

Lecture 9

Chosen-Ciphertext Security

Stefan Dziembowski
University of Rome
La Sapienza



SAPIENZA
UNIVERSITÀ DI ROMA

BiSS 2009
Bertinoro International
Spring School
2-6 March 2009



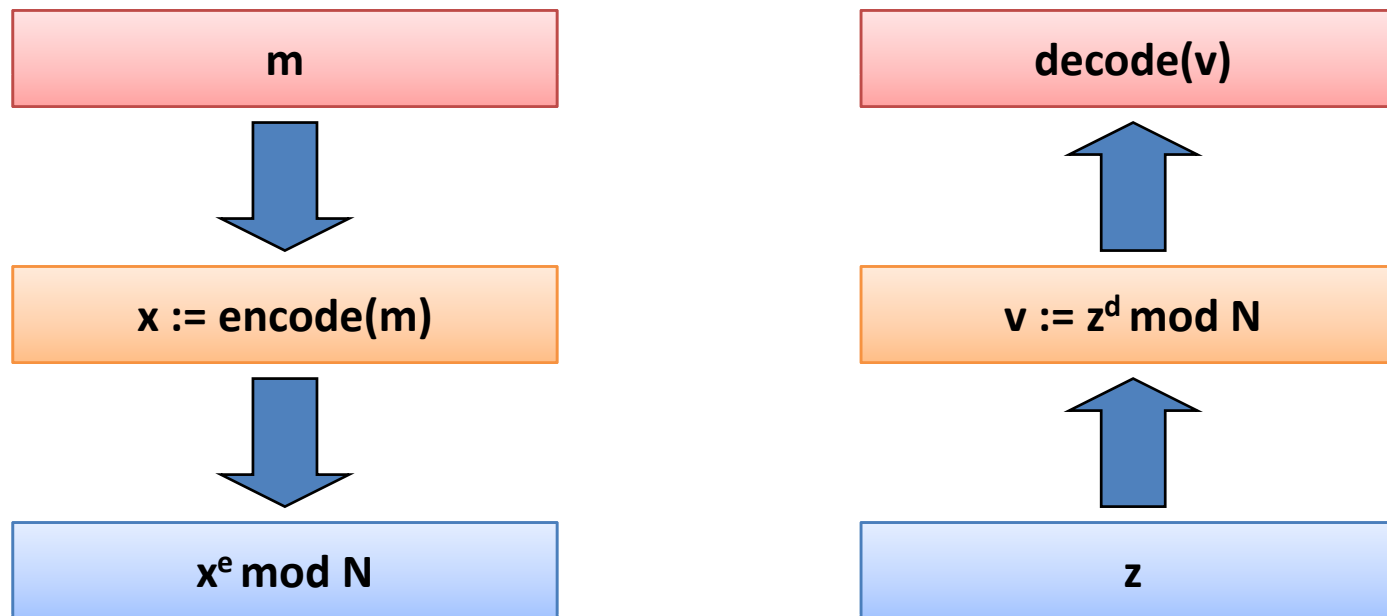
Plan



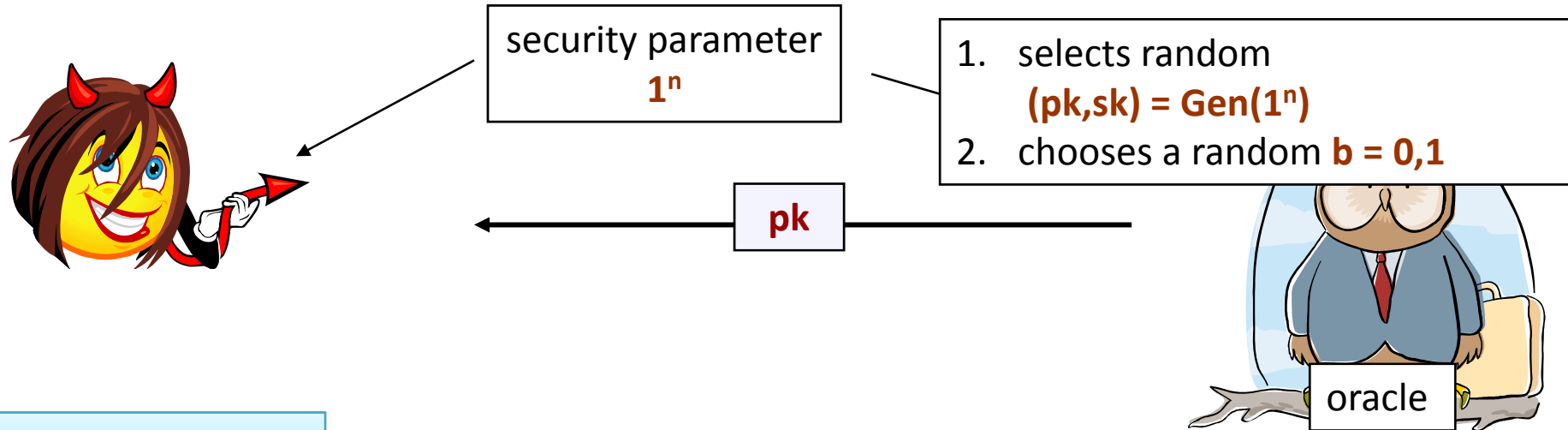
1. CCA-security – the motivation and the definition
2. CCA-security in the private-key settings
3. CCA-security in the public-key settings
 1. a scheme that is efficient and simple,
 2. a scheme that is even more efficient and a bit less simple.

Problem

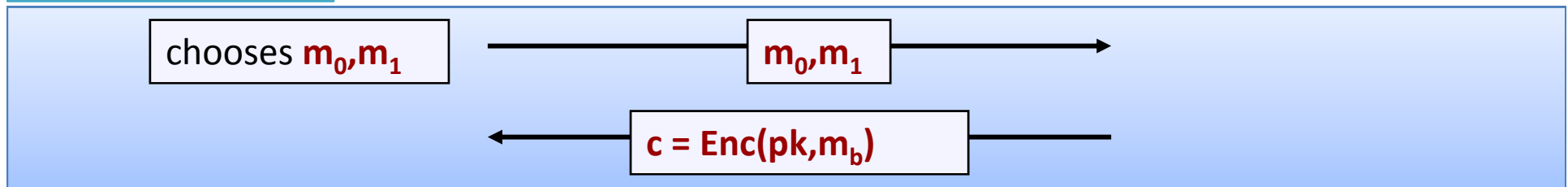
How to encode a message **m** before encrypting it (with **RSA**, for example)?



Remember the chosen-plaintext attack?



challenge phase:

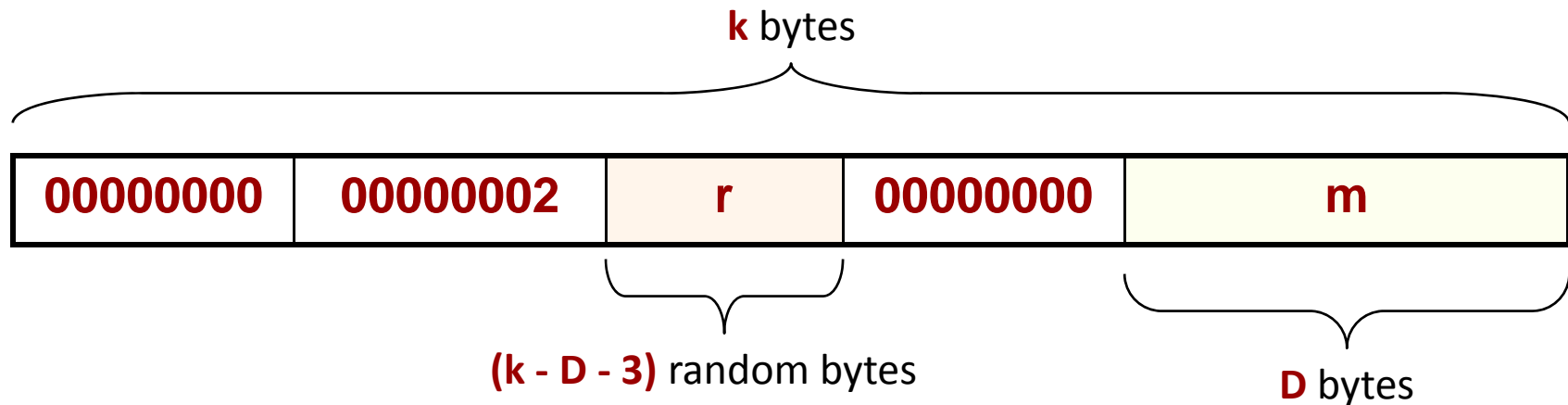


has to guess b

The PKCS #1 v1.5 encoding

We observed that encoding has to be randomized.

This is the encoding that we presented:



This lecture

The **PKCS #1 v1.5** encoding looks ad-hoc...

Today we present a more “scientific” encoding.

For this, we are going to use a stronger security definition.

Chosen **Ciphertext** Attack (CCA)

The adversary may also choose a **ciphertext** and learn the corresponding plaintext.

Does it make sense?

- Aren't we too paranoid?
- How to formalize it?

Aren't we too paranoid?

No!

