Stream Ciphers

Outline

- One-Time Pad
- Perfect Secrecy
- Pseudorandom Generators (PRGs) and Stream Ciphers
- Attacks
- Security of PRGs
- Semantic Security

Symmetric Ciphers

Definition.

A (symmetric) **cipher** defined over (K, M, C)

is a pair of "efficient" algorithms **(E,D)** where

- **E:** $K \times M \to C$
- **D:** $K \times C \to M$

such that $\forall m \in M, \forall k \in K$: $\mathbf{D(k, E(k,m)) = m}$

- E is often **randomized**.
- D is **always deterministic**.

The One-Time Pad (Vernam 1917)

First example of a "secure" cipher

- $K = M = C = \{0,1\}^n$
- $E(k, m) = k \oplus m$
- $D(k, c) = k \oplus c$
- k used <u>**only once**</u>
- k is a **random** key (i.e., <u>**uniform**</u> distribution over K)

$$
\begin{array}{lccccccc}
m: & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
k: & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
c: & 1 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
\quad \oplus
$$

The One-Time Pad (Vernam 1917)

The one-time pad is a **cipher**:

- $D(k, E(k,m)) =$
- $D(k, k \oplus m) =$
- $k \oplus (k \oplus m) =$
- $(k \oplus k) \oplus m =$
- $0 \oplus m =$
- m

One-time pad definition:

- $E(k, m) = k \oplus m$
- $D(k, c) = k \oplus c$

The One-Time Pad (Vernam 1917)

- **Pro:**
  - Very **fast** encryption and decryption
- **Con:**
  - **Long keys** (as long as the plaintext),
    If Alice wants to send a message to Bob,
    she first has to transmit a key of the same length to Bob **in a secure way**.
    If Alice has a secure mechanism to transmit the key, she might use that same mechanism to transmit the message itself!

Is the OTP secure? **What is a secure cipher?**

What is a secure cipher?

Attacker's abilities: **CT only attack** (for now)

Possible security requirements:

attempt #1: **attacker cannot recover secret key**

$E(k, m) = m$ would be secure

attempt #2: **attacker cannot recover all of plaintext**

$E(k, m_0 \| m_1) = m_0 \| k \oplus m_1$ would be secure

Shannon's idea:

CT should reveal no "info" about PT

Information Theoretic Security (Shannon 1949)

Definition.

A cipher (E, D) over (K, M, C) has **perfect secrecy** if

$\forall \mathbf{m_0}, \mathbf{m_1} \in M$ with $\mathbf{len(m_0) = len(m_1)}$ and $\forall \mathbf{c} \in C$

$$\Pr[E(k, m_0)=c] = \Pr[E(k, m_1)=c]$$

where **k is uniform in K** $(k \longleftarrow K)$

Information Theoretic Security

- Given CT, can't tell if PT is m_0 or m_1 (for all m_0, m_1)
- Most powerful adversary learns nothing about PT from CT
- No CT only attack! (but other attacks are possible…)

Is OTP ''secure''?

OTP has perfect secrecy.

Proof:

$$\forall m, c \qquad \Pr_k[E(k,m) = c] \;=\; \frac{\#keys\ k \in K\ \ s.t.\ E(k,m) = c}{|K|}$$

$$\text{So if } \forall m, c \;\; \#\{k \in K : E(k,m) = c\} = const.$$

$$\Rightarrow \text{Cipher has perfect secrecy}$$

Let **m** $\in$ M and **c** $\in$ C.

How many OTP keys map **m** to **c** ?

- None
- 1
- 2
- It depends on **m**

$$
\begin{array}{lccccccc}
\text{m:} & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
\text{k:} & ? & ? & ? & ? & ? & ? & ? \\
\hline
\text{c:} & 1 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
\quad \oplus
$$

Is OTP "secure"?

OTP has perfect secrecy.

Proof:

$$\forall m, c \qquad \Pr_k[E(k,m) = c] = \frac{1}{|K|}$$

$$\text{So if } \forall m, c \;\; \#\{k \in K : E(k,m) = c\} = const.$$

$$\boxed{\Rightarrow \text{Cipher has perfect secrecy}}$$

The bad news …

- OTP drawback: **key-length=msg-length**

- Are there ciphers with perfect secrecy that use shorter keys?

  Theorem: perfect secrecy ⇒ $|K| \geq |M|$

  i.e. perfect secrecy ⇒ key-length ≥ msg-length

- Hard to use in practice!!!!

Pseudorandom Generators and Stream Ciphers

Review

Cipher over (K,M,C): a pair of "efficient" algorithms (E, D) s.t.

$\forall\, m \in M,\ \forall\, k \in K: \quad D(k, E(k, m)) = m$

Weak ciphers: substitution cipher, Vigener, …

A good cipher: **OTP** $\quad M = C = K = \{0,1\}^n$

$\mathbf{E(k, m) = k \oplus m} \quad , \quad \mathbf{D(k, c) = k \oplus c}$

OTP has perfect secrecy (i.e., no CT only attacks)

Bad news: perfect-secrecy ⇒ key-len ≥ msg-len

Stream Ciphers: making OTP practical

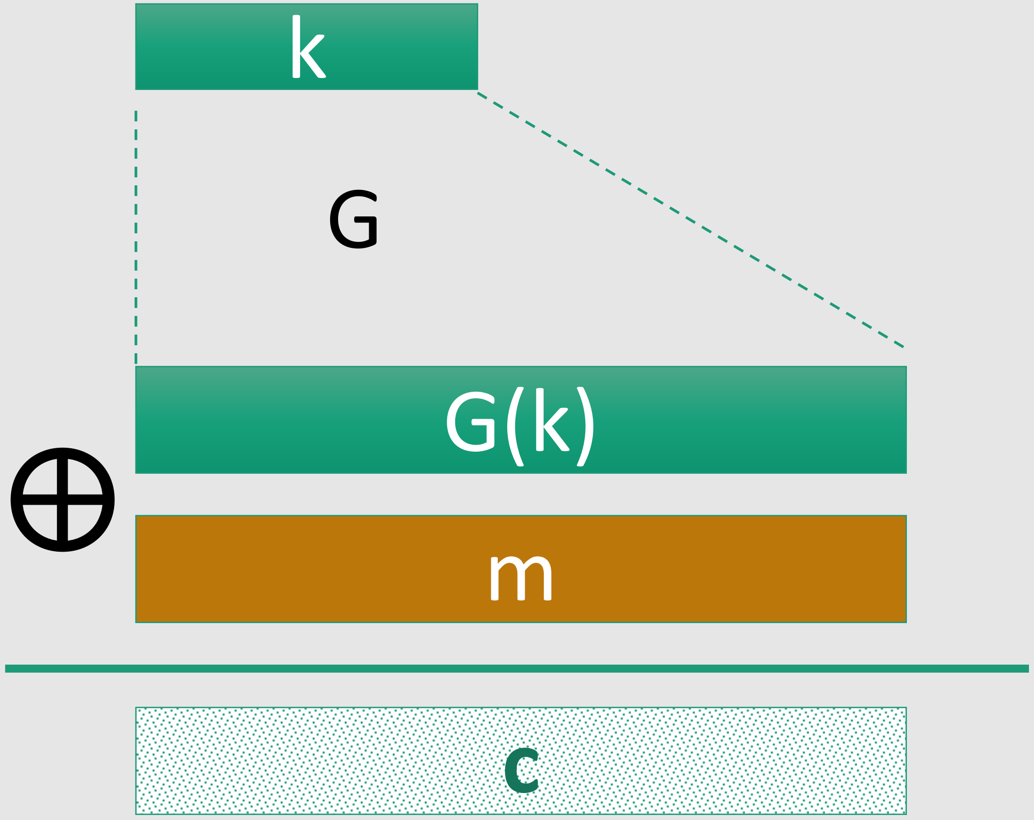
Idea: replace "**random**" key by "**pseudorandom**" key

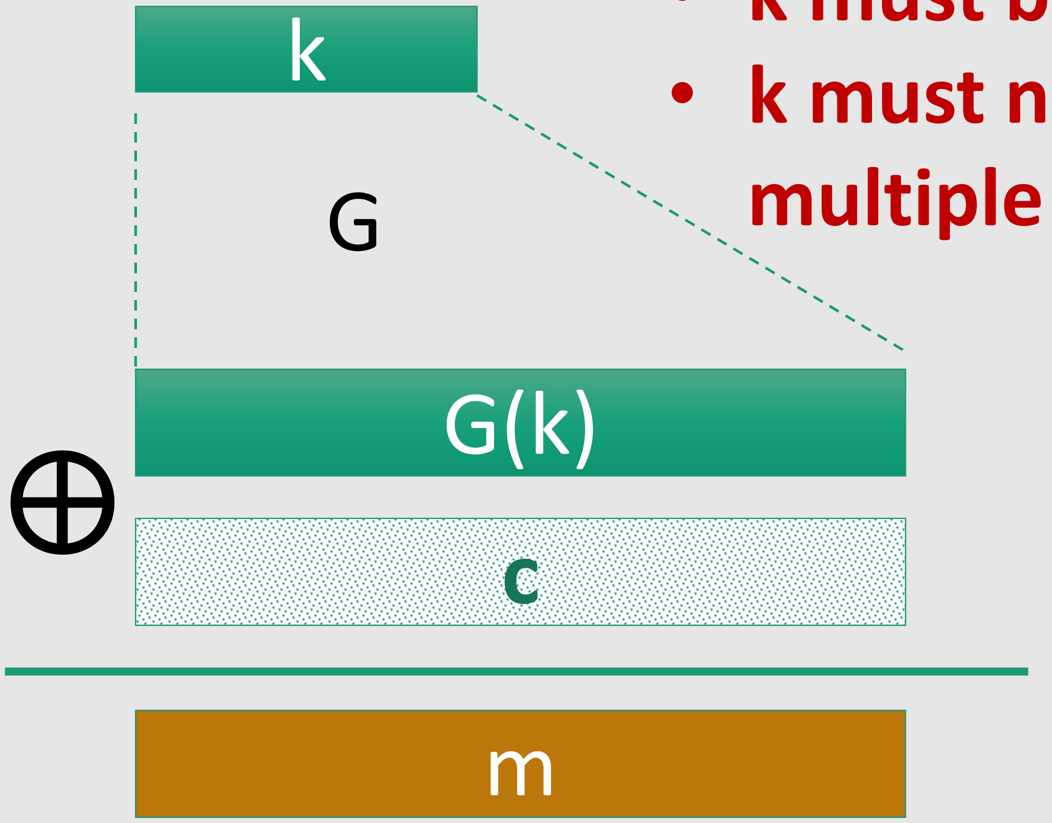
Pseudorandom Generator (PRG):

PRG is a function $G: \underbrace{\{0,1\}^s}_{\text{seed space}} \rightarrow \{0,1\}^n \qquad n >> s$

(efficiently computable by a deterministic algorithm)

Stream Ciphers: making OTP practical





- **k must be random**
- **k must not be used multiple times**

$E(k, m) = G(k) \oplus m$

$D(k, c) = G(k) \oplus c$

Can a stream cipher have perfect secrecy?

