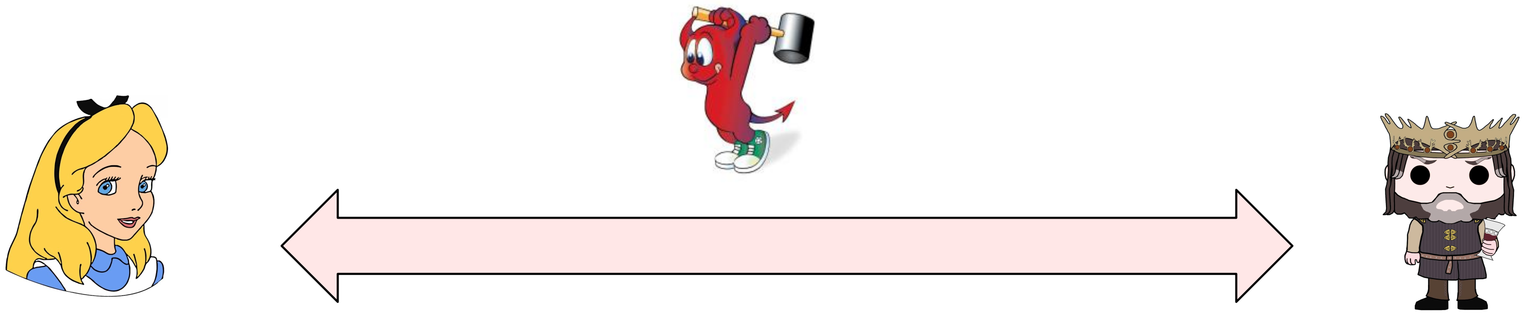
Reminder: Passive vs Active Attacks

We are now considering **active** attacks:

- The attacker has full control over the channel
- Can alter the message contents
- Can drop messages
- Can forge new messages



Reminder: Secrecy vs Integrity

There are two important guarantees that we would like to achieve against an active adversary 

Secrecy:

- This is what we have been concerned with so far.
- The adversary should not be able to (easily) learn (any information about) the plaintexts

Integrity (& Authentication):

- The adversary is not able to tamper with the messages
- The message originated from the intended party
- The message has not been modified in transit

Integrity and Secrecy are **orthogonal** concerns

Reminder: Secrecy vs Integrity

There are two important guarantees that we would like to achieve against an active adversary

Secrecy:

Secrecy against active adversaries?

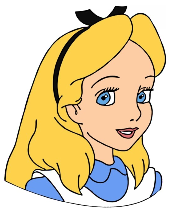
Integrity (& Authentication):

- The adversary is not able to tamper with the messages
- The message originated from the intended party
- The message has not been modified in transit

Integrity and Secrecy are **orthogonal** concerns

Reminder: Malleability of OTP

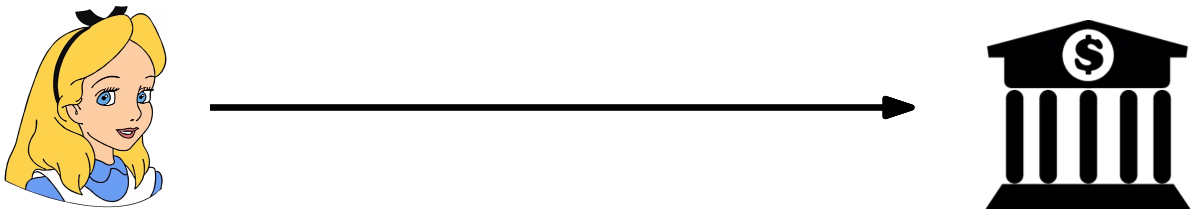
- Alice buys an item from the adversary for 5.20€
- Alice makes a wire transfer from her bank's website



Reminder: Malleability of OTP

- Alice buys an item from the adversary for 5.20€
- Alice makes a wire transfer from her bank's website
- The bank website sends a message of the form
  PAY <RECIPIENT_IBAN> <AMOUNT> to the bank's backend

$$m = \underbrace{\texttt{010100000100000101011001}}_{\texttt{PAY}}\underbrace{\texttt{01001001}\ldots\texttt{00110010}}_{\texttt{IBAN}}\underbrace{\texttt{0000000100000100}}_{\texttt{AMOUNT (520)}}$$

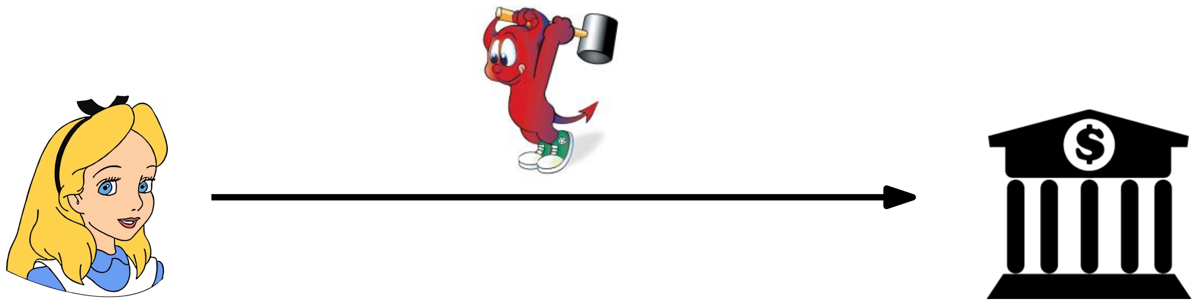


Reminder: Malleability of OTP

- Alice buys an item from the adversary for 5.20€
- Alice makes a wire transfer from her bank's website
- The bank website sends a message of the form
  PAY <RECIPIENT_IBAN> <AMOUNT> to the bank's backend

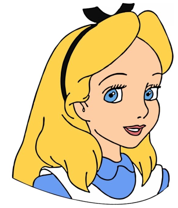
$$m = \underbrace{\texttt{010100000100000101011001}}_{\texttt{PAY}}\underbrace{\texttt{01001001}\ldots\texttt{00110010}}_{\texttt{IBAN}}\underbrace{\texttt{0000000100000100}}_{\texttt{AMOUNT (520)}}$$

- The message is encrypted with a one-time pad

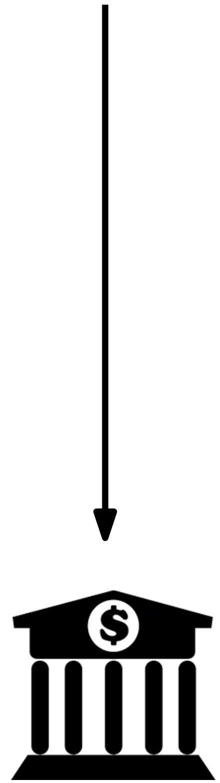


Reminder: Malleability of OTP

$$m = \underbrace{010100000100000101011001}_{\texttt{PAY}}\,\underbrace{01001001\ldots00110010}_{\texttt{IBAN}}\,\underbrace{0000000100000100}_{\texttt{AMOUNT (520)}}$$

