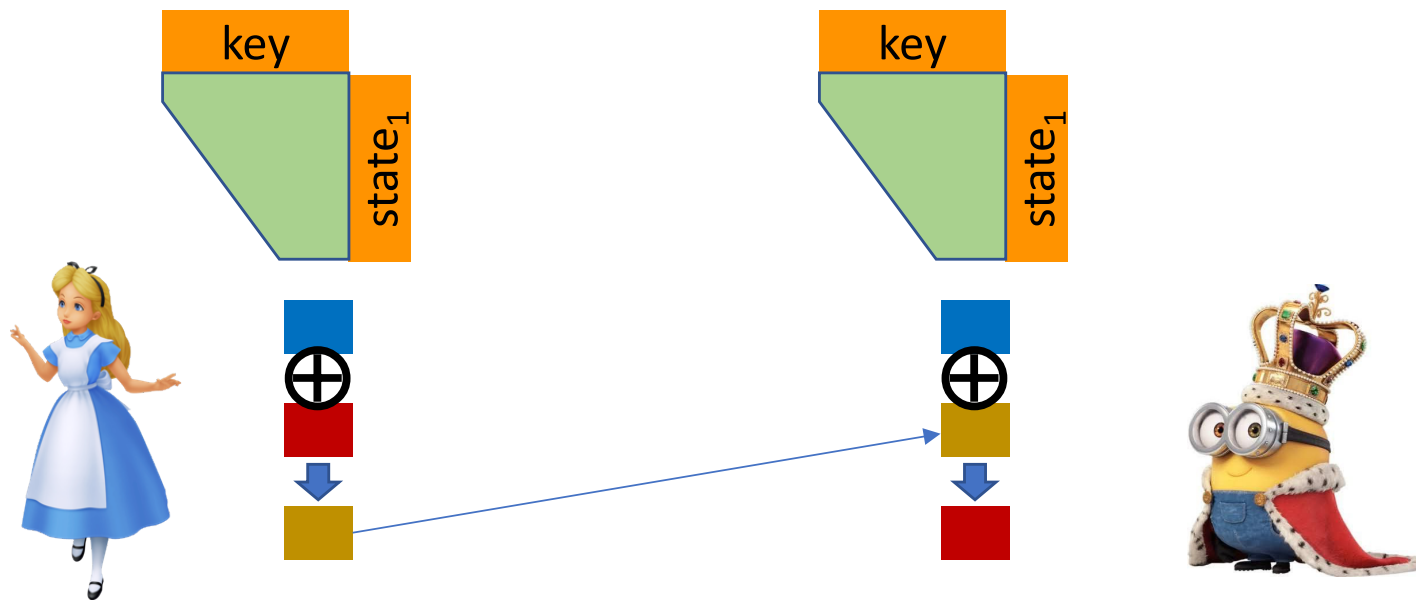
In this way, can encrypt arbitrarily long messages

- security proof similar to last time

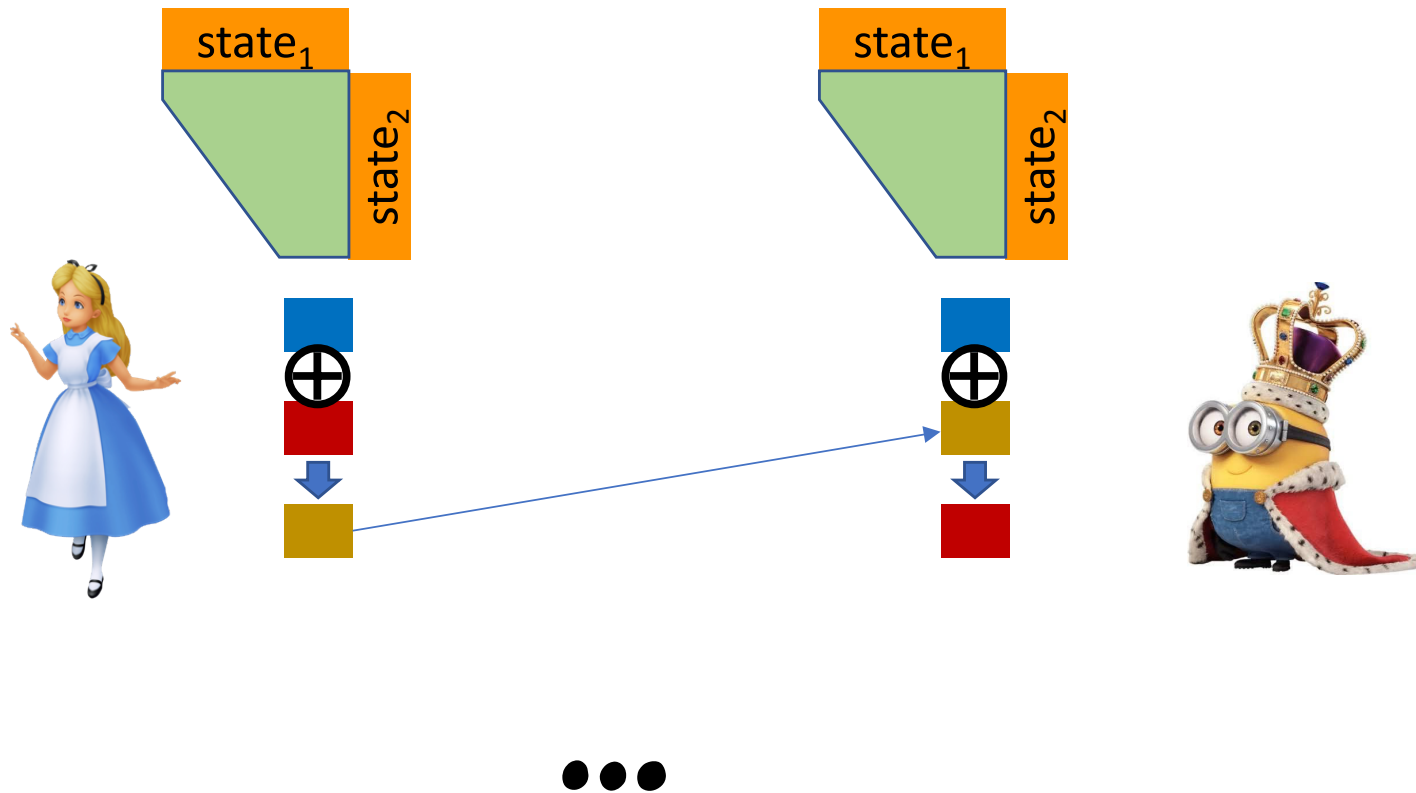
But remember, stream ciphers are really just OTP's, so still cannot encrypt twice with the same part of keystream