- Yes, if the PRG is really “secure”
- No, there are no ciphers with perfect secrecy
- Yes, every cipher has perfect secrecy
- No, since the key is shorter than the message

Can a stream cipher have perfect secrecy?

- Yes, if the PRG is really “secure”
- No, there are no ciphers with perfect secrecy
- Yes, every cipher has perfect secrecy
- No, since the key is shorter than the message

Stream Ciphers: making OTP practical

Stream ciphers cannot have perfect secrecy !!

- Need a different definition of security
- Security will **depend on specific PRG**

Weak PRGs (do not use for crypto)

Linear congruential generator with parameters a, b, p:
(a, b are integers, p is a prime)

r[0] := seed
r[i] ← a r[i-1] + b mod p
output few bits of r[i]
i++

has some good statistical properties
But it's easy to predict

glibc random():

r[i] ← (r[i-3] + r[i-31]) % 2^{32}
output r[i] >> 1

Do not use random() for crypto
(e.g., Kerberos v4)

Attacks on OTP and Stream Ciphers

Review

- **One-time pad**:
  - $E(k,m) = \mathbf{k} \oplus m$
  - $D(k,c) = \mathbf{k} \oplus c$

- **k** is random (**uniform**)
- **k** used only once

- **Stream ciphers**
  making OTP practical using a **PRG** $G: K \longrightarrow \{0,1\}^n$
  - $E(k,m) = \mathbf{G(k)} \oplus m$
  - $D(k,c) = \mathbf{G(k)} \oplus c$

Attack 1: **two time** pad is insecure !!

Never use stream cipher **key more than once** !!

$$c_1 \leftarrow m_1 \oplus PRG(k)$$
$$c_2 \leftarrow m_2 \oplus PRG(k)$$

Eavesdropper does:

$$c_1 \oplus c_2 \rightarrow$$

Enough redundancy in English and ASCII encoding that:

$$m_1 \oplus m_2 \rightarrow m_1 , m_2$$

Real-world examples

- Project Venona (1941 – 1946)

Real-world examples

- Project Venona (1941 – 1946)

- MS-PPTP (windows NT):

$k = (k_{C \to S}, k_{S \to C})$

k

m_1

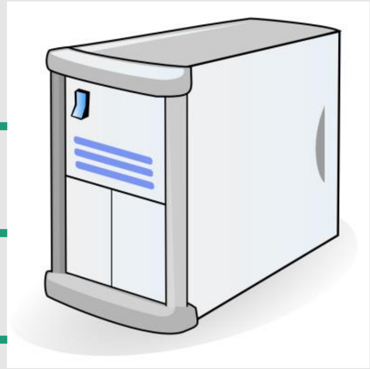
m_2

m_3

s_1

s_2

s_3



k

$[\, m_1 \,||\, m_2 \,||\, m_3 \,] \oplus \boxed{PRG(k)}$

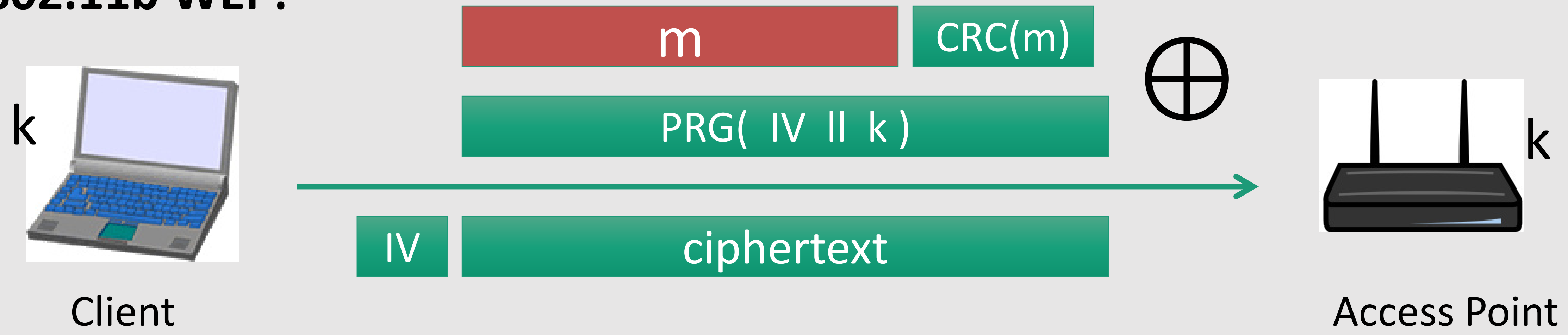
$[\, s_1 \,||\, s_2 \,||\, s_3 \,] \oplus \boxed{PRG(k)}$

Need different keys for $C \longrightarrow S$ and $S \longrightarrow C$

Real-world examples

k: LONG-TERM KEY

802.11b WEP:

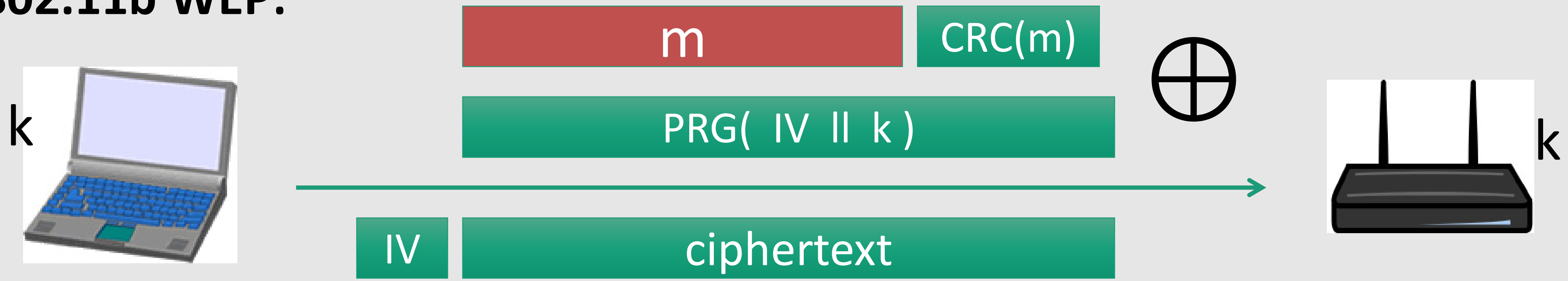


Length of IV: 24 bits

- Repeated IV after $2^{24} \approx$ 16M frames
- On some 802.11 cards: IV resets to 0 after power cycle

Avoid related keys

802.11b WEP:



24 bits 104 bits

key for frame #1: (1 ll k)

key for frame #2: (2 ll k)

⋮

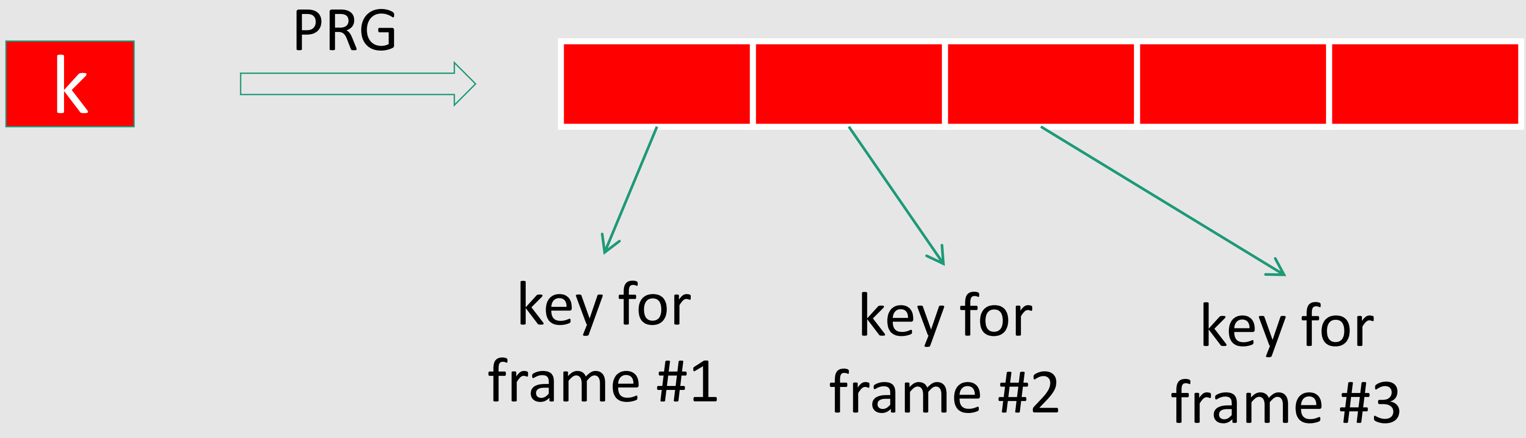
Very related keys!!
Not random keys!

