Lecture 8

Public-Key Encryption I

Stefan Dziembowski

www.crypto.edu.pl/Dziembowski

University of Warsaw

21.11.18

version 1.1

Plan

1. Problems with the "handbook RSA"
2. Definition of the **CPA security**
3. Constructions of **CPA-secure** **RSA** encryption schemes
   1. theoretical
   2. practical
4. The **hybrid encryption** and the **KEM/DEM** paradigm
5. Definition of the **CCA security**
6. Constructions of **CCA-secure** symmetric encryption
7. Constructions of **CCA-secure** **RSA** encryption schemes

"Handbook RSA" encryption

Take $\boldsymbol{Z_N^*}$(where $\boldsymbol{N = pq}$ and $\boldsymbol{p, q}$ are **two distinct odd primes**), defined as follows:

$\boldsymbol{e \leftarrow Z_{\varphi(N)}^*}$

$\boldsymbol{d = e^{-1} \bmod \varphi(N)}$

Let $\boldsymbol{pk = (N, e)}$ and $\boldsymbol{sk = (N, d)}$

Handbook RSA encryption scheme:

messages and **ciphertexts**: $\boldsymbol{Z_N}$

- $\boldsymbol{\mathrm{Enc}_{N,e}(m) = m^e \bmod N}$
- $\boldsymbol{\mathrm{Dec}_{N,d}(c) = c^d \bmod N}$

Is it secure?

Issues with the "handbook RSA"

1. It is **deterministic**.
2. It has some "**algebraic properties**".
3. It is defined over Z_N^* and not over Z_N.

this is not really a problem (exercise)

Algebraic properties of RSA

1. **RSA** is homomorphic:

$$\mathbf{RSA}_{e,N}(m_0 \cdot m_1) = (m_0 \cdot m_1)^e$$

$$= m_0^e \cdot m_1^e$$

$$= \mathbf{RSA}_{e,N}(m_1) \cdot \mathbf{RSA}_{e,N}(m_2)$$

why is it bad?

By checking if $c = c_0 \cdot c_1$ the adversary can check if the messages m, m_0, m_1 corresponding to c, c_0, c_1 satisfy:

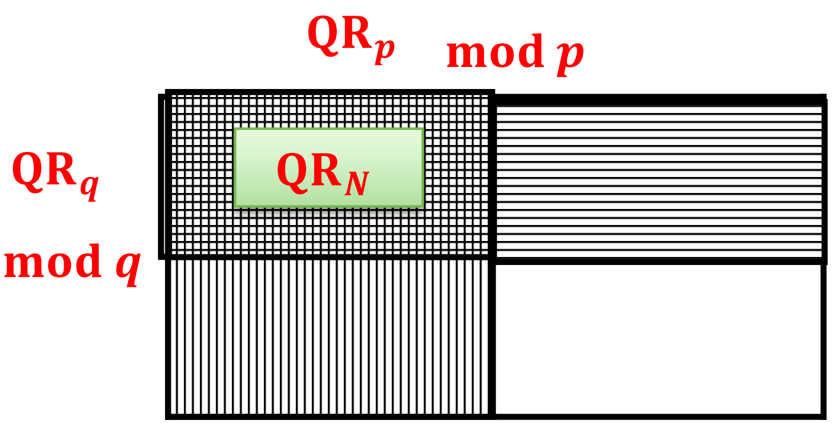
$$m = m_0 \cdot m_1$$

2. The **Jacobi symbol** leaks.

Jacobi Symbol (from the last lecture)

for any prime $\boldsymbol{p}$ define $J_p(x) := \begin{cases} +1 & \text{if } x \in \mathbf{QR}_p \\ -1 & \text{otherwise} \end{cases}$

for $N = pq$ define $J_N(x) := J_p(x) \cdot J_q(x)$



$J_N(x) :=$

+1	-1
-1	+1

It is a subgroup of Z_N^*

$Z_N^+ := \{x \in Z_N^* : J_N(x) = +1\}$

Jacobi symbol can be computed efficiently!

(even in $\boldsymbol{p}$ and $\boldsymbol{q}$ are unknown)

Fact: the RSA function "preserves" the **Jacobi symbol**

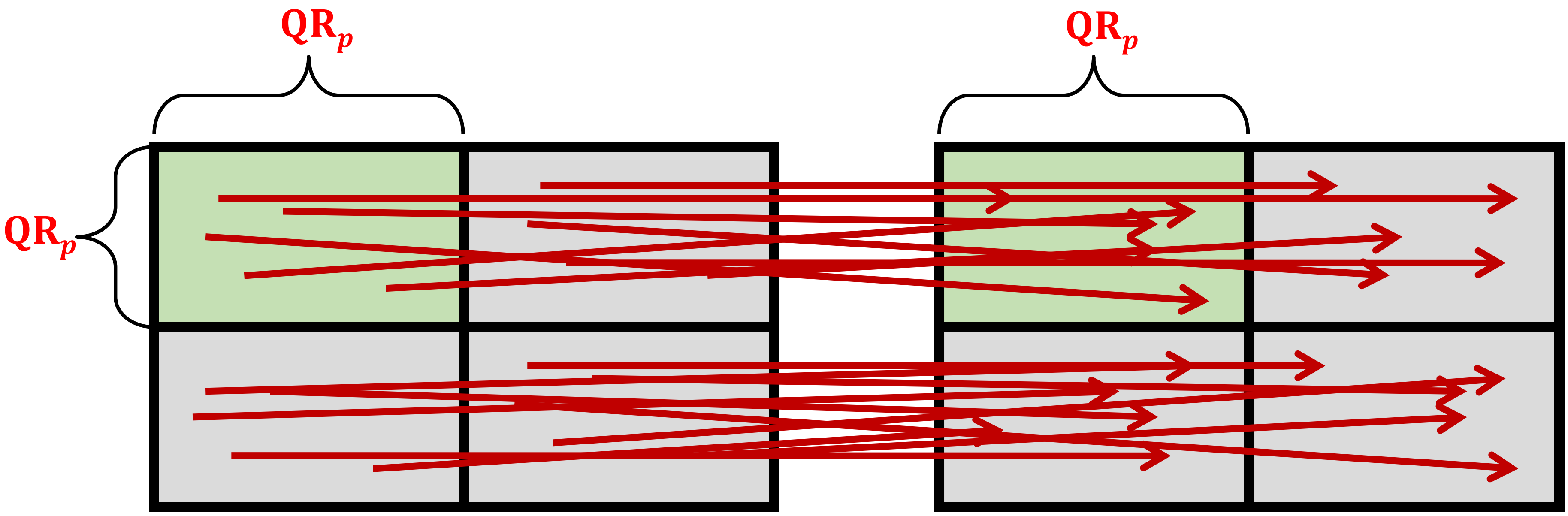
$N = pq$ - **RSA** modulus

e is such that $e \perp \varphi(N)$

$$J_N(x) = J_N(x^e \bmod N)$$

Actually, something even stronger holds:

$\mathbf{RSA}_{N,e}$ is a permutation on each "quarter" of Z_N^*



In other words:

- $m \bmod p \in \mathrm{QR}_p$ **iff** $m^e \bmod p \in \mathrm{QR}_p$
- $m \bmod q \in \mathrm{QR}_q$ **iff** $m^e \bmod q \in \mathrm{QR}_q$

Example Z_{35}^{*}

We calculate $\mathbf{RSA}_{23,35}(m) = m^{23} \bmod 35$

QR$_7$ — mod 7

QR$_5$ — mod 5

	1	2	4	3	5	6
1	1	16	11	31	26	6
4	29	9	4	24	19	34
2	22	2	32	17	12	27
3	8	23	18	3	33	13

	1	4	2	5	3	6
1	1	11	16	26	31	6
4	29	4	9	19	24	34
3	8	18	23	33	3	13
2	22	32	2	12	17	27

How to prove it?

By the **CRT** and by the fact that $\boldsymbol{p}$ and $\boldsymbol{q}$ are symmetric it is enough to show that

$$\boldsymbol{m} \text{ is a } \mathbf{QR}_{\boldsymbol{p}}$$

iff

$$\boldsymbol{m}^{\boldsymbol{e}} \text{ is a } \mathbf{QR}_{\boldsymbol{p}}$$

Fact

For an odd e:

$$m^e \bmod p \text{ is a } \mathrm{QR}_p$$
$$\text{iff}$$
$$m \bmod p \text{ is a } \mathrm{QR}_p$$

Proof:

Let g be the generator of Z_p^*. Let y be such that $m = g^y$.

Recall that x is a QR_p iff x is an even power of g

We have that

$m^e \bmod p$ is a QR_p

iff

$(g^y)^e \bmod p$ is an **even** power of g

remember that p and e are **odd**

iff

$g^{ye \bmod (p-1)}$ is an **even** power of g

iff

$g^y \bmod p$ is an **even** power of g.

$= m \bmod p$

QED

Conclusion

The Jacobi symbol "leaks", i.e.:

from $\boldsymbol{c}$

one can compute $\boldsymbol{J_N(\mathbf{Dec}_{N,d}(c))}$

(without knowing the factorization of $\boldsymbol{N}$)

Is it a big problem?

Depends on the application...

Plan for today

1. Provide a formal security definition of public key encryption.

2. Modify **RSA** so that it is secure according to this definition.

Plan

1. Problems with the "handbook RSA"
2. Definition of the **CPA security**
3. Constructions of **CPA-secure RSA** encryption schemes
   1. theoretical
   2. practical
4. The **hybrid encryption** and the **KEM/DEM** paradigm
5. Definition of the **CCA security**
6. Constructions of **CCA-secure** symmetric encryption
7. Constructions of **CCA-secure RSA** encryption schemes

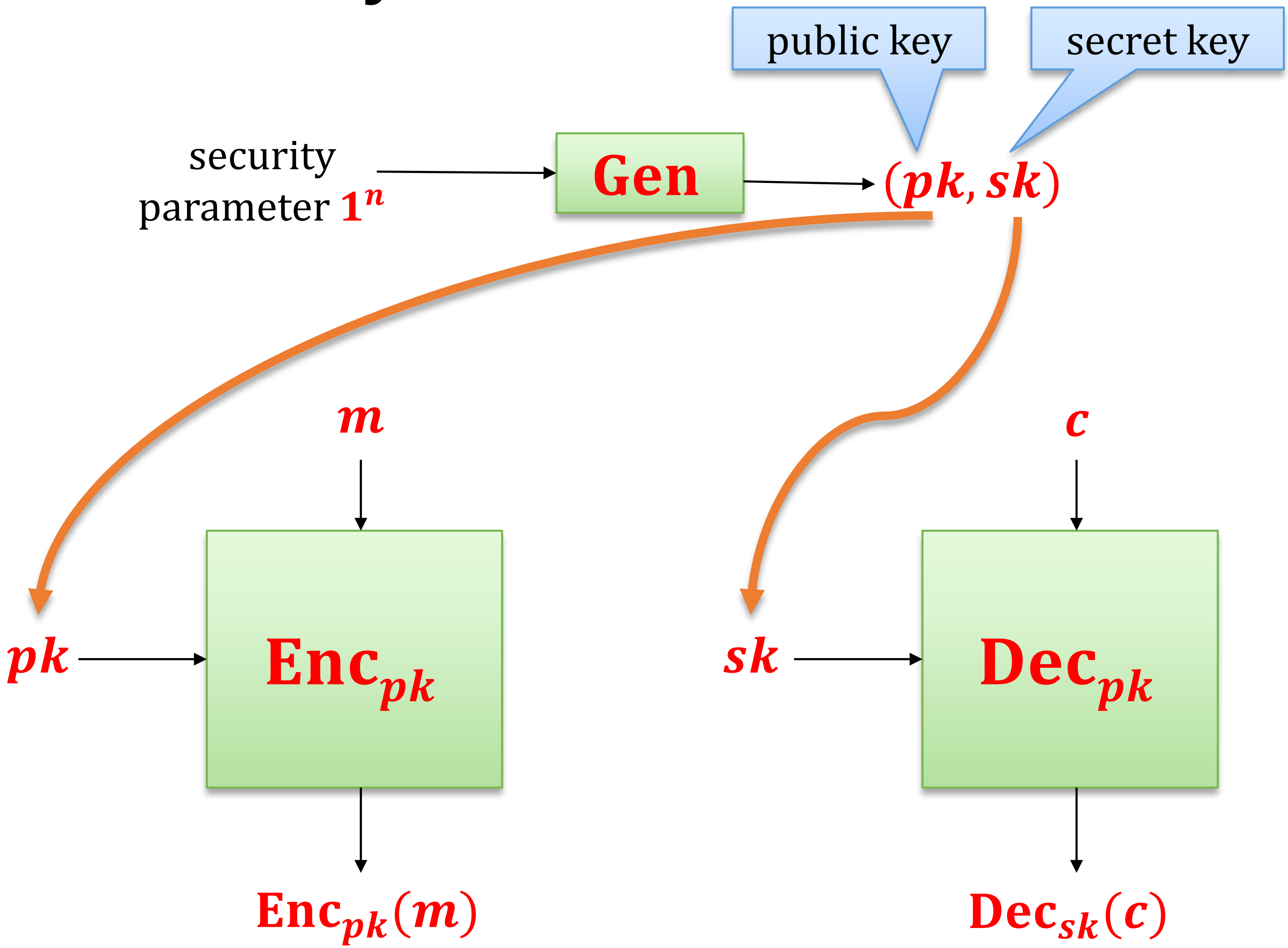
A mathematical view

A **public-key encryption (PKE)** scheme is a triple **(Gen, Enc, Dec)** of poly-time algorithms, where

- **Gen** is a **key-generation** randomized algorithm that takes as input a security parameter $\mathbf{1}^{n}$ and outputs a key pair $(\boldsymbol{pk}, \boldsymbol{sk}) \in (\{\mathbf{0}, \mathbf{1}\}^{*})^{\mathbf{2}}$.
- **Enc** is an **encryption** algorithm that takes as input the **public key pk** and a **message m** (from some set that **may depend** on $\boldsymbol{pk}$), and outputs a **ciphertext** $\boldsymbol{c}$,
- **Dec** is a **decryption** algorithm that takes as input the **private key** $\boldsymbol{sk}$ and the **ciphertext** $\boldsymbol{c}$, and outputs a **message** $\boldsymbol{m}' \in \{\mathbf{0}, \mathbf{1}\}^{*} \cup \{\perp\}$.

We will sometimes write $\mathbf{Enc}_{\boldsymbol{pk}}(\boldsymbol{m})$ and $\mathbf{Dec}_{\boldsymbol{sk}}(\boldsymbol{c})$ instead of $\mathbf{Enc}(\boldsymbol{pk}, \boldsymbol{m})$ and $\mathbf{Dec}(\boldsymbol{sk}, \boldsymbol{c})$.

Pictorially



Correctness

We will require that it always holds that

$$\boldsymbol{P(\mathbf{Dec}_{sk}(\mathbf{Enc}_{pk}(m)) \neq m)} \text{ is negligible in } \boldsymbol{n}$$

assuming that:

- $(pk, sk) \leftarrow \mathbf{Gen}(1^n)$
- and $\boldsymbol{m}$ is a "legal" plaintext for $\boldsymbol{pk}$.

The security definition

Remember the symmetric-key case?

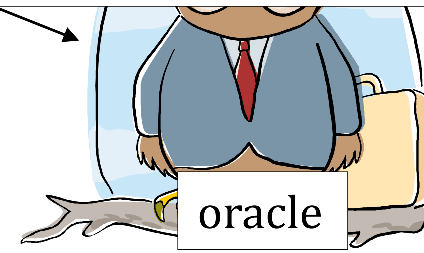
We considered a **chosen-plaintext attack**.

How would it look in the case of the **public-key encryption**?