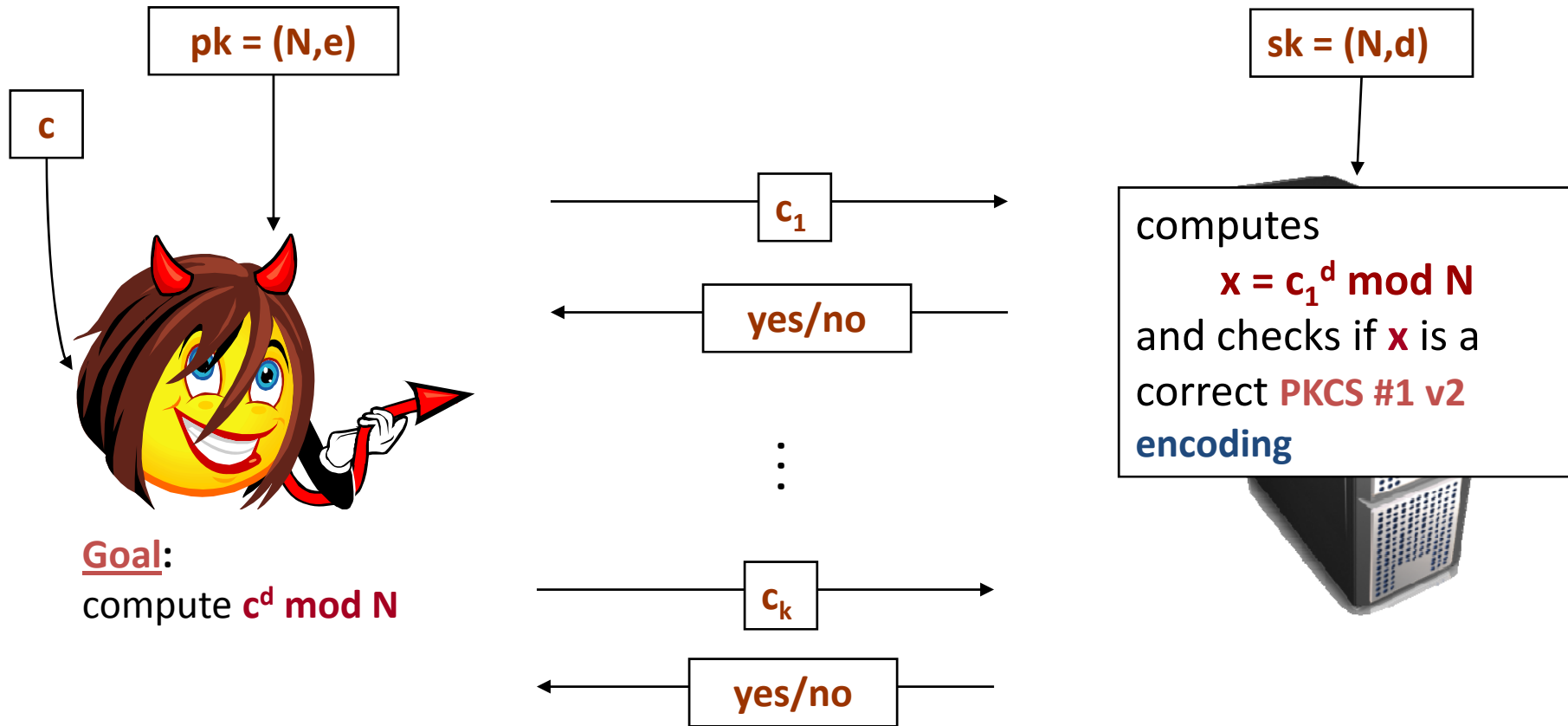
Bleichenbacher [1998] showed a “practical” chosen ciphertext attack on encoding proposed for the PKCS #1 v.2 standard.

[see also: Bleichenbacher, D., Kaliski B., Staddon J., "Recent results on PKCS #1: RSA encryption standard", *RSA Laboratories' bulletin #7*, <ftp://ftp.rsasecurity.com/pub/pdfs/bulletn7.pdf>]

Why is Bleichenbacher's attack practical?

Because it assumes that the adversary can get only one bit of information about the plaintext...

Bleichenbacher's attack – the scenario



Bleichenbacher [1998]:

There exists a successful attack that requires $k = 2^{20}$ questions for $N = 1024$.

So, chosen ciphertext attacks are practical!

In **Bleichenbacher's attack** the adversary could obtain just **one bit** of information.

Conservative approach:

assume that he can get the **entire** plaintext.

Idea

1. Provide a security definition that “covers” this type of an attack.

2. Propose a scheme that is “provably secure” according to this definition.

This will lead to an encoding that is less ad-hoc than PKCS #1 v1.5

CCA - security

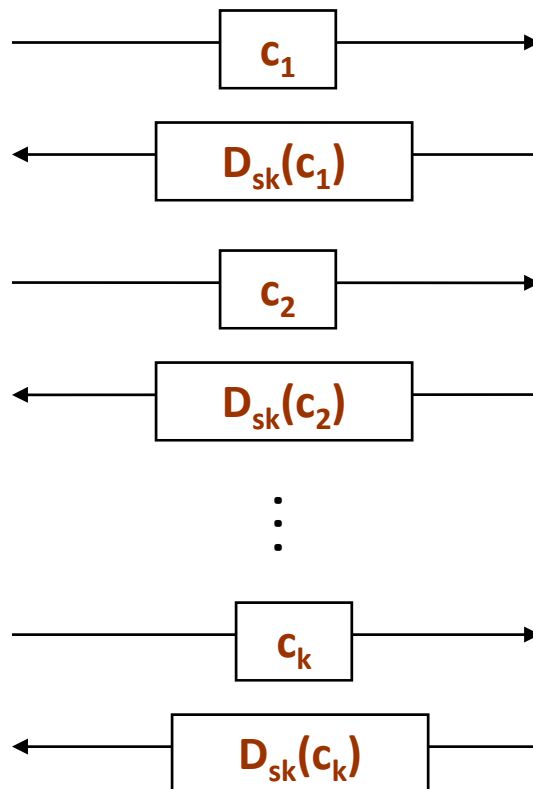
It makes sense to consider CCA-security in

- private-key settings
- public-key settings



Decryption oracle

To define the CCA-security we consider a **decryption** oracle.



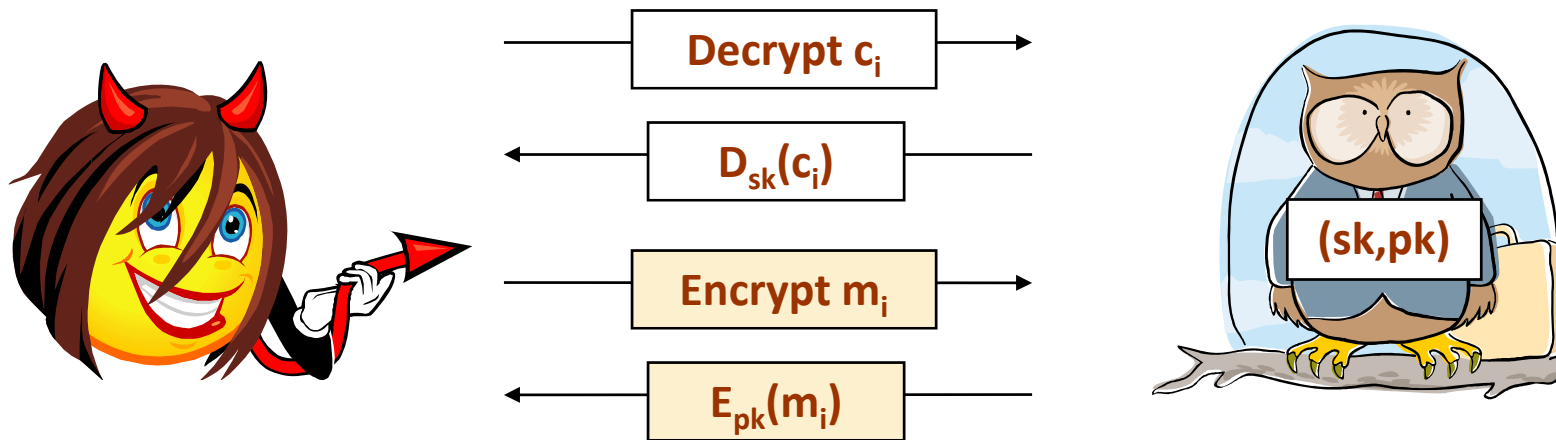
convention:

$D_{sk}(c_i) := \text{error}$
if c_i cannot be
decrypted

Decryption/encryption oracle

We assume that also CPA is allowed.

Two types of queries:



CCA-security– the game

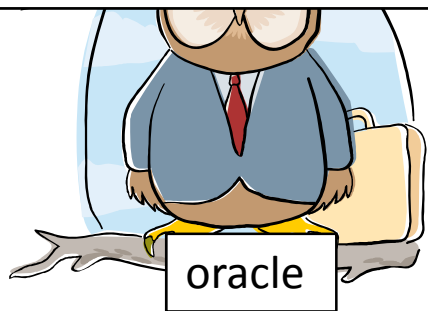
in the private-key settings: $pk = sk$



security parameter
 1^n

1. selects random $(pk, sk) = \text{Gen}(1^n)$
2. chooses a random $b = 0, 1$

pk (in the public-key settings)



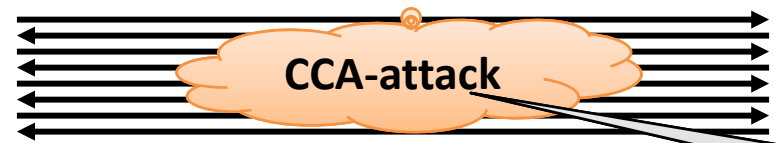
oracle

challenge phase:

chooses m_0, m_1

m_0, m_1

$c = \text{Enc}(pk, m_b)$



$|m_0| = |m_1|$

has to guess b

Here **Eve** cannot ask for decryption of c .

CCA-security

Alternative name: **CCA-secure**

Security definition:

We say that **(Gen, Enc, Dec)** has **indistinguishable encryptions under a chosen-ciphertext attack (CCA)** if any

randomized polynomial time adversary

guesses **b** correctly

with probability at most **$0.5 + \epsilon(n)$** , where **ϵ** is negligible.