The PRG used in WEP (called RC4) is not secure for such related keys

- Attack that can recover k after 10^6 frames (FMS 2001)
- Recent attack => 40.000 frames

A better construction

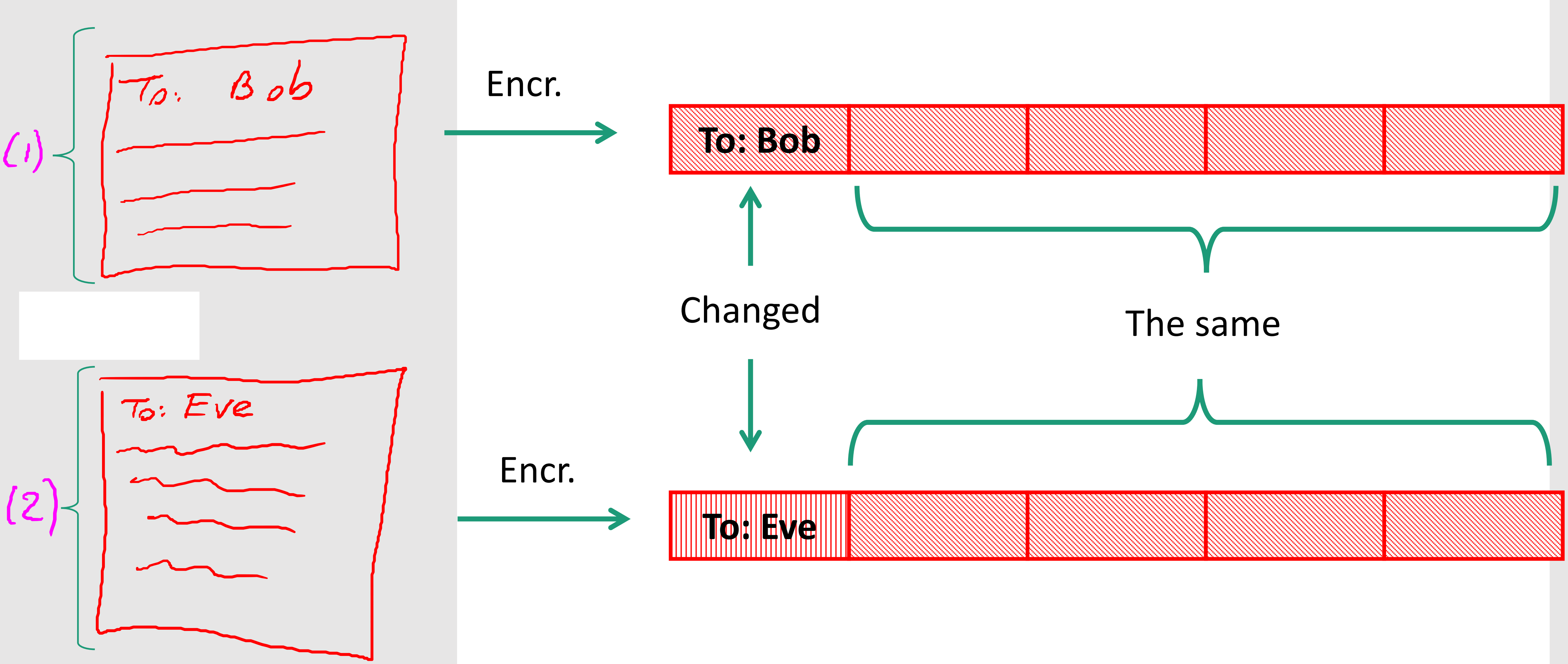
k



⇒ now each frame has a pseudorandom key

better solution: use stronger encryption method (as in WPA2)

Yet another example: disk encryption



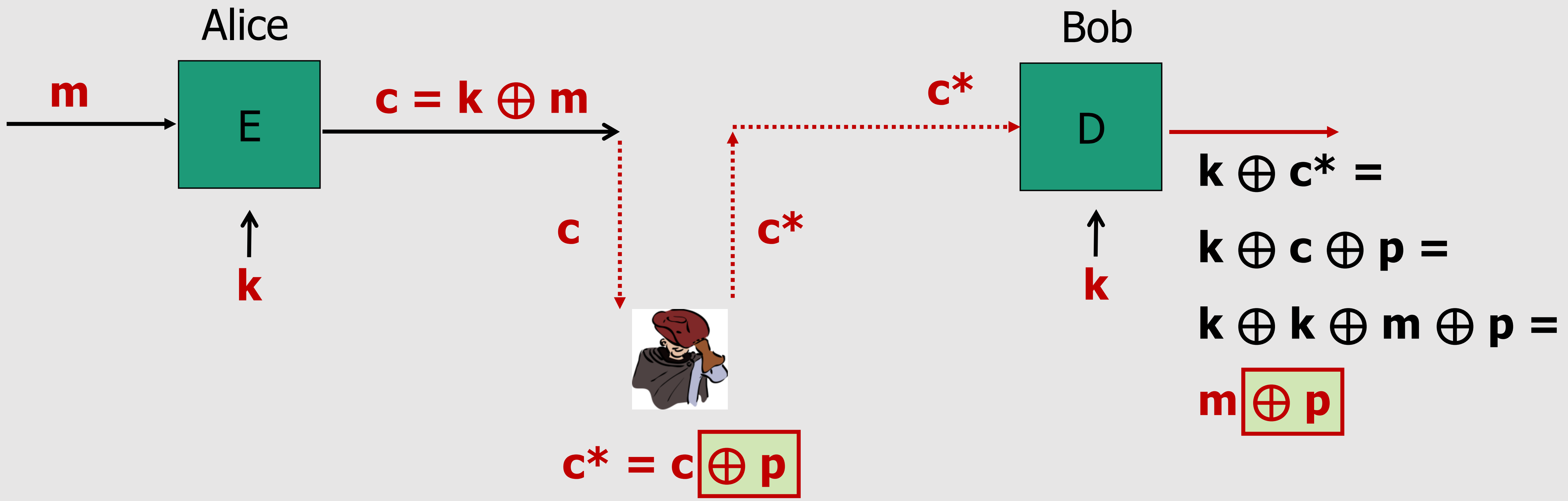
Two time pad: summary

Never use stream cipher key **more than once** !!

- Network traffic: negotiate new key for every session (e.g. TLS)
  - One key (or ''sub-key'') for traffic **from Client to Server**
  - One key (or ''sub-key'') for traffic **from Server to Client**

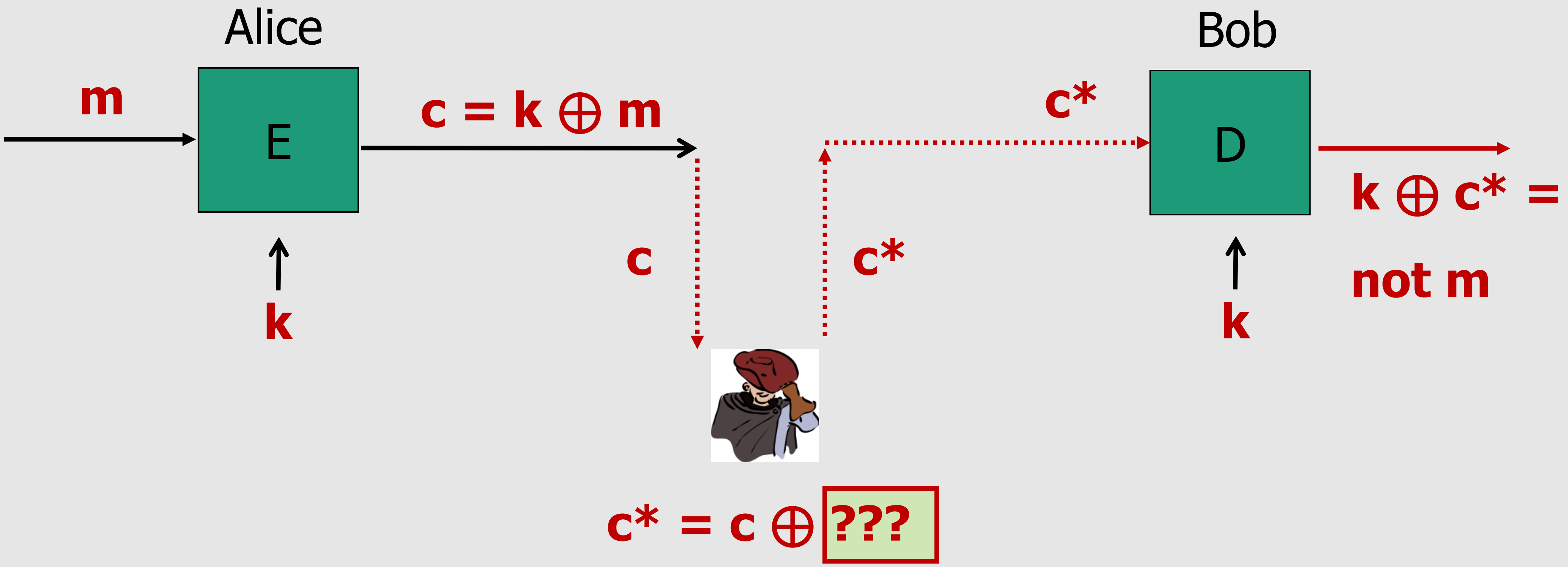
- Disk encryption: typically do not use a stream cipher