CPA in the **symmetric** settings

security parameter
1^n

1. chooses random k
2. chooses random $b \leftarrow \{0,1\}$



oracle

chooses m'_1 $\xrightarrow{m'_1}$

$\xleftarrow{c_1 = \mathrm{Enc}(k, m'_1)}$

$\vdots$

chooses m'_t $\xrightarrow{m'_t}$

$\xleftarrow{c_t = \mathrm{Enc}(k, m'_t)}$

challenge phase:

chooses m_0, m_1 $\xrightarrow{m_0, m_1}$

$\xleftarrow{c = \mathrm{Enc}(k, m_b)}$

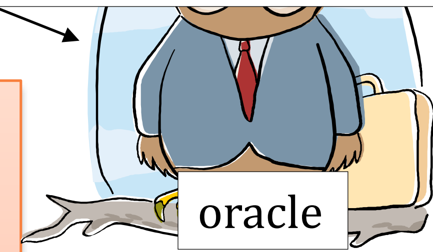
the interaction continues . . .

has to guess b

CPA in the <u>**asymmetric**</u> settings

security parameter
$\mathbf{1^n}$

1. generates $(\boldsymbol{pk}, \boldsymbol{sk}) \leftarrow \mathbf{Gen}(\mathbf{1}^n)$
2. chooses random $\boldsymbol{b} \leftarrow \{\mathbf{0}, \mathbf{1}\}$



oracle

$\boldsymbol{pk}$

chooses $\boldsymbol{m'_1}$

$\boldsymbol{m'_1}$

$\boldsymbol{c_1} = \mathbf{Enc}(\boldsymbol{pk},\boldsymbol{m'_1})$

⋮

chooses $\boldsymbol{m'_t}$

$\boldsymbol{m'_t}$

$\boldsymbol{c_t} = \mathbf{Enc}(\boldsymbol{pk},\boldsymbol{m'_t})$

This is not needed.

<u>**Why?**</u>

Because if **Eve** knows $\boldsymbol{pk}$ she can compute all these ciphertexts herself!

challenge phase:

chooses $\boldsymbol{m_0},\boldsymbol{m_1}$

$\boldsymbol{m_0},\boldsymbol{m_1}$

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

the interaction continues . . .

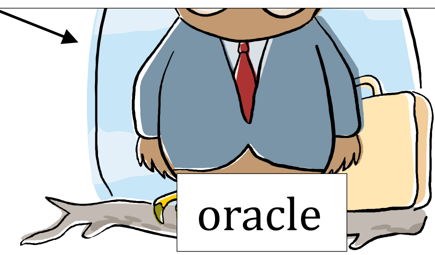
has to guess $\boldsymbol{b}$

The game after simplifications

security parameter
$\mathbf{1}^n$

1. generates $(pk, sk) \leftarrow \mathbf{Gen}(1^n)$
2. chooses random $b \leftarrow \{0, 1\}$

pk



oracle

challenge phase:

chooses m_0, m_1

m_0, m_1

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

has to guess b

CPA-security

Alternative name: CPA-secure

Security definition:

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

randomized polynomial time adversary

guesses b correctly

with probability at most $1/2 + \varepsilon(n)$, where ε is negligible.

Is the "**handbook RSA**" CPA-secure?

$N = pq$, such that p and q are random primes, and $|p| = |q|$
e – random such that $e \perp (p-1)(q-1)$
d – random such that $ed = 1 \pmod{(p-1)(q-1)}$
$pk := (N, e)$ $sk := (N, d)$

$$\mathbf{Enc}_{pk}(m) = m^e \bmod N.$$
$$\mathbf{Dec}_{sk}(c) = c^d \bmod N.$$

Not CPA-secure!

In fact: **no deterministic encryption scheme is secure**.

How can the adversary win the game?

1. he chooses any m_0, m_1,
2. computes $c_0 = \mathbf{Enc}_{pk}(m_0)$ himself
3. compares the result.

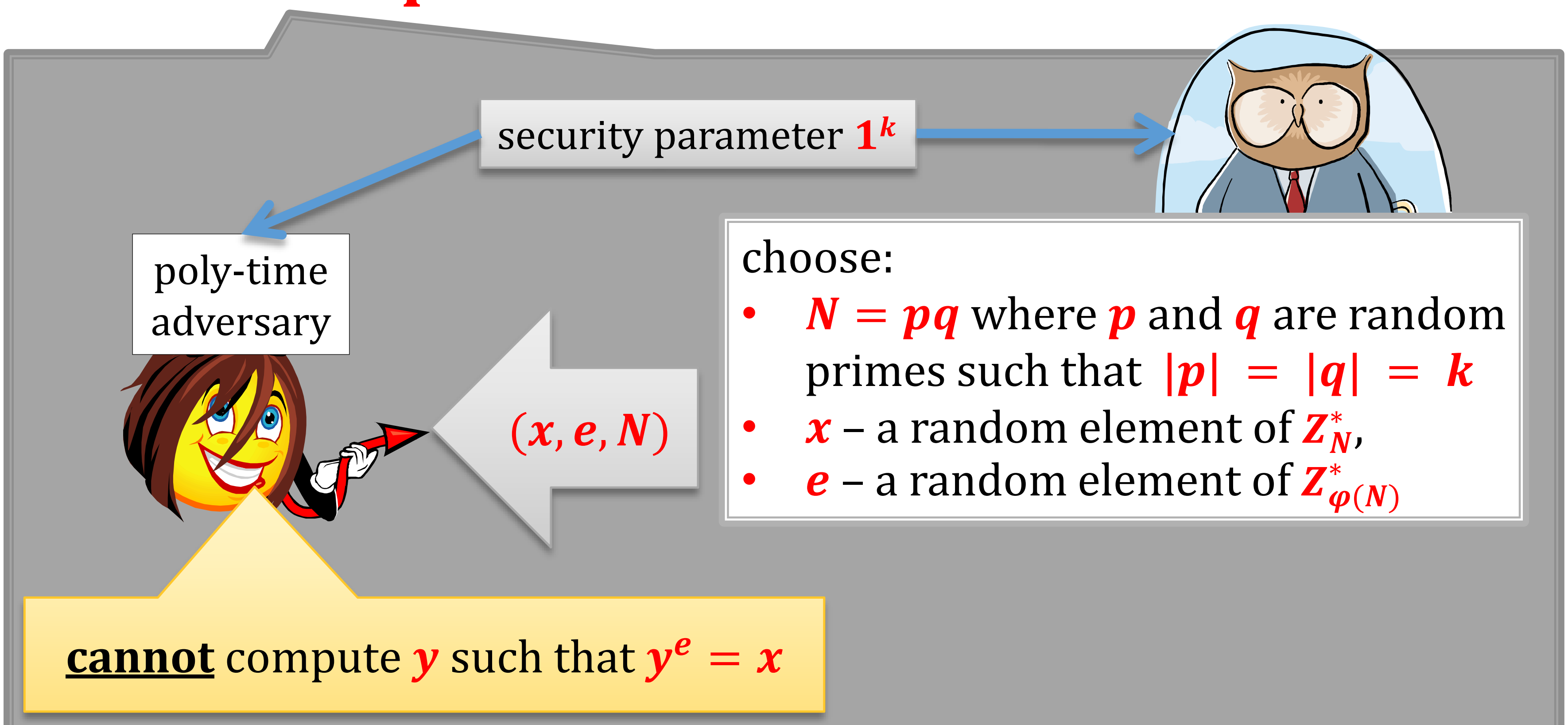
Moral: encryption has to be randomized.

Plan

1. Problems with the "handbook RSA"
2. Definition of the **CPA security**
3. Constructions of **CPA-secure RSA** encryption schemes
   1. theoretical
   2. practical
4. The **hybrid encryption** and the **KEM/DEM** paradigm
5. Definition of the **CCA security**
6. Constructions of **CCA-secure** symmetric encryption
7. Constructions of **CCA-secure RSA** encryption schemes

CPA-secure encryption from the RSA assumption

We now show how to construct a provably secure encryption scheme whose security is based on the **RSA assumption**.



Outline of the construction

1. We prove that the **least significant bit** is a **hard to compute** for **RSA**.

2. We show how to **"encrypt using this bit"**

RSA hardcore bit

Question: does **RSA** have a bit that is for sure well-hidden?

Answer: if **RSA assumption** doesn't hold, then: **no**.

But what if it holds?

Answer: **yes** – the **least significant bit of the argument is hard to compute**.

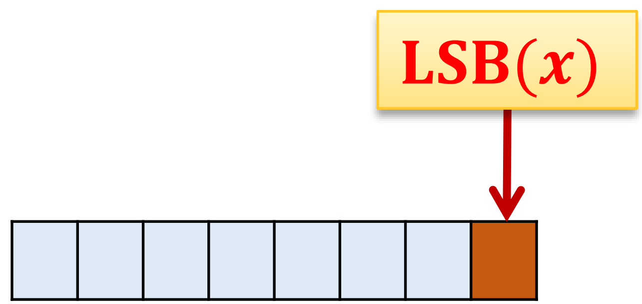
Notation

For an integer x we will write

$$\mathbf{LSB}(x)$$

to denote the least significant bit of x.

x:



In other words: $\mathbf{LSB}(x) = x \bmod 2$

Fact (informally)

LSB is the "hardest bit to compute" in **RSA**.

(it is called a "hard-core bit").

More precisely:

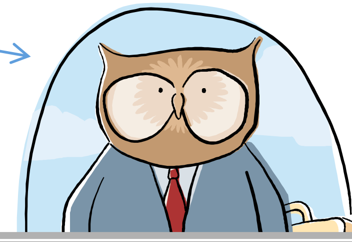
If you can compute **LSB** then you can invert **RSA**.

Note:

In some sense it is a "dual" predicate to Jacobi symbol...

"LSB game":

security parameter $\mathbf{1}^k$



choose:

- $N = pq$ where p and q are random primes such that $|p| = |q| = k$
- x – a random element of Z_N^*,
- e – a random element of $Z_{\varphi(N)}^*$

(x, e, N)

outputs b

$= x^d \bmod N$

The adversary **wins** if
b is the **least significant bit** of $y = \mathbf{Enc}_{e,N}^{-1}(x)$

Theorem

Suppose the **RSA assumption** holds.
Then every poly-time adversary wins **Game 2** with a probability at most

$$\mathbf{0.5 + \boldsymbol{\varepsilon}(\boldsymbol{k})}$$

where $\boldsymbol{\varepsilon}$ is negligible.

W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr
On the hardness of the least-signficant bits of the RSA and Rabin functions, 1984

In other words:

The least significant bit is a **hard-core bit for RSA.**

Proof strategy

Suppose we are given a poly-time adversary

that wins the **LSB game**.

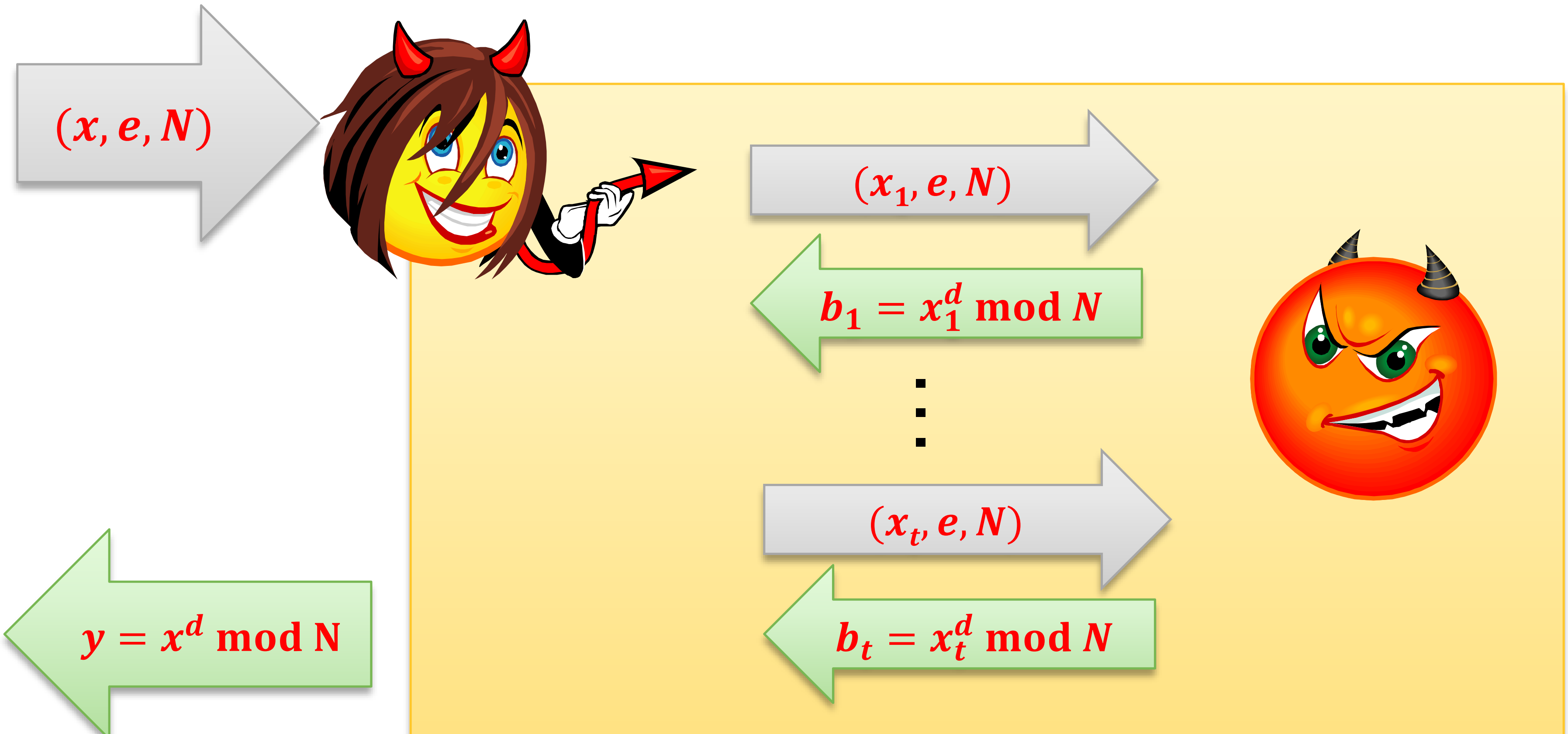
For simplicity suppose that this happens with probability **1**

(not: $\mathbf{0.5 + \varepsilon(k)}$)

We construct a poly-time adversary

that breaks the **RSA assumption**.

Outline of the construction



Observation

Adversary  that can compute

LSB of $\boldsymbol{x^d}$ **mod** $\boldsymbol{N}$.

can also be used to compute (for any $\boldsymbol{c} \in \boldsymbol{Z_N^*}$)

LSB of $\boldsymbol{c \cdot x^d}$ **mod** $\boldsymbol{N}$.

How?