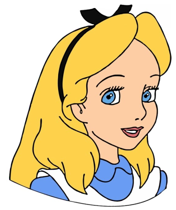


$$c = \underbrace{001101100010101000110101}_{\texttt{PAY}}\,\underbrace{11010001\ldots10001101}_{\texttt{IBAN}}\,\underbrace{1011111010010010}_{\texttt{AMOUNT}}$$



Reminder: Malleability of OTP

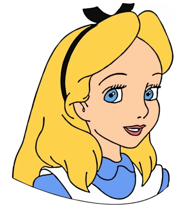
$$m = \underbrace{\texttt{010100000100000101011001}}_{\texttt{PAY}}\,\underbrace{\texttt{01001001}\ldots\texttt{00110010}}_{\texttt{IBAN}}\,\underbrace{\texttt{0000000100000100}}_{\texttt{AMOUNT (520)}}$$



$$c = \underbrace{\texttt{001101100010101000110101}}_{\texttt{PAY}}\,\underbrace{\texttt{11010001}\ldots\texttt{10001101}}_{\texttt{IBAN}}\,\underbrace{\texttt{1011111010010010}}_{\texttt{AMOUNT}}$$

$\oplus$ $\ldots\texttt{1000000000000000}$

Reminder: Malleability of OTP



$$m = \underbrace{\texttt{010100000100000101011001}}_{\texttt{PAY}}\,\underbrace{\texttt{01001001}\ldots\texttt{00110010}}_{\texttt{IBAN}}\,\underbrace{\texttt{0000000100000100}}_{\texttt{AMOUNT (520)}}$$

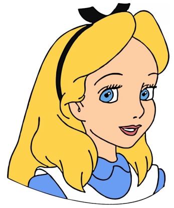
$$c = \underbrace{\texttt{001101100010101000110101}}_{\texttt{PAY}}\,\underbrace{\texttt{11010001}\ldots\texttt{10001101}}_{\texttt{IBAN}}\,\underbrace{\texttt{1011111010010010}}_{\texttt{AMOUNT}}$$

$$\oplus \qquad \ldots\texttt{1000000000000000}$$

$$c' = \underbrace{\texttt{001101100010101000110101}}_{\texttt{PAY}}\,\underbrace{\texttt{11010001}\ldots\texttt{10001101}}_{\texttt{IBAN}}\,\underbrace{\texttt{0011111010010010}}_{\texttt{AMOUNT}}$$

Reminder: Malleability of OTP

$$m = \underbrace{010100000100000101011001}_{\texttt{PAY}}\ \underbrace{01001001\ldots00110010}_{\texttt{IBAN}}\ \underbrace{0000000100000100}_{\texttt{AMOUNT (520)}}$$



$$c = \underbrace{001101100010101000110101}_{\texttt{PAY}}\ \underbrace{11010001\ldots10001101}_{\texttt{IBAN}}\ \underbrace{1011111010010010}_{\texttt{AMOUNT}}$$

$$\oplus \qquad \ldots 1000000000000000$$

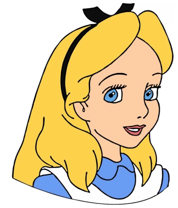
$$c' = \underbrace{001101100010101000110101}_{\texttt{PAY}}\ \underbrace{11010001\ldots10001101}_{\texttt{IBAN}}\ \underbrace{0011111010010010}_{\texttt{AMOUNT}}$$

$$m' = \underbrace{010100000100000101011001}_{\texttt{PAY}}\ \underbrace{01001001\ldots00110010}_{\texttt{IBAN}}\ \underbrace{1000000100000100}_{\texttt{AMOUNT (33028)}}$$

Reminder: Malleability of OTP

$m =$ 010100000100000101011001 01001001 . . . 00110010 0000000100000100

PAY — IBAN — AMOUNT (520)



$c =$ 0011011000101010000110101 11010001 . . . 10001101 1011111010010010

PAY — IBAN

$\oplus$ . . . 00000

c' . . . 10001101 0011111010010010

IBAN — AMOUNT

010100000100000101011001 01001001 . . . 00110010 1000000100000100

PAY — IBAN — AMOUNT (33028)

Changes to the ciphertext result in predictable changes to the plaintext

The scheme is **malleable**!

Malleability and CCA security

- All the encryption schemes we have seen so far are malleable!
- Malleable encryption schemes are problematic in the presence of an active adversary

Malleability and CCA security

- All the encryption schemes we have seen so far are malleable!
- Malleable encryption schemes are problematic in the presence of an active adversary

How do we formalize the **threat model** when the attacker has control over the channel?

Malleability and CCA security

- All the encryption schemes we have seen so far are malleable!
- Malleable encryption schemes are problematic in the presence of an active adversary

How do we formalize the **threat model** when the attacker has control over the channel?

What **security guarantee** do we want to achieve against such an adversary?

Malleability and CCA security

- All the encryption schemes we have seen so far are malleable!
- Malleable encryption schemes are problematic in the presence of an active adversary

How do we formalize the **threat model** when the attacker has control over the channel?

Chosen Ciphertext Attacks (CCA)

What **security guarantee** do we want to achieve against such an adversary?

Chosen-ciphertext attack

The adversary can learn the ciphertexts corresponding to one or more plaintexts of its choice

and

the plaintexts corresponding to one or more ciphertexts of its choice

The adversary wants to deduce information about the underlying plaintext of some other ciphertext produced using the same key

Chosen-ciphertext attack

The adversary can learn the ciphertexts corresponding to one or more plaintexts of its choice

and

the plaintexts corresponding to one or more ciphertexts of its choice

The adversary wants to deduce information about the underlying plaintext of some other ciphertext produced using the same key

How can the adversary learn the (some information about the) plaintexts of the desired ciphertext?