Attack 2: no integrity (OTP is **malleable**)



Modifications to ciphertext are **<u>undetected</u>** and have **<u>predictable</u>** impact on plaintext

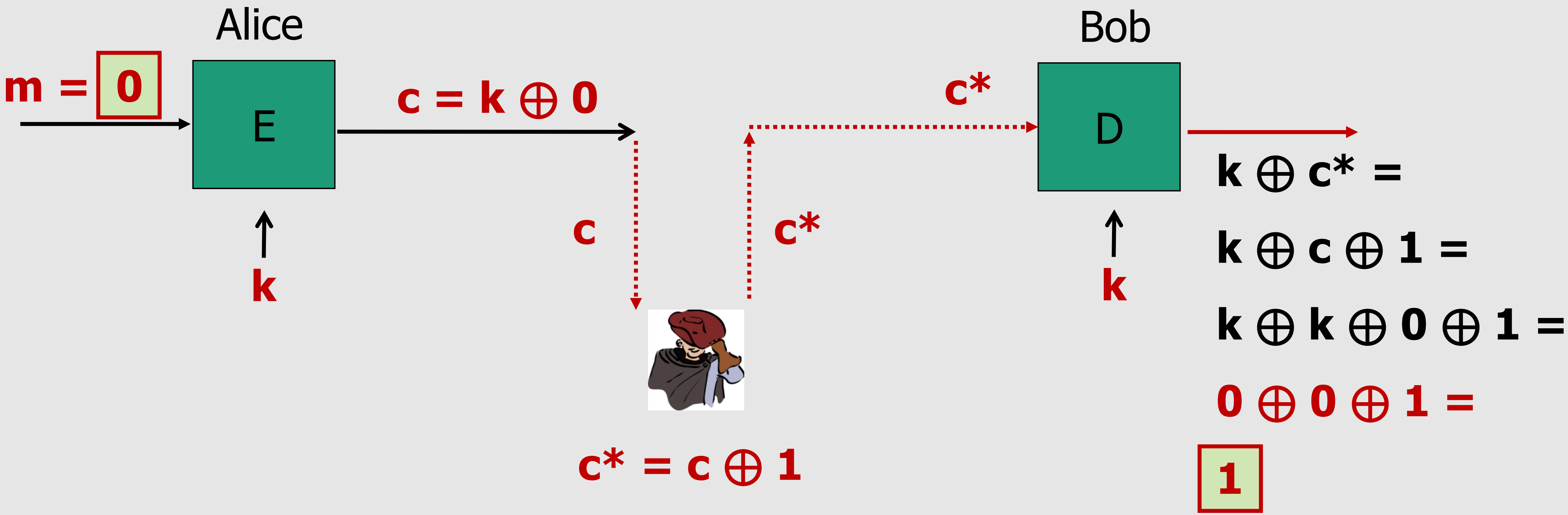
Attack 2: no integrity (OTP is **malleable**)



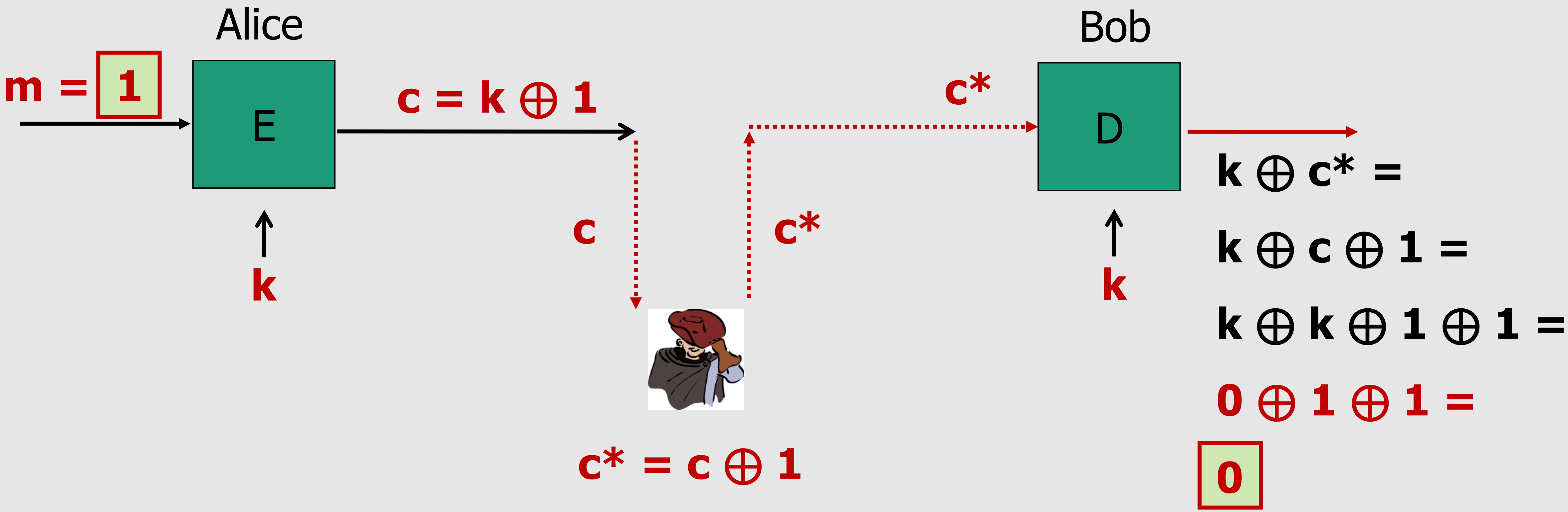
- Alice has to answer yes (**1**) or no (**0**) to Bob's invitation. She'll encrypt the answer with OTP.
- The attacker cannot recover Alice's answer from CT.
- **Still, can the attacker "flip" Alice's answer?**

  Yes !! Apply ⊕ 1 to the intercepted CT

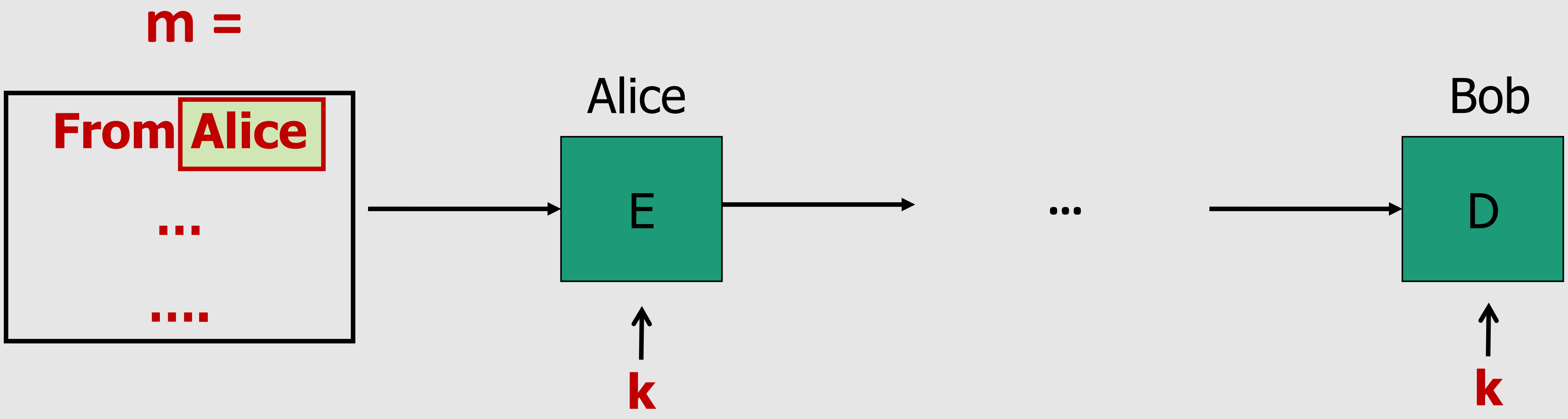
Attack 2: no integrity (OTP is **malleable**)



Attack 2: no integrity (OTP is **malleable**)



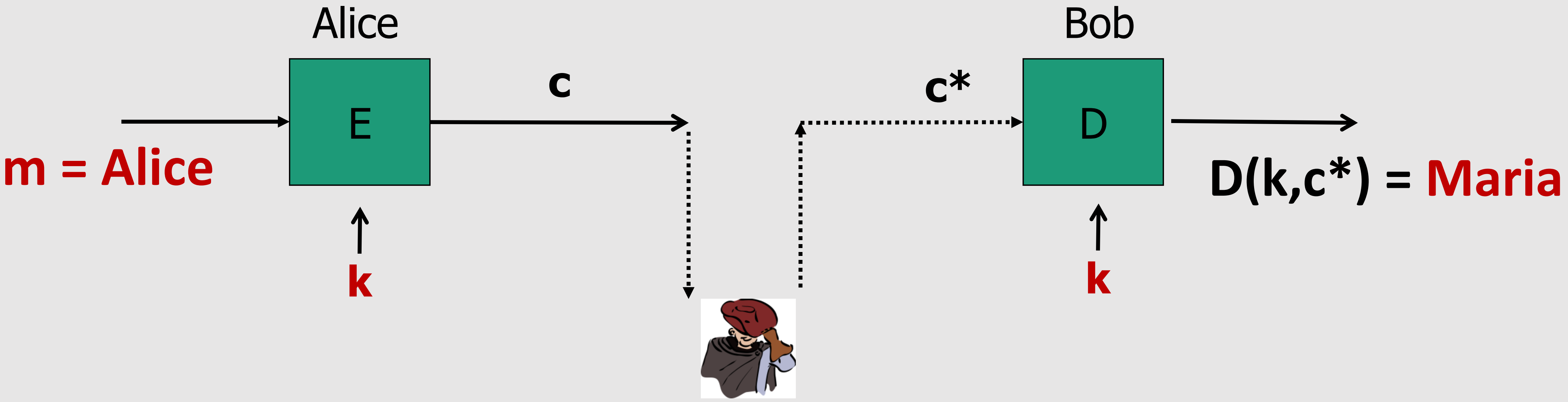
Attack 2: no integrity (OTP is **malleable**)



Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)

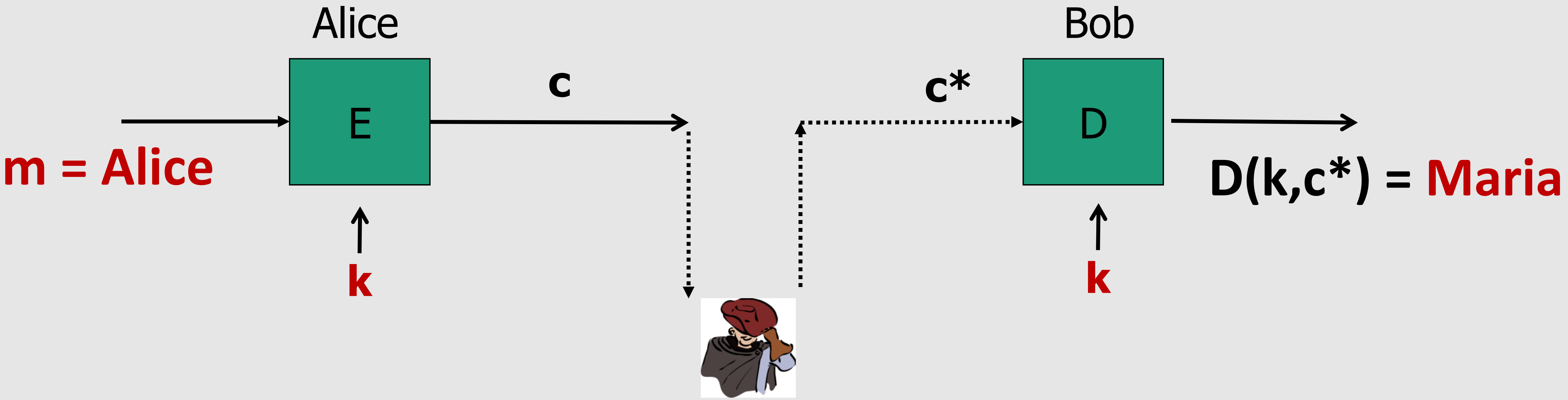


$c^* = c \oplus \boxed{???}$

Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)

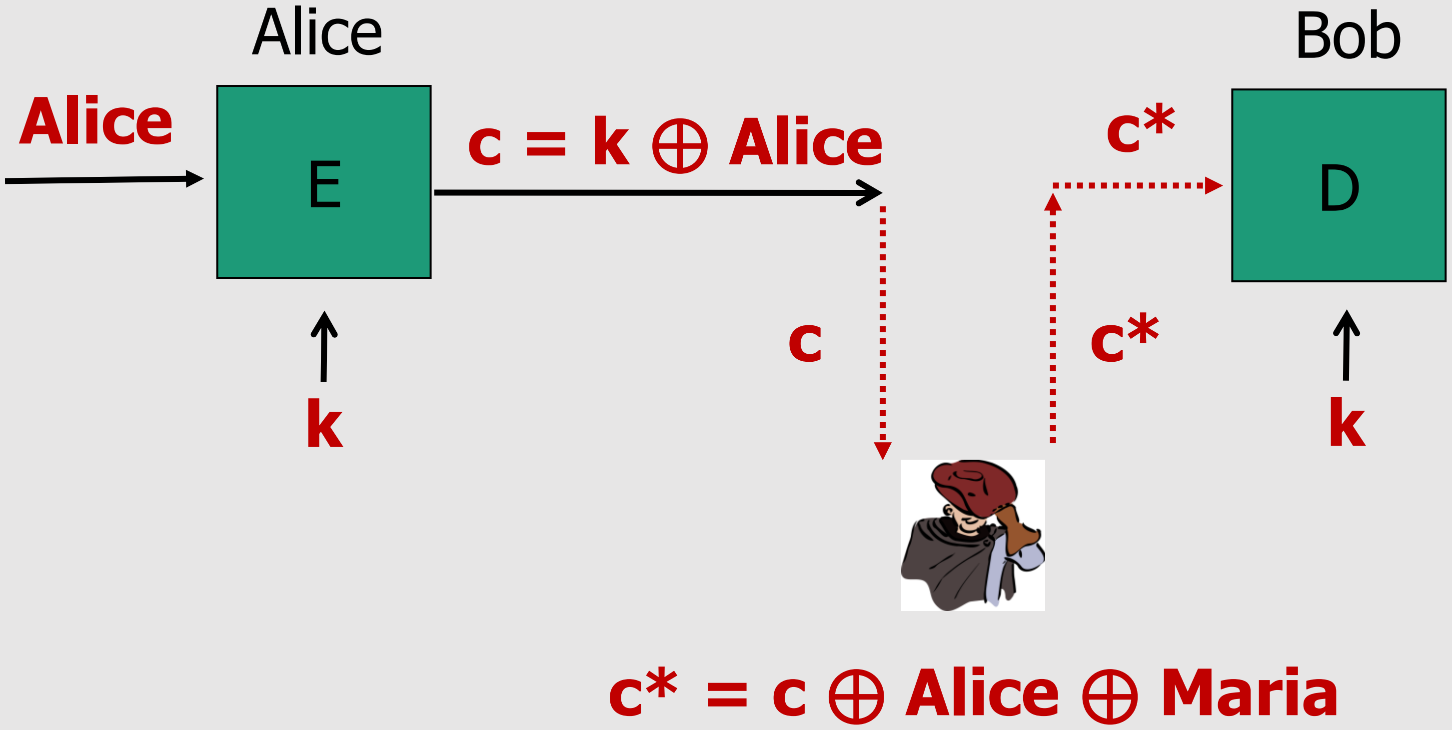


$$c^* = c \oplus \boxed{\text{Alice} \oplus \text{Maria}}$$

Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)



k ⊕ c* =

k ⊕ c ⊕ Alice ⊕ Maria =

k ⊕ k ⊕ Alice ⊕ Alice ⊕ Maria =

0 ⊕ Alice ⊕ Alice ⊕ Maria =

0 ⊕ 0 ⊕ Maria =

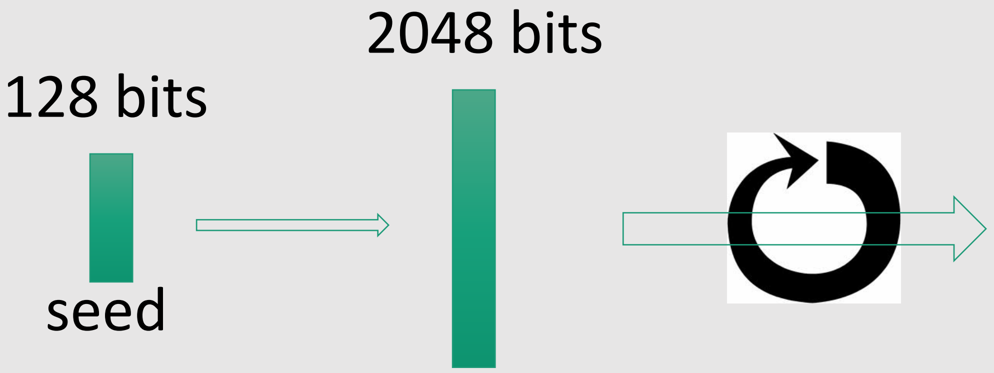
Maria

Consider the bank account number in a wire transfer…

Real-world Stream Ciphers

Old example (software): RC4 (1987)

Variable size seed
(e.g., 128 bits)



1 byte
per round

- Used in HTTPS and WEP

RC4 PRG

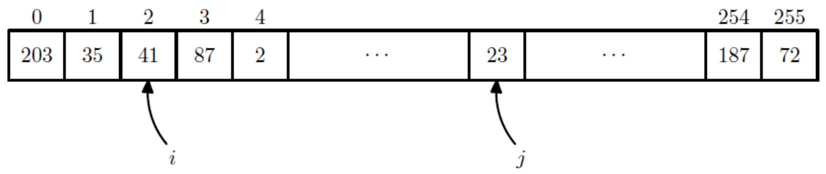


Figure 3.12: An example RC4 internal state

The RC4 stream cipher key s is a seed for the PRG and is used to initialize the array S to a pseudo-random permutation of the numbers 0 : : : 255. Initialization is performed using the following **setup algorithm**:

input: string of bytes s

for $i \leftarrow 0$ to 255 do: $\quad S[i] \leftarrow i$

$j \leftarrow 0$

for $i \leftarrow 0$ to 255 do

$\quad k \leftarrow s\big[i \bmod |s|\big] \quad$ // *extract one byte from seed*

$\quad j \leftarrow \big(\, j + S[i] + k \,\big) \bmod 256$

$\quad \mathrm{swap}(S[i], S[j])$

During the loop the index i runs linearly through the array while the index j jumps around. At each iteration the entry at index i is swapped with the entry at index j.