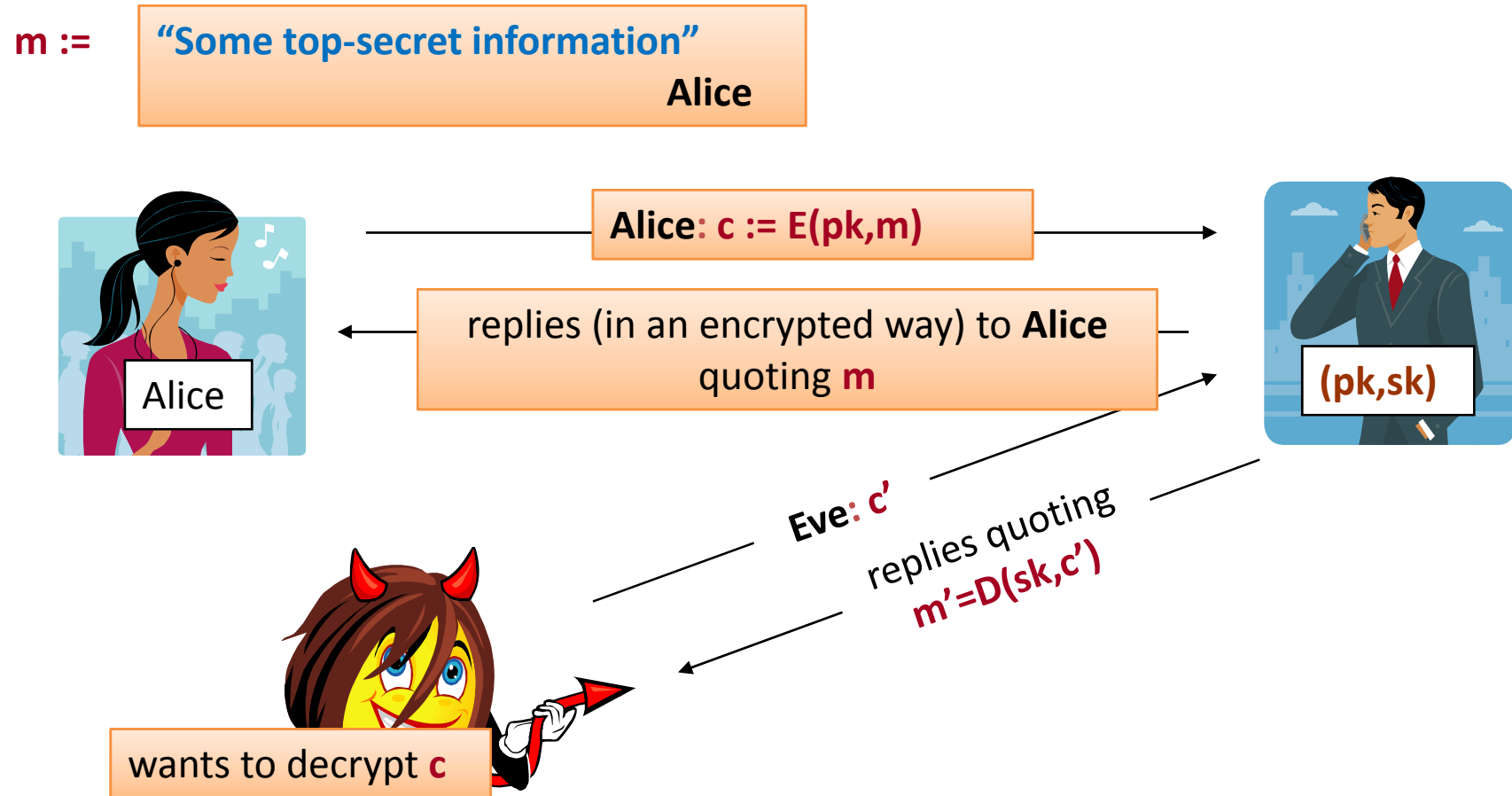
CCA in practice (1/2)

Some actions of the receiver may depend on the decrypted message.

For example, the receiver may communicate an error if the message “looks strange”.

(like in the **Bleichenbacher’s** attack)

CCA in practice (2/2)



Why **Eve** cannot just set $c' := c$?

Because **Bob** would get suspicious (why message from **Eve** has **Alice's** name inside?)

Plan

1. CCA-security – the motivation and the definition



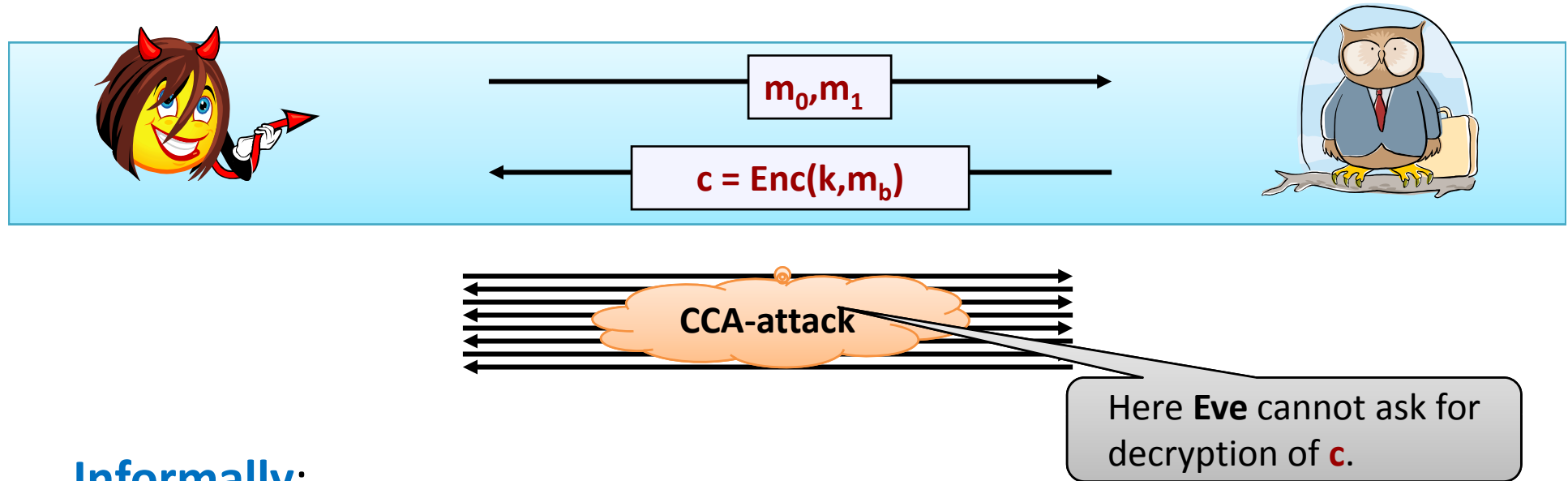
2. CCA-security in the private-key settings

3. CCA-security in the public-key settings

1. a scheme that is efficient and simple,

2. a scheme that is even more efficient and a bit less simple.

CPA-security does not imply the CCA-security



Informally:

To win the game it is enough that Eve computes some c' such that $\text{Dec}(k, c')$ is "related to" $\text{Dec}(k, c)$.

(**Why?** Because then she is allowed to do as for it.)

For example: this is possible for any stream cipher!

What to do?

CCA-security in the private-key settings can be achieved by adding authentication.

How to combine authentication with encryption?

We already considered this problem some time ago.

Authentication and Encryption

Options:

- **Encrypt-and-authenticate:**

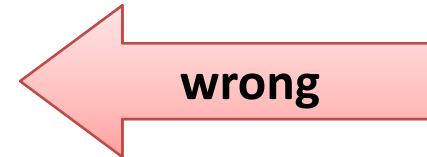
$$c \leftarrow \text{Enc}_{k_1}(m) \text{ and } t \leftarrow \text{Tag}_{k_2}(m)$$

- **Authenticate-then-encrypt:**

$$t \leftarrow \text{Tag}_{k_2}(m) \text{ and } c \leftarrow \text{Enc}_{k_1}(m || t)$$

- **Encrypt-then-authenticate:**

$$c \leftarrow \text{Enc}_{k_1}(m) \text{ and } t \leftarrow \text{Tag}_{k_2}(c)$$



A CCA-secure encryption scheme

(Enc,Dec) – a CPA-secure symmetric encryption scheme
(Tag,Vrfy) – a MAC.

Create a new encryption scheme **(Enc',Dec')** where:

- The secret key **k** has a form **k = (k₀,k₁)**, where
 - **k₀** is a key for **(Enc,Dec)**, and
 - **k₁** is a key for **(Tag,Vrfy)**
- **Enc'((k₀,k₁),m) := (Enc(k₀,m), Tag(k₁,Enc(k₀,m)))**
- **Dec'((k₀,k₁),m) := decrypt and verify the tag**

Why is it secure?

Intuition

An adversary cannot create a new valid pair

$(\text{Enc}(k_0, m), \text{Tag}(k_1, \text{Enc}(k_0, m)))$

without knowing k_1 .

So he will always receive an error message from the oracle

(unless he replays the ciphertexts that he already received from the oracle – but this gives him no extra information)

We need one technical assumption:

For every m and k there exists exactly one t such that:

$\text{Vrfy}_k(m, t) = \text{yes.}$

Is the “**authenticate-then-encrypt**” secure?

Authenticate-then-encrypt:

$$t \leftarrow \text{Tag}_{k_2}(m) \quad \text{and} \quad c \leftarrow \text{Enc}_{k_1}(m || t)$$

Not always!

There exists (artificial) counter-examples...

The first counter-example

Authenticate-then-encrypt:

$$t \leftarrow \text{Tag}_{k_2}(m) \quad \text{and} \quad c \leftarrow \text{Enc}_{k_1}(m || t)$$

Suppose the encryption scheme adds a random bit at the end of the ciphertext.