$(\boldsymbol{c^e \cdot x, e, N})$

outputs

$$\boldsymbol{b' = \mathrm{LSB}((c^e \cdot x)^d) = \mathrm{LSB}(c^{ed} \cdot x^d) = \mathrm{LSB}(c \cdot x^d)}$$

The method

Let $y := x^d \bmod N$

The adversary  will use  to

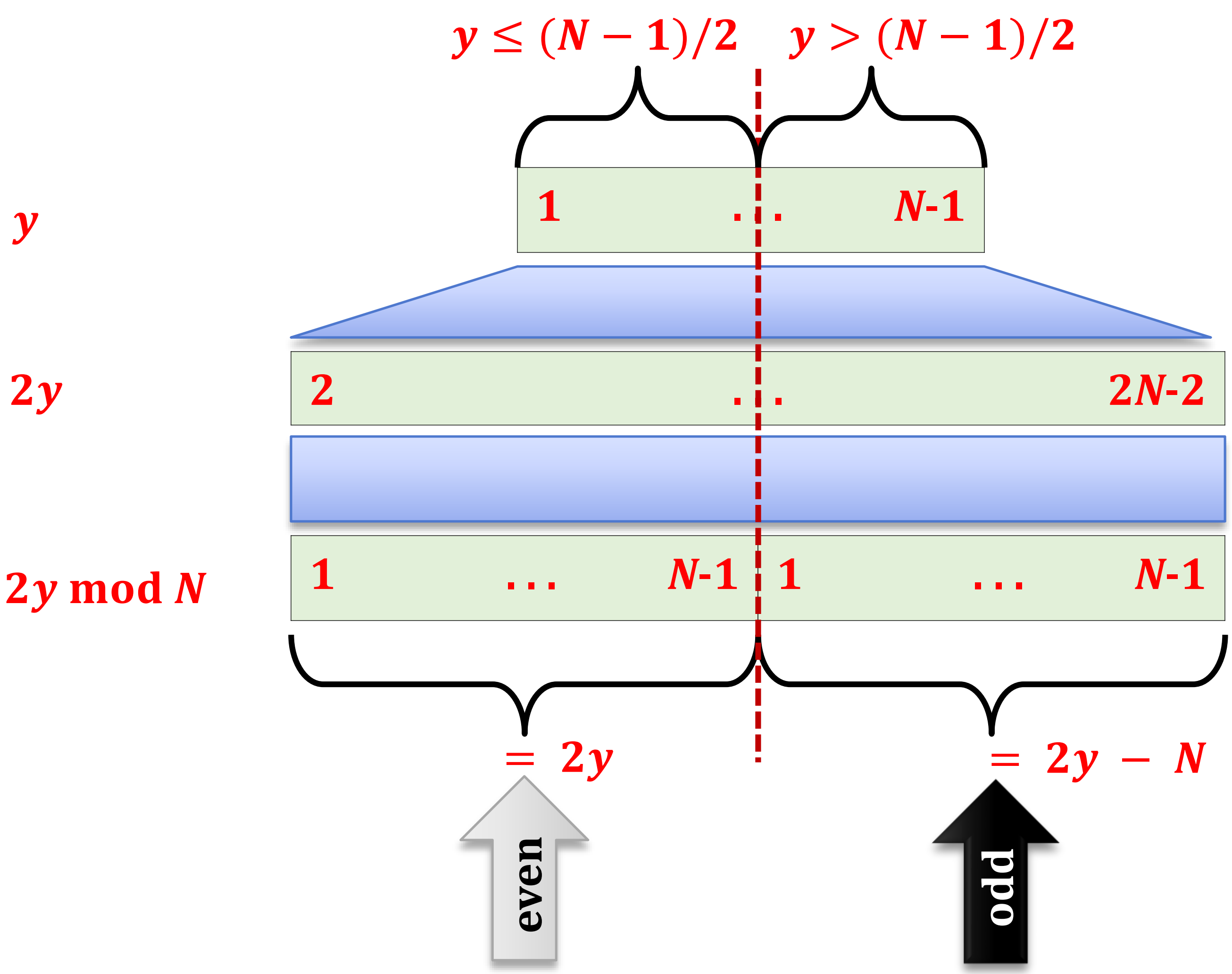
compute:

- $\mathbf{LSB}(2y)$
- $\mathbf{LSB}(4y)$
- $\mathbf{LSB}(8y)$

$\vdots$

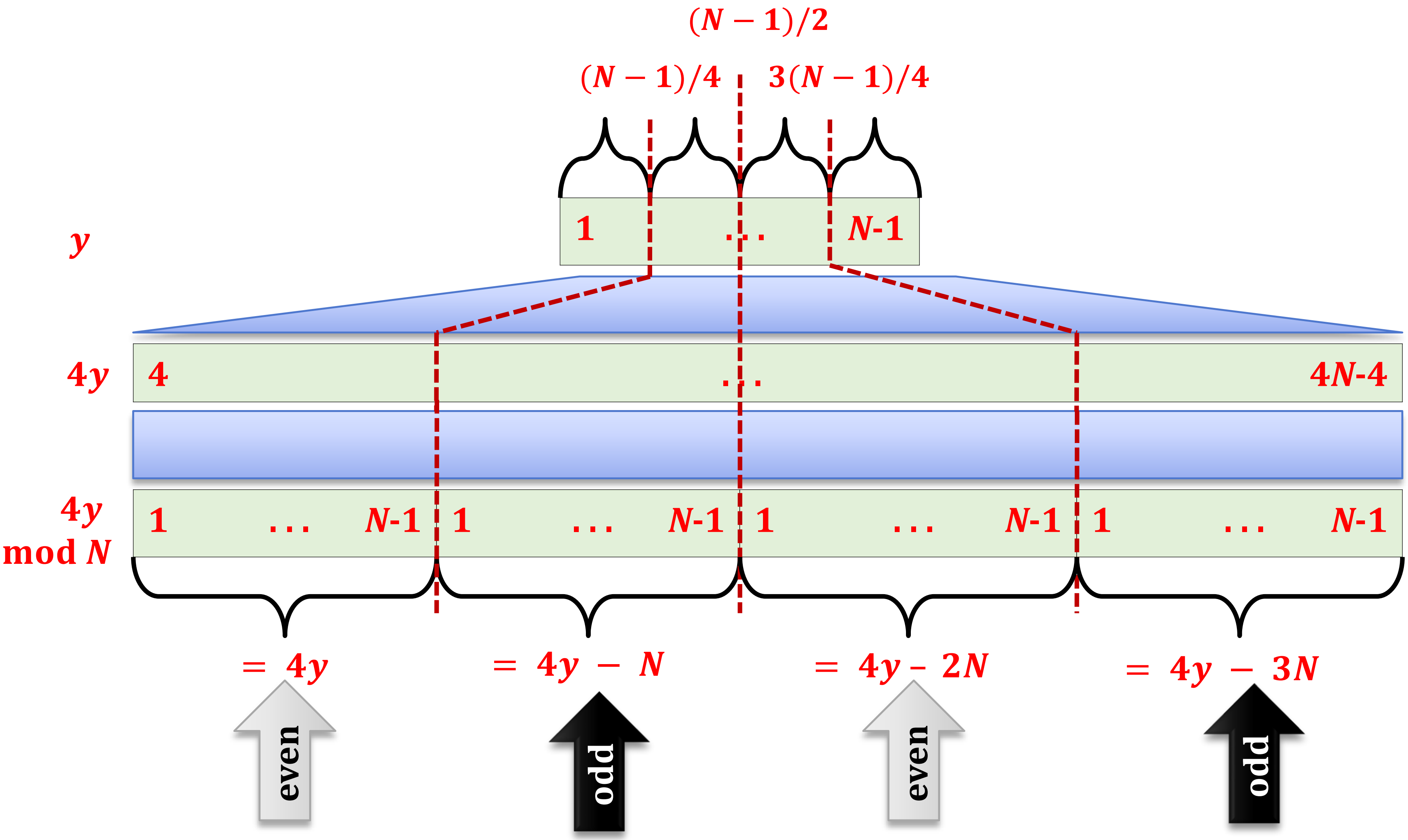
Why is it useful?

Observation

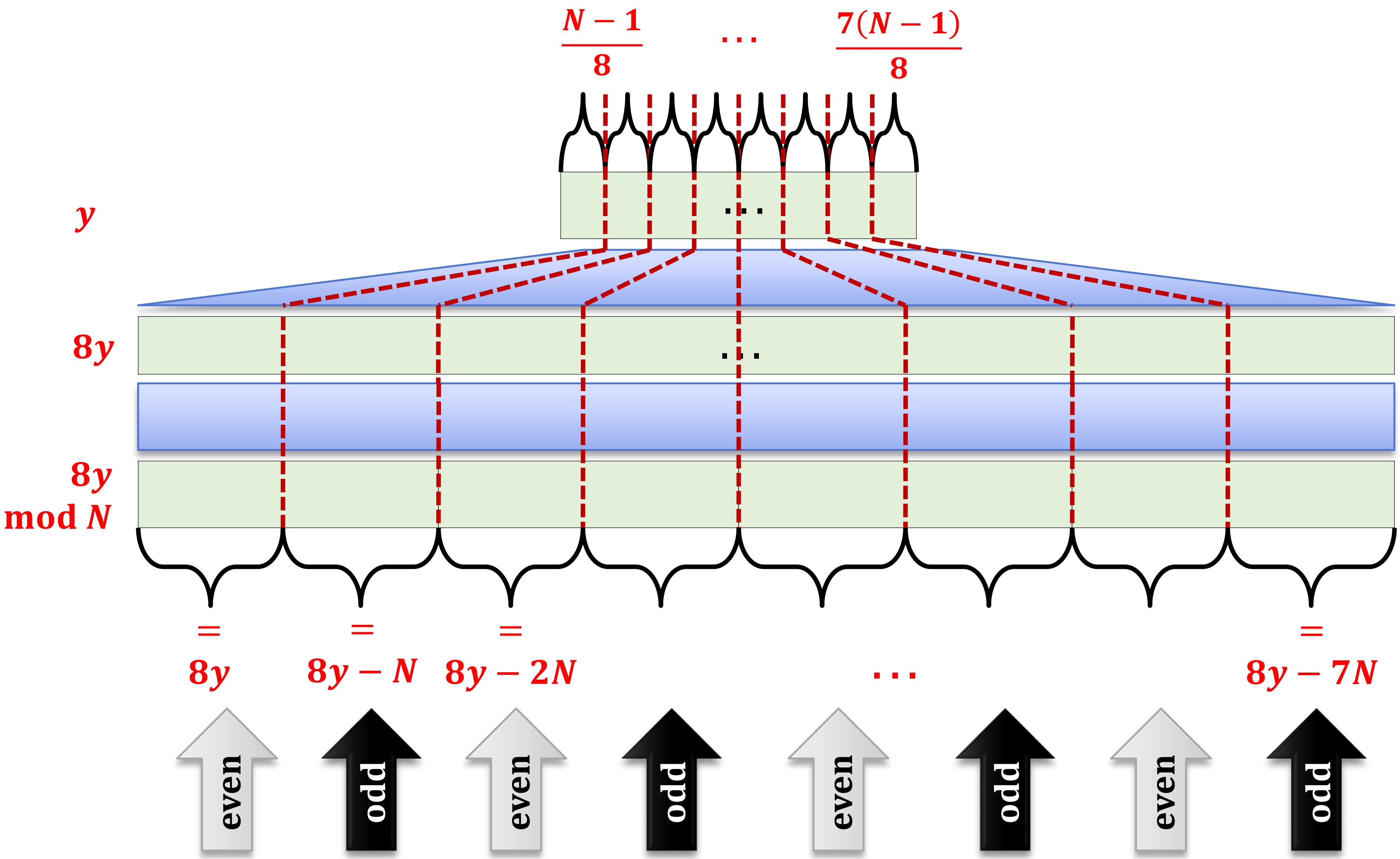


Remember:
$N = pq$ is odd

Moral: $y \in [1, \ldots, (N-1)/2]$ iff $2y \bmod N$ is even



<u>Moral</u>: $y \in \left[1, \ldots, \frac{N-1}{4}\right] \cup \left[\frac{N}{2}+1, \ldots, \frac{3(N-1)}{4}\right]$ iff $4y \bmod N$ is even



Moral: $y \in \left[1, \dots, \frac{N-1}{8}\right] \cup \left[\frac{2N}{8} + 1, \dots, \frac{3(N-1)}{8}\right] \cup \left[\frac{4N}{8} + 1, \dots, \frac{5(N-1)}{8}\right] \cup \left[\frac{6N}{8} + 1, \dots, \frac{7(N-1)}{8}\right]$ iff $8y \bmod N$ is even

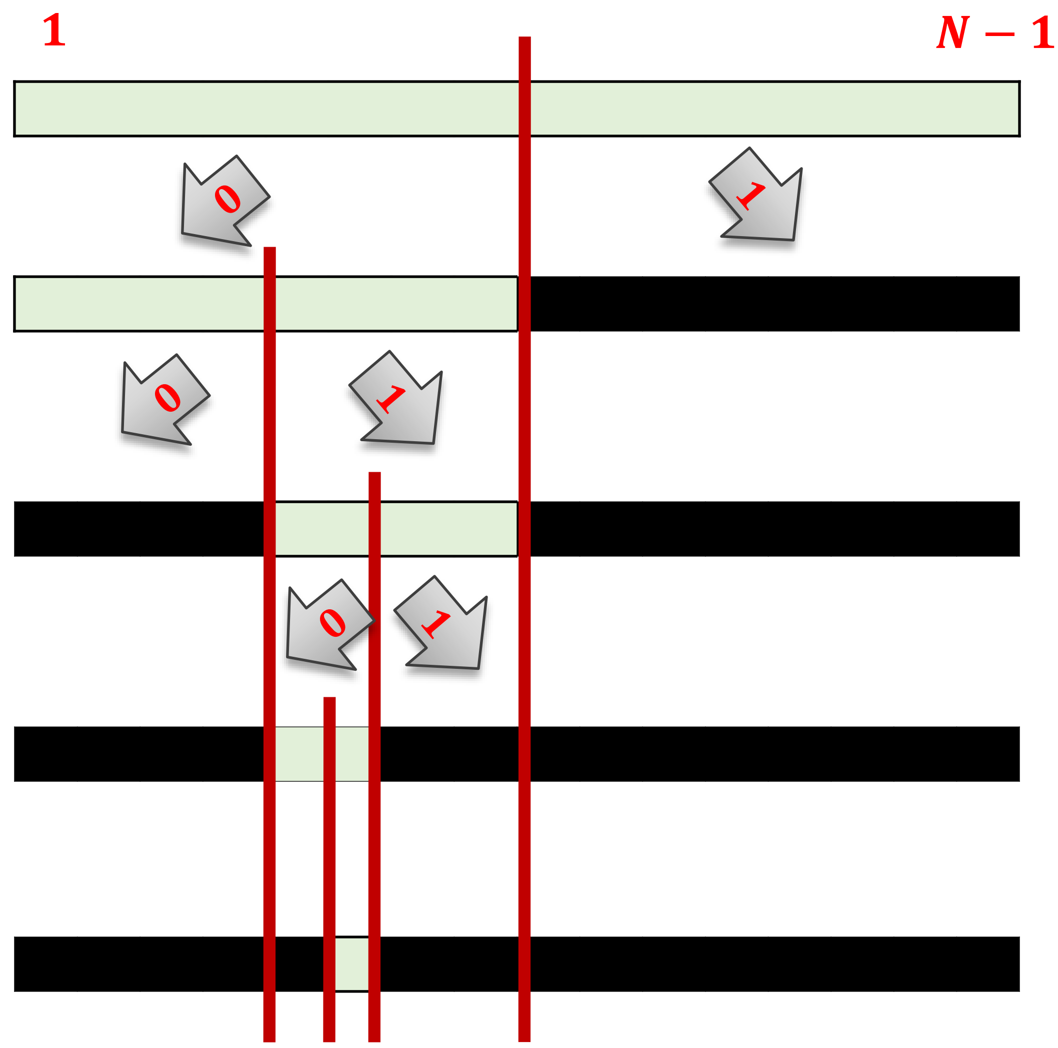
So we can use bisection

calculate $\mathbf{LSB}(2y)$

calculate $\mathbf{LSB}(4 \cdot y)$

calculate $\mathbf{LSB}(8 \cdot y)$

calculate $\mathbf{LSB}(16 \cdot y)$



⋮

QED

Why is it interesting?

We can encrypt **one bit messages** as follows:

(N, e) – **public key**
(N, d) – **private key**

$$\mathbf{Enc}_{e,N}(b) = (\mathbf{LSB}(y) \oplus b, y^e)$$

(where $y \leftarrow Z_N^*$)

$$\mathbf{Dec}_{d,N}(c, x) = \mathbf{LSB}\left(x^d\right) \oplus c$$

This is secure **under the RSA assumption**

How to extend it to longer messages?

Encrypt **bit-by-bit**:

$$\mathbf{Enc}_{e,N}(m_1, \dots, m_k) = \big((\mathbf{LSB}(y_1) \oplus m_1, \dots, \mathbf{LSB}(y_k) \oplus m_k), (y_1^e, \dots, y_k^e)\big)$$

where $y_1, \dots, y_k \leftarrow Z_N^*$

$$\mathbf{Dec}_{d,N}((c_1, \dots, c_k), (x_1, \dots, x_k)) = \big(\mathbf{LSB}(x_1^d) \oplus c_1, \dots, \mathbf{LSB}(x_k^d) \oplus c_k\big)$$

Lemma

Assume that the **RSA** **assumption holds**. Then the encryption scheme from the previous slide is **CPA-secure**.

Proof: exercise

Conclusion

Advantage:

Security of this scheme is implied by the RSA assumption.

RSA assumption holds ⇄ the public-key encryption scheme that we just constructed is secure

Disadvantage:

The ciphertext is much longer than the plaintext.

It is a rather theoretical construction!