RC4 PRG

Once the array S is initialized, the PRG generates pseudo-random output one byte at a time using the following **stream generator**:

$i \leftarrow 0, \quad j \leftarrow 0$

repeat

  $i \leftarrow (i+1) \bmod 256$

  $j \leftarrow (j + S[i]) \bmod 256$

  $\mathrm{swap}(S[i], S[j])$

  output $\; S\big[\; (S[i] + S[j]) \bmod 256 \;\big]$

forever

The procedure runs for as long as necessary. Again, the index i runs linearly through the array while the index j jumps around. Swapping S[i] and S[j] continuously shuffles the array S.

Security of RC4

Weaknesses:

1. Bias in initial output: let us assume that the RC4 **setup algorithm is perfect** and generates a uniform permutation from the set of all 256! permutations.
Mantin and Shamir showed that, even assuming perfect initialization, the output of RC4 is biased: Pr[2^{nd} byte = 0] = 2/256 → RC4-drop[n]

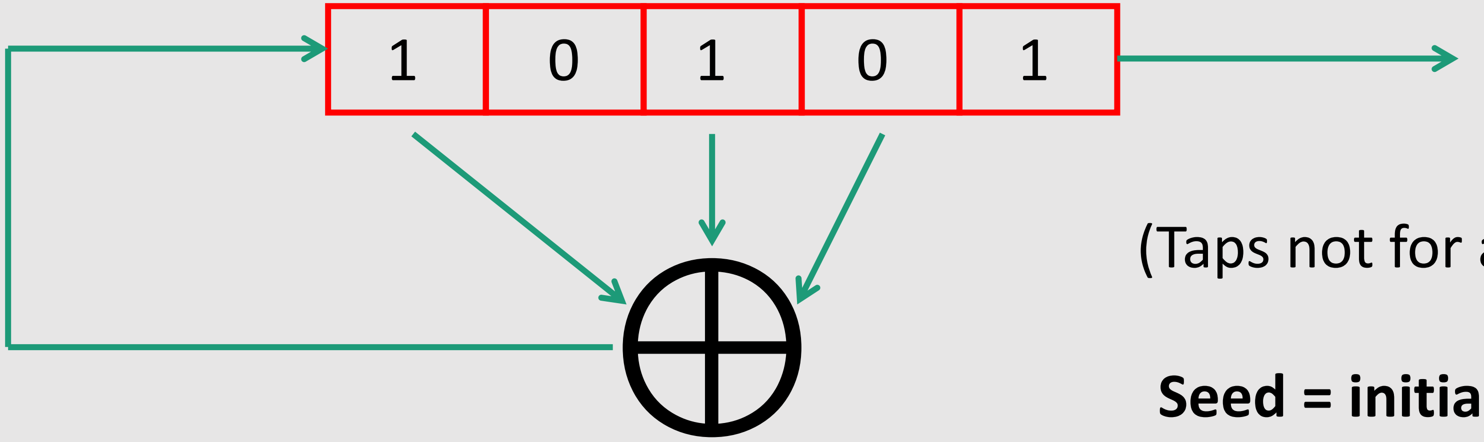
2. Fluhrer and McGrew: Prob. of (0,0) is $1/256^2 + 1/256^3$

3. Related key attacks: attack on WEP

Old example (hardware): CSS (badly broken)

Content Scrambling System

Linear feedback shift register (LFSR):



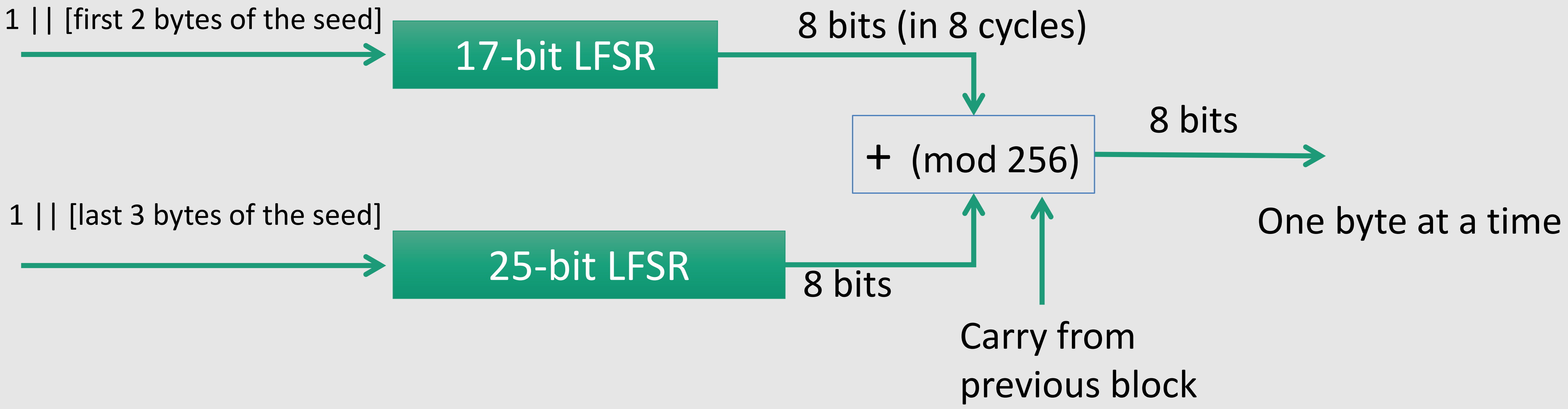
(Taps not for all cells)

Seed = initial state of the LFSR

DVD encryption (CSS):	2 LFSRs	all broken
GSM encryption (A5/1,2):	3 LFSRs	
Bluetooth (E0):	4 LFSRs	

Old example (hardware): CSS (badly broken)

CSS: seed = 5 bytes = 40 bits



Easy to break in time $\approx 2^{17}$

Modern stream ciphers: eStream

PRG: $\{0,1\}^s \times R \longrightarrow \{0,1\}^n$ $\quad n >> s$

Seed ***Nonce***

Nonce: a non-repeating value for a given key, that is

a pair (k,r) is never used more than once

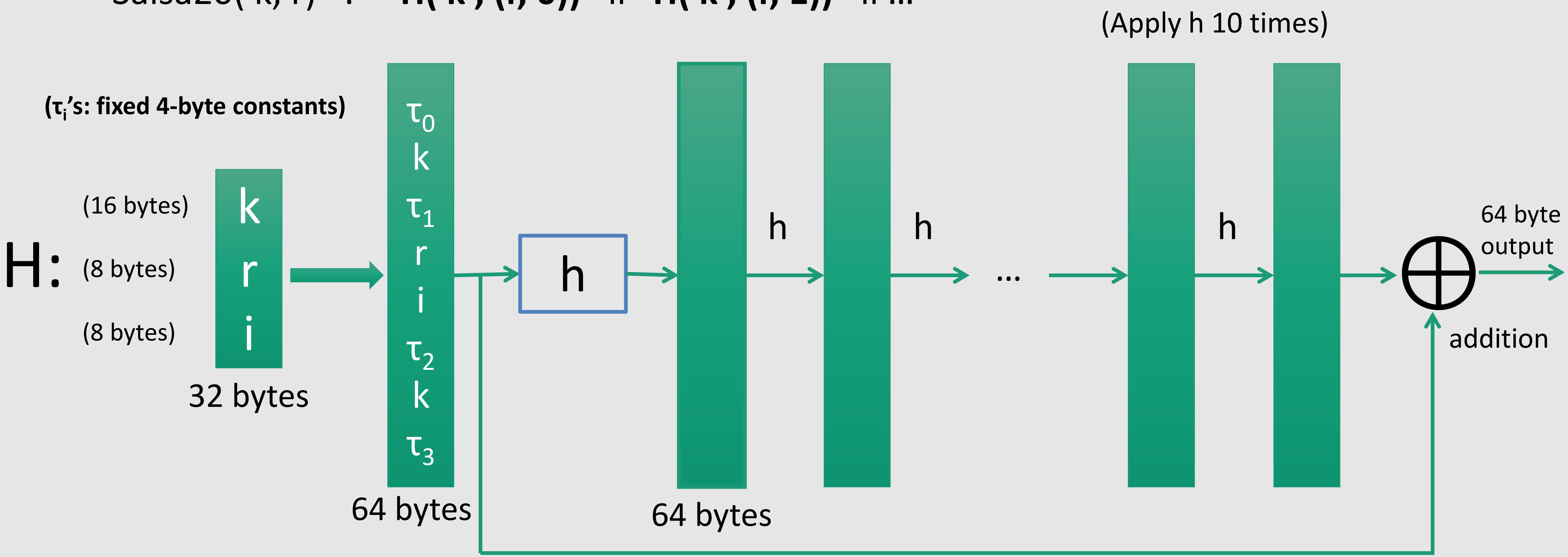
=> can re-use the key as long as the nonce changes

$E(k, m, r) = m \oplus PRG(k, r)$

eStream: Salsa 20 (SW+HW)

Salsa20: $\{0,1\}^{128 \text{ or } 256} \times \{0,1\}^{64} \longrightarrow \{0,1\}^n$ (max n = 2^{73} bits)

Salsa20(k, r) := **H(k , (r, 0)) ll H(k , (r, 1)) ll …**



h: invertible function. designed to be fast on x86 (SSE2)

Performance: Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	PRG	Speed (MB/sec)
	RC4	126
eStream	Salsa20/12	643
eStream	Sosemanuk	727

When is a PRG ‘‘secure’’?

When is a PRG “secure”?