Then



is a different ciphertext and the adversary is allowed to ask the oracle to decrypt it.

This example is really artificial. There exist better ones...

The second counter-example

Consider the following transformation

$$T : \{0,1\}^* \rightarrow \{0,1\}^*$$

defined on every (x_1, x_2, \dots, x_n) as

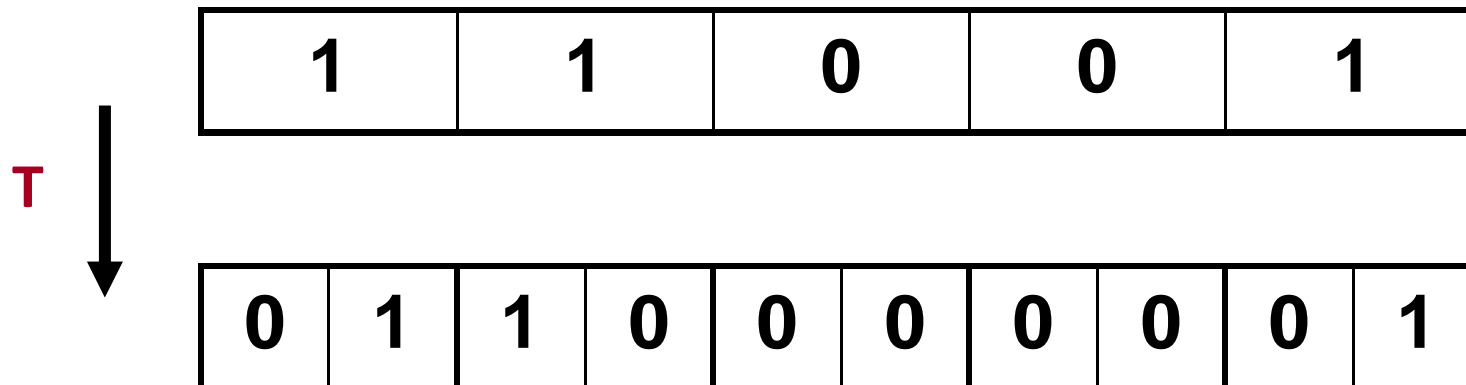
$$T(x_1, x_2, \dots, x_n) = (U(x_1), U(x_2), \dots, U(x_n)),$$

where

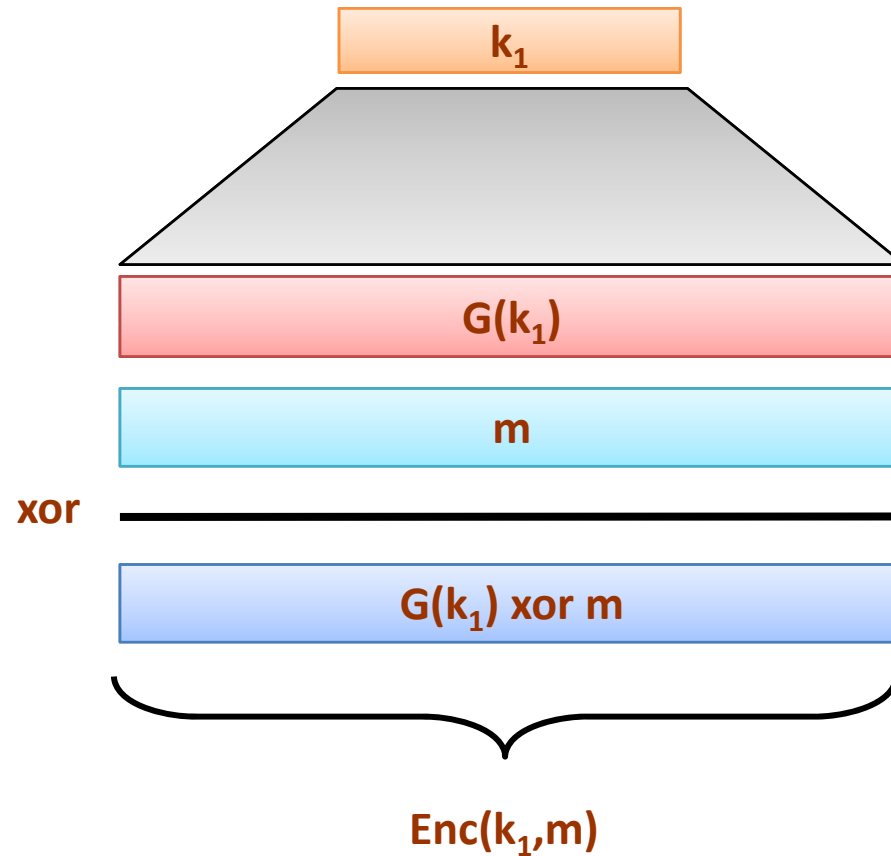
- $U(0) = 00$
- $U(1) = 01$, or 10 , randomly.

This transformation is of course invertible.

Example:

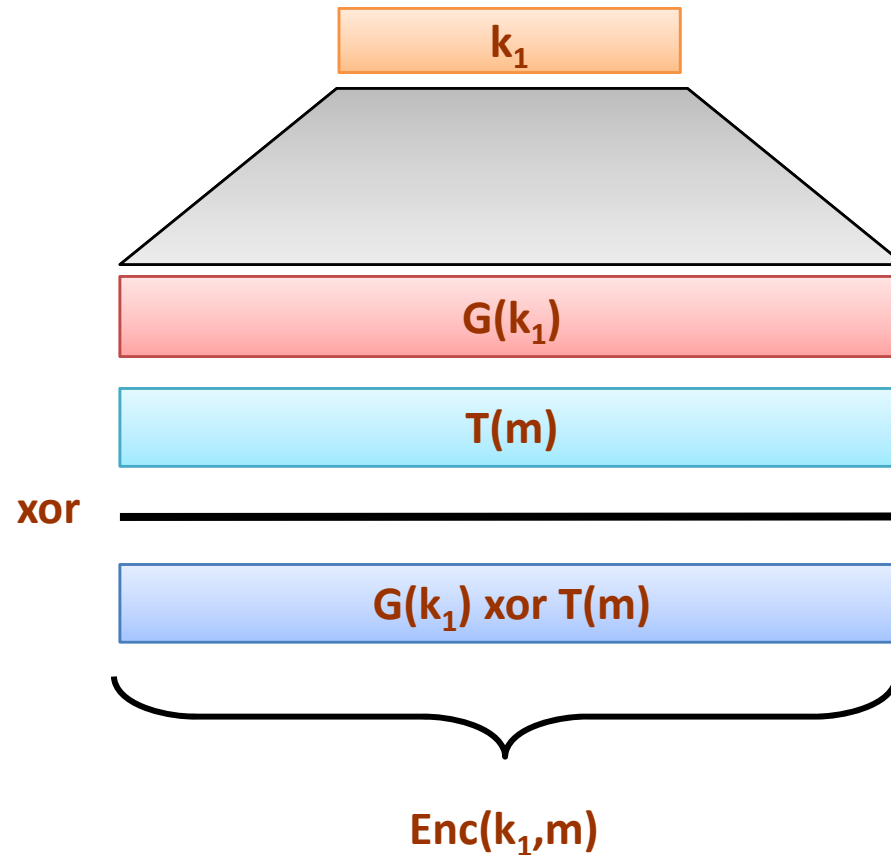


Remember the stream ciphers?



Our new (artificial) encryption scheme

To encrypt a message m do the following:

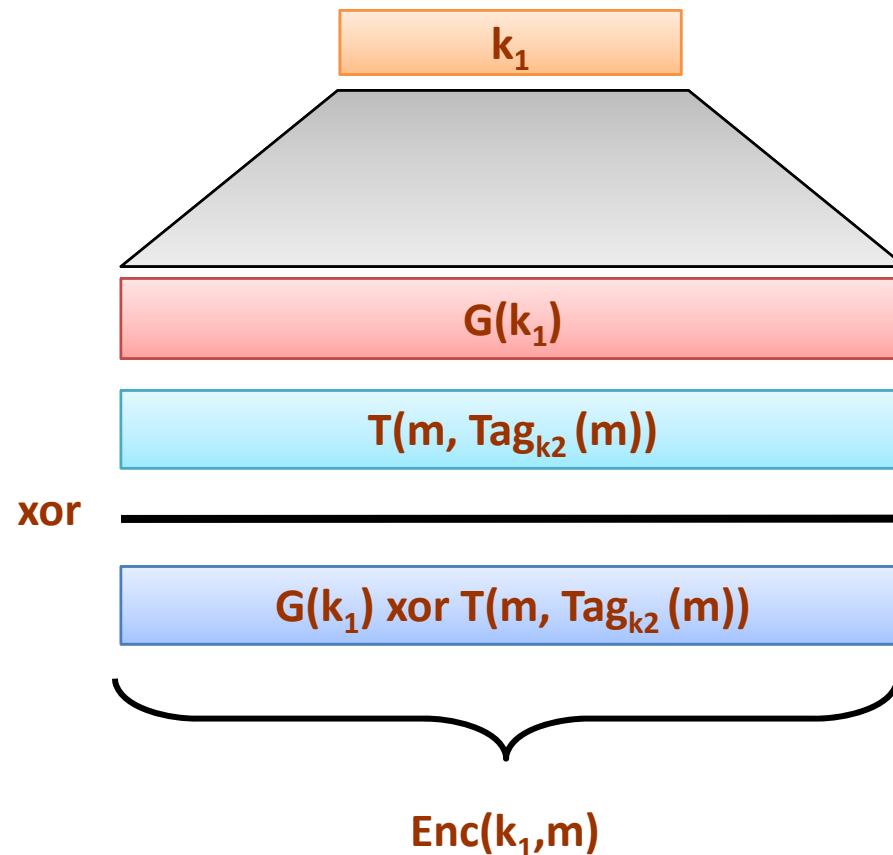


This scheme is **CPA**-secure.