Plan

1. Problems with the “handbook RSA”
2. Definition of the **CPA security**
3. Constructions of **CPA-secure RSA** encryption schemes
    1. theoretical
    2. practical
4. The **hybrid encryption** and the **KEM/DEM** paradigm
5. Definition of the **CCA security**
6. Constructions of **CCA-secure** symmetric encryption
7. Constructions of **CCA-secure RSA** encryption schemes

Encoding (also called: “padding”)

Before encrypting a message we usually **encode it** (adding some randomness).

This has the following advantages:

- it makes the encryption **non-deterministic**
- it **breaks the “algebraic properties”** of encryption.

How is it done in real-life?

PKCS #1: RSA Encryption Standard Version 1.5:

public-key: (N, e)
k := length on N in bytes.
D := length of the plaintext
requirement: $D \leq k - 11$.

$\mathbf{Enc}\big((N, e), m\big) := \ x^e \bmod N$, where x is equal to:

$$\overbrace{\boxed{00000000}\boxed{00000001}\underbrace{\boxed{\quad r \quad}}_{(k - D - 3)\text{ random non-zero bytes}}\boxed{00000000}\underbrace{\boxed{\quad m \quad}}_{D\text{ bytes}}}^{k\text{ bytes}}$$

How to encrypt?

$\boldsymbol{m}$

$\textbf{Encoding}(\boldsymbol{m}) :=$

00000000	00000001	r	00000000	m

$\textbf{Enc}_{e,N}$

$\textbf{Enc}_{e,N}(\textbf{Encoding}(\boldsymbol{m}))$

How to decrypt?

ciphertext y

$\mathbf{Dec}_{d,N}$

check if the format agrees....

00000000	00000001	r	00000000	m

if not then output $\perp$, otherwise

output
m

Example

If the adversary can calculate the Jacobi symbol of

00000000	**00000001**	***r***	**00000000**	***m***

most probably it doesn't help him in learning any information about ***m***...

Security of the **PKCS #1: RSA Encryption Standard Version 1.5** – security

It is **believed** to be **CPA-secure**.

(as we will later learn: it's not "**CCA-secure**")

Plan

1. Problems with the "handbook RSA"
2. Definition of the **CPA security**
3. Constructions of **CPA-secure** **RSA** encryption schemes
    1. theoretical
    2. practical
4. The **hybrid encryption** and the **KEM/DEM** paradigm
5. Definition of the **CCA security**
6. Constructions of **CCA-secure** symmetric encryption
7. Constructions of **CCA-secure** **RSA** encryption schemes

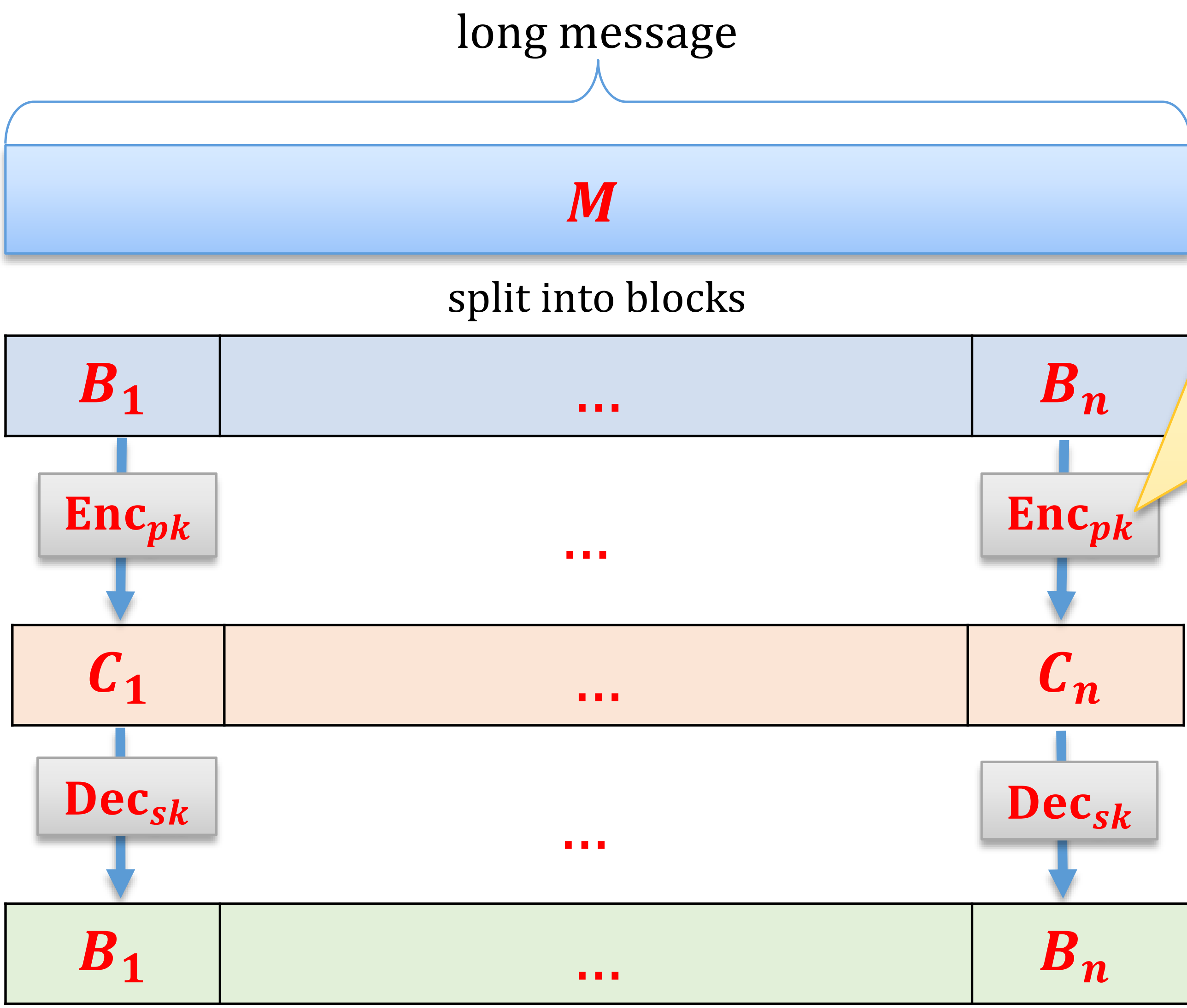
How to encrypt longer messages?

Two options:

1. divide the message in blocks and **encrypt each block separately**.

2. combine the **public-key encryption with the private-key encryption**.

Encrypting block-by-block



note: this is **randomized**, so we don't have the same problem as with the **ECB mode**

A problem with this solution

It's rather inefficient (the number of public-key operations is proportional to $|M|$)

A more efficient solution:

hybrid encryption

Ingredients for the hybrid encryption

- $(\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$– a **public key encryption scheme**

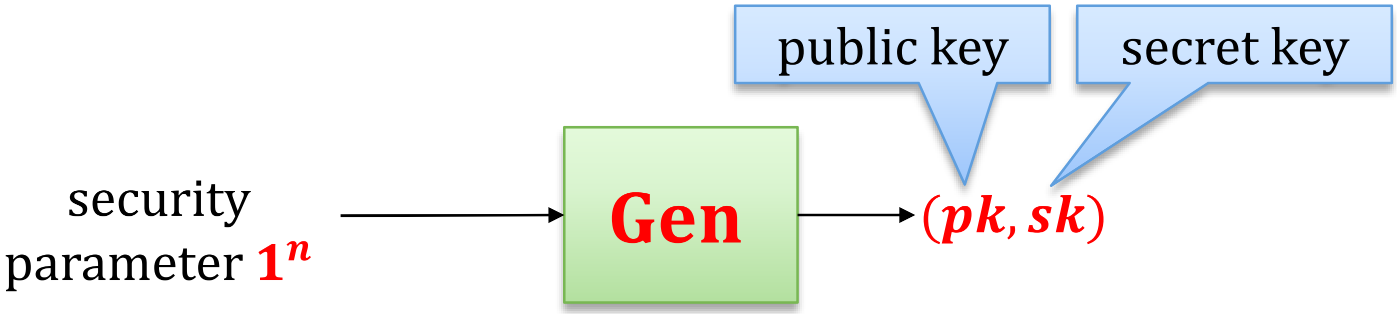
- $(\mathbf{Enc}', \mathbf{Dec}')$– a **private key encryption scheme**

Main idea:

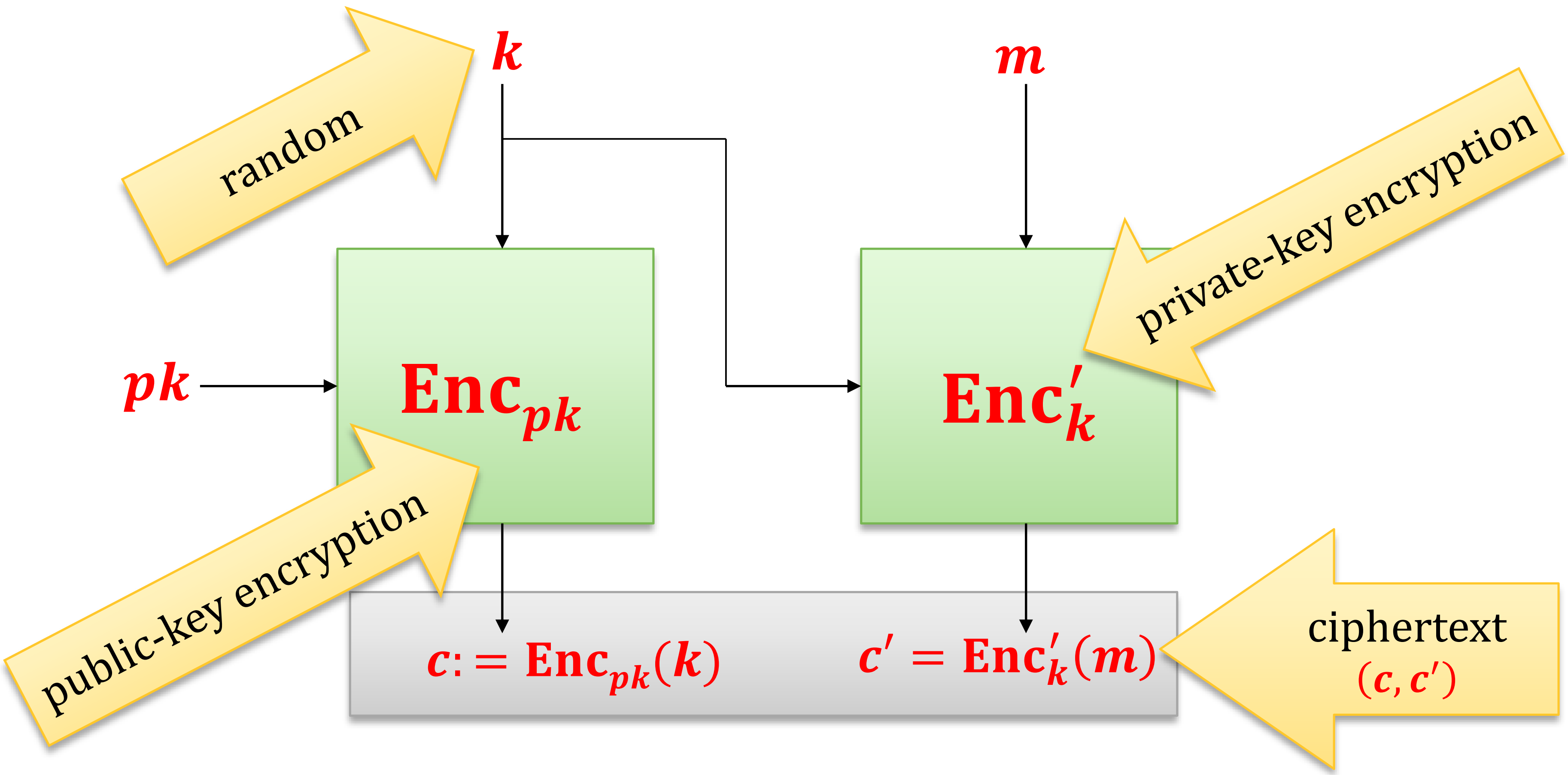
Encrypt the symmetric key with a public-key encryption scheme.

Key generation

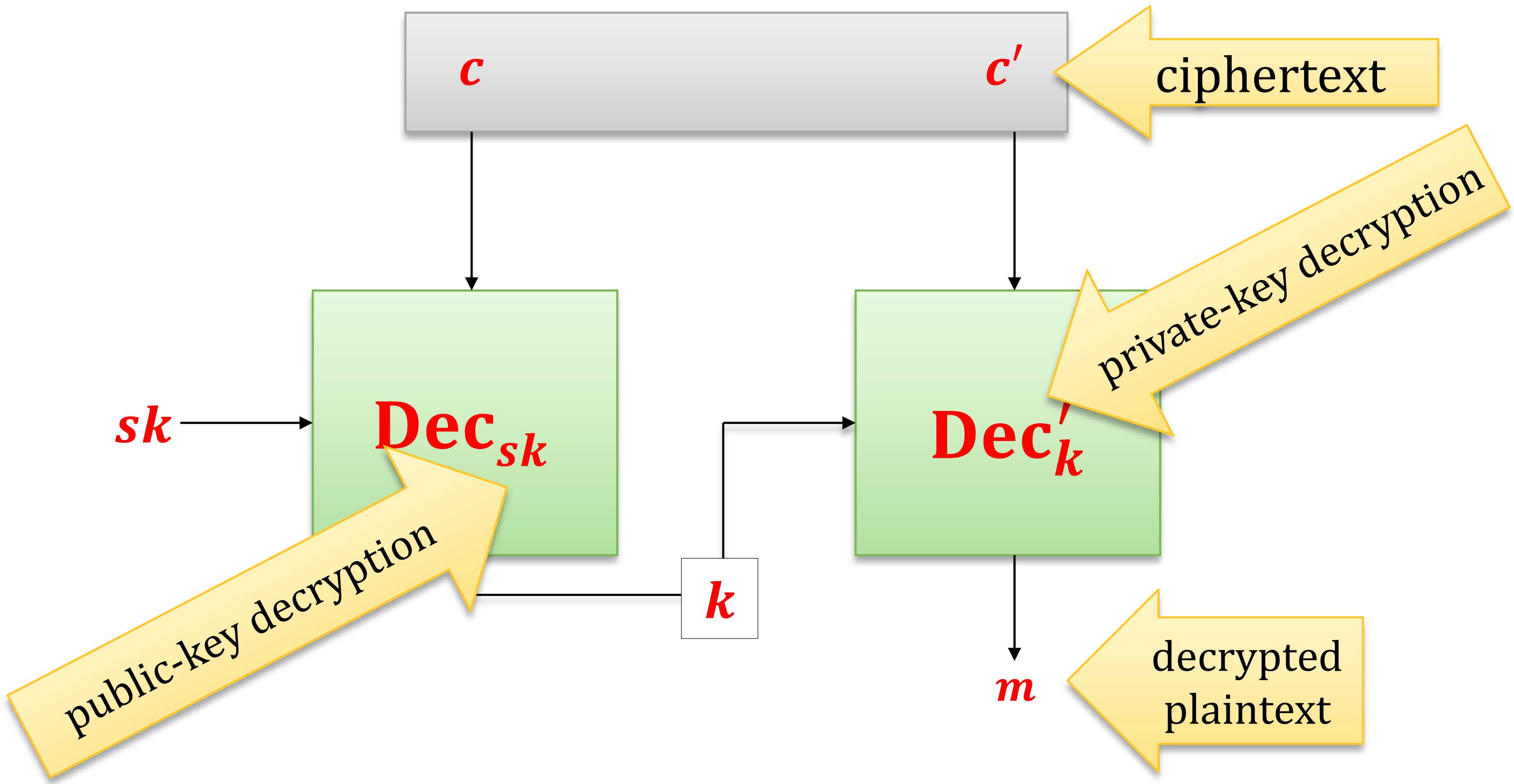
The same as in the public-key scheme:



How to encrypt?



How to decrypt?