Chosen-ciphertext attack

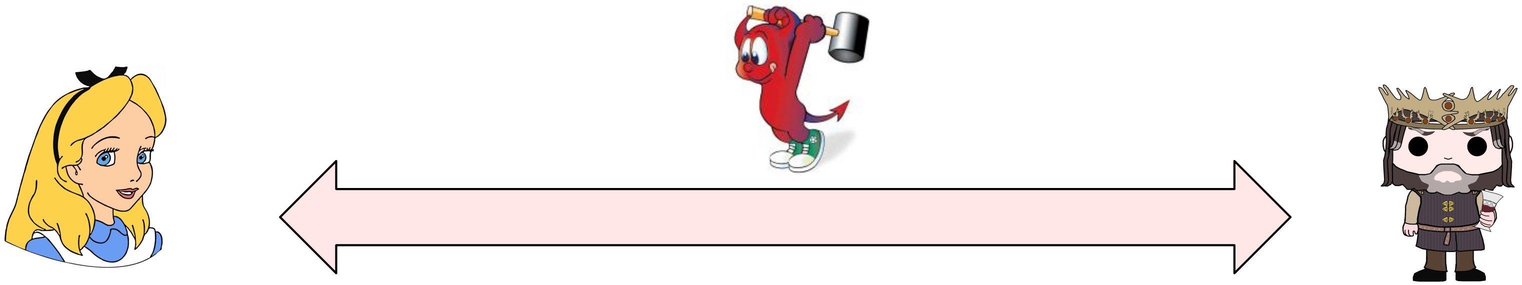
The adversary can learn the ciphertexts corresponding to one or more plaintexts of its choice

and

the plaintexts corresponding to one or more ciphertexts of its choice

The adversary wants to deduce information about the underlying plaintext of some other ciphertext produced using the same key

How can the adversary learn the (some information about the) plaintexts of the desired ciphertext?



Chosen-ciphertext attack

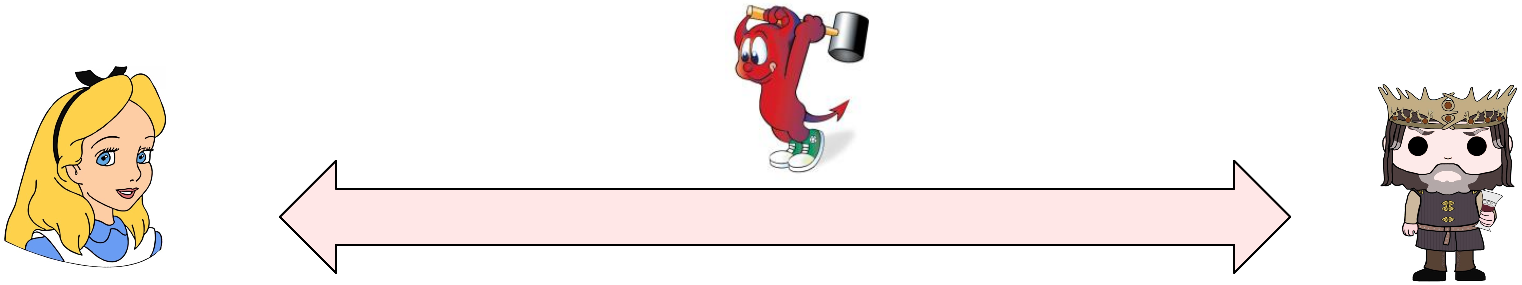
The adversary can learn the ciphertexts corresponding to one or more plaintexts of its choice

and

the plaintexts corresponding to one or more ciphertexts of its choice

The adversary wants to deduce information about the underlying plaintext of some other ciphertext produced using the same key

How can the adversary learn the (some information about the) plaintexts of the desired ciphertext?



The adversary modifies/injects traffic and observes Bob response

Chosen-ciphertext attack

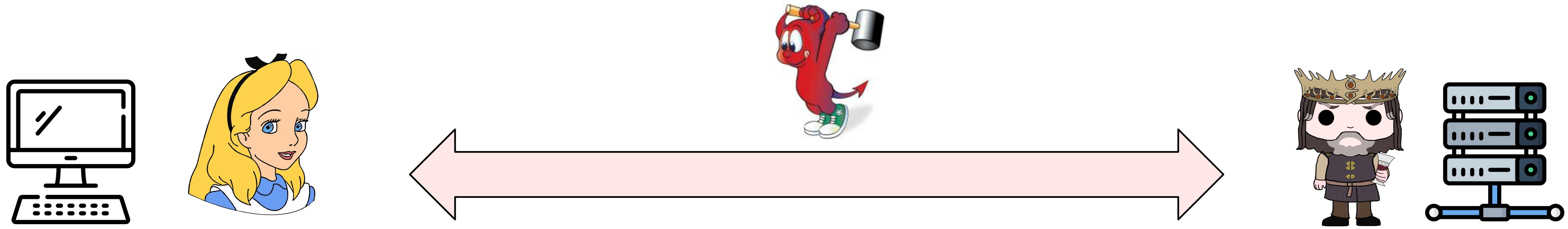
The adversary can learn the ciphertexts corresponding to one or more plaintexts of its choice

and

the plaintexts corresponding to one or more ciphertexts of its choice

The adversary wants to deduce information about the underlying plaintext of some other ciphertext produced using the same key

How can the adversary learn the (some information about the) plaintexts of the desired ciphertext?



The adversary modifies/injects traffic and observes Bob response

Many protocols close a connection or request a retransmission when a bad message is received

Malleability and CCA security

- All the encryption schemes we have seen so far are malleable!
- Malleable encryption schemes are problematic in the presence of an active adversary

How do we formalize the **threat model** when the attacker has control over the channel?

Chosen Ciphertext Attacks (CCA)

What **security guarantee** do we want to achieve against such an adversary?

Malleability and CCA security

- All the encryption schemes we have seen so far are malleable!
- Malleable encryption schemes are problematic in the presence of an active adversary

How do we formalize the **threat model** when the attacker has control over the channel?

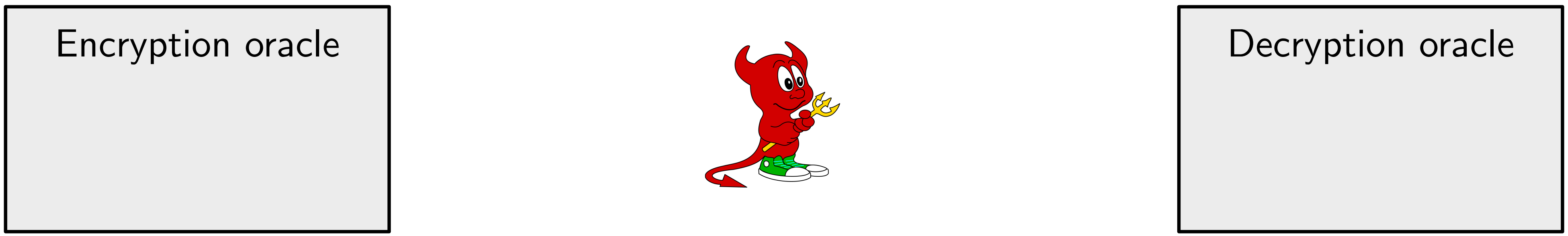
Chosen Ciphertext Attacks (CCA)

What **security guarantee** do we want to achieve against such an adversary?