Instead, encrypt like we did with the one-time pad

Multiple Messages with Stream Ciphers



Multiple Messages with Stream Ciphers



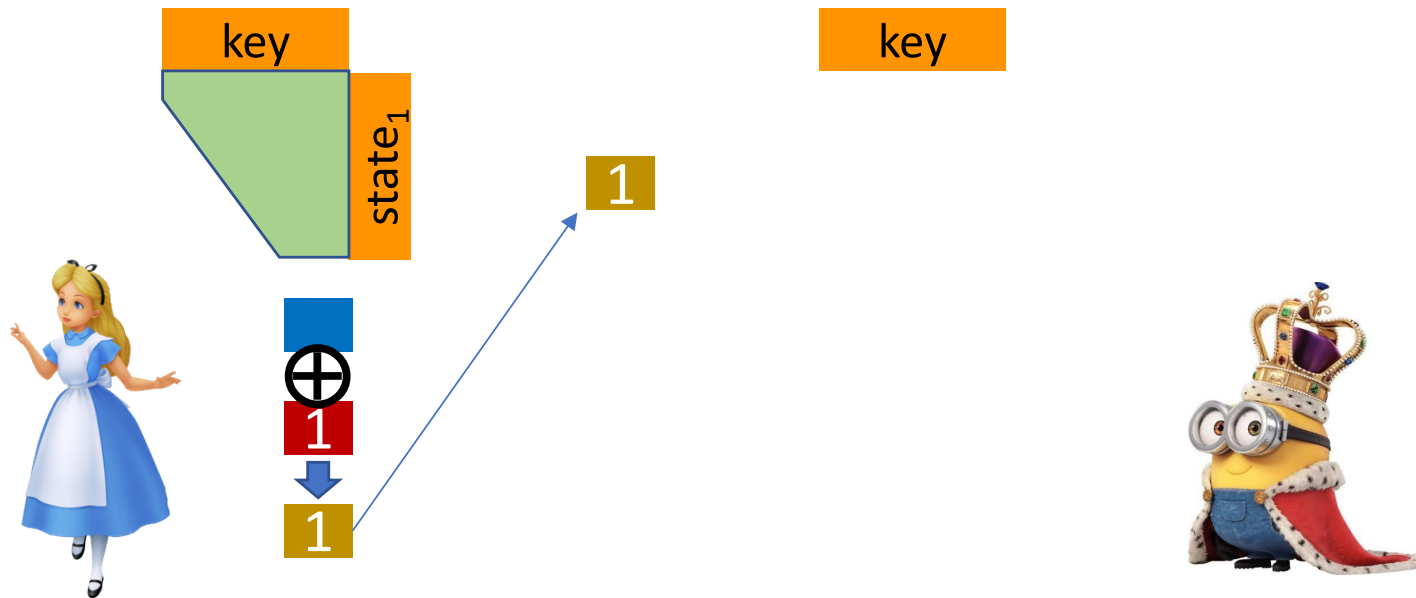
Limitations of Stream Ciphers

Just like with OTP, need to be careful because communication may be asynchronous

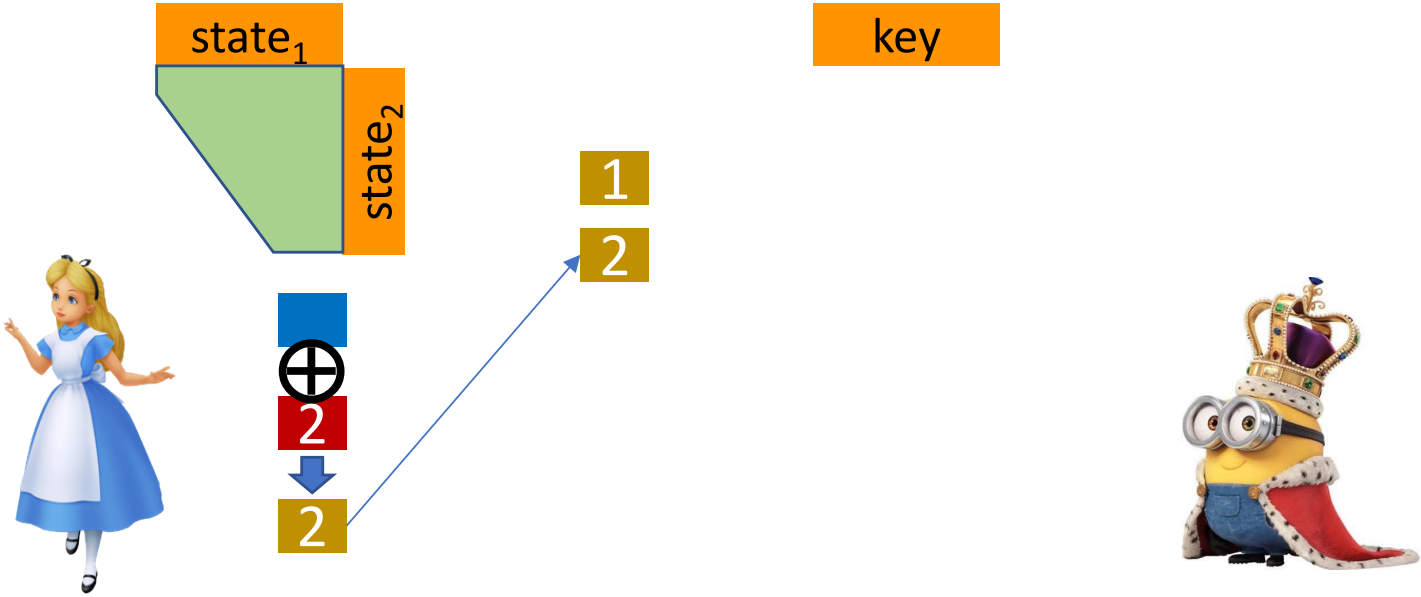
- Keep a different key/state for each direction of communication