A more direct method: the **KEM/DEM** paradigm

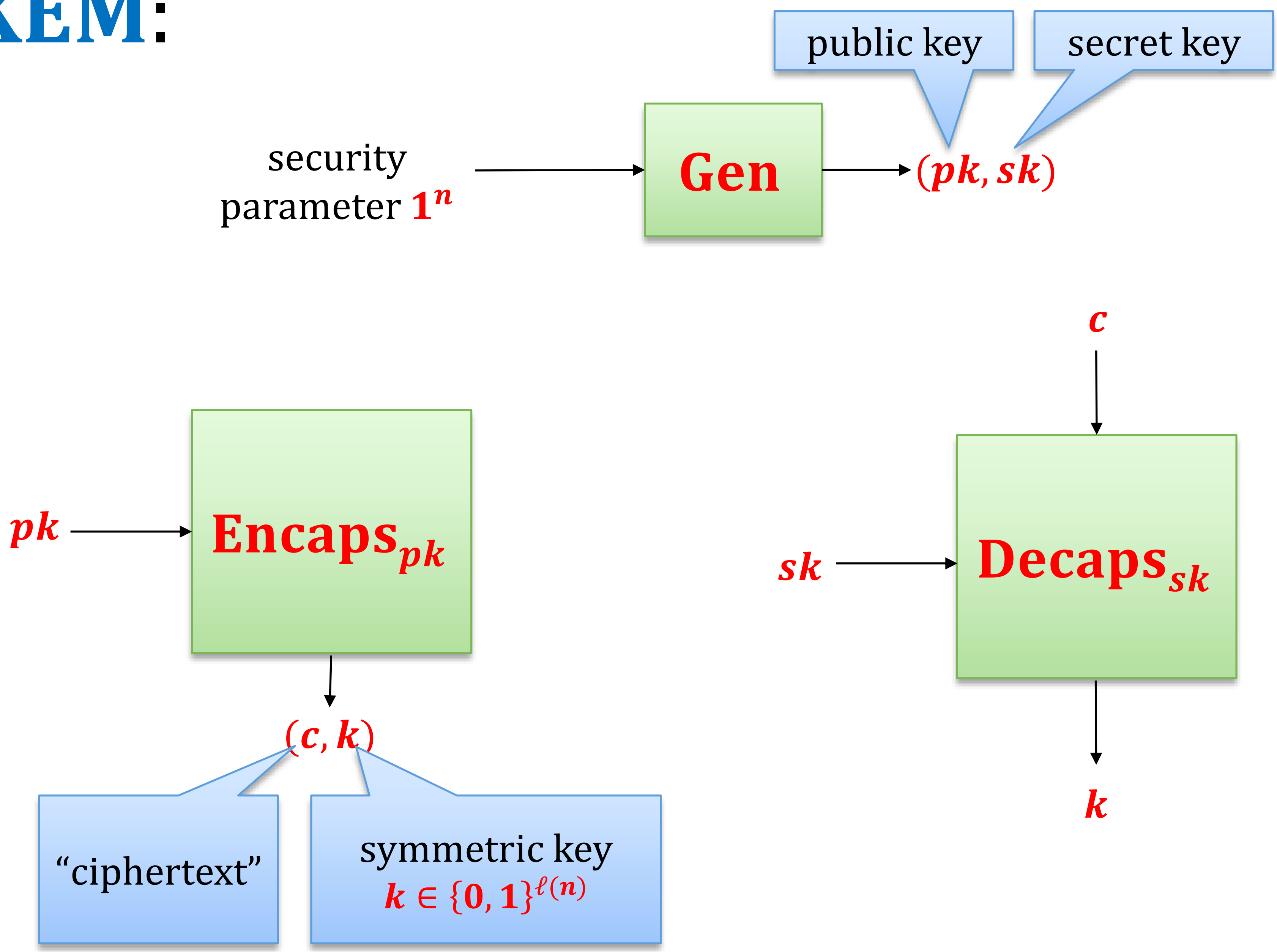
DEM – **Data Encapsulation Mechanism**

= private key encryption

KEM – **Key Encapsulation Mechanism**
consists of the following algorithms:

- **key generation algorithm Gen** – as in **PKE**,
- **encapsulation algorithm Encaps**,
- **decapsulation algorithm Decaps**.

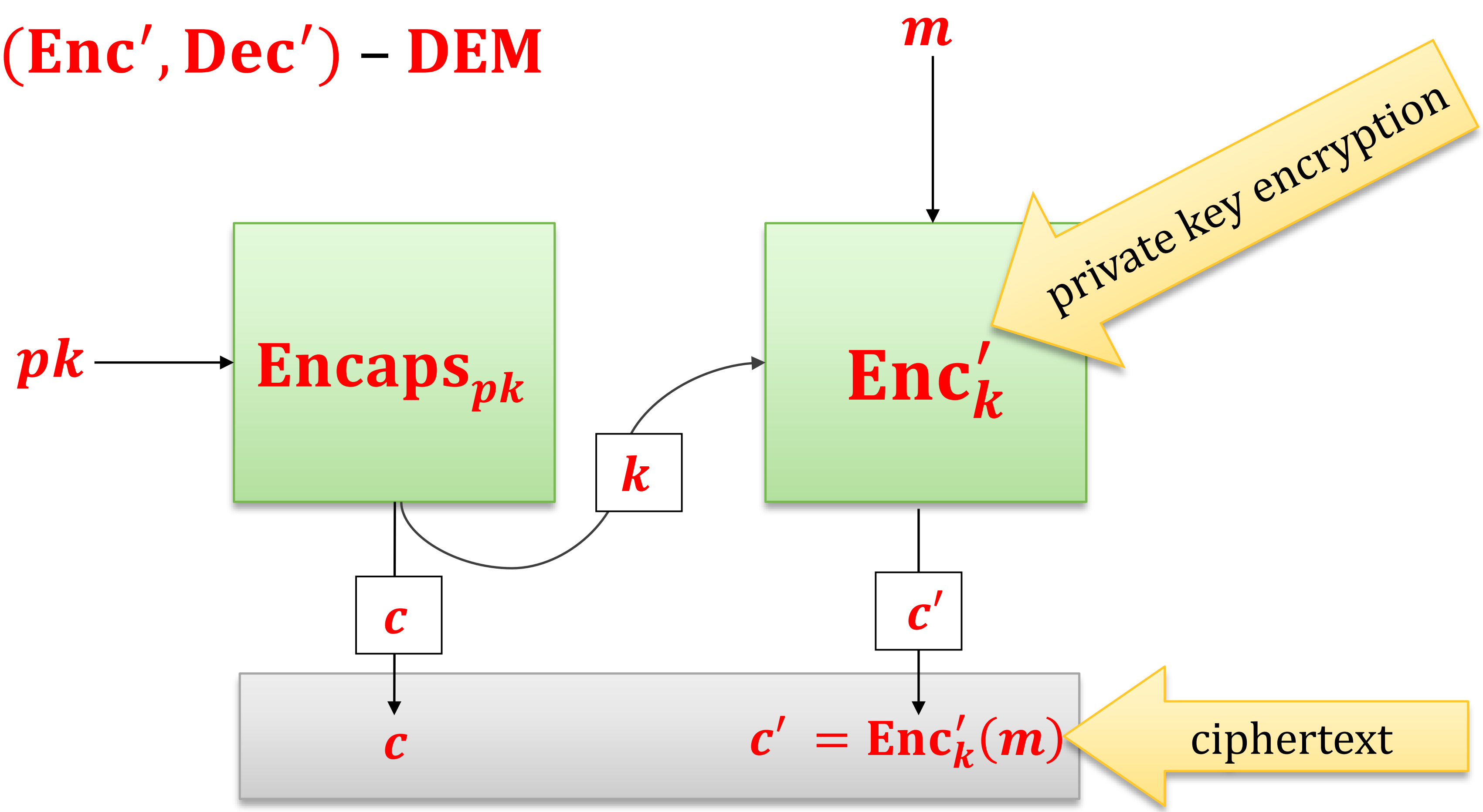
KEM:


public key
secret key
security
parameter 1^n
Gen
(pk, sk)
pk
Encaps_{pk}
(c, k)
"ciphertext"
symmetric key
$k \in \{0,1\}^{\ell(n)}$
c
sk
Decaps_{sk}
k

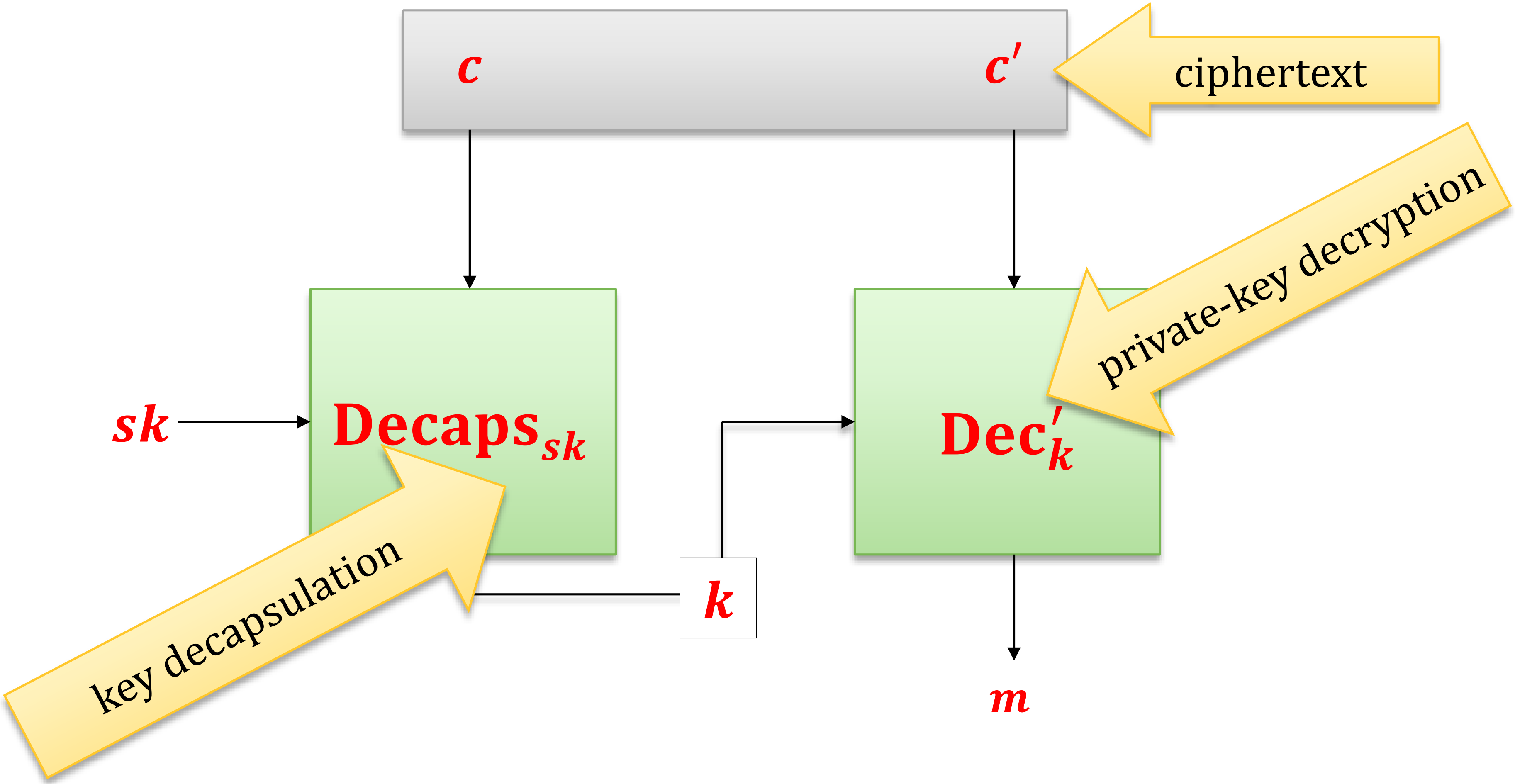
How to encrypt?

Gen – as in **KEM**

$(\mathbf{Enc}', \mathbf{Dec}')$ – **DEM**



How to decrypt?



One method to implement **KEM**

Take a **public-key encryption** scheme $(\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$.

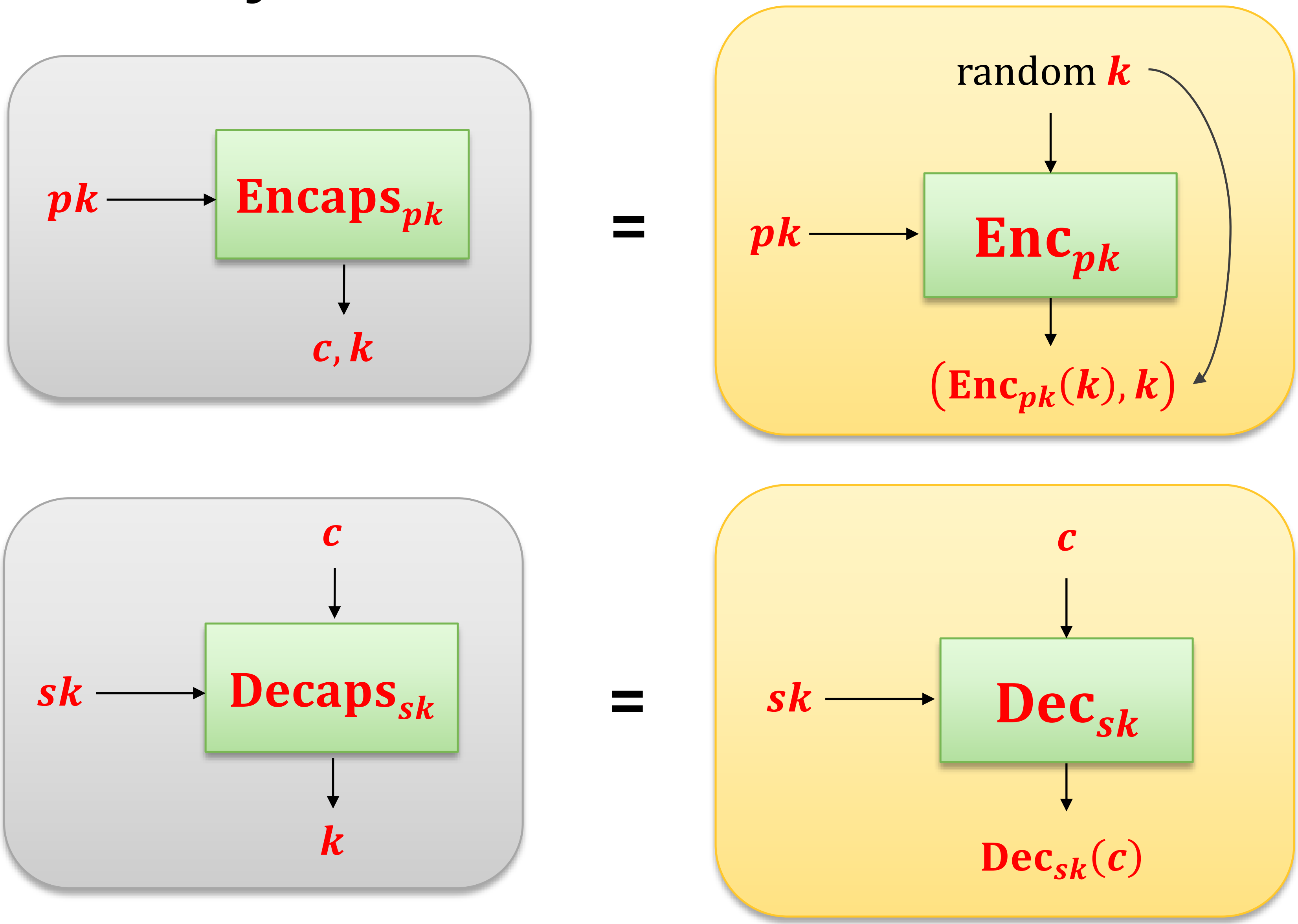
Define **KEM** as follows:

- **Gen** is the same
- $\mathbf{Encaps}_{pk}$ = generate a **random symmetric key** $\boldsymbol{k}$ and **output**

$$\left(\mathbf{Enc}_{pk}(\boldsymbol{k}), \boldsymbol{k}\right)$$

- $\mathbf{Decaps}_{sk}$ = on input $\boldsymbol{c}$ **output** $\mathbf{Dec}_{sk}(\boldsymbol{c})$

Pictorially:



Note

In this case **KEM/DEM** method is simply equal to the **hybrid encryption**.

However: there exist other, direct methods for key encapsulation.