We define a suitable experiment to capture the security guarantee

Modeling CCA security

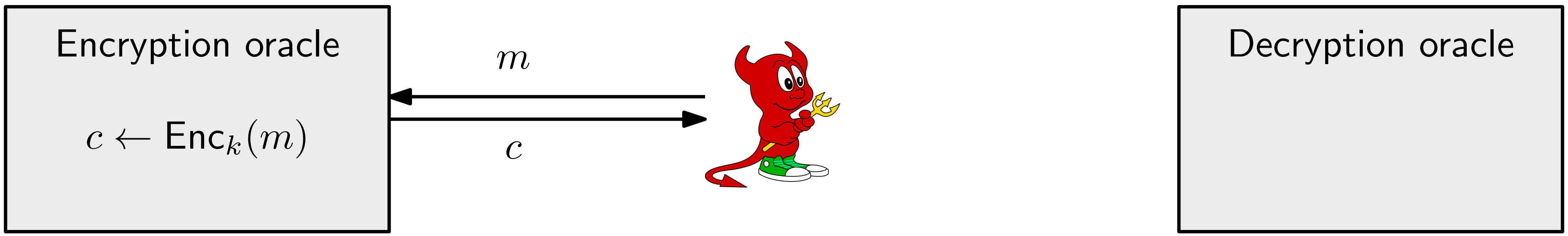
A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated and the adversary $\mathcal{A}$ has access to both an **encryption oracle** and a **decryption oracle**



Modeling CCA security

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated and the adversary $\mathcal{A}$ has access to both an **encryption oracle** and a **decryption oracle**

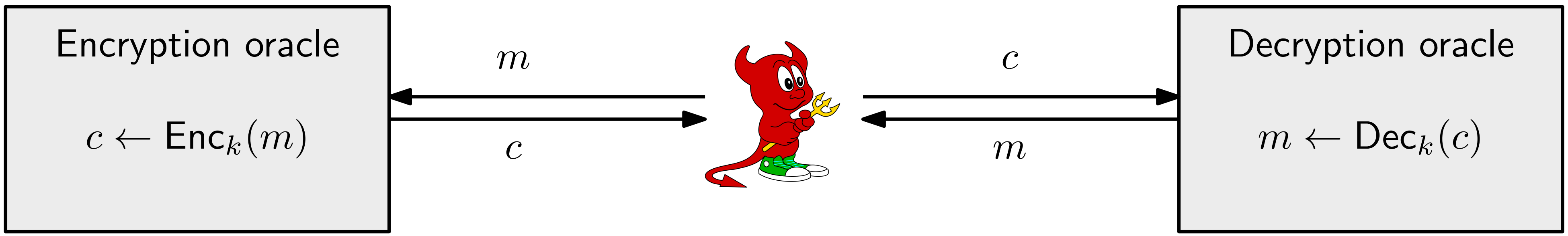
- The encryption oracle acts as a black-box that can be **queried** with a message m and returns an encryption of m



Modeling CCA security

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated and the adversary $\mathcal{A}$ has access to both an **encryption oracle** and a **decryption oracle**

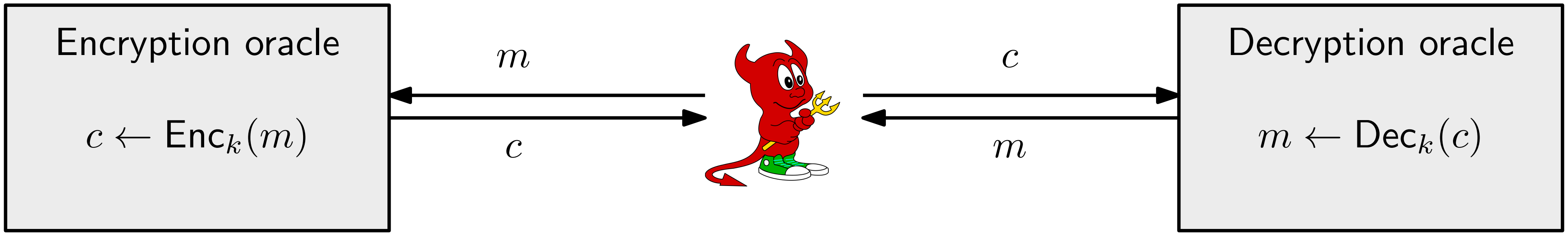
- The encryption oracle acts as a black-box that can be **queried** with a message m and returns an encryption of m
- The decryption oracle can be **queried** with a ciphertext and returns the corresponding plaintext



Modeling CCA security

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated and the adversary $\mathcal{A}$ has access to both an **encryption oracle** and a **decryption oracle**

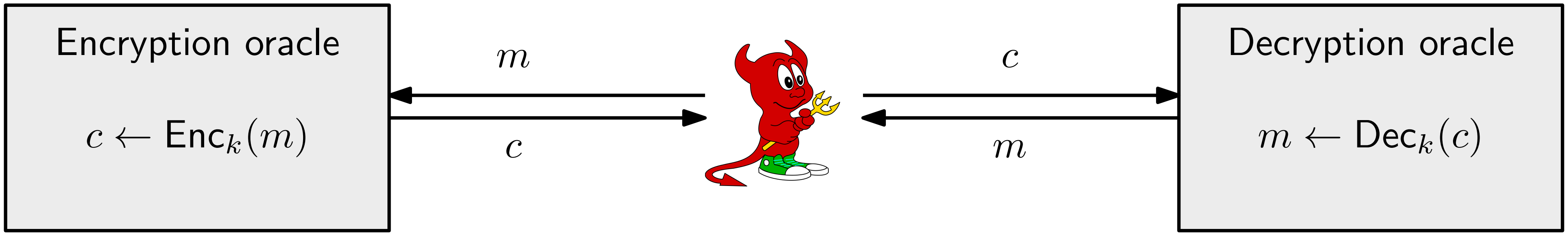
- The encryption oracle acts as a black-box that can be **queried** with a message m and returns an encryption of m
- The decryption oracle can be **queried** with a ciphertext and returns the corresponding plaintext
- There is no limit on the number of queries the adversary can make



Modeling CCA security

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated and the adversary $\mathcal{A}$ has access to both an **encryption oracle** and a **decryption oracle**