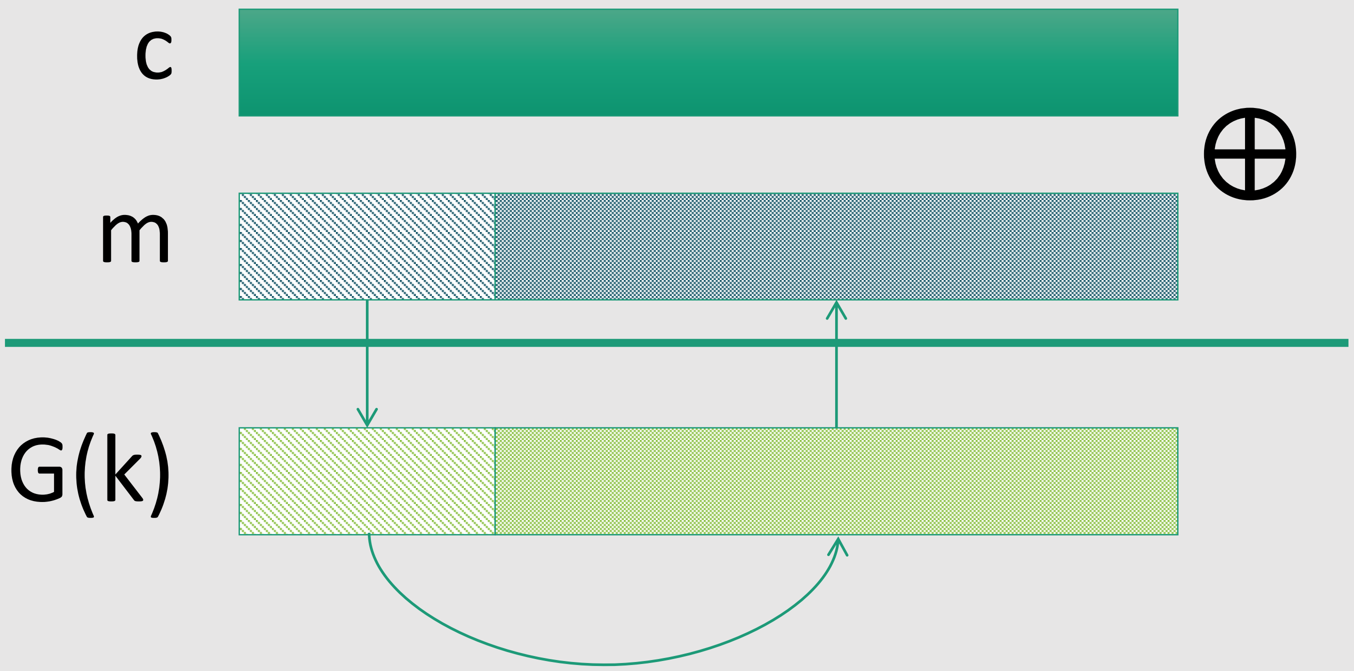
1. **Unpredictable** PRG
2. **Secure** PRG

We’ll see that they are **equivalent** notions

PRG must be unpredictable

Suppose PRG is **predictable**:

$$\exists i : \quad G(k)|_{1,\ldots,i} \xrightarrow{Alg} G(k)|_{i+1,\ldots,n}$$



Even

$$G(k)|_{1,\ldots,i} \xrightarrow{Alg} G(k)|_{i+1}$$

is a problem

PRG must be unpredictable

We say that G: K ⟶ $\{0,1\}^n$ is **predictable** if:

$\exists$ "efficient" algorithm A and $\exists 1 \leq i \leq n-1$ s.t.

$$\Pr_{k \leftarrow K}\left[A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}\right] > \frac{1}{2} + \epsilon$$

for non-negligible ϵ (e.g., $\epsilon = \frac{1}{2^{30}}$)

PRG is **unpredictable** if it **is not predictable**

⇒ ∀i: no "efficient" adversary can predict bit (i+1) for "non-neg" ε

- Suppose $G{:}K \longrightarrow \{0,1\}^n$ is such that for all **k**: **XOR(G(k)) = 1**
- Is G predictable ??

1. Yes, given the first bit I can predict the second
2. No, G is unpredictable
3. Yes, given the first (n-1) bits I can predict the n-th bit
4. It depends

- Suppose $G{:}K \longrightarrow \{0,1\}^n$ is such that for all **k**: **XOR(G(k)) = 1**
- Is G predictable ??

1. Yes, given the first bit I can predict the second
2. No, G is unpredictable
3. Yes, given the first (n-1) bits I can predict the n-th bit
4. It depends

One more definition of ''secure'' PRG

Let **G:K ⟶ {0,1}^{n}** be a PRG

Goal:

define what it means that

$$[k \leftarrow K, \ \text{output } G(k)]$$

is "indistinguishable" from

$$[r \leftarrow \{0,1\}^n, \ \text{ouput } r]$$

G: $\{0,1\}^{10} \longrightarrow \{0,1\}^{1000}$

[k ⟵ $\{0,1\}^{10}$, output G(k)]

[r ⟵ $\{0,1\}^{1000}$, output r]

Note

A minimum security requirement for a PRG is that

the length **s** of the random seed should be **sufficiently large**

so that a search over $\mathbf{2^s}$ elements (the total number of possible seeds) is infeasible for the adversary.

Statistical Tests

Statistical test on $\{0,1\}^n$:

An algorithm A s.t. A(x) outputs "0" or "1",
that is $\mathbf{A : \{0,1\}^n \longrightarrow \{0,1\}}$

Examples:

1. A(x)=1 iff $|\#0(x) - \#1(x)| \leq 10\sqrt{n}$
2. A(x)=1 iff $|\#00(x) - n/4| \leq 10\sqrt{n}$
3. A(x)=1 iff $\text{max-run-of-0}(x) < 10 \log_2(n)$

.....

Advantage

- Let **G:K ⟶$\{0,1\}^n$** be a **PRG**
- Let **A: $\{0,1\}^n$ ⟶ {0,1}** be a **statistical test** on $\{0,1\}^n$

Define: $$Adv_{PRG}[A,G] = \left| \Pr_{k \leftarrow K}[A(G(k)) = 1] - \Pr_{r \leftarrow \{0,1\}^n}[A(r) = 1] \right| \in [0,1]$$

- Adv close to 0 => A cannot distinguish G from random
- Adv non-negligible => A can distinguish G from random
- Adv close to 1 => A can distinguish G from random very well

A silly example: $A(x) = 0 \;\Rightarrow\; Adv_{PRG}[A,G] =$

Example of Advantage

- Suppose $G:K \longrightarrow \{0,1\}^n$ satisfies **msb(G(k)) = 1** for 2/3 of keys in K
- Define statistical test $A(x)$ as:

if [msb(x)=1] output "1" else output "0"

Then

$$\mathrm{Adv}_{\mathrm{PRG}}[A,G] = \Big|\, \Pr[\,A(G(k))=1] - \Pr[\,A(r)=1\,] \,\Big| =$$

$$\Big|\, 2/3 - 1/2 \,\Big| = 1/6$$

A breaks G with advantage 1/6 (which is not negligible)
hence **G is not a good PRG**

Secure PRGs: crypto definition

Definition:

We say that **G : K ⟶ {0,1}^{n}** is a **secure PRG** if

for every "*efficient*" statistical test **A**, **Adv_{PRG}[A,G] is "negligible"**

Are there provably secure PRGs? Unknown (=> P ≠ PN)

A secure PRG is unpredictable

We show: PRG predictable $\Rightarrow$ PRG is insecure

Suppose A is an efficient algorithm s.t.

$$\Pr_{k \leftarrow K}\left[A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}\right] > \frac{1}{2} + \epsilon$$

for non-negligible ε (e.g. ε = 1/1000)

A secure PRG is unpredictable

Define statistical test B as:

$$B(X) = \begin{cases} \text{if } A(X|_{1,\ldots,i}) = X_{i+1} \text{ output } 1 \\ \text{else output } 0 \end{cases}$$

$$k \leftarrow K : \; Pr[B(G(k)) = 1] > \frac{1}{2} + \epsilon$$

$$r \leftarrow \{0,1\}^n : \; Pr[B(r) = 1] = \frac{1}{2}$$

$$\Rightarrow Adv_{PRG}[B, G] = |Pr[B(G(k)) = 1] - Pr[B(r) = 1]| > \epsilon$$

Thm (Yao'82): an unpredictable PRG is secure

Let **G : K ⟶ {0,1}^{n}** be **PRG**

"Thm": if **∀ i ∈ {0, … , n-1}** **G** is **unpredictable** at position **i**

then **G** is a **secure PRG**.

If next-bit predictors cannot distinguish G from random

then no statistical test can !!

More Generally

Let $\mathbf{P_1}$ and $\mathbf{P_2}$ be two distributions over $\{0,1\}^n$

We say that $\mathbf{P_1}$ and $\mathbf{P_2}$ are **computationally indistinguishable** (denoted $P_1 \approx_p P_2$)

$$\text{if } \forall \text{ "efficient" statistical test } A$$

$$\left| \Pr_{X \leftarrow P_1}[A(X)=1] - \Pr_{X \leftarrow P_2}[A(X)=1] \right| < \text{ negligible}$$

Example: a PRG is secure if **$\{ k \longleftarrow K : G(k) \} \approx_p \text{uniform}(\{0,1\}^n)$**

Semantic Security

What is a secure cipher?