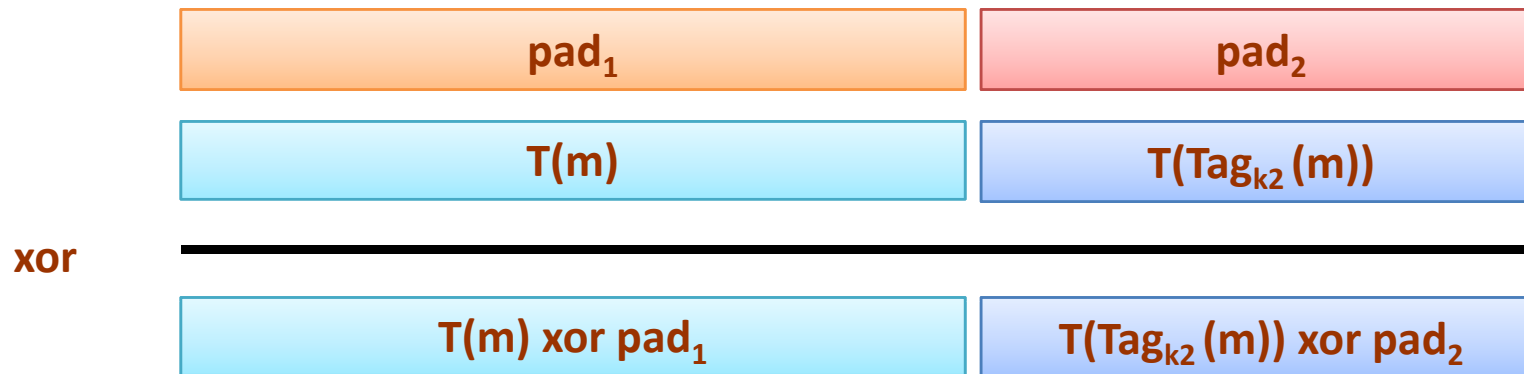
CCA-security?

Suppose we use this encryption scheme with the **authenticate-then-encrypt** method:

$$t \leftarrow \text{Tag}_{k_2}(m) \text{ and } c \leftarrow \text{Enc}_{k_1}(m || t)$$



How does the ciphertext look?

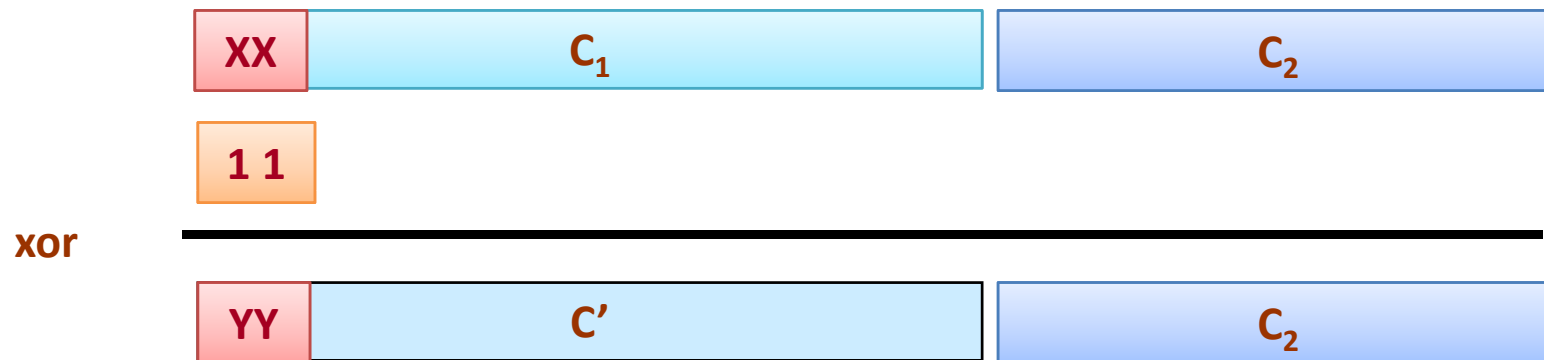


The attack

The adversary that wants to decrypt the first bit of

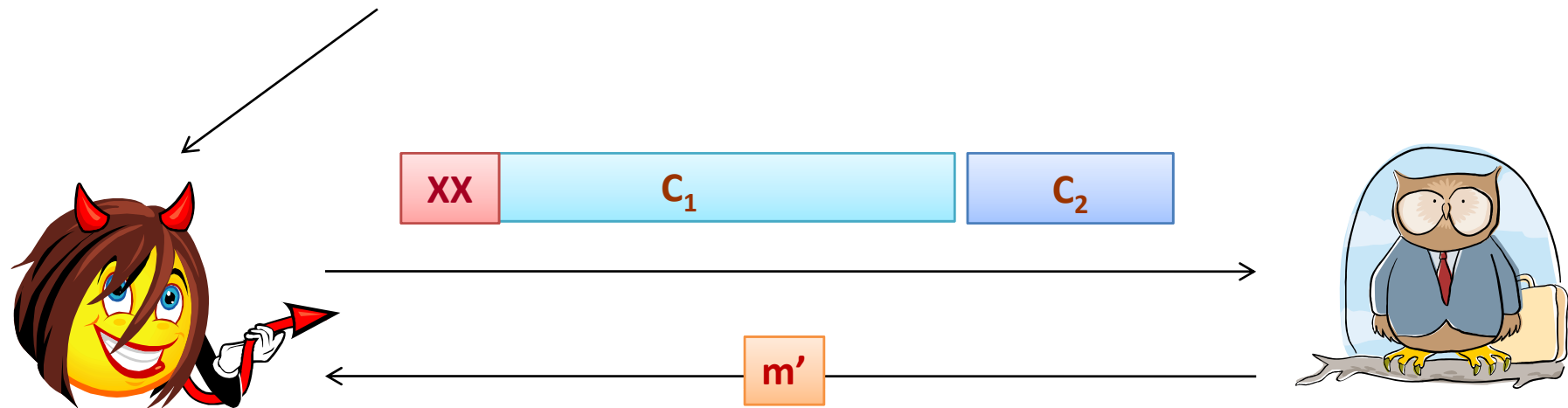
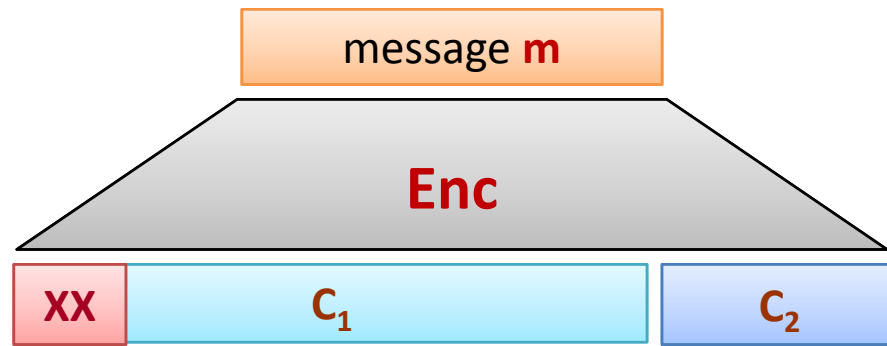


can modify the ciphertext by flipping the first two bits:



If the first two bits are **01** or **10** then the corresponding plaintext doesn't change.

If the first two bits are **00** then the plaintext changes and the tag becomes **invalid!**



if $m' = \text{error}$ then the first bit of m is equal to **0**, otherwise it is equal to **1**.

The same can be done for any other bit.

These examples are artificial

It is likely that for many “normal” schemes this combination is secure.

However, these examples show that the **authenticate-then-encrypt** method cannot be proven secure...

(from the standard assumptions)

Plan

1. CCA-security – the motivation and the definition
2. CCA-security in the private-key settings?
3. CCA-security in the public-key settings
 1. a scheme that is efficient and simple,
 2. a scheme that is even more efficient and a bit less simple.



How does it look in the public-key settings

There are many constructions of a CCA-secure public-key encryption scheme.

Probably the most famous is the one of **Cramer and Shoup**:

[Ronald Cramer and Victor Shoup: "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack." 1998].

It is based on hardness of **discrete logarithm** and is quite efficient.

Still, many practitioners prefer more efficient schemes (with a weaker security proof).

First attempt

Idea: take a **symmetric-key CCA-secure** scheme **(Enc', Dec')** and use something similar to hybrid encryption.

r is random

public key: **(N,e)**

private key: **(N,d)**

$$\text{Enc}((N,e),m) := (r^e \bmod N, \text{Enc}'(r,m))$$

$$\text{Dec}((N,d),(c_0,c_1)) := \text{Dec}'(c_0^d \bmod N, c_1)$$

Problem

$$\text{Enc}((N,e),m) := (r^e \bmod N, \text{Enc}'(r,m))$$

$|N|$ is normally much larger than the length of a key for symmetric encryption.

Typically $|N| = 1024$ and length of the key is **128**.

First idea: **truncate**.

But is it secure?

It may be the case that

- **RSA** is hard to invert, but
- **128** first bits are easy to compute...

Idea

Instead of truncating – hash!

t – length of the symmetric key

$H : \{0,1\}^* \rightarrow \{0,1\}^t$ – a hash function

$$\text{Enc}((N,e),m) := (r^e \bmod N, \text{Enc}'(H(r),m))$$

$$\text{Dec}((N,d),(c_0,c_1)) := \text{Dec}'((H(c_0)^d \bmod N, c_1))$$

But can we prove anything about it?

depends...