(they are more efficient)

Consequences of these approaches

For longer messages the cost of encryption is **dominated by the cost of symmetric operations**.

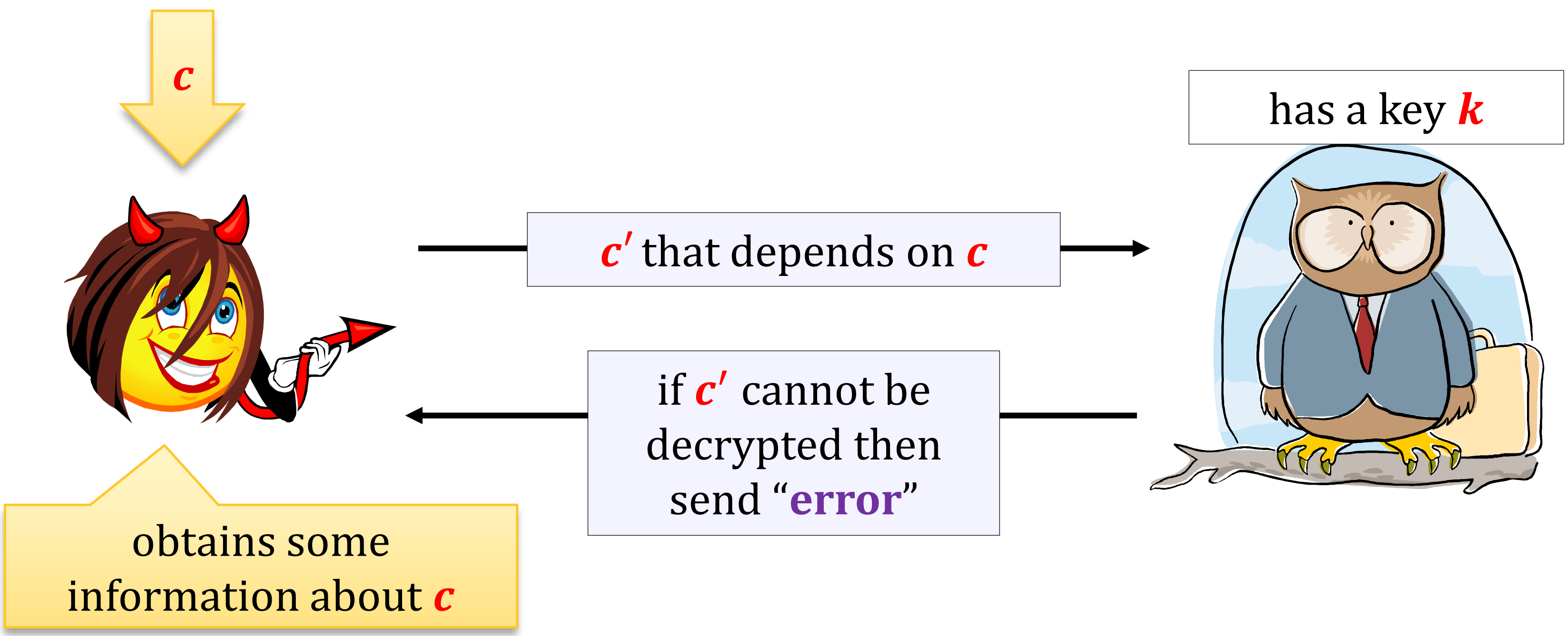
Hence: the public-key operations (amortized over the length of the messages) are almost “for free”.

Plan

1. Problems with the “handbook RSA”
2. Definition of the **CPA security**
3. Constructions of **CPA-secure** **RSA** encryption schemes
    1. theoretical
    2. practical
4. The **hybrid encryption** and the **KEM/DEM** paradigm
5. Definition of the **CCA security**
6. Constructions of **CCA-secure** symmetric encryption
7. Constructions of **CCA-secure** **RSA** encryption schemes

Chosen-ciphertext attacks – motivation

Remember the attack on the **symmetric** encryption based on the **error messages from the decryption oracle**?

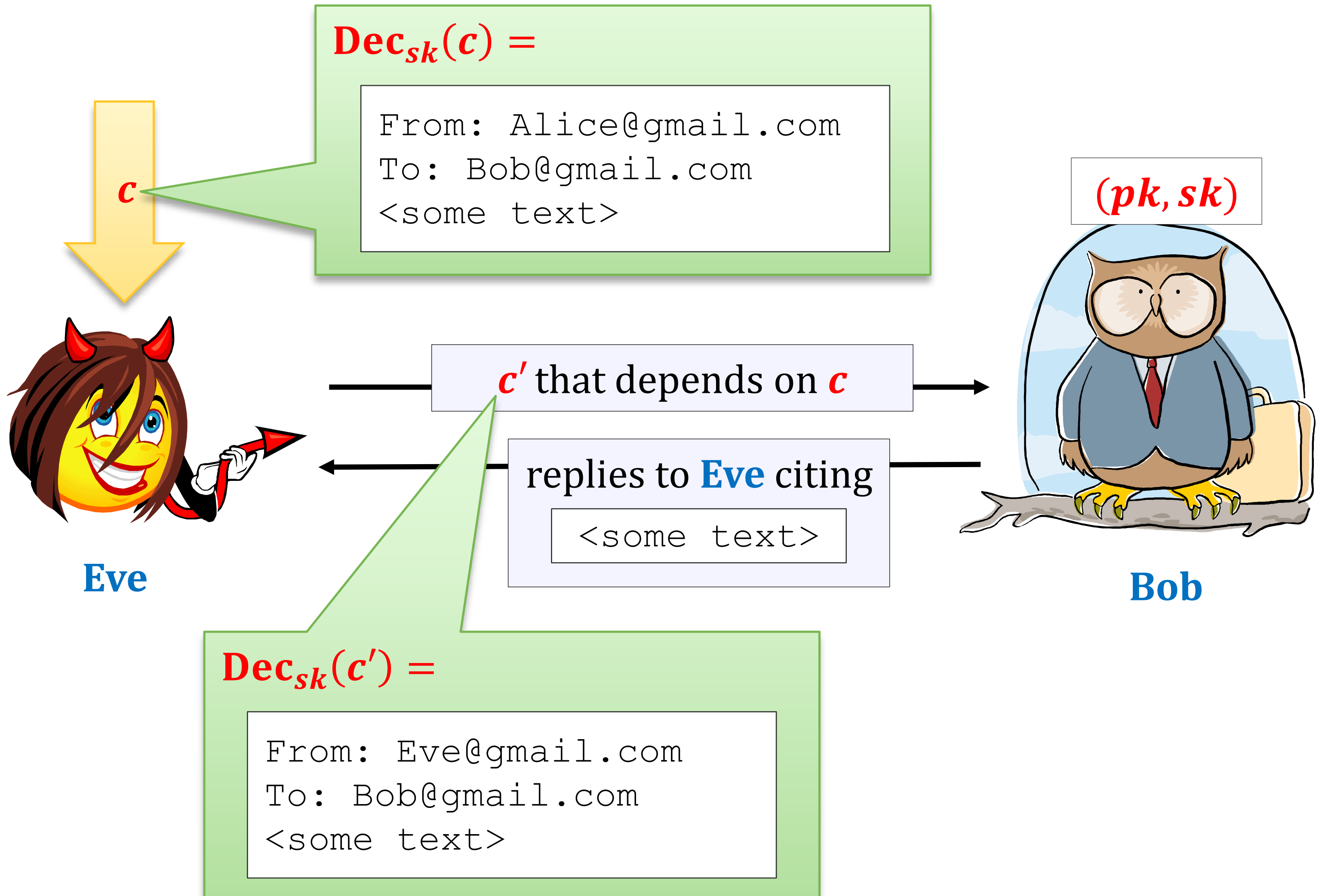


Another scenario

c

if $\mathbf{Dec}_{sk}(c)$ = "**alarm**"
then announce
"**alarm**" to everybody.

A more advanced example



Note

CPA security does not imply that such attacks are impossible.

We need a **stronger** security definition.

This will be called:

chosen-ciphertext security (CCA)

It can be defined both for the **symmetric** and **asymmetric** case.

Decryption oracle

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

c_1 →

← $\mathbf{Dec}_{sk}(c_1)$

c_2 →

← $\mathbf{Dec}_{sk}(c_2)$

⋮

c_k →

← $\mathbf{Dec}_{sk}(c_k)$

convention:

$\mathbf{Dec}_{sk}(c_i) := \perp$
if c_i cannot be decrypted

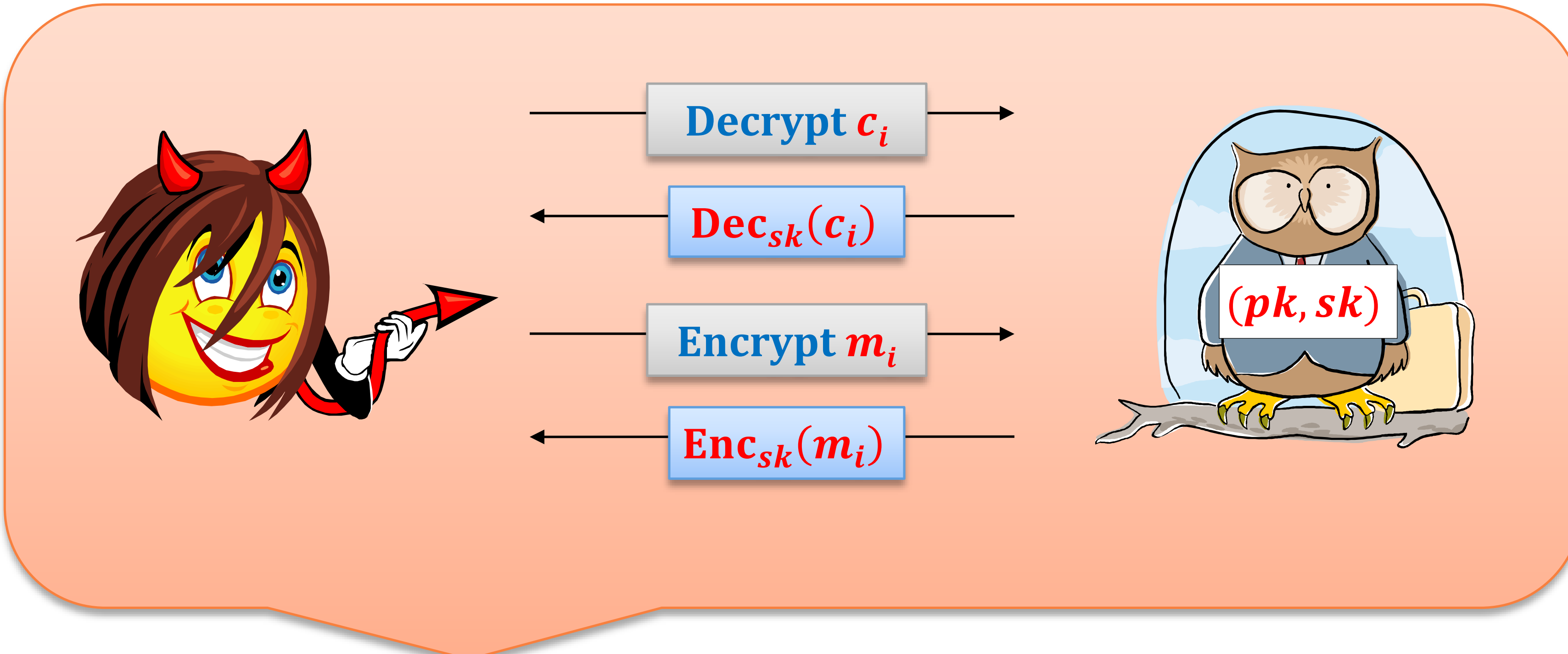
we call such a ciphertext "**invalid**"

Decryption/encryption oracle

We assume that **also CPA** is allowed.

this will be used in the symmetric case

Two types of queries:

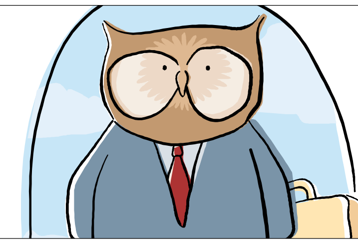


this is called a **CCA-attack**

CCA-security– the game in the symmetric case

1. generates $k \leftarrow \mathbf{Gen}(1^n)$
2. chooses random $b \leftarrow \{0, 1\}$

security parameter
1^n



decryption/encryption oracle

CCA-attack

challenge phase:

chooses m_0, m_1

m_0, m_1

$c = \mathbf{Enc}_k(m_b)$

CCA-attack

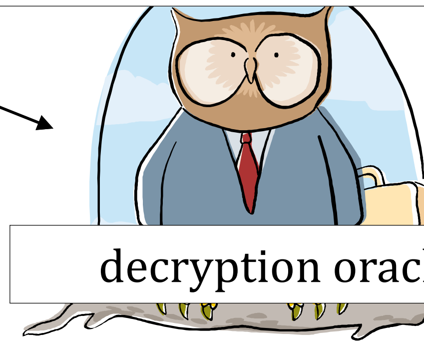
has to guess b

Here the adversary **cannot** ask for decryption of c.

CCA-security– the game in the asymmetric case

security parameter 1^n

1. generates $(pk, sk) \leftarrow \mathrm{Gen}(1^n)$
2. chooses random $b \leftarrow \{0,1\}$



decryption oracle

pk

CCA-attack

challenge phase:

chooses m_0, m_1

m_0, m_1

$c = \mathrm{Enc}_{pk}(m_b)$

CCA-attack

m_0, m_1 have to be **legal for pk**

has to guess b

Here the adversary **cannot** ask for decryption of c.

CCA security

Alternative name: **CCA-secure**

Security definition (in the **asymmetric** case):

In the **symmetric** case: **(Enc, Dec)**

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 $1/2 + \varepsilon(n)$, where ε is negligible.

Easy to see

CCA-security implies **CPA security**

(because the adversary in the “**CCA game**” is at least as powerful as the one in the “**CPA game**”)

What about the implication in the other direction?

CPA-security does **not** imply the CCA-security

$\boldsymbol{m_0, m_1}$

$\boldsymbol{c = \mathbf{Enc}(k, mb)}$

here ask about the “related” c'

CCA-attack

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

Informally:

To win the game it is enough that Eve is computes some c' such that $\mathbf{Dec}_k(c')$ is “*related to*” $\mathbf{Dec}_k(c)$.
(**Why?** Because then she is allowed to ask for it.)

For example: it is possible for any stream cipher!

if $c' = c' \oplus (1, \ldots, 1)$ then $\mathbf{Dec}_k(c') = \mathbf{Dec}_k(c) \oplus (1, \ldots, 1)$

How to construct CCA-secure schemes?

- in the **symmetric case**: **easy**
- in the **asymmetric case**: usually **harder** (also: in this case the "**CCA attacks**" are more realistic).

Plan

1. Problems with the "handbook RSA"
2. Definition of the **CPA security**
3. Constructions of **CPA-secure RSA** encryption schemes
   1. theoretical
   2. practical
4. The **hybrid encryption** and the **KEM/DEM** paradigm
5. Definition of the **CCA security**
6. Constructions of **CCA-secure** symmetric encryption
7. Constructions of **CCA-secure RSA** encryption schemes

Symmetric case

Simplest method: **authenticate** every ciphertext with a MAC.

Ingredients:

- **$(\mathbf{Enc}, \mathbf{Dec})$** – a **CPA-secure symmetric encryption** scheme
- **$(\mathbf{Tag}, \mathbf{Vrfy})$** – a **message authentication code** that is **strongly secure**.