Attacker's abilities: **CT only attack: obtains one ciphertext**

Possible security requirements:

attempt #1: **attacker cannot recover secret key**

$E(k, m) = m$ would be secure

attempt #2: **attacker cannot recover all of plaintext**

$E(k, m_0 \| m_1) = m_0 \| k \oplus m_1$ would be secure

Shannon's idea:

CT should reveal no "info" about PT

Recall Shannon's perfect secrecy

Let (E,D) be a cipher over (K,M,C)

Shannon's perfect secrecy:

(E,D) has perfect secrecy if $\forall\ m_0, m_1 \in M \quad (\ |m_0| = |m_1|\ )$

$$\{ E(k,m_0) \} = \{ E(k,m_1) \} \quad \text{where} \quad k \longleftarrow K$$

- The two distributions must be **identical**
- Too strong definition
- It requires long keys
- Stream Ciphers can't satisfy it

Weaker Definition:

(E,D) has perfect secrecy if $\forall\ m_0, m_1 \in M \quad (\ |m_0| = |m_1|\ )$

$$\{ E(k,m_0) \} \approx_p \{ E(k,m_1) \} \quad \text{where} \quad k \longleftarrow K$$

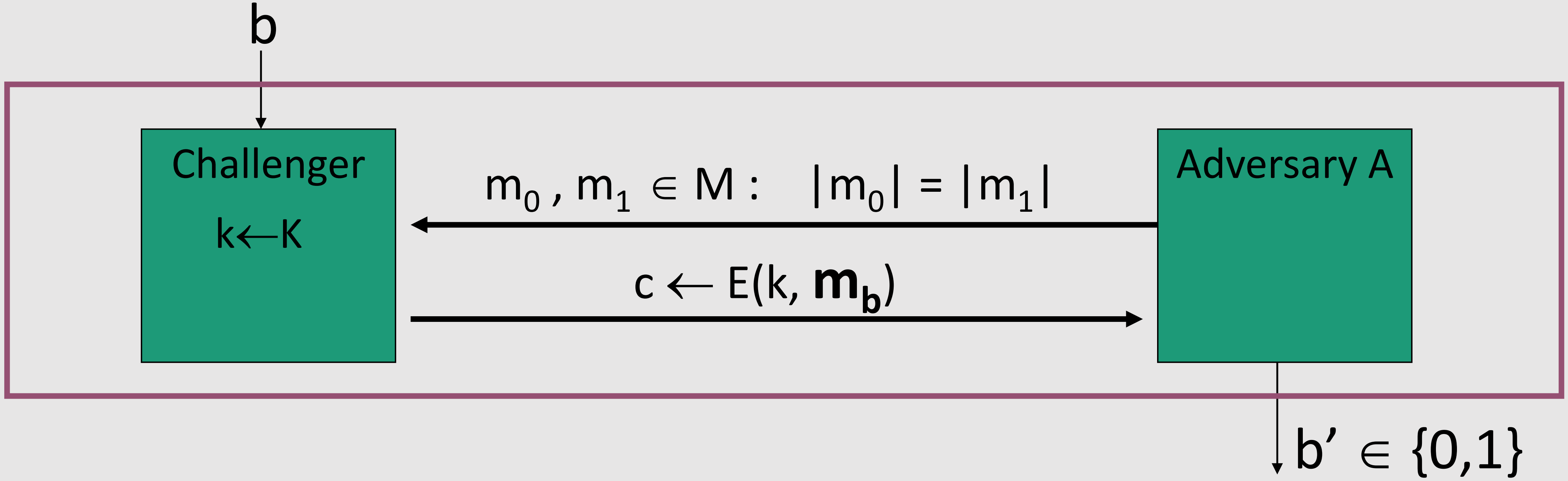
Rather than requiring the two distributions to be identical, we require them to be **COMPUTATIONALLY INDISTINGUISHABLE**

(One more requirement) … but also need adversary to exhibit $m_0, m_1 \in M$ explicitly

Semantic Security (one-time key)

For a cipher **Q = (E,D)** and an adversary **A** define a game as follows.

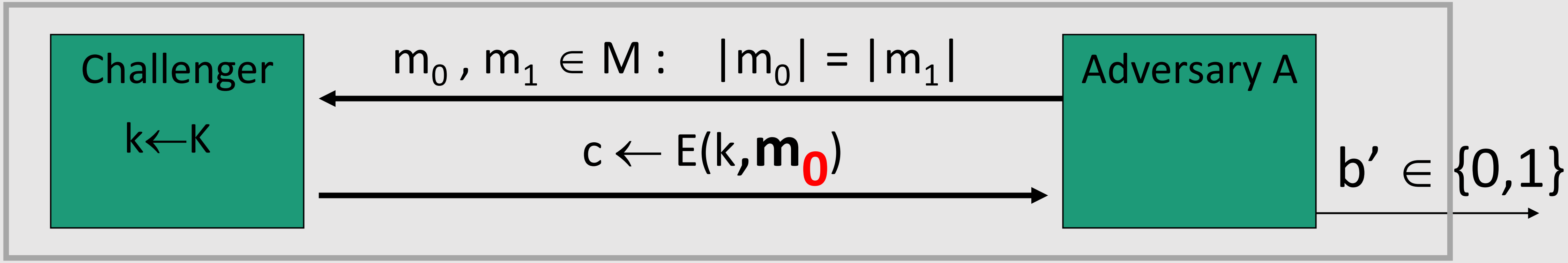
For b=0,1 define experiments EXP(0) and EXP(1) as:



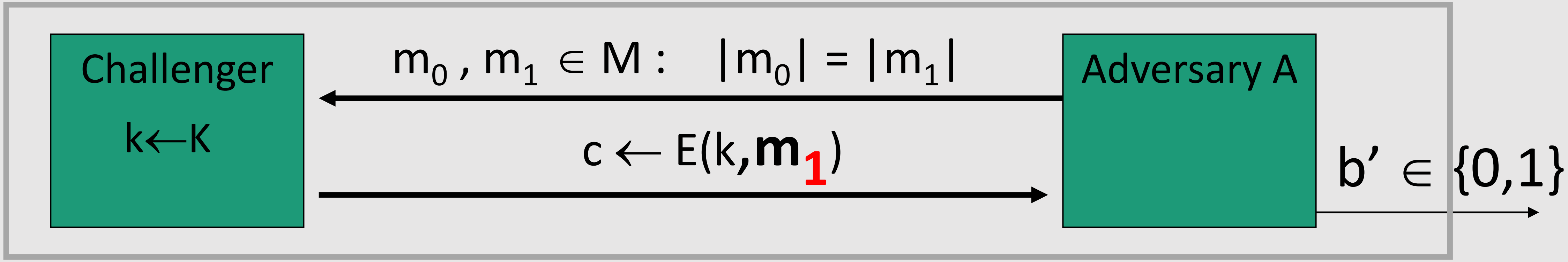
$$\mathbf{Adv_{SS}[A,Q] := |\ Pr[EXP(0)=1\ ] - Pr[\ EXP(1)=1\ ]\ |}$$

Semantic Security (one-time key)

EXP(0):



EXP(1):



$\text{Adv}_{SS}[A,Q] = \Big| \Pr[\,\mathbf{EXP(0)}=1\,] - \Pr[\,\mathbf{EXP(1)}=1\,] \Big|$ should be “negligible” for all “efficient” A

Semantic Security (one-time key)

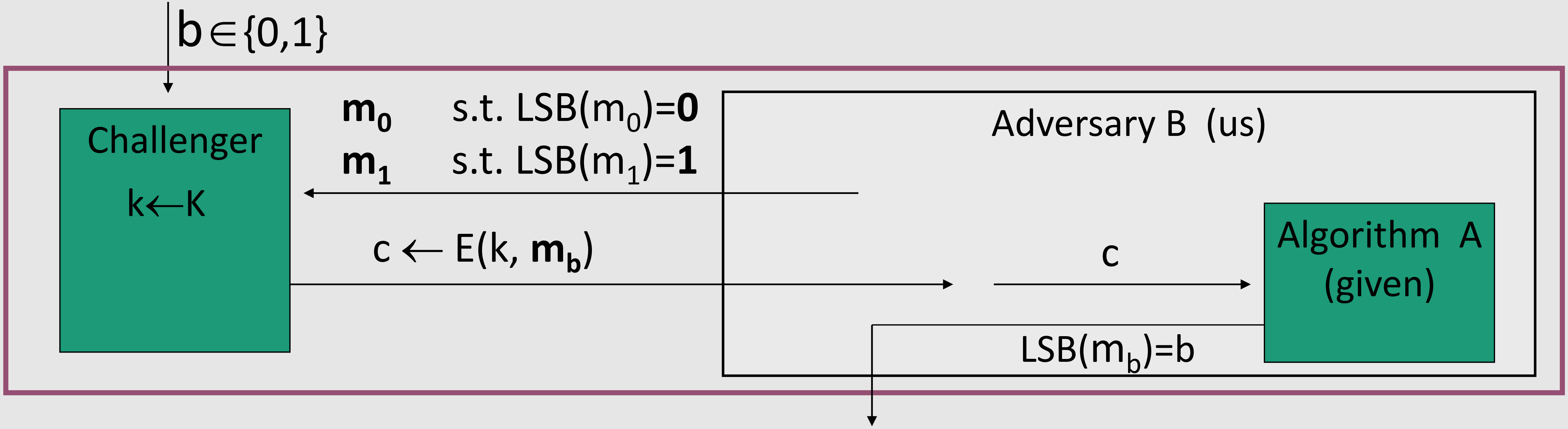
Definition:

Q is **semantically secure** if for all “efficient” **A**,

$$\mathbf{Adv_{SS}[A,Q]} \textbf{ is “negligible”.}$$

Example

Suppose efficient **A can always deduce LSB of PT from CT**
$\Rightarrow$ **Q** is **not** semantically secure.



Then $\textbf{Adv}_{SS}[B,Q] = \Big|\ \Pr[\ \textbf{EXP(0)}=1\ ] - \Pr[\ \textbf{EXP(1)}=1\ ]\ \Big| =$

Stream ciphers are semantically secure

Theorem:

G is a **secure PRG** $\Rightarrow$ stream cipher **Q** derived from G is **semantically secure**

In particular:

∀ semantic security adversary **A**, ∃ a PRG adversary **B** (i.e., a statistical test) s.t.

$$\mathbf{Adv_{SS}[A,Q] \le 2 \cdot Adv_{PRG}[B,G]}$$