Which properties should **H** have?

If we just assume that **H** is collision-resistant we cannot prove anything...

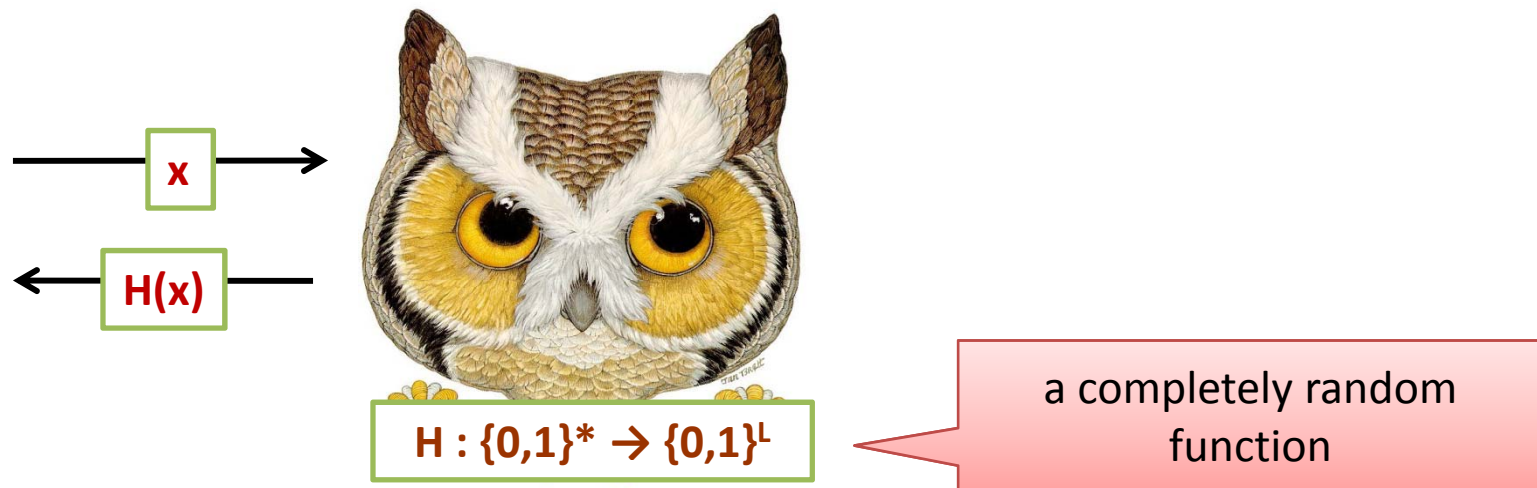
We have to assume that **H** “outputs random values on different inputs”.

This can be formalized by modeling **H** as **random oracle**.

Remember the **Random Oracle Model**?

Random oracle model

Model the hash function as a **random oracle**.



Security proof – the intuition

H – a hash function $\text{Enc}((N,e),m) := (r^e \bmod N, \text{Enc}'(H(r), m))$

Why is this scheme secure in the **random oracle model**?

Because, as long as the adversary did not query the oracle on r , the value of $H(r)$ is completely random.

To learn r the adversary would need to compute it from $r^e \bmod N$, so he would need to invert **RSA**.

So (with a very high probability) from the point of view of the adversary $H(r)$ is random.

Therefore the **CCA-security** of (Enc, Dec) follows from the **CCA-security** of $(\text{Enc}', \text{Dec}')$.

Disadvantages of this method

$$\text{Enc}((N,e),m) := (r^e \bmod N, \text{Enc}'(H(r), m))$$

The ciphertext is longer than **N** even if the message is short.

Therefore in practice another method is used:

Optimal Asymmetric Encryption Padding (OAEP).

Plan

1. CCA-security – the motivation and the definition
2. CCA-security in the private-key settings?
3. **CCA-security in the public-key settings**
 1. a scheme that is efficient and simple,
 2. a scheme that is even more efficient and a bit less simple.



Optimal Asymmetric Encryption Padding (OAEP) – the history

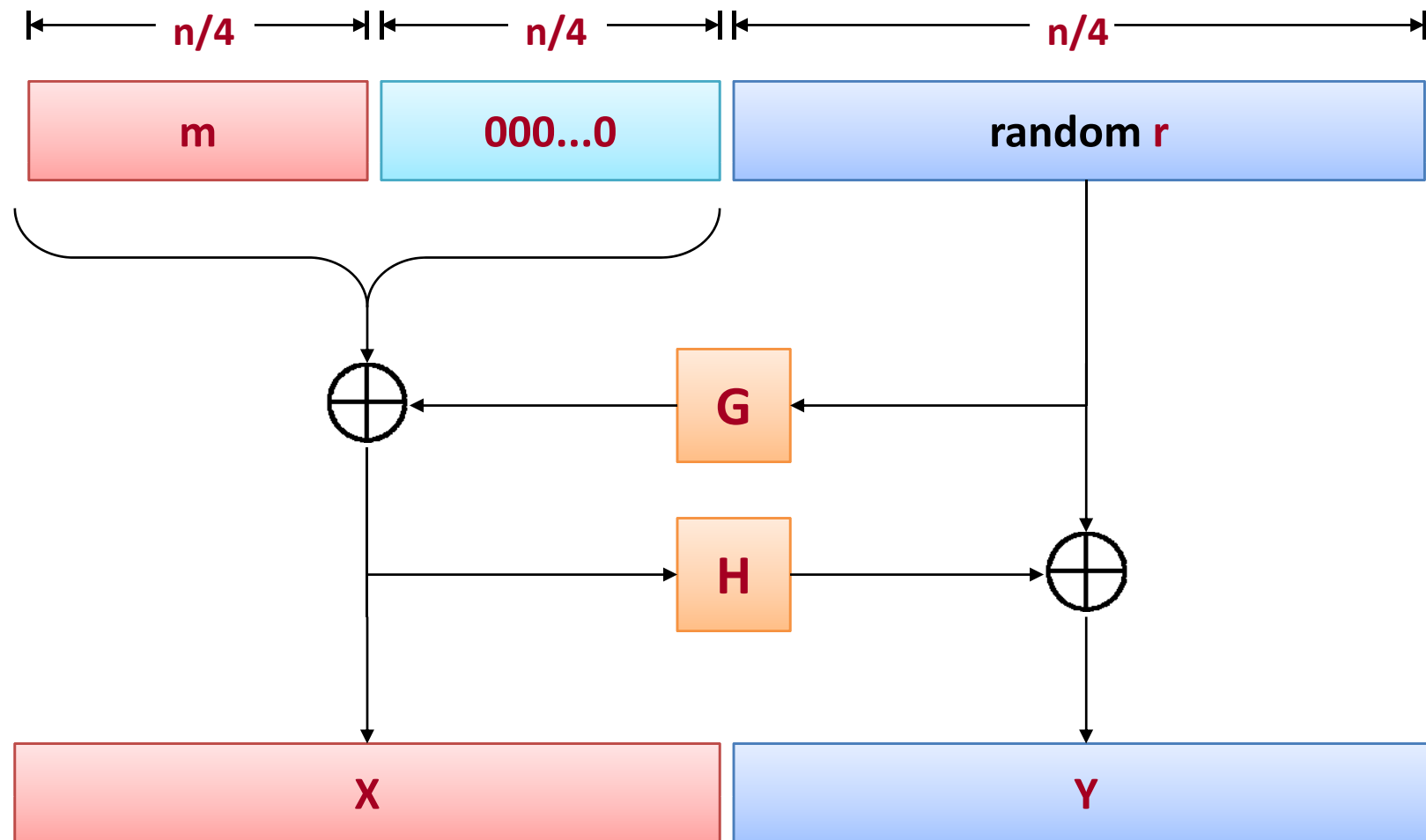
- **Introduced in:**
[M. Bellare, P. Rogaway. *Optimal Asymmetric Encryption -- How to encrypt with RSA*. Eurocrypt '94]
- **An error in the security proof was spotted in**
[V. Shup. *OAEP Reconsidered*. Crypto '01]
- **This error was repaired in**
[E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. *RSA-OAEP is secure under the RSA assumption*. Crypto '01]

It is now a part of a PKCS#1 v. 2.0 standard.