A MAC is **strongly secure** if the adversary cannot produce a valid tag t' on a message m even if he saw a valid pair (m, t) (where $t \neq t$)

The method from previous lectures

encrypt-then-authenticate:

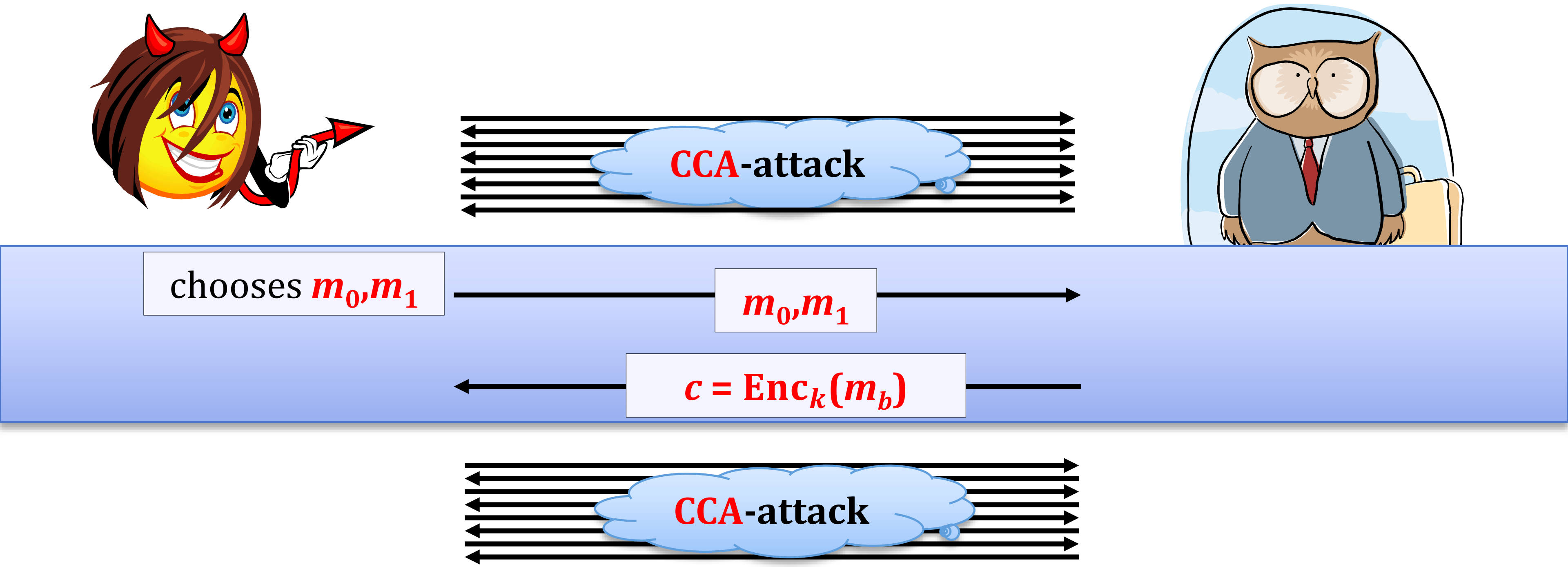
key: a pair $(\boldsymbol{k_1}, \boldsymbol{k_2})$

- to **encrypt** $\boldsymbol{m}$ compute $\boldsymbol{c} := \mathbf{Enc}_{\boldsymbol{k_1}}(\boldsymbol{m})$ and $\boldsymbol{t} := \mathbf{Tag}_{\boldsymbol{k_2}}(\boldsymbol{c})$, and output $(\boldsymbol{c}, \boldsymbol{t})$
- to **decrypt** $(\boldsymbol{c}, \boldsymbol{t})$:

  if $\mathbf{Vrfy}_{\boldsymbol{k_2}}(\boldsymbol{c}, \boldsymbol{t}) = \mathbf{no}$ then output $\perp$

  otherwise output $\mathbf{Dec}_{\boldsymbol{k_1}}(\boldsymbol{c})$

Why is this secure?



The adversary **cannot "produce himself" a valid ciphertext.**

The only decryption queries **Decrypt** c' on which he doesn't get $\perp$ are such that he received c' from the oracle before.

But he already knows the decryptions of such c's.

So: the CCA attack does not help him!

Plan

1. Problems with the “handbook RSA”
2. Definition of the **CPA security**
3. Constructions of **CPA-secure RSA** encryption schemes
    1. theoretical
    2. practical
4. The **hybrid encryption** and the **KEM/DEM** paradigm
5. Definition of the **CCA security**
6. Constructions of **CCA-secure** symmetric encryption
7. Constructions of **CCA-secure RSA** encryption schemes

PKCS #1 v.2 is not CCA-secre

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 Blaichenbacher's attack practical?

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

Bleichenbacher's attack – the scenario

$pk = (N, e)$

c

Goal:
compute $c^d \bmod N$

c_1 →

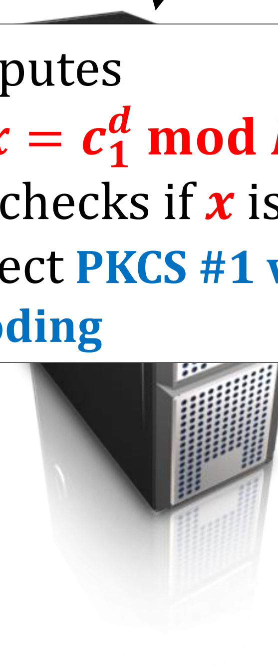
← yes/no

⋮

c_k →

← yes/no

$sk = (N, d)$



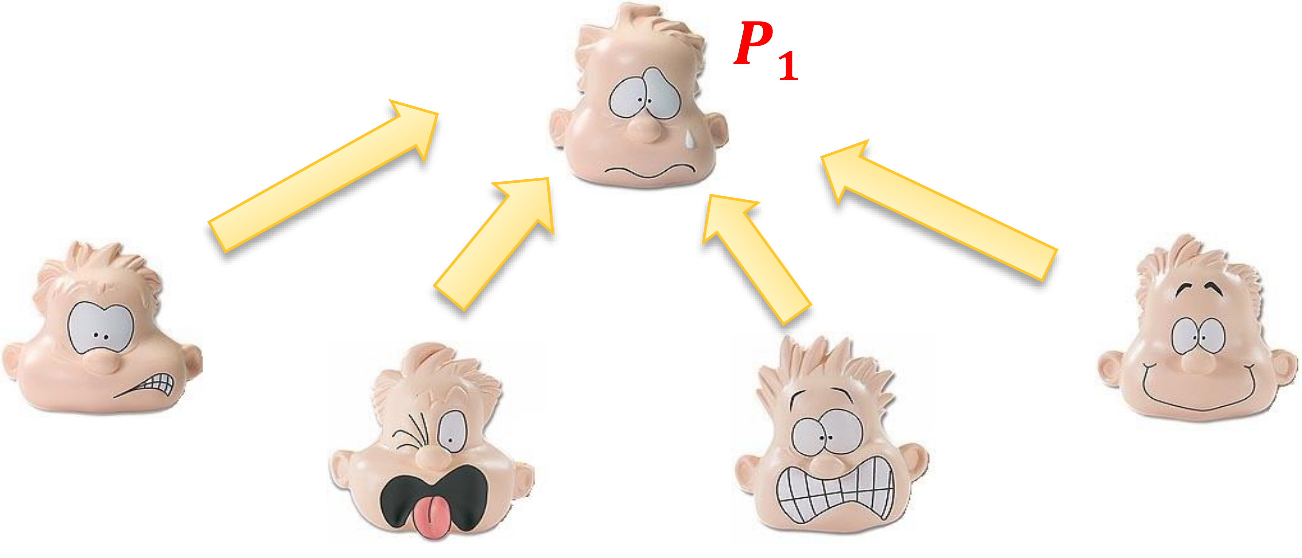
computes
$x = c_1^d \bmod N$
and checks if x is a correct **PKCS #1 v2 encoding**

Bleichenbacher [1998]:
There exists a successful attack that requires $k = 2^{20}$ questions for $|N| = 1024$.

How to construct CCA-secure encryption scheme from RSA?

Observation: MACs don't help (at least directly).

Because in the asymmetric case the parties don't share a key for a MAC.



First attempt

Idea: take a **symmetric-key CCA-secure** scheme **(Enc', Dec')** and use it in the **KEM/DEM** method.

r is random from Z_N^*

public key: (N, e)

private key: (N, d)

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

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

Problem

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

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

Typically $|N| = 1024$ and length of the symmetric 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!

$\boldsymbol{t}$ – length of the symmetric key
$\boldsymbol{H}: \{\mathbf{0},\mathbf{1}\}^* \rightarrow \{\mathbf{0},\mathbf{1}\}^{\boldsymbol{t}}$ – a hash function

$$\mathbf{Enc}((\boldsymbol{N},\boldsymbol{e}),\boldsymbol{m}) := (\boldsymbol{r}^{\boldsymbol{e}} \,\mathbf{mod}\, \boldsymbol{N}, \mathbf{Enc}'(\boldsymbol{H}(\boldsymbol{r}),\boldsymbol{m}))$$

$$\mathbf{Dec}((\boldsymbol{N},\boldsymbol{d}),(\boldsymbol{c}_0,\boldsymbol{c}_1)) \coloneqq \mathbf{Dec}'\left(\boldsymbol{H}\left(\boldsymbol{c}_0^{\boldsymbol{d}} \,\mathbf{mod}\, \boldsymbol{N}\right),\boldsymbol{c}_1\right)$$

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.**

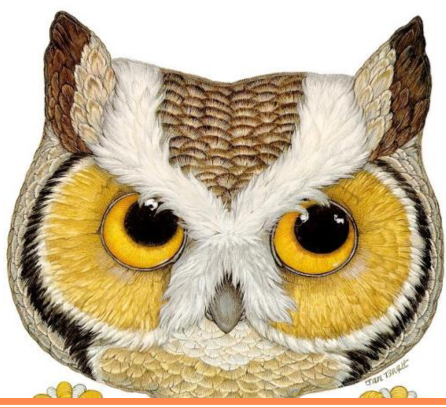
Remember the **Random Oracle Model**?

Random oracle model

hash functions ≈ **random oracles**

$\boldsymbol{x}$

$\boldsymbol{H(x)}$



$\boldsymbol{H}: \{\mathbf{0}, \mathbf{1}\}^* \to \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{L}}$

a completely random function

Security proof – the intuition

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

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 $r^e \mathbf{\ mod\ } N$, 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 $(\mathbf{Enc}, \mathbf{Dec})$ follows from the **CCA-security** of $(\mathbf{Enc}', \mathbf{Dec}')$.

A drawback of this method

$$\mathbf{Enc}((\boldsymbol{N}, \boldsymbol{e}), \boldsymbol{m}) := (\boldsymbol{r}^{\boldsymbol{e}} \ \mathbf{mod}\ \boldsymbol{N}, \mathbf{Enc}'(\boldsymbol{H}(\boldsymbol{r}), \boldsymbol{m}))$$

The ciphertext is longer than $\boldsymbol{N}$ even if the message is short.

Therefore in practice another method is used:

Optimal Asymmetric Encryption Padding (OAEP).

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 spoted 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

N – RSA modulus

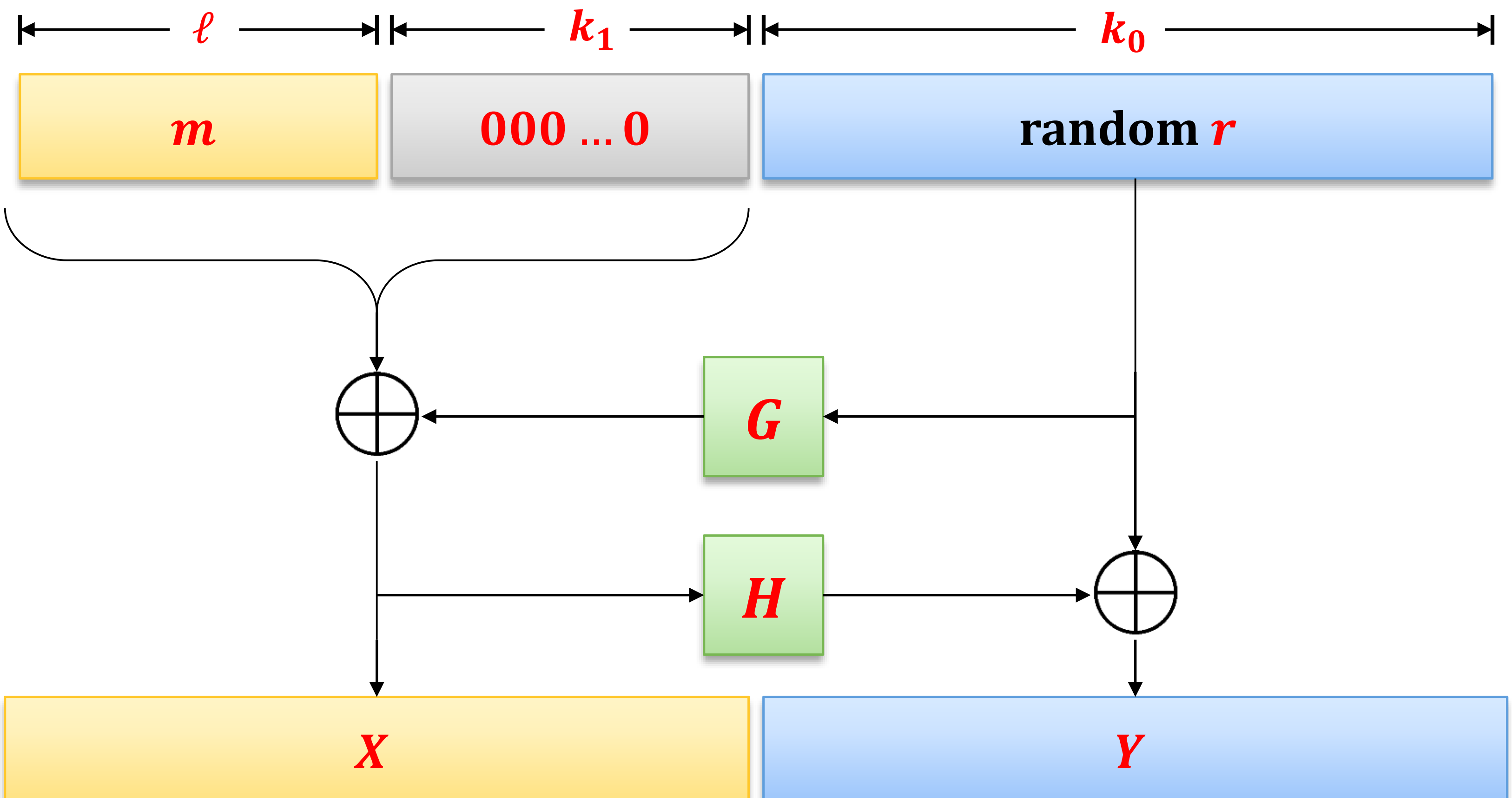
ℓ, k_1, k_2 – parameters such that

$\ell + k_1 + k_2 \leq \lfloor \log_2 N \rfloor$

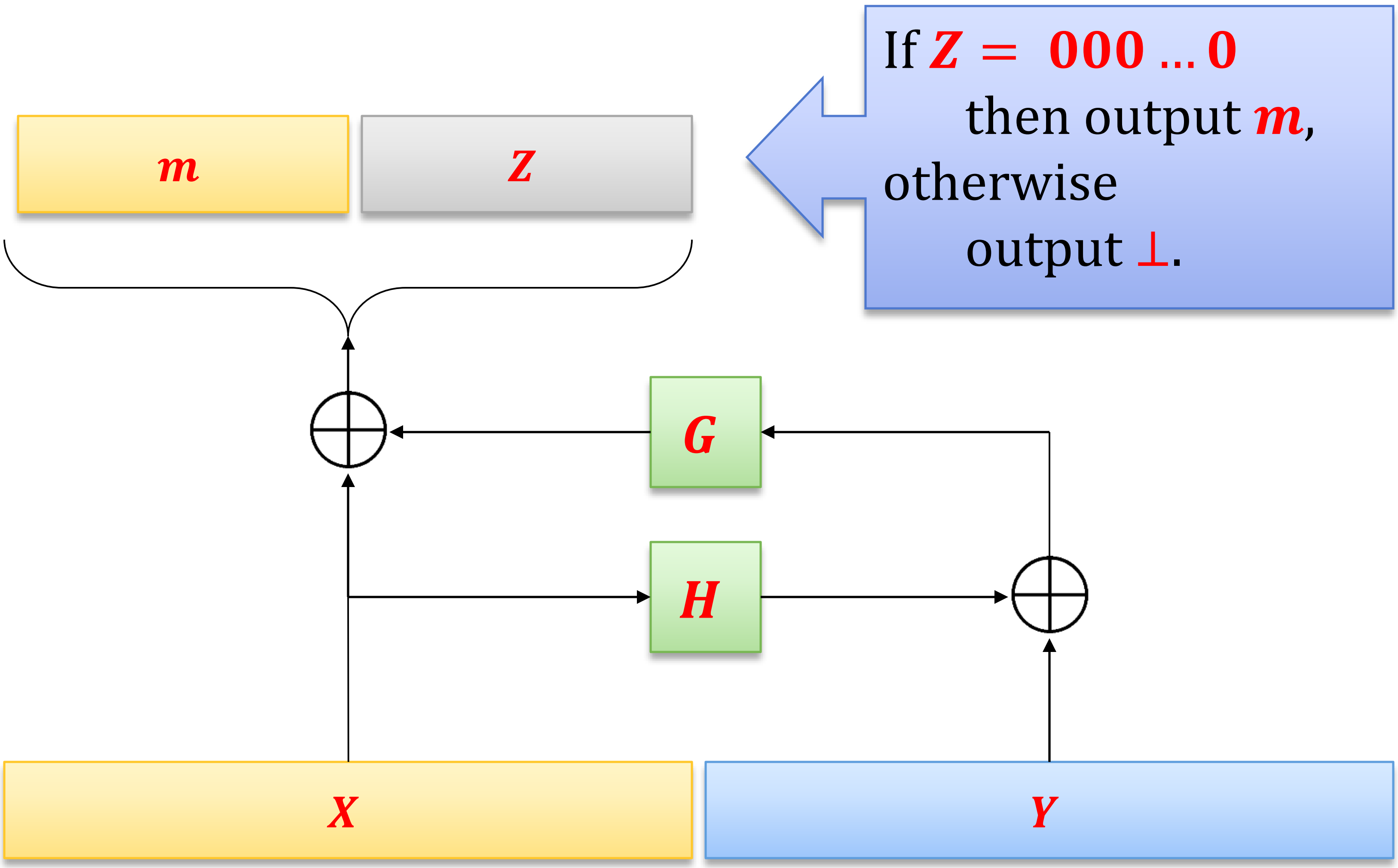
hash functions:

- $G: \{0,1\}^{k_0} \to \{0,1\}^{\ell + k_1}$,
- $H: \{0,1\}^{\ell + k_1} \to \{0,1\}^{k_0}$

OAEP(m) :=



How to invert?