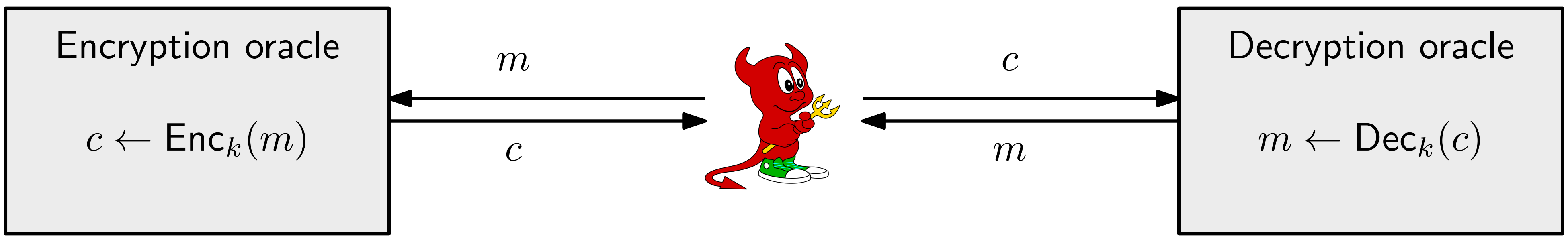
- The encryption oracle acts as a black-box that can be **queried** with a message m and returns an encryption of m
- The decryption oracle can be **queried** with a ciphertext and returns the corresponding plaintext
- There is no limit on the number of queries the adversary can make
- All messages are encrypted/decrupted using the same key k



Modeling CCA security

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated and the adversary $\mathcal{A}$ has access to both an **encryption oracle** and a **decryption oracle**

- The encryption oracle acts as a black-box that can be **queried** with a message m and returns an encryption of m
- The decryption oracle can be **queried** with a ciphertext and returns the corresponding plaintext
- There is no limit on the number of queries the adversary can make
- All messages are encrypted/decrupted using the same key k
- The key k is **unknown** to the adversary



The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

Formally, if $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private key encryption scheme with message space $\mathcal{M}$, we denote the following experiment by $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$

- A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated
- $\mathcal{A}$ can interact with an encryption oracle that provides access to $\mathsf{Enc}_k(\cdot)$ **and** with a decryption oracle that provides access to $\mathsf{Dec}_k(\cdot)$
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$ with $|m_0| = |m_1|$
- A uniform random bit $b \in \{0, 1\}$ is generated
- The *challenge ciphertext* c is computed by $\mathsf{Enc}_k(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ can interact with an encryption oracle that provides access to $\mathsf{Enc}_k(\cdot)$ **and** with a decryption oracle that provides access to $\mathsf{Dec}_k(\cdot)$
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *output of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

Formally, if $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private key encryption scheme with message space $\mathcal{M}$, we denote the following experiment by $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$

- A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated
- $\mathcal{A}$ can interact with an encryption oracle that provides access to $\mathsf{Enc}_k(\cdot)$ **and** with a decryption oracle that provides access to $\mathsf{Dec}_k(\cdot)$
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$ with $|m_0| = |m_1|$
- A uniform random bit $b \in \{0, 1\}$ is generated
- The *challenge ciphertext* c is computed by $\mathsf{Enc}_k(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ can interact with an encryption oracle that provides access to $\mathsf{Enc}_k(\cdot)$ **and** with a decryption oracle that provides access to $\mathsf{Dec}_k(\cdot)$
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *output of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

k and b are unknown to $\mathcal{A}$

The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

Formally, if $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a private key encryption scheme with message space $\mathcal{M}$,

we denote the following experiment by $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$

- A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated
- $\mathcal{A}$ can interact with an encryption oracle that provides access to $\mathsf{Enc}_k(\cdot)$ **and** with a decryption oracle that provides access to $\mathsf{Dec}_k(\cdot)$
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$ with $|m_0| = |m_1|$
- A uniform random bit $b \in \{0, 1\}$ is generated
- The *challenge ciphertext* c is computed by $\mathsf{Enc}_k(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ can interact with an encryption oracle that provides access to $\mathsf{Enc}_k(\cdot)$ **and** with a decryption oracle that provides access to $\mathsf{Dec}_k(\cdot)$
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *output of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

k and b are unknown to $\mathcal{A}$

$\mathcal{A}$ is not allowed to request a decryption of c

The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated

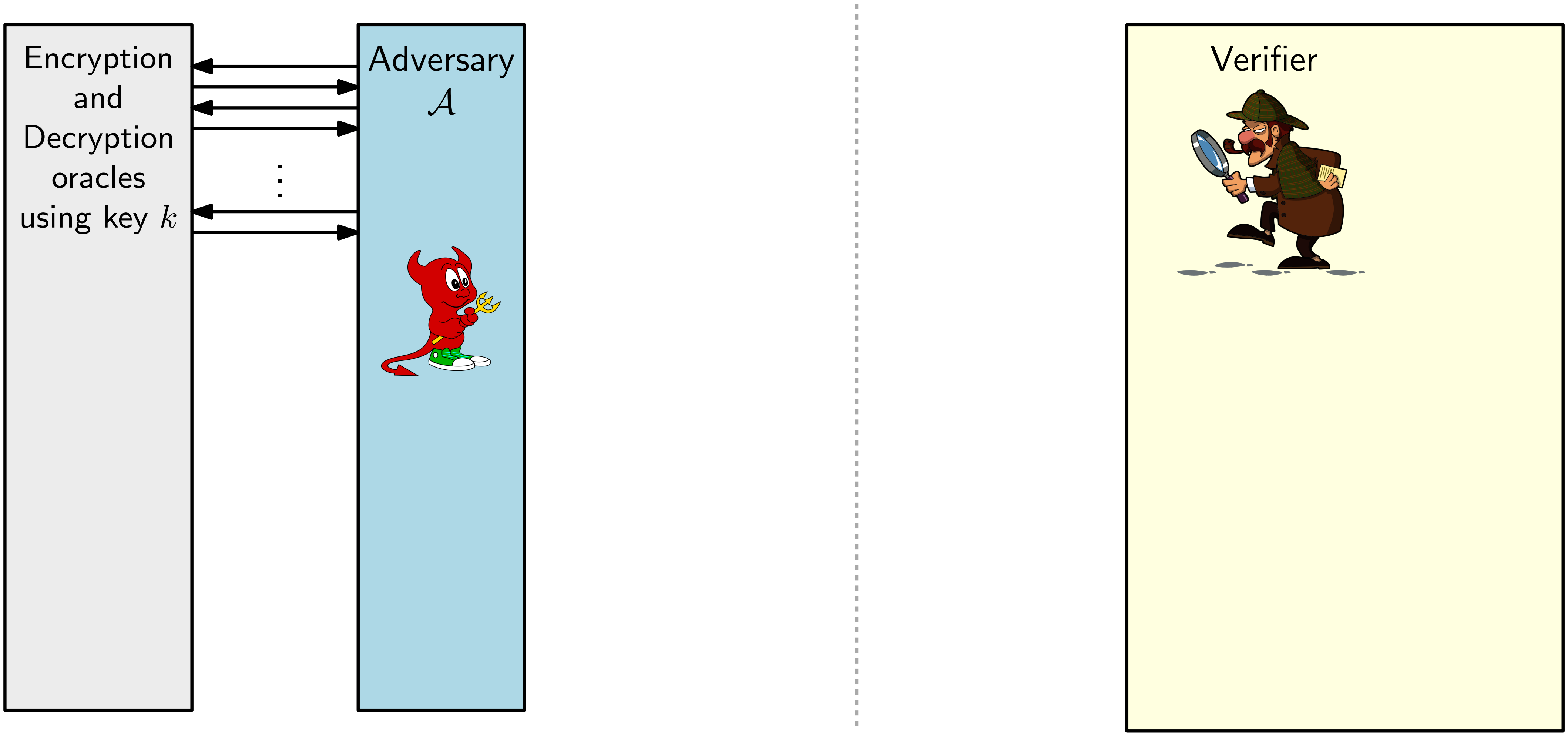
Encryption and Decryption oracles using key k