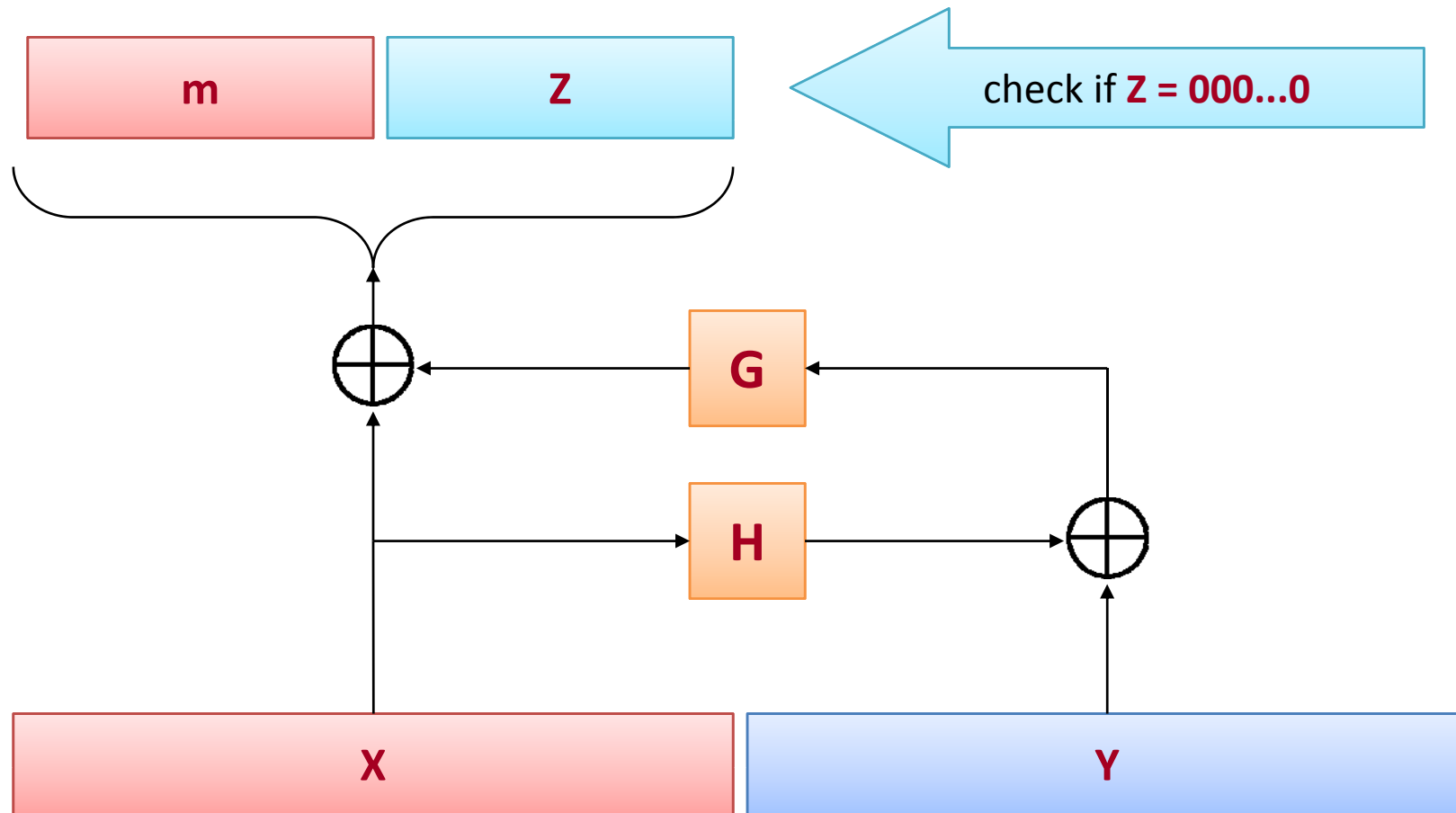
OAEP

G,H – hash functions

OAEP(m) :=



How to invert?



RSA-OAEP

private key: (N,d)

public key: (N,e)

$$\text{Enc}((N,e),m) = (\text{OAEP}(m))^e \bmod N$$

$$\text{Dec}((N,e),m) = (\text{OAEP}^{-1}(m))^d \bmod N$$

Security of RSA-OAEP

Security of **RSA-OAEP** can be proven if

- one models **H** and **G** as random oracles
- the **RSA assumption** holds.

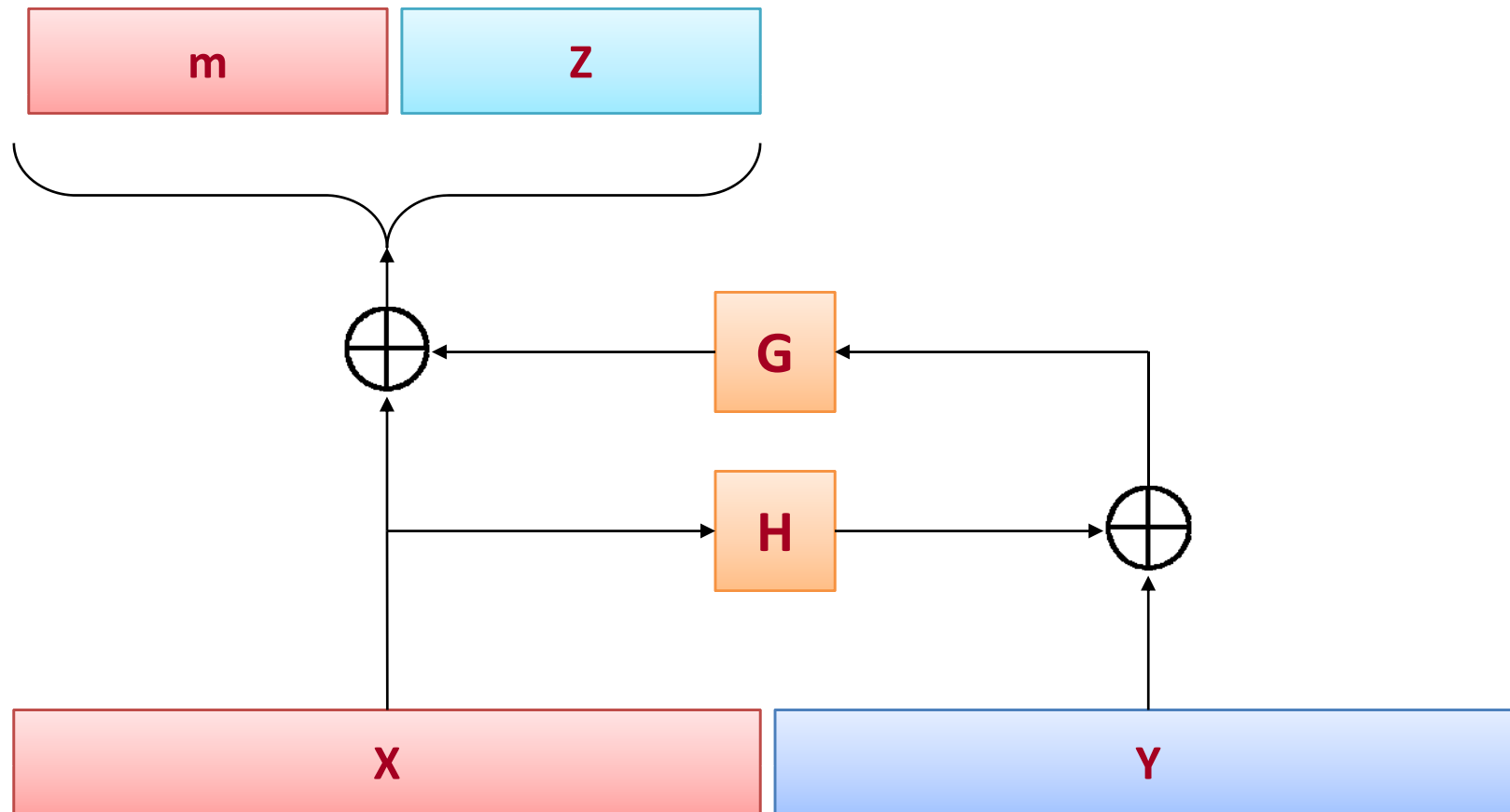
We do not present the proof here.

We just mention some nice properties of this encoding.

Nice properties of OAEP

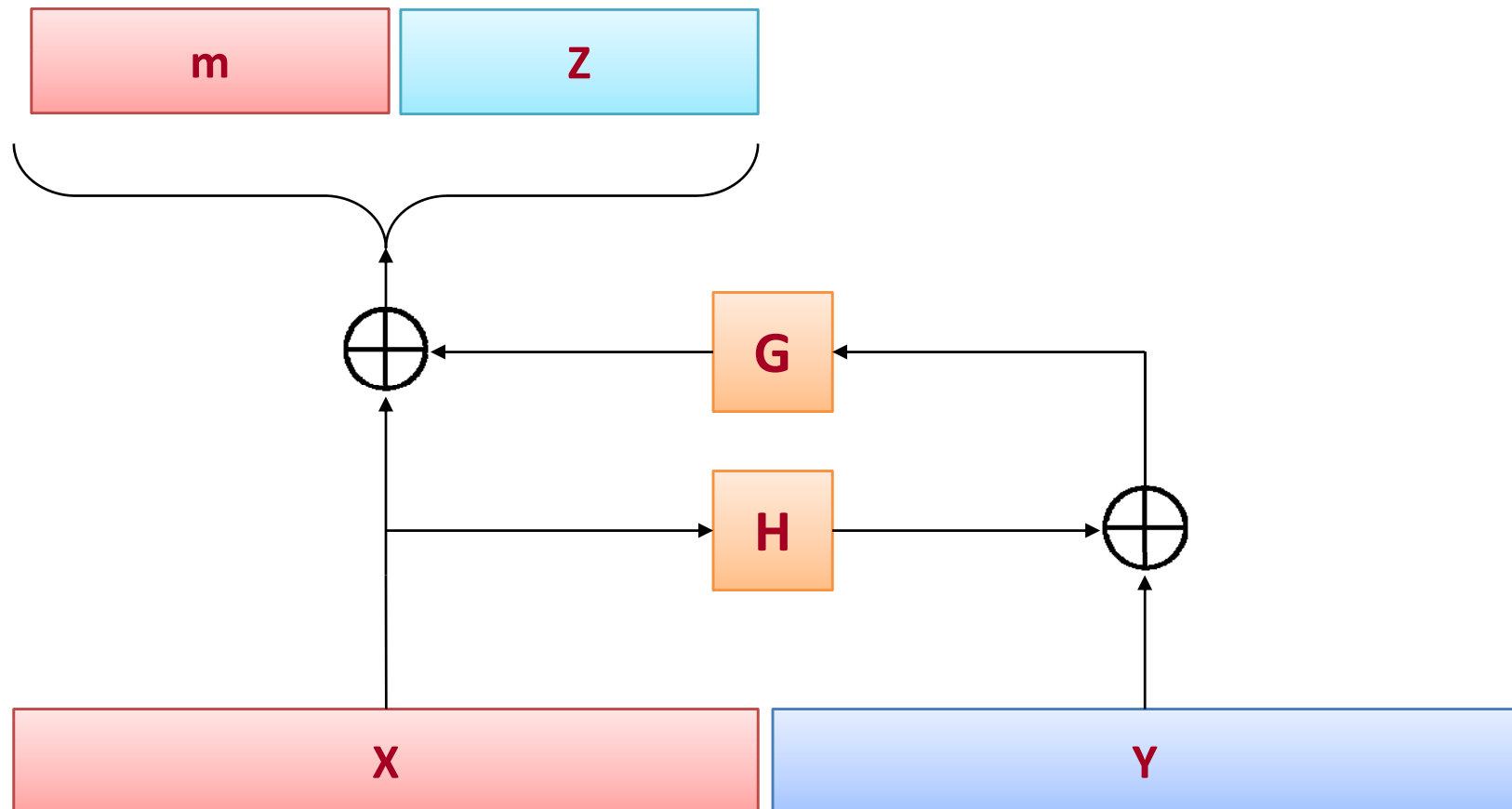
- it is invertible
- but to invert you need to know (X,Y) completely
- for any messages m_1, m_2, m_3, \dots :
the encodings
 $OAEP(m_1), OAEP(m_2), OAEP(m_3), \dots$
are independent and random
- It is hard to produce a valid (X,Y) “without knowing m ”

OAEP is hard to invert if you don't know **X** and **Y** completely.