RSA-OAEP

key pair like in the **handbook RSA**:

private key: (N, d)

public key: (N, e)

$$\mathbf{Enc}\big((N, e), m\big) := (\mathbf{OAEP}(m))^e \bmod N$$

$\mathbf{Dec}\big((N, e), c\big) :=$ let $x := c^d \bmod N$

if $x > 2^{\ell + k_1 + k_2}$ then output $\perp$

otherwise output $\mathbf{OAEP}^{-1}(x)$

Security of RSA-OAEP

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

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

We do not present the proof here.

We just mention some **nice properties** of this encoding.

Nice properties of OAEP
(for the right choice of parameters)

- it is **invertible**

good for the **CPA-security**

- but **to invert you need to know** (X, Y) **completely**
- for every message m the encoding

$$\mathbf{OAEP}(m)$$

is uniformly **random**

good for the **CCA-security**

- It is hard to produce a valid (X, Y) "without knowing m first"

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

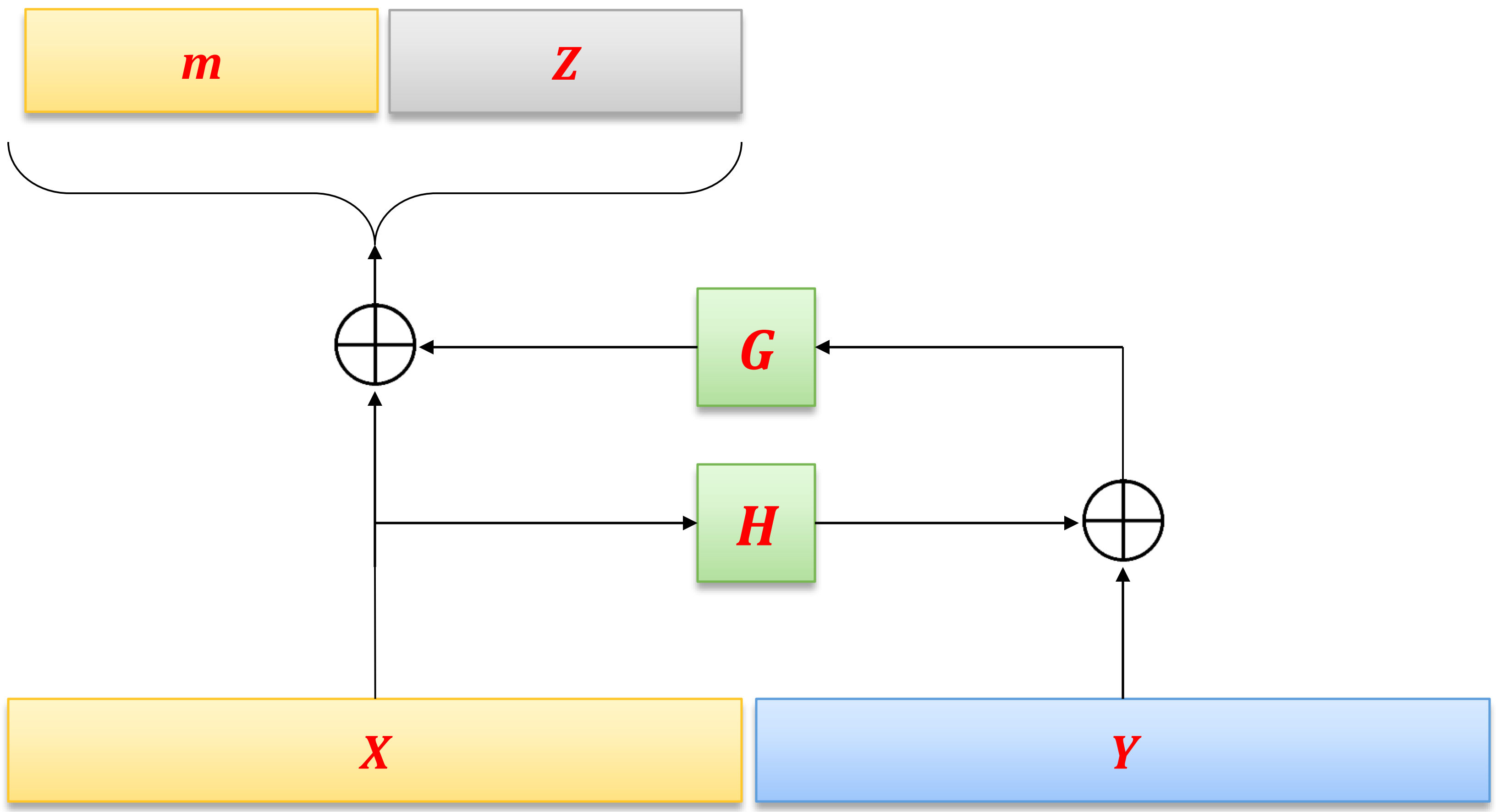
Actually:

m is completely hidden in such a case.

(assuming G and H are random oracles)

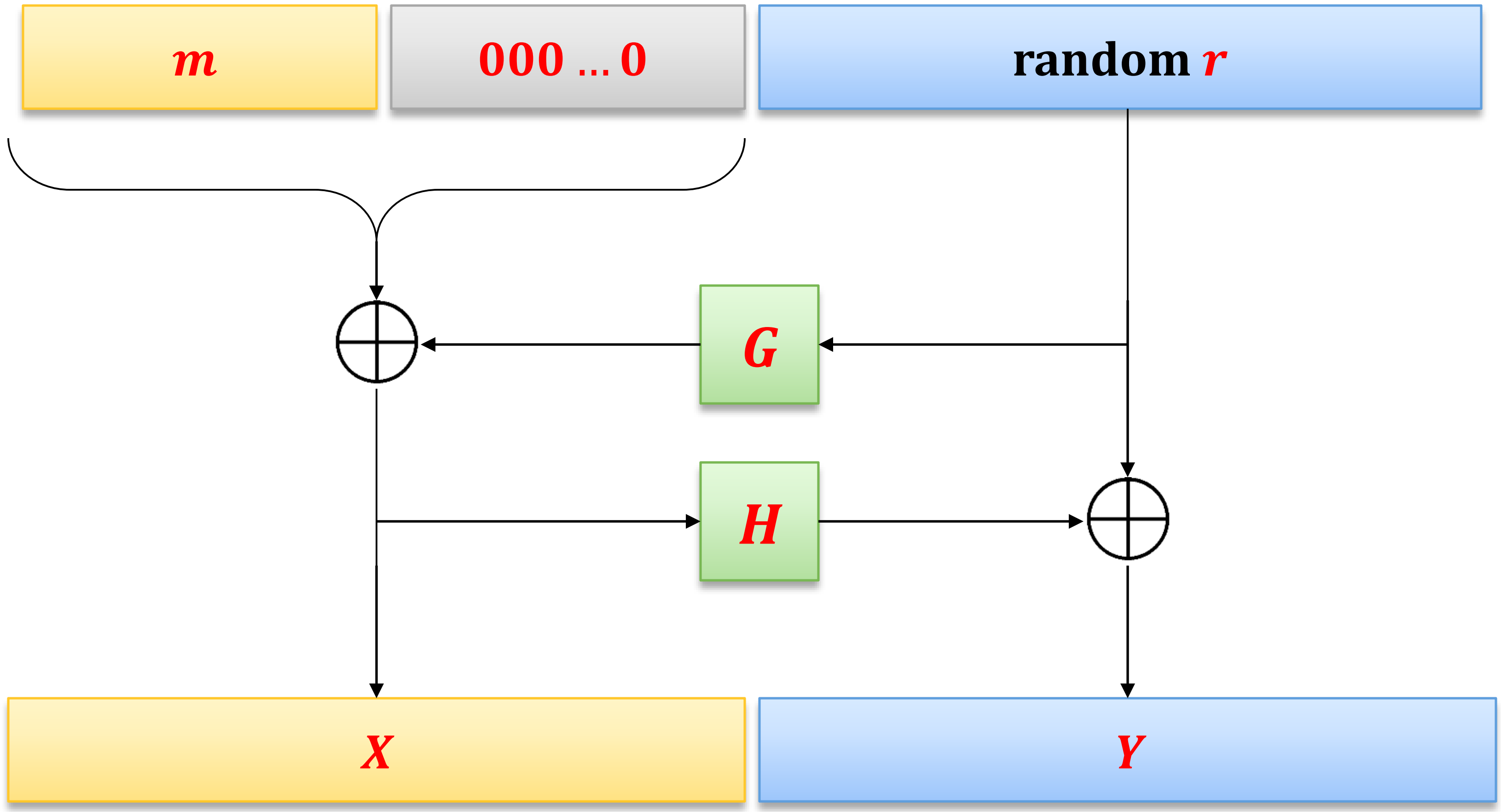
Why?

Look at the picture:



The encoding OAEP(m) is uniformly random

Again look at the picture:



Why are these two properties useful for **CPA-security**?

The adversary obtains

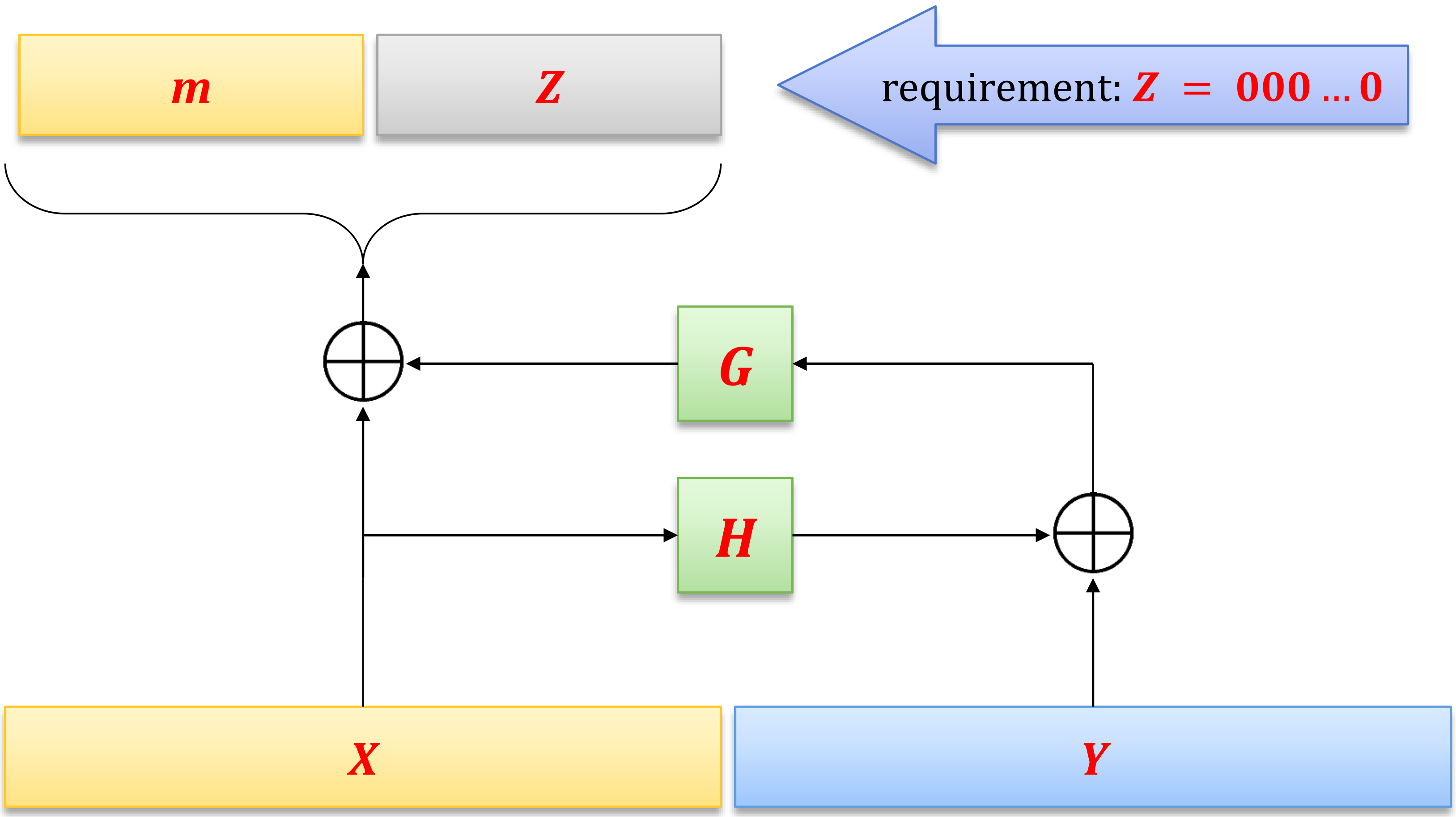
$$\mathbf{Enc}_{pk}(m_b) = x^e \bmod N$$

where $x = \mathrm{OAEP}(m_b)$.

In order to get **any information about** m_b needs to compute the **entire value of** x, where x is uniformly random.

Hardness of this problem is equivalent to the **RSA assumption.**

It is hard to produce a valid (X, Y) "without knowing m first"



This last property is useful for CCA-security

Why?

Informally:

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

The only way to produce a valid ciphertext is to do the following:

- choose $\boldsymbol{m}$
- compute $\boldsymbol{c} := (\mathbf{OAEP}(\boldsymbol{m}))^{\boldsymbol{e}} \mathbf{mod}\ \boldsymbol{N}$.

Note

In “handbook RSA” this is not the case since every $\boldsymbol{c} \in \boldsymbol{Z}_{\boldsymbol{N}}^{*}$ is a valid ciphertext.

Also in the **PKCS #1: RSA Encryption Standard Version 1.5** standard the probability of producing a valid ciphertext is noticeable.

An interesting attack on OAEP

J. Manger: **A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0.** CRYPTO 2001

Based on the following fact:

the decryption algorithm outputs $\perp$ in two cases:

1. "$x > 2^{\ell+k_1+k_2}$",
2. or $Z \neq 000\ldots0$.

The attack exploits the fact that in the **PKCS #1 v2.0** standard the **error messages in these two cases were different**.

Moral: implementation details matter!