Here, even bigger problem cause by out of order messages

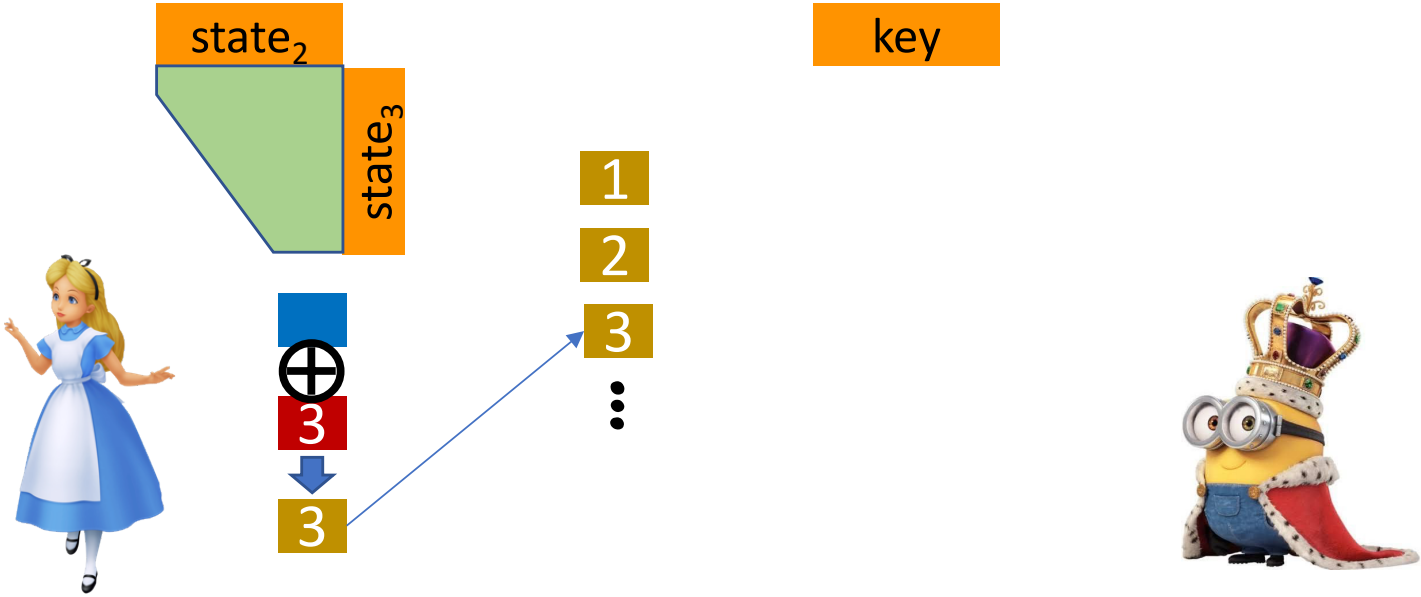
Multiple Messages with Stream Ciphers



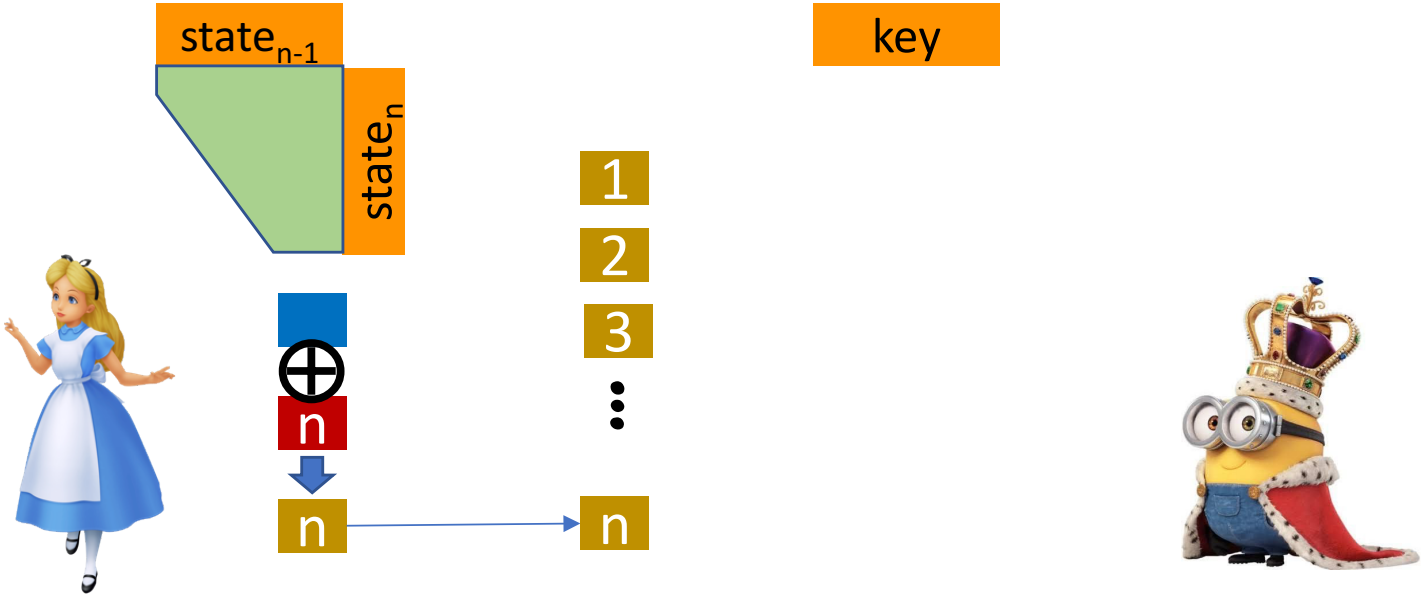
Multiple Messages with Stream Ciphers



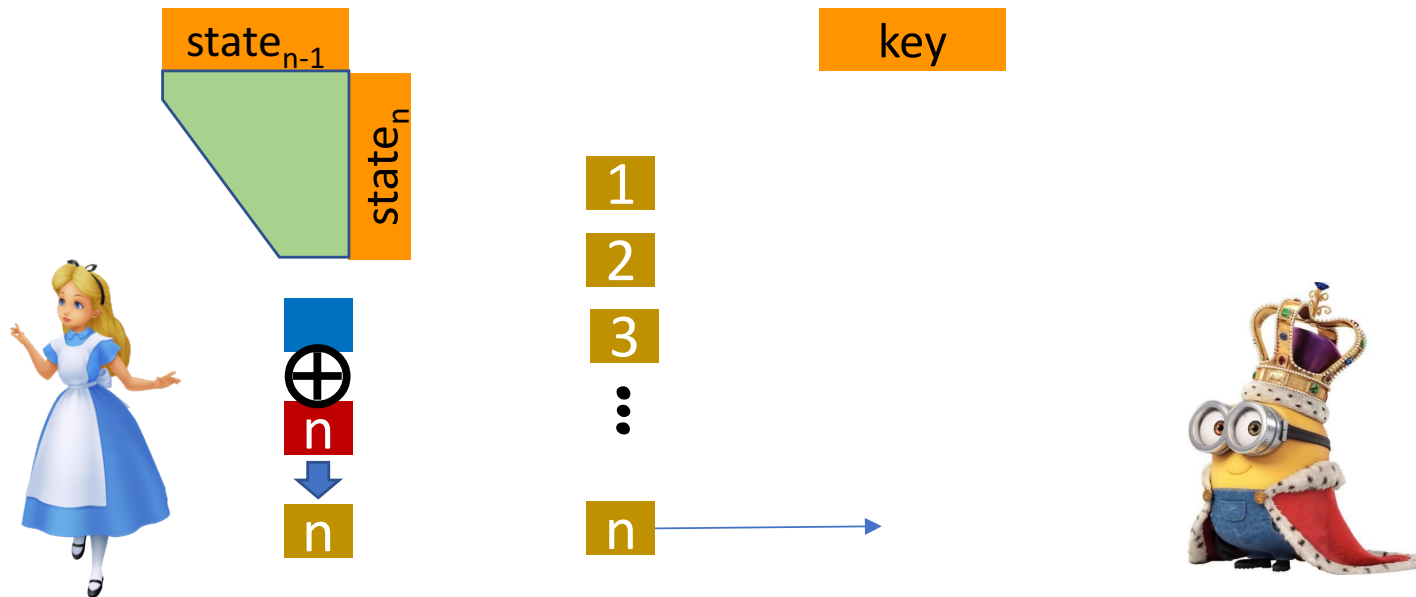
Multiple Messages with Stream Ciphers



Multiple Messages with Stream Ciphers



Multiple Messages with Stream Ciphers



Bob needs to either:

- Store entire keystream until he receives message 1
- Or re-compute keystream from scratch every message

Multiple Messages with Stream Ciphers

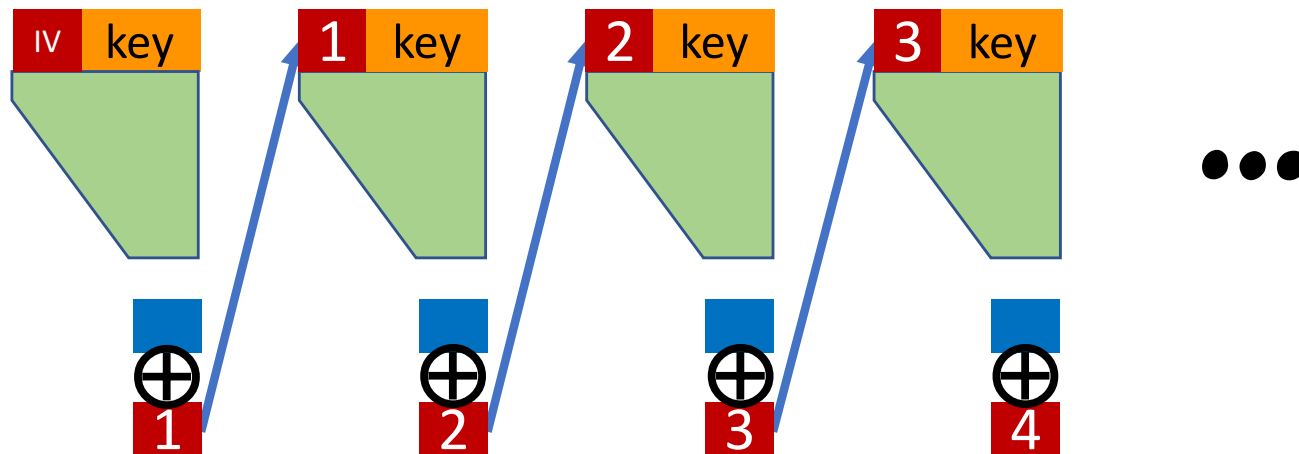
Out of order messages cause implementation difficulties

Mitigation?

- Self-synchronizing stream cipher

Self-Synchronizing Stream Ciphers

“state” is just (last several ciphertext bits seen, key)



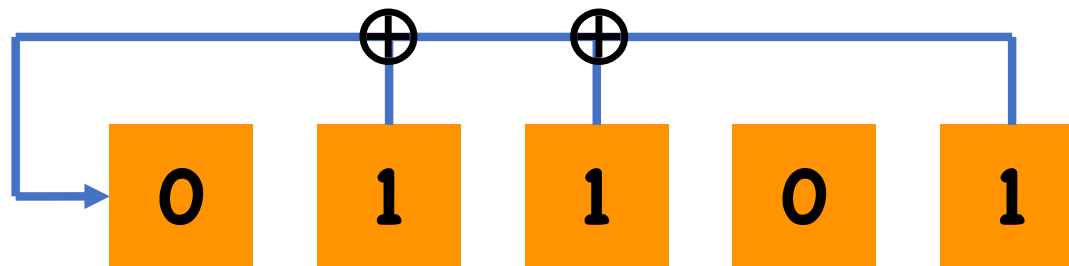
Thus, you can always decrypt if the last several ciphertext bits were correct

How do we build PRGs?

Linear Feedback Shift Registers

In each step,

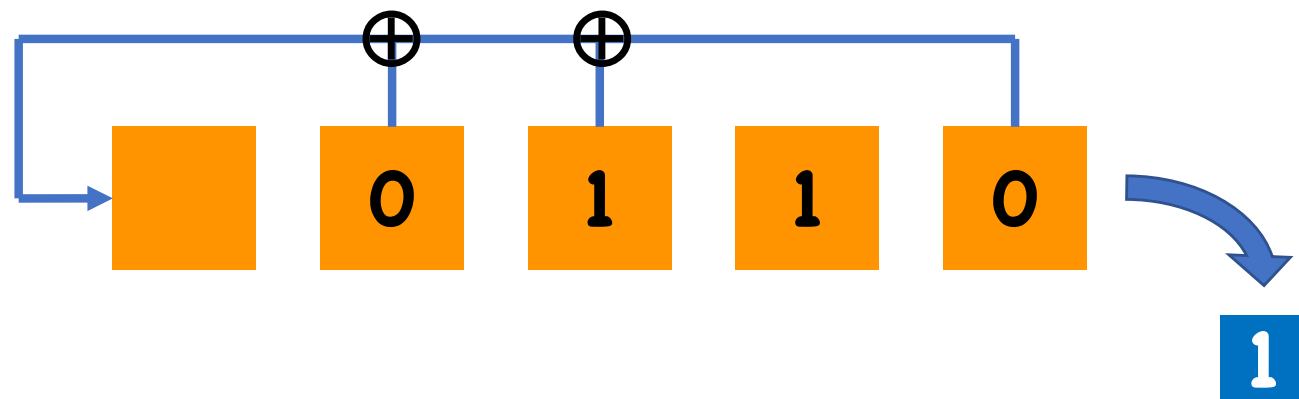
- Last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

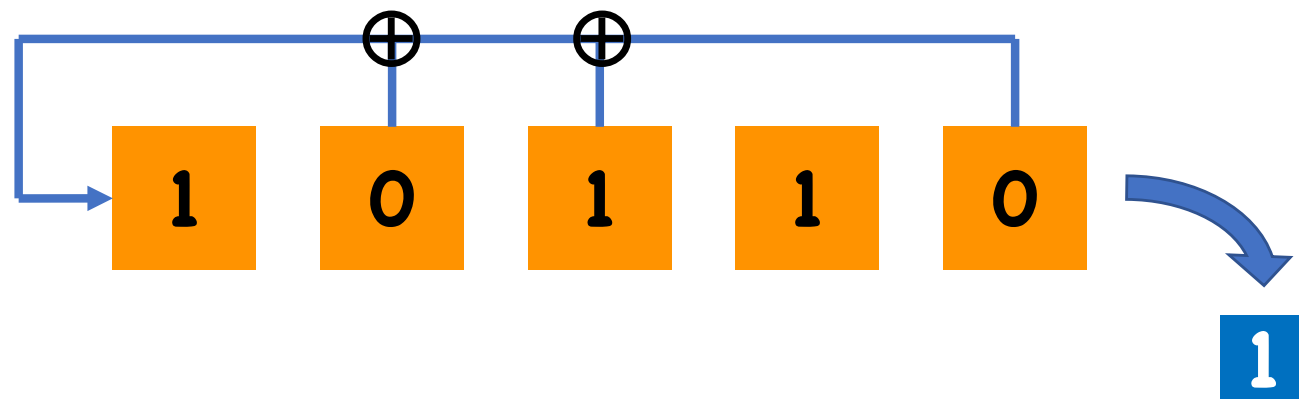
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