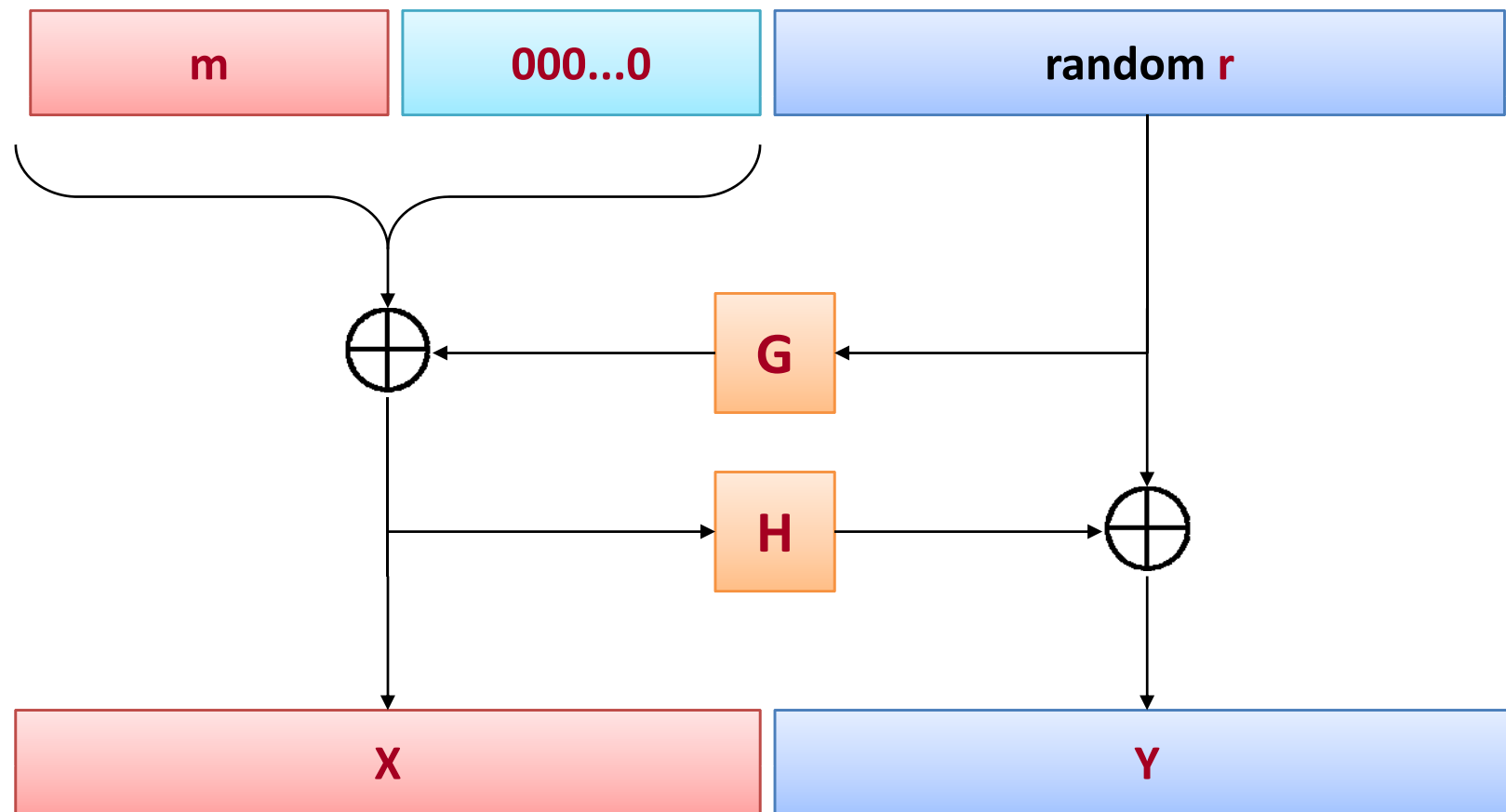


Why?

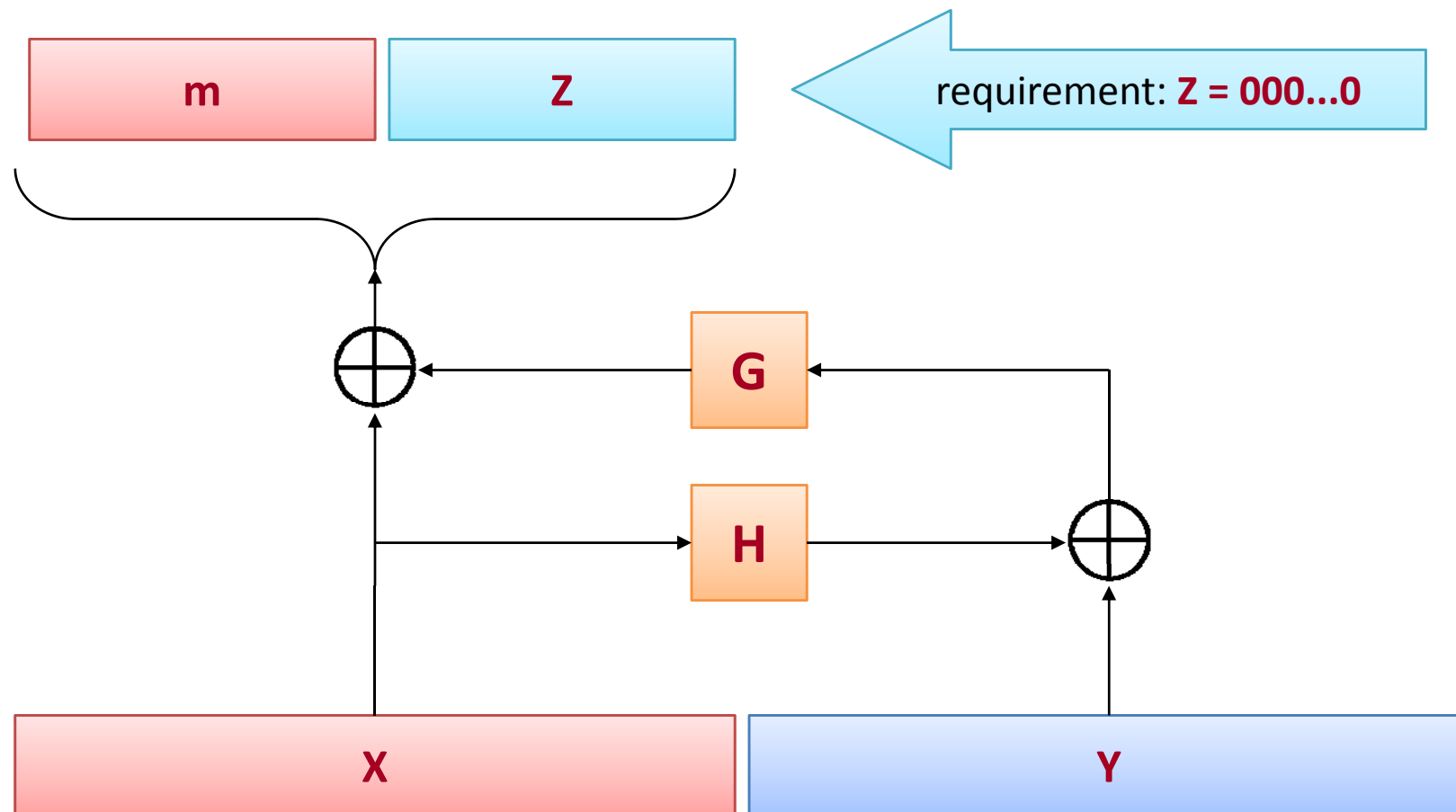
Assume **G** and **H** are random oracles...



Any encodings $\text{OAEP}(m_1), \text{OAEP}(m_2), \text{OAEP}(m_3), \dots$ are independent and random



It is hard to produce a valid (X,Y) “without knowing m ”



This last property is useful for CCA-
security

Why?

Informally:

Eve can produce valid ciphertexts only of those
messages that she knows...

Security – the conclusion

If it is hard to produce a valid (X,Y) “without knowing m ”, then CCA should not help the adversary.

Because he will only receive the error messages from the oracle.

(that was just an intuition – in reality the proof is complicated)

©2009 by Stefan Dziembowski. Permission to make digital or hard copies of part or all of this material is currently granted without fee *provided that copies are made only for personal or classroom use, are not distributed for profit or commercial advantage, and that new copies bear this notice and the full citation.*