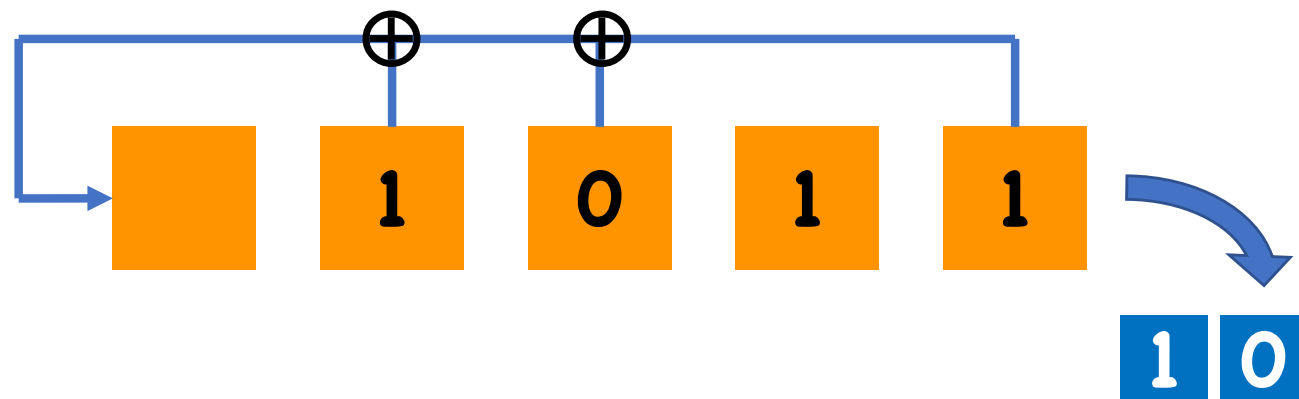
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

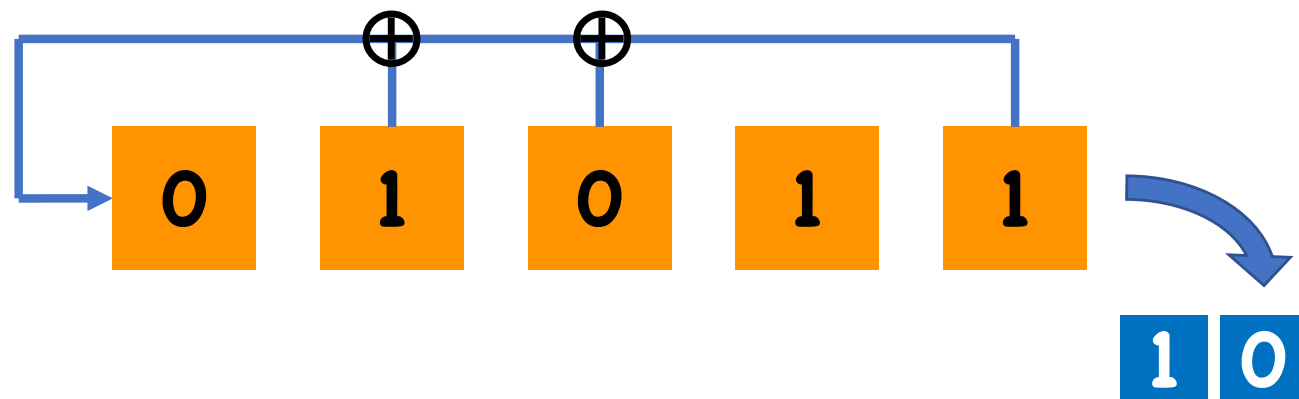
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

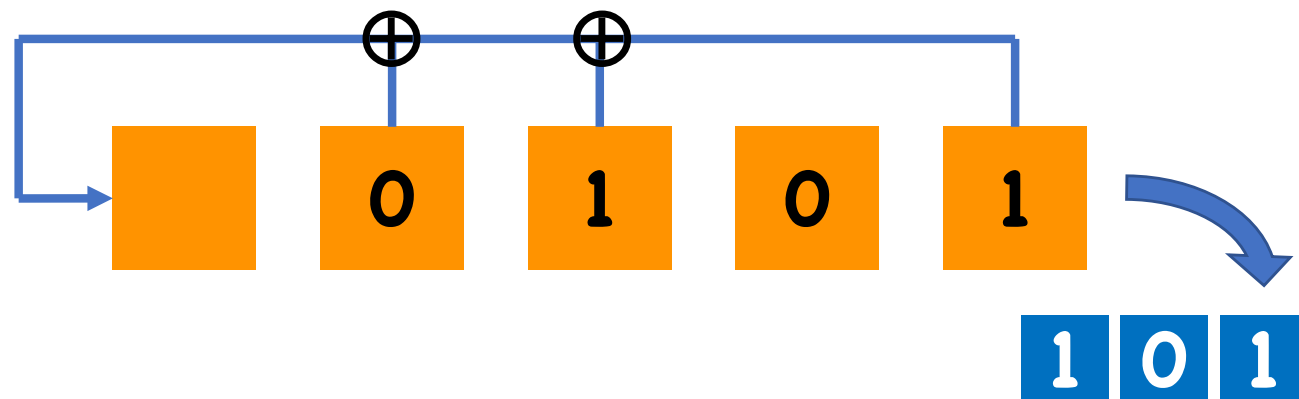
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

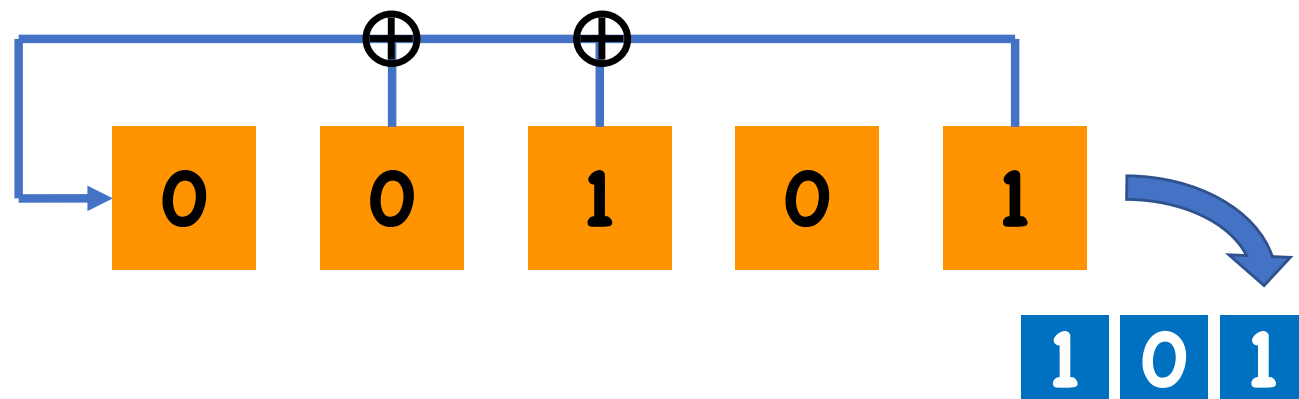
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

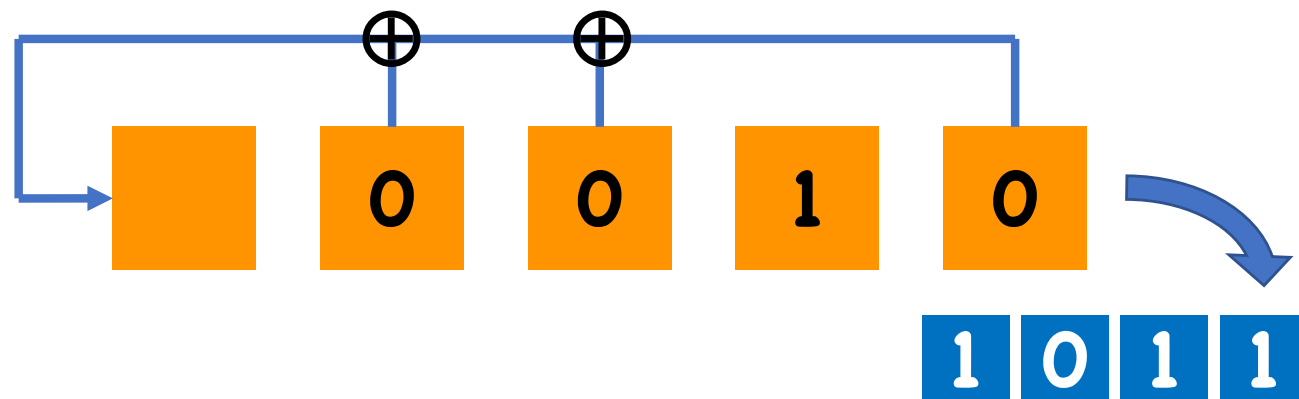
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



Linear Feedback Shift Registers

In each step,

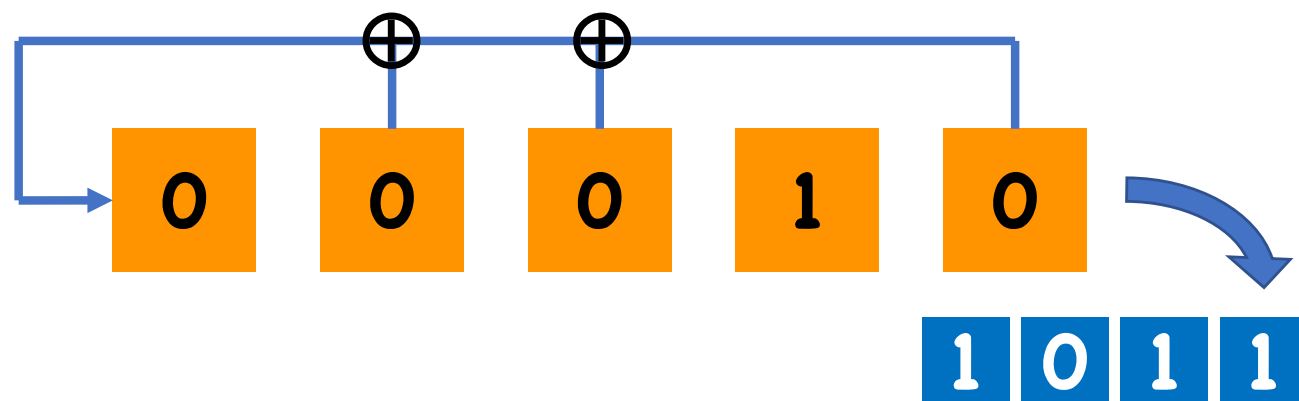
- last bit of state is removed and outputted
- Rest of bits are shifted right
- First bit is XOR of subset of remaining bits



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



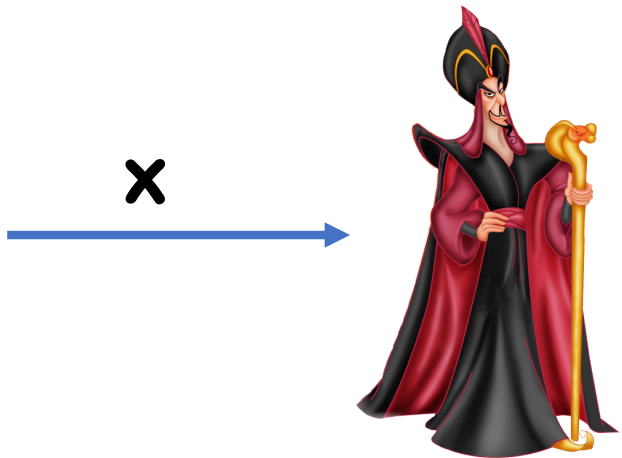
...

Linear Feedback Shift Registers

Are LFSR's secure PRGs?

No!

First n bits of output = initial state



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


Initialize LFSB to have state x_1, \dots, x_n

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

Check if $\mathbf{y} = \mathbf{x}$

PRGs should be Unpredictable

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

Definition: G is **unpredictable** if, for all probabilistic polynomial time (PPT)  and any polynomial p , there exists a negligible function ϵ such that

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


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 \rightarrow Unpredictability

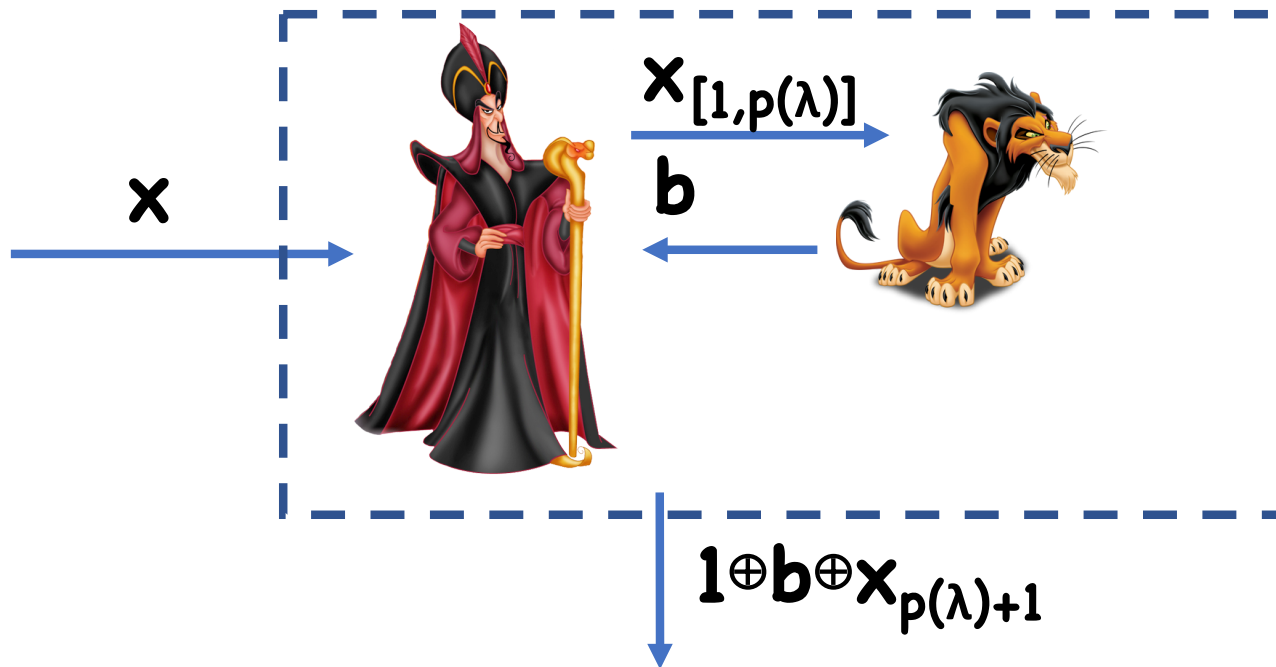
Assume towards contradiction PPT , polynomial p , non-negligible function ϵ s.t.

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

Proof

Pseudorandomness \rightarrow Unpredictability

Construct 



Proof

Pseudorandomness \rightarrow Unpredictability

Analysis:

- If \mathbf{x} is random, $\Pr[\mathbf{1} \oplus \mathbf{b} \oplus \mathbf{x}_{p(\lambda)+1} = 1] = \frac{1}{2}$
- If \mathbf{x} is pseudorandom,

$$\begin{aligned} & \Pr[\mathbf{1} \oplus \mathbf{b} \oplus \mathbf{x}_{p(\lambda)+1} = 1] \\ &= \Pr[G(\mathbf{s})_{p(\lambda)+1} \leftarrow \text{🦁} (G(\mathbf{s})_{[1,p(\lambda)]})] \\ &= \frac{1}{2} \pm \varepsilon(\lambda) \end{aligned}$$

Proof

Unpredictability \rightarrow Pseudorandomness

Assume towards contradiction PPT , non-negligible function ϵ s.t.

$$\left| \Pr[\img alt="wizard" data-bbox="288 536 328 618" (G(s))=1 : s \leftarrow \{0,1\}^\lambda] \right. \\ \left. - \Pr[\img alt="wizard" data-bbox="338 648 378 730" (x)=1 : x \leftarrow \{0,1\}^{t(\lambda)}] \right| = \epsilon(\lambda)$$

Proof

Unpredictability \rightarrow Pseudorandomness

Hybrids:

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

H_0 : truly random x

H_t : pseudorandom t

Proof

Unpredictability \rightarrow Pseudorandomness

Hybrids:

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

$$\left| \Pr[\text{A}(x)=1 : x \leftarrow H_s] \right.$$

$$\left. - \Pr[\text{A}(x)=1 : x \leftarrow H_0] \right| = \epsilon(\lambda)$$

$$\text{Let } q_i = \Pr[\text{A}(x)=1 : x \leftarrow H_i]$$

Proof

Unpredictability \rightarrow Pseudorandomness

Hybrids:

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

$$| q_t - q_0 | = \epsilon(\lambda)$$

$$\text{Let } q_i = \Pr[\img alt="wizard" data-bbox="470 830 510 910" style="vertical-align: middle;"/> (x)=1 : x \leftarrow H_i]$$

Proof

Unpredictability \rightarrow 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.

$$|q_i - q_{i-1}| \geq \varepsilon(\lambda)/t$$

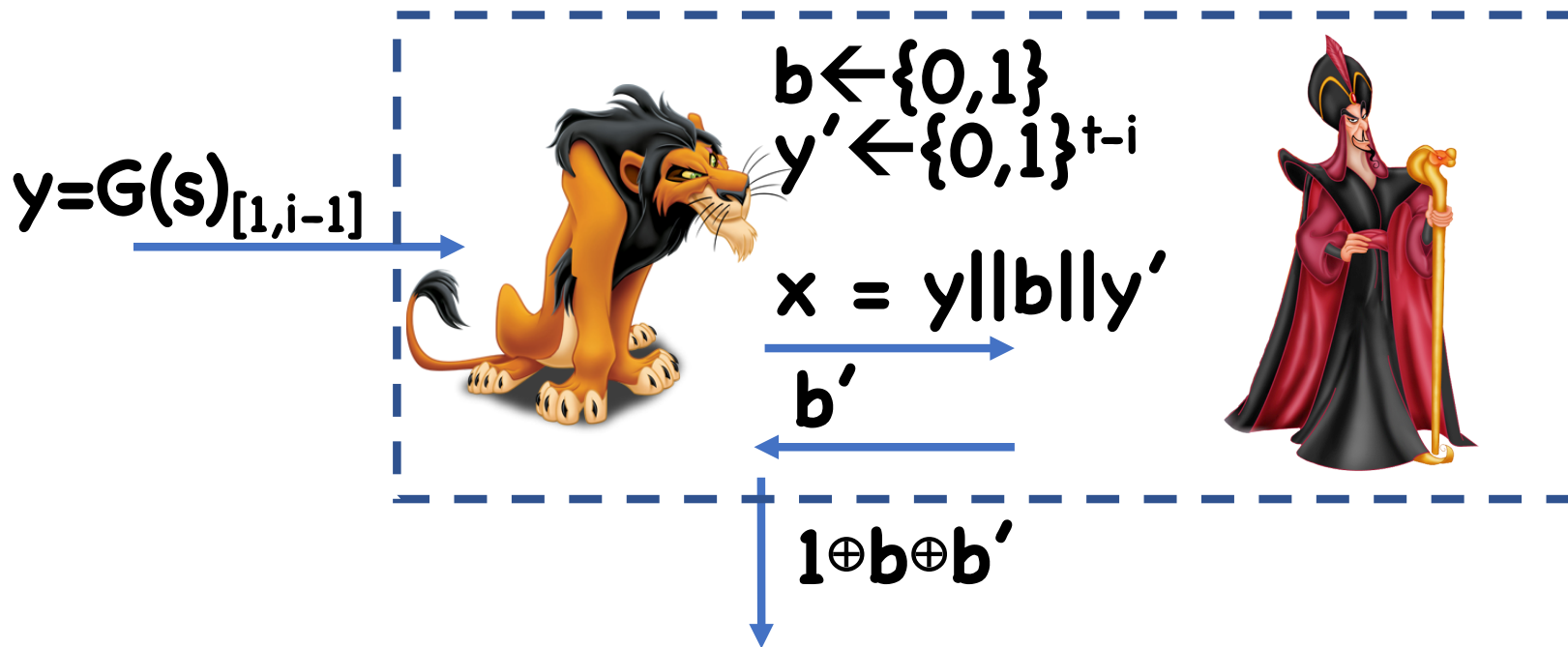
Can assume wlog that

$$q_i - q_{i-1} \geq \varepsilon(\lambda)/t$$

Proof

Unpredictability \rightarrow Pseudorandomness




Construct 



Proof

Unpredictability \rightarrow Pseudorandomness

Analysis:

- If $\mathbf{b} = \mathbf{G}(\mathbf{s})_i$, then  sees \mathbf{H}_i
 - \Rightarrow  outputs $\mathbf{1}$ with probability q_i
 - \Rightarrow  outputs $\mathbf{b}=\mathbf{G}(\mathbf{s})_i$ with probability q_i

Proof

Unpredictability \rightarrow Pseudorandomness

Analysis:

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

Define q_i' as $\Pr[\text{👑 outputs } \mathbf{1}]$

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

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

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

Proof

Unpredictability \rightarrow Pseudorandomness

Analysis:

• $\Pr[\text{🦊 outputs } G(s)_i]$

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

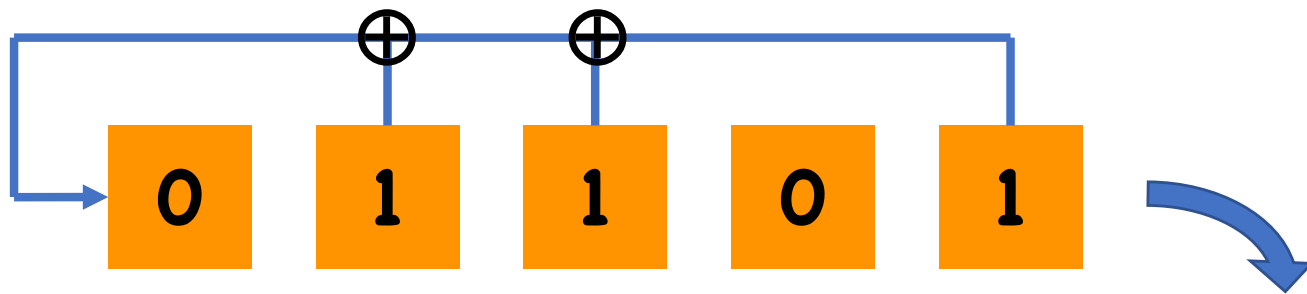
$$= \frac{1}{2} + q_i - q_{i-1}$$

$$\geq \frac{1}{2} + \varepsilon(\lambda)/t$$

Linearity

Linearity

LFSR's are linear:



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

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

Linearity

LFSR's are linear:

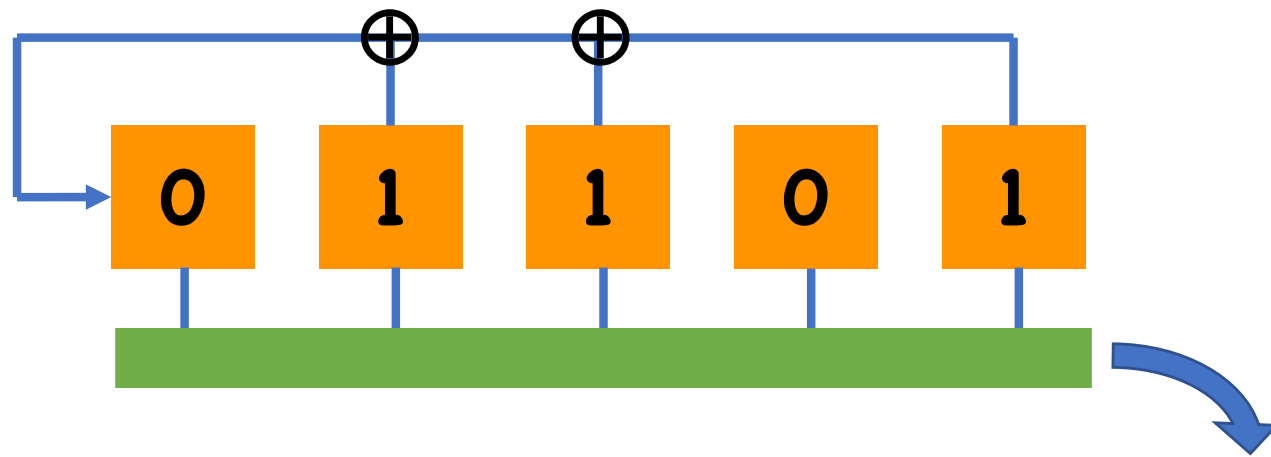
- Each output bit is a linear function of the initial state (that is, $\mathbf{G}(\mathbf{s}) = \mathbf{A} \bullet \mathbf{s} \pmod{2}$)

Any linear \mathbf{G} cannot be a PRG

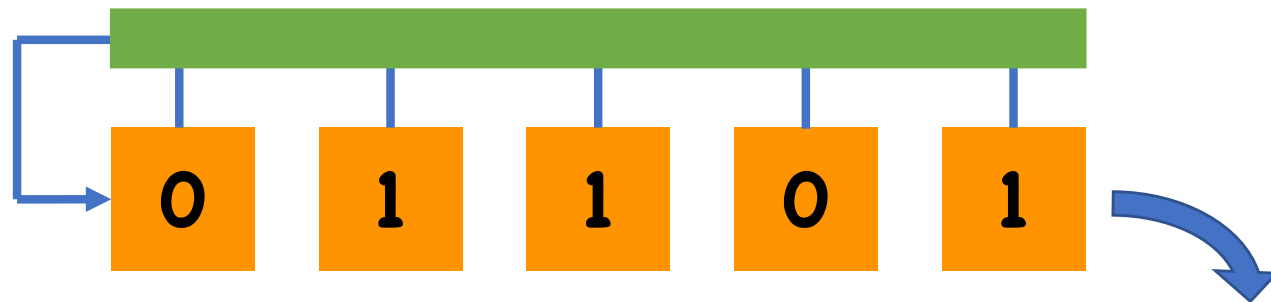
- Can check if \mathbf{x} is in column-span of \mathbf{A} using linear algebra

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^n
- Possible to design LFSR's with period $2^n - 1$

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 **$i=0, \dots, 255$**
 $S[i] = i$
- **$j = 0$**
- For **$i=0, \dots, 255$**
 $j = j + S[i] + k[i \bmod 16] \pmod{256}$
 Swap **$S[i]$** and **$S[j]$**
- Output **$(S, 0, 0)$**

RC4

GetBits(S,i,j):

- $i++ \pmod{256}$
- $j += S[i] \pmod{256}$
- Swap $S[i]$ and $S[j]$
- $t = S[i] + S[j] \pmod{256}$
- Output $(S,i,j), S[t]$

New state

Next output byte



Insecurity of RC4

Second byte of output is slightly biased towards 0

- $\Pr[\text{second byte} = 0^8] \approx 2/256$
- Should be $1/256$

Means RC4 is not secure according to our definition

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

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

Insecurity of RC4

Possible to extend attack to actually recover the input \mathbf{k} in some use cases

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

Can be used to completely break WEP encryption standard

Extending the Stretch of a PRG

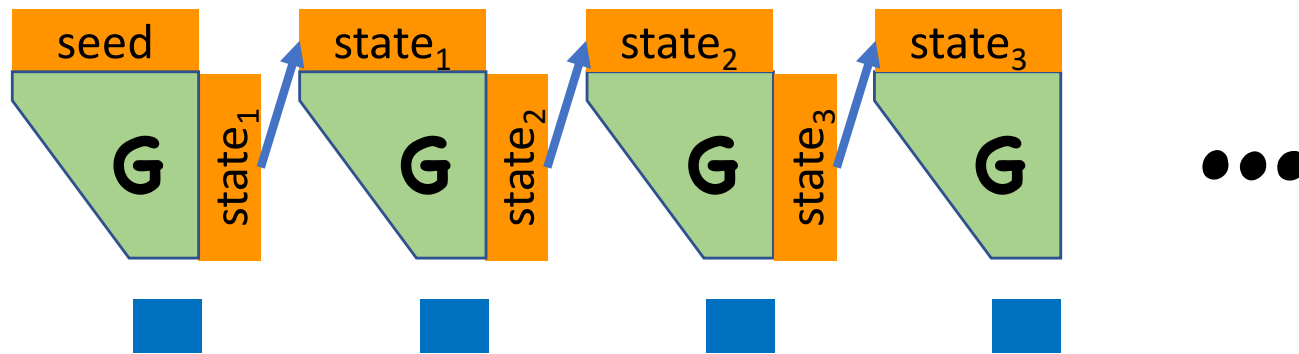
Suppose you have a fixed-stretch PRG \mathbf{G}

- Better yet, a PRG that expands by a single bit

$$\mathbf{G}: \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda+1}$$

Construct a PRG \mathbf{G}' of arbitrary output length

Extending the Stretch of a PRG

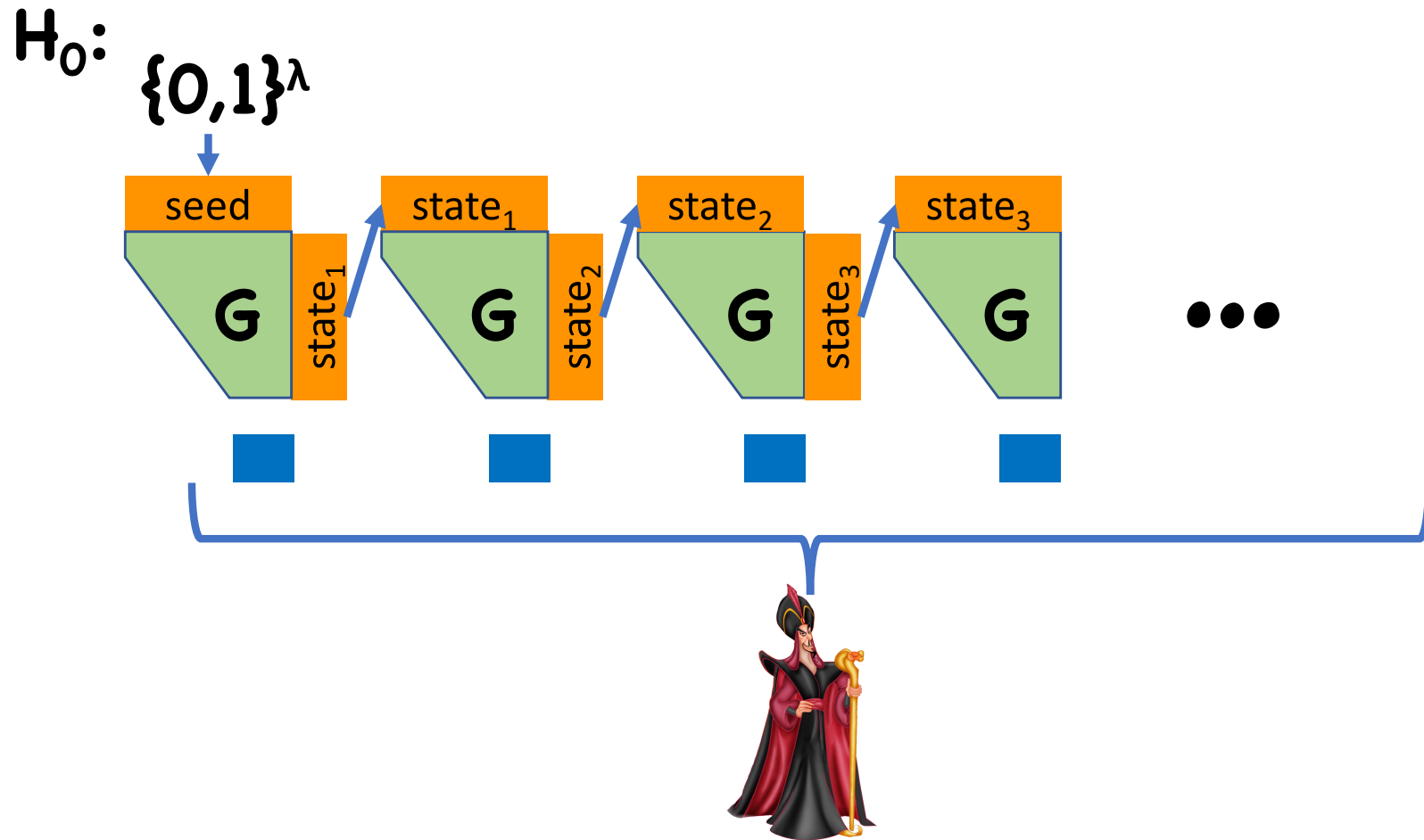


Security Proof

Assume towards contradiction PPT , non-negligible ϵ ...

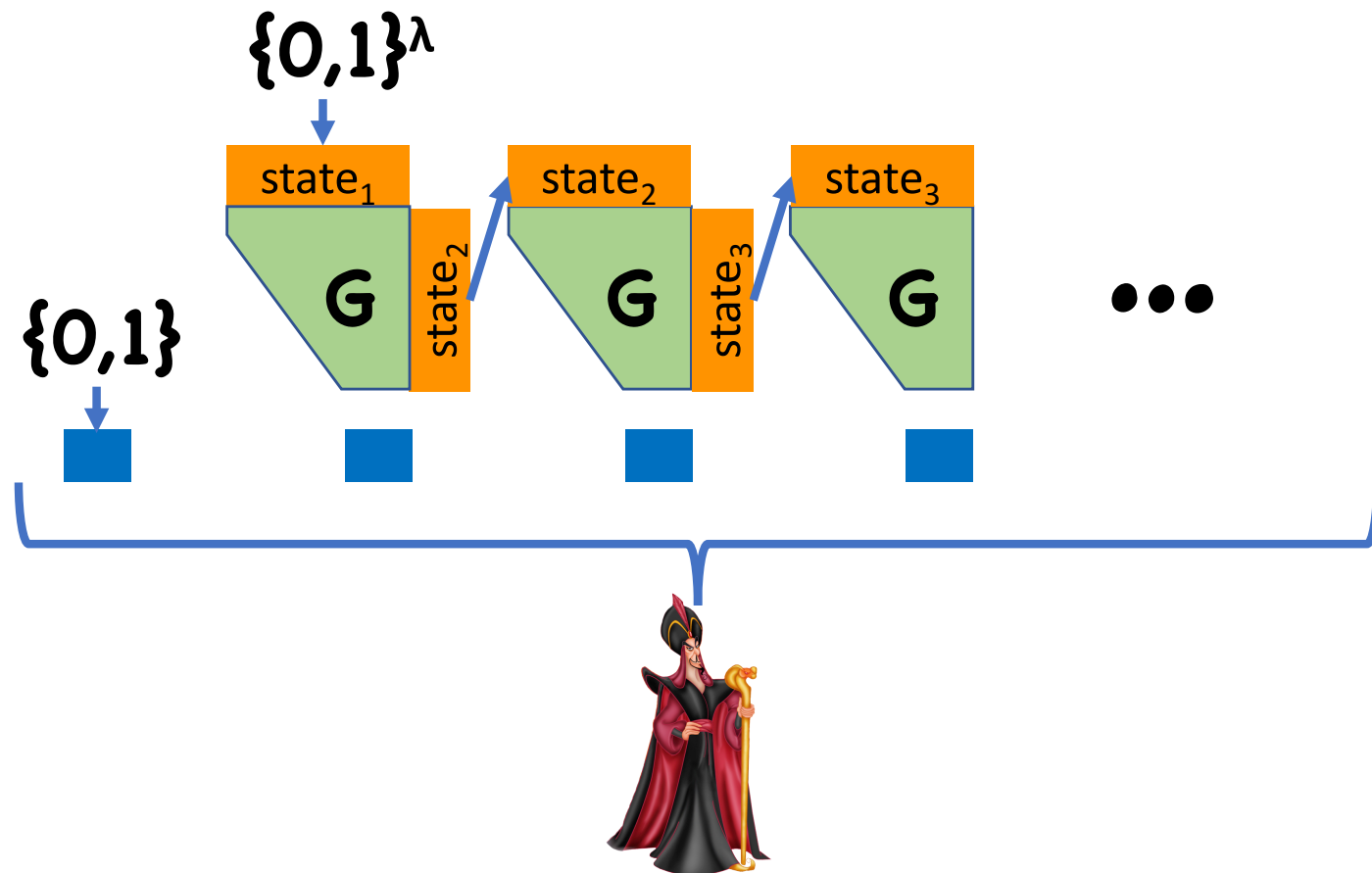
Define hybrids...

Security Proof



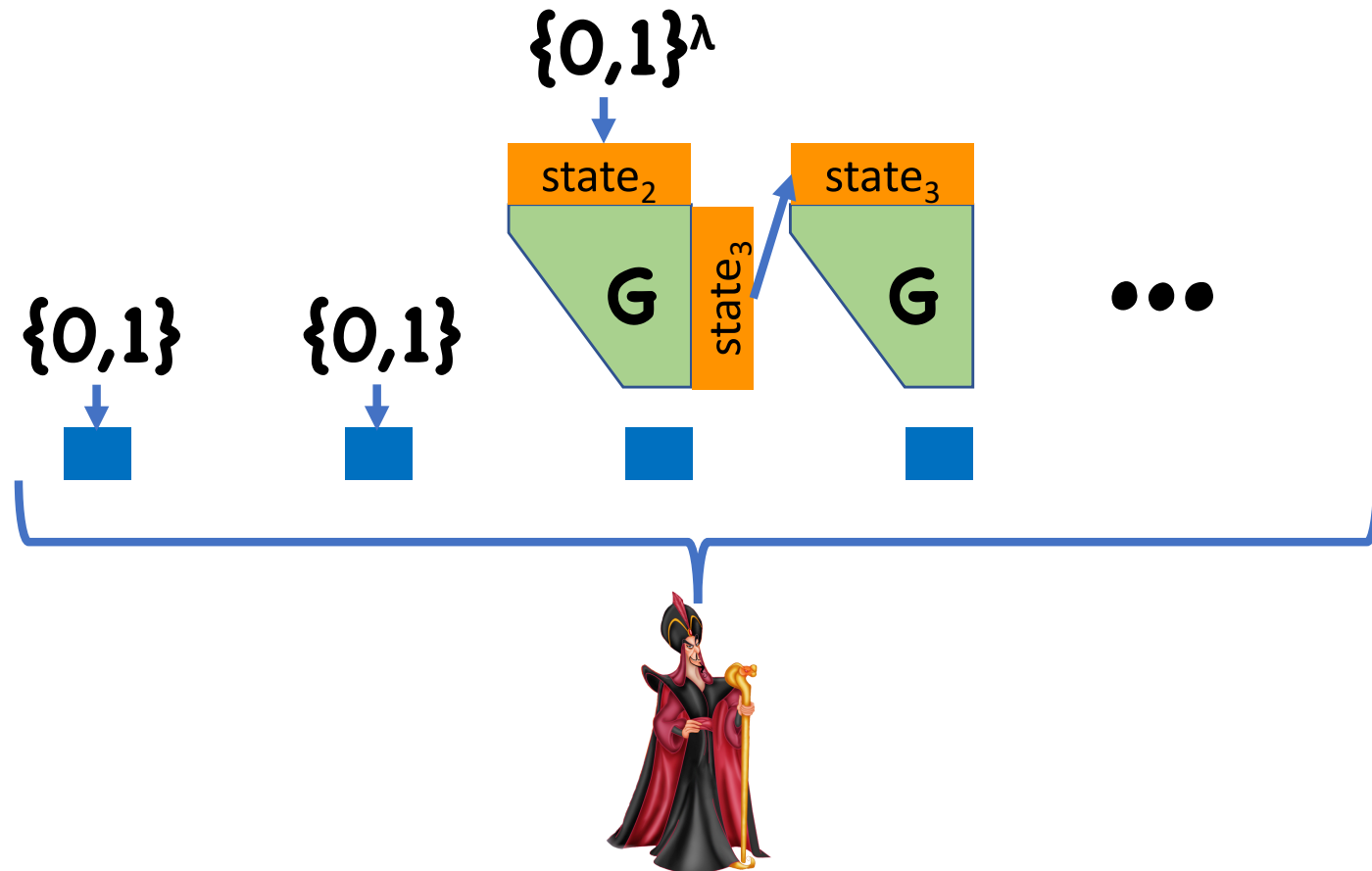
Security Proof

H_1 :



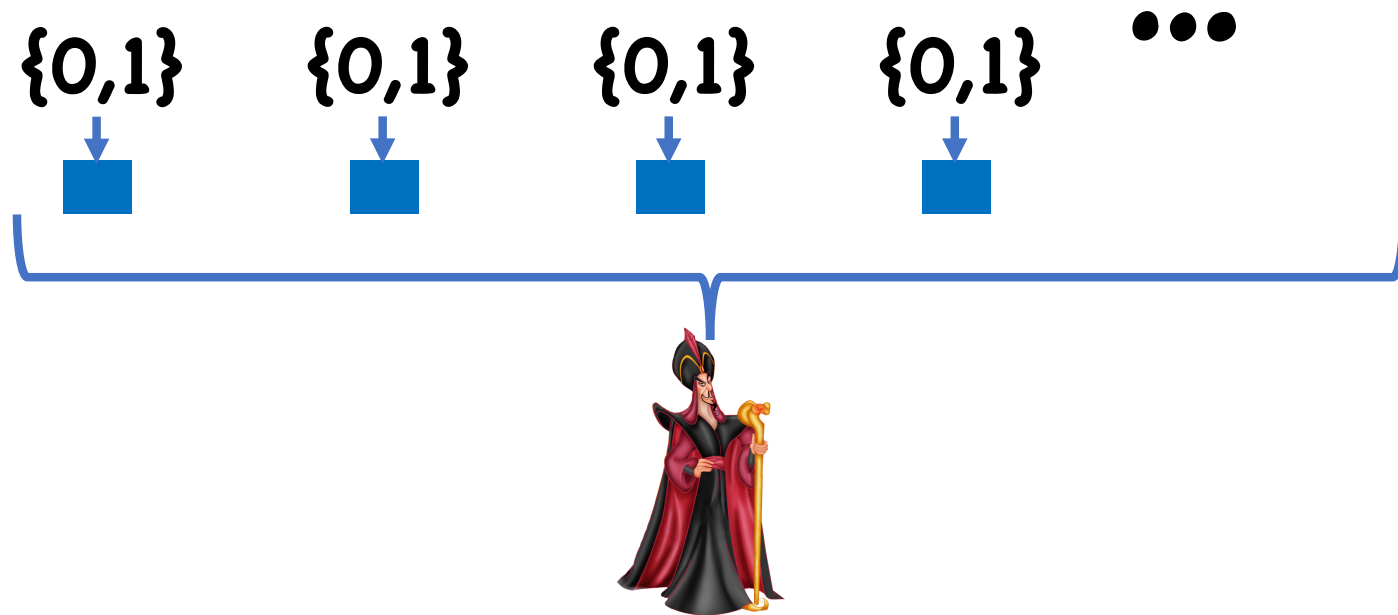
Security Proof

H_2 :



Security Proof

H_t :



Security Proof

H_0 corresponds to pseudorandom x

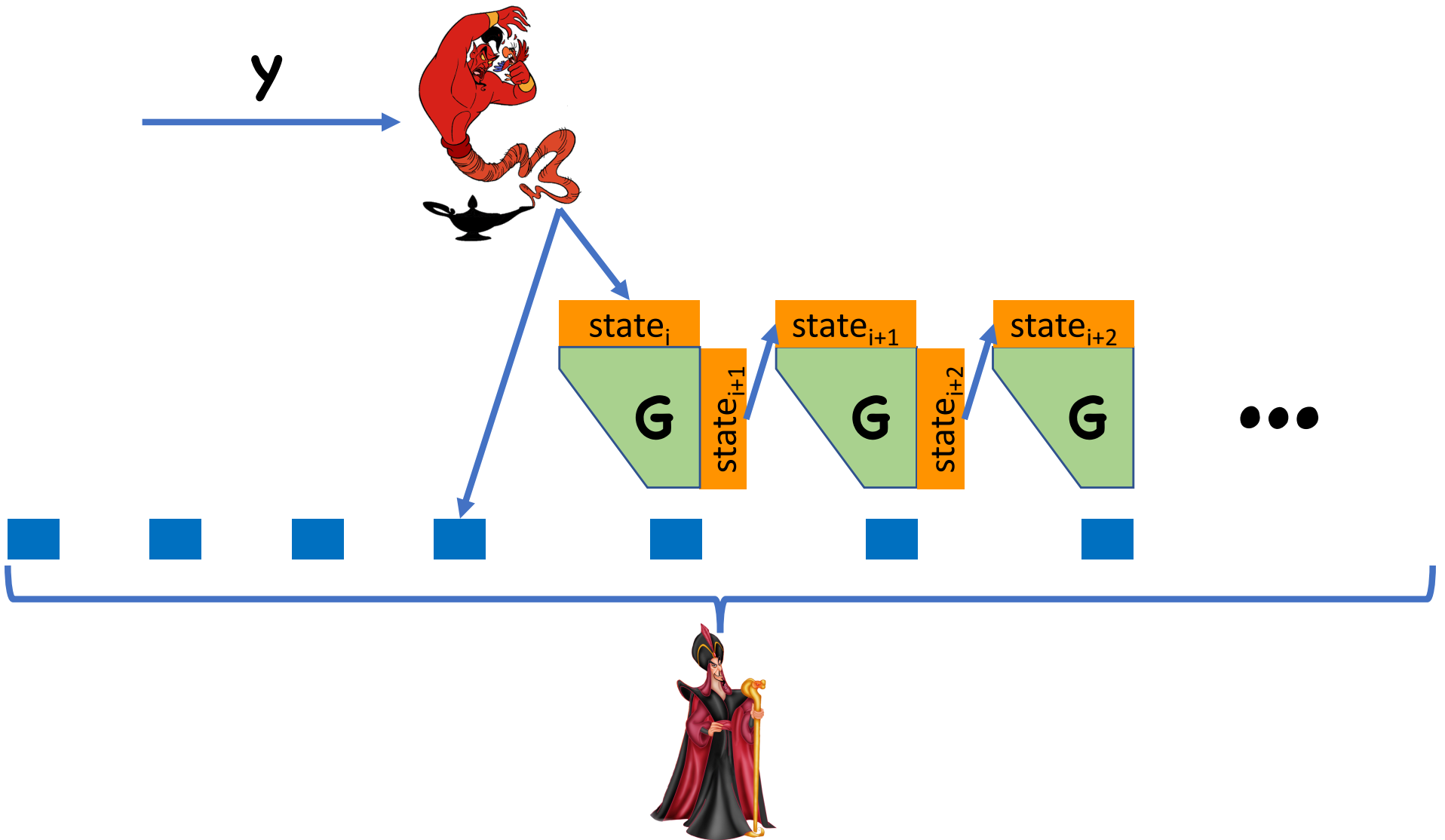
H_t corresponds to truly random x

Let $q_i = \Pr[\text{👩} (x)=1 : x \leftarrow H_i]$

By assumption, $|q_t - q_0| = \epsilon(\lambda)$







$\Rightarrow \exists i$ s.t. $|q_i - q_{i-1}| = \epsilon(\lambda)/t$

Security Proof



Security Proof

Analysis

- If $\mathbf{y} = \mathbf{G}(\mathbf{s})$, then  sees \mathbf{H}_{i-1}
 - $\Rightarrow \Pr[\text{ outputs 1}] = q_{i-1}$
 - $\Rightarrow \Pr[\text{ outputs 1}] = q_{i-1}$
- If \mathbf{y} is random, then  sees \mathbf{H}_i
 - $\Rightarrow \Pr[\text{ outputs 1}] = q_i$
 - $\Rightarrow \Pr[\text{ outputs 1}] = q_i$

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

Next Time

Stateless encryption for arbitrary messages

Pseudorandom Functions (PRFs)