Adversary $\mathcal{A}$

Verifier

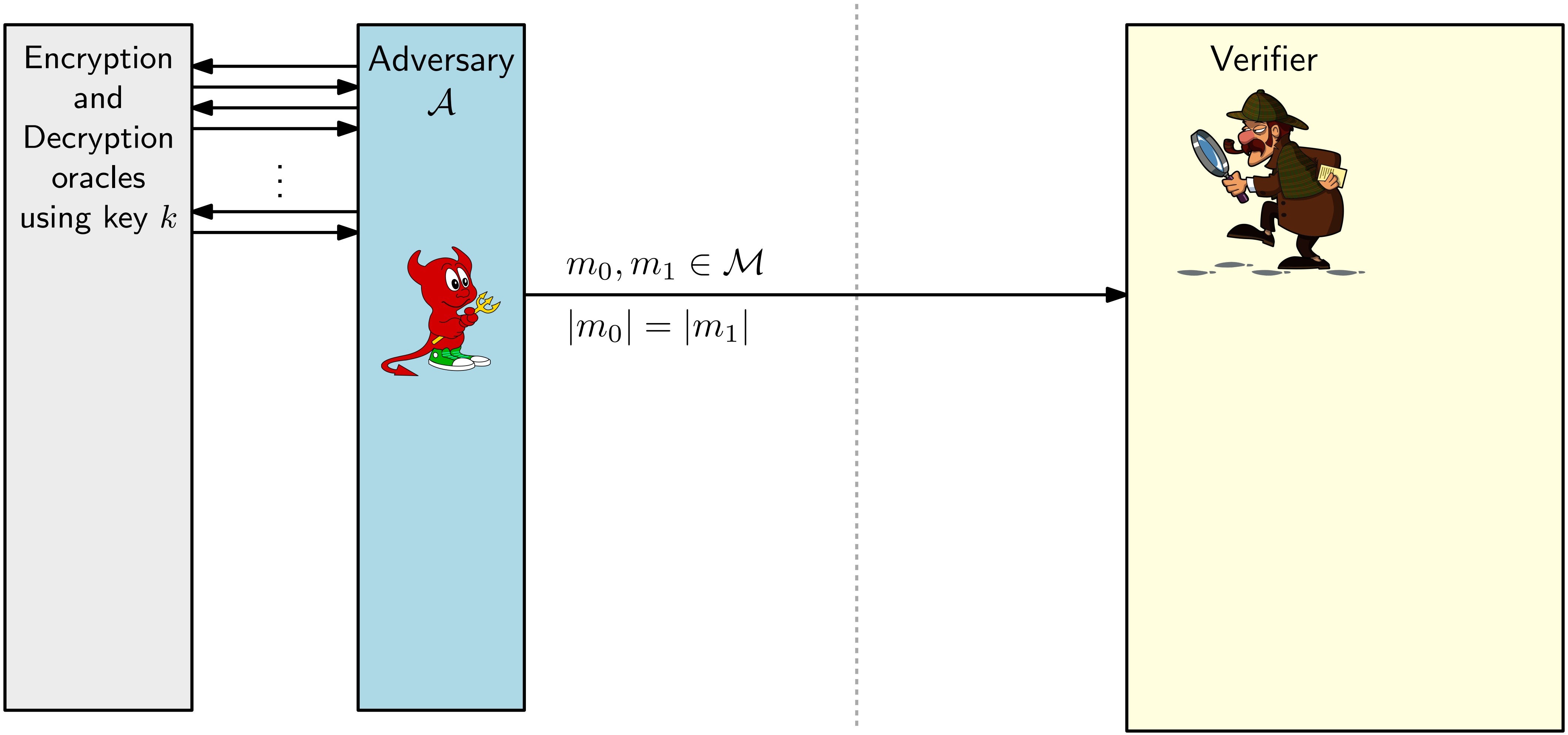
The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated



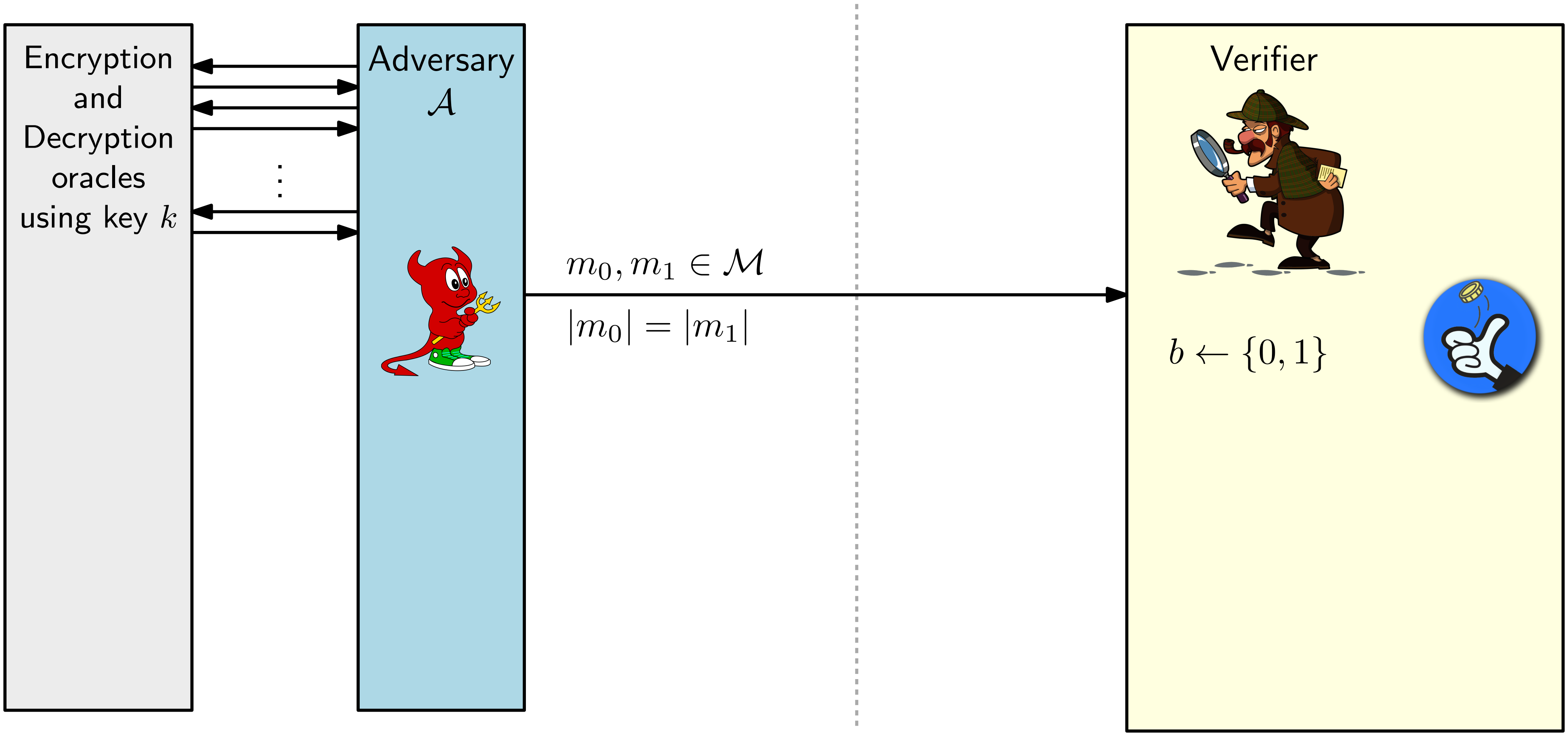
The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated



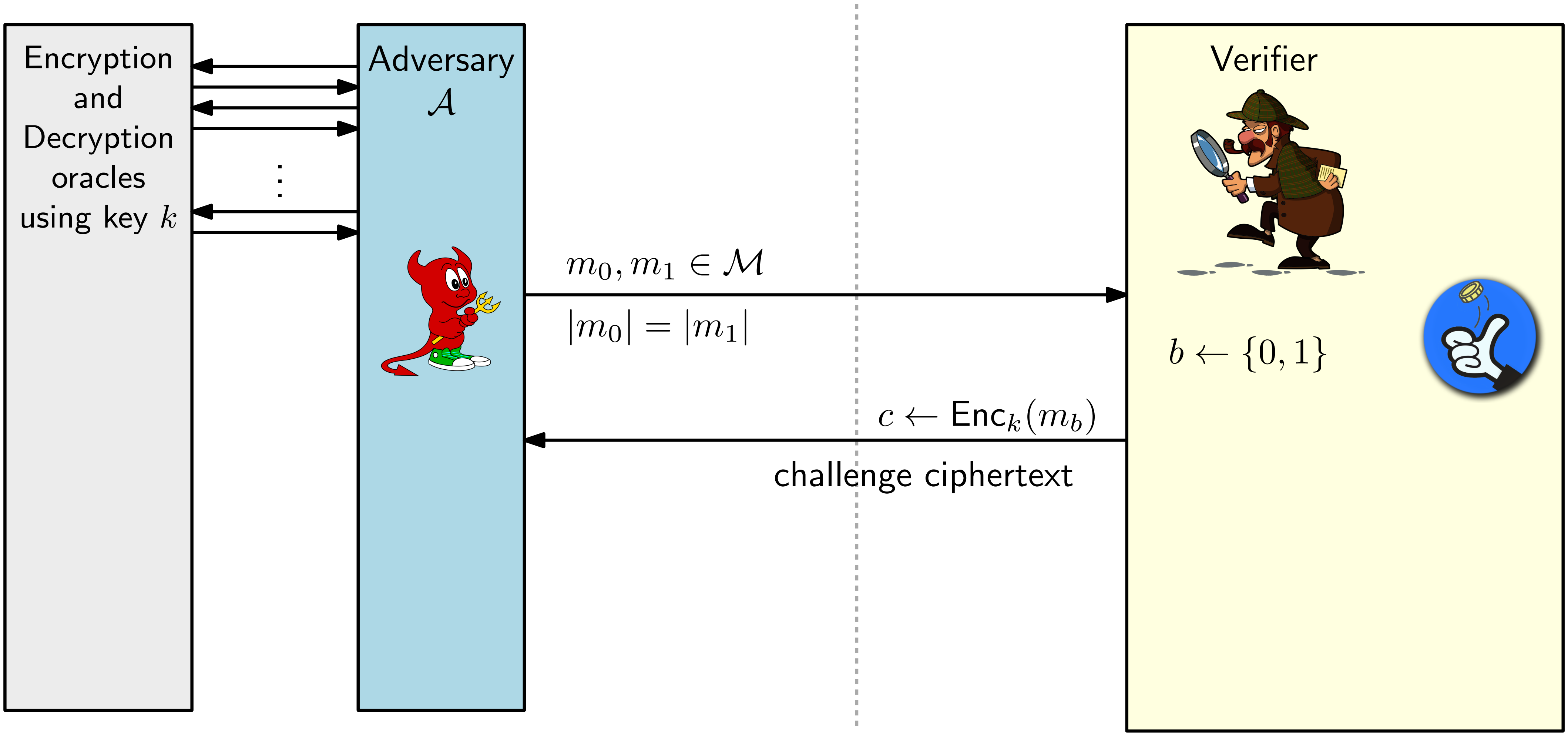
The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated



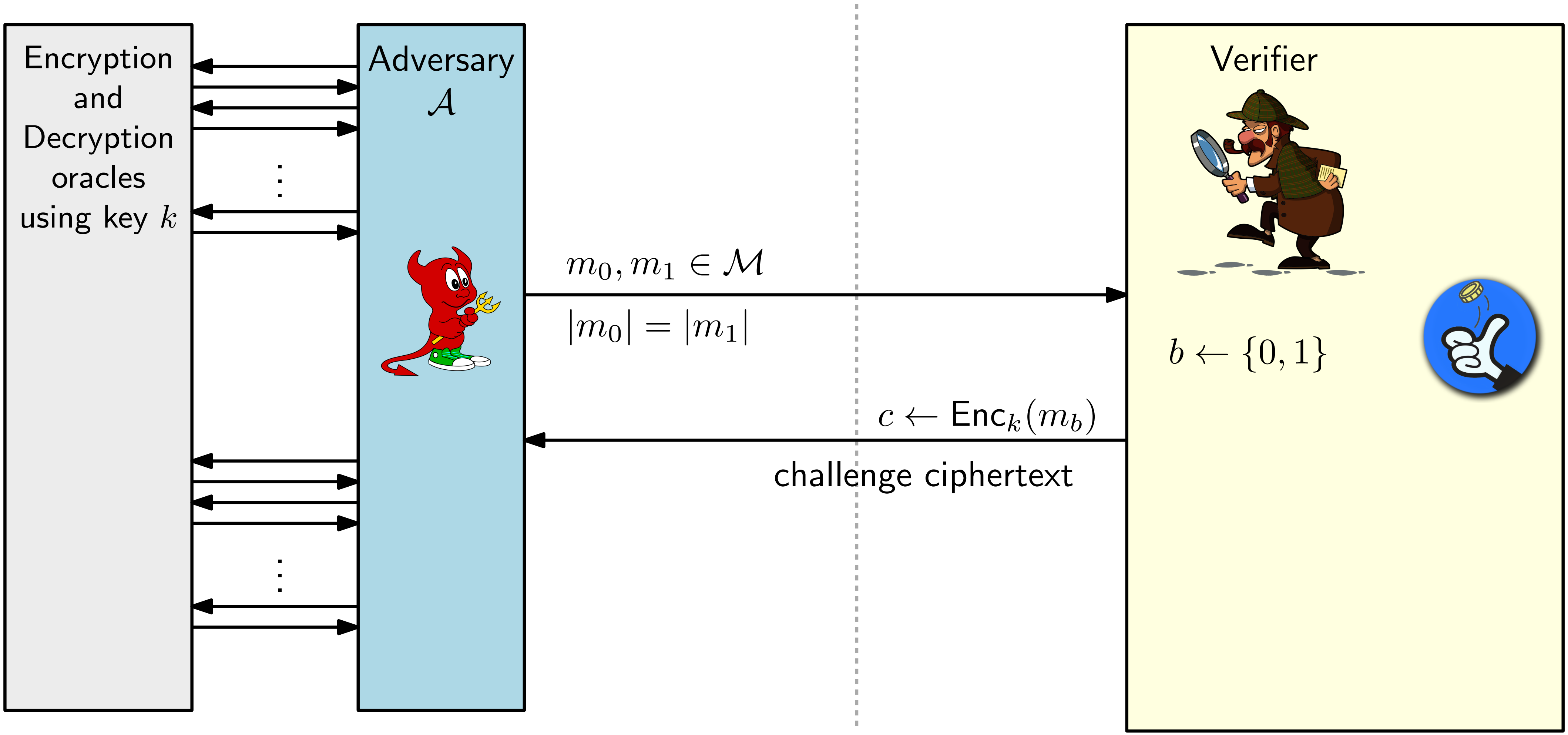
The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated



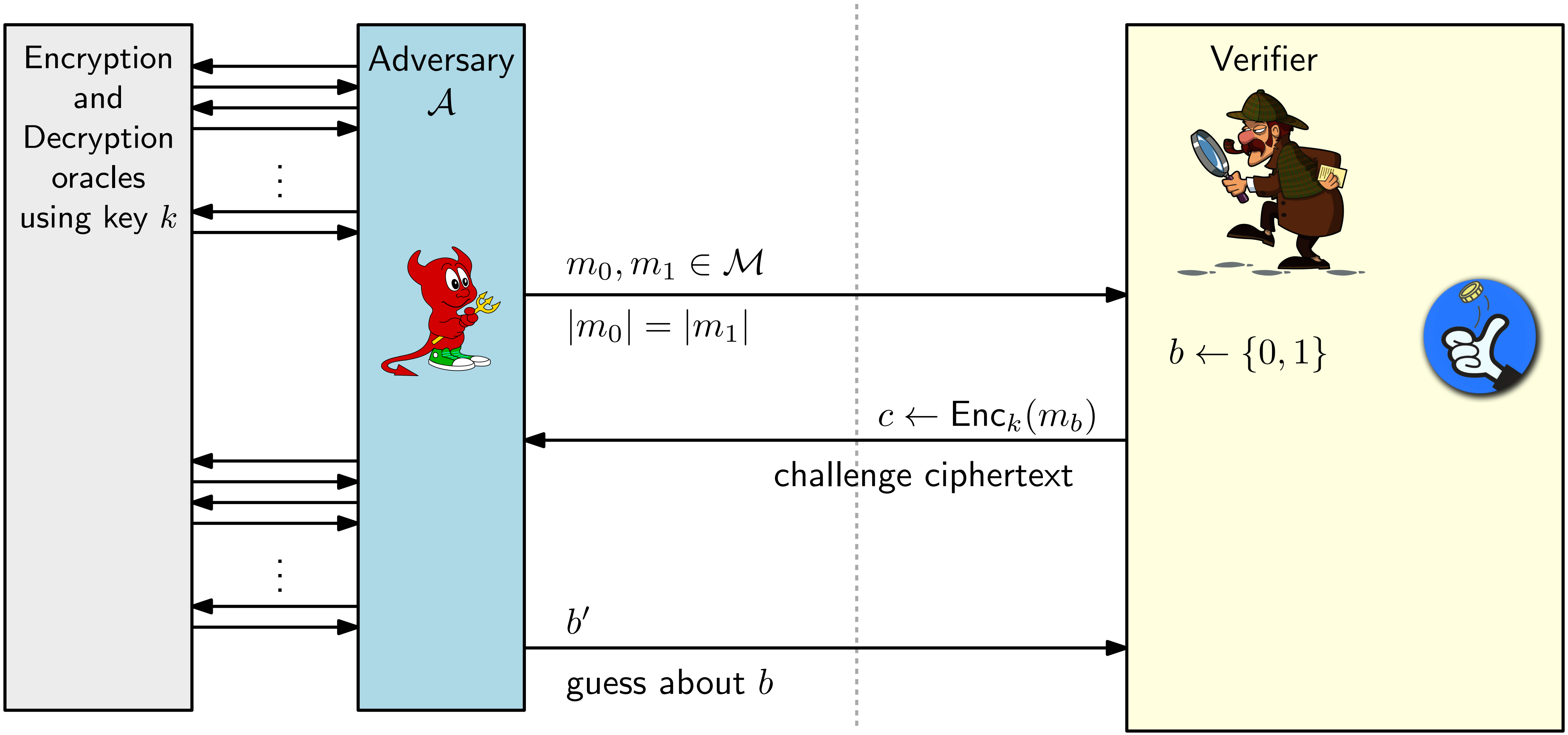
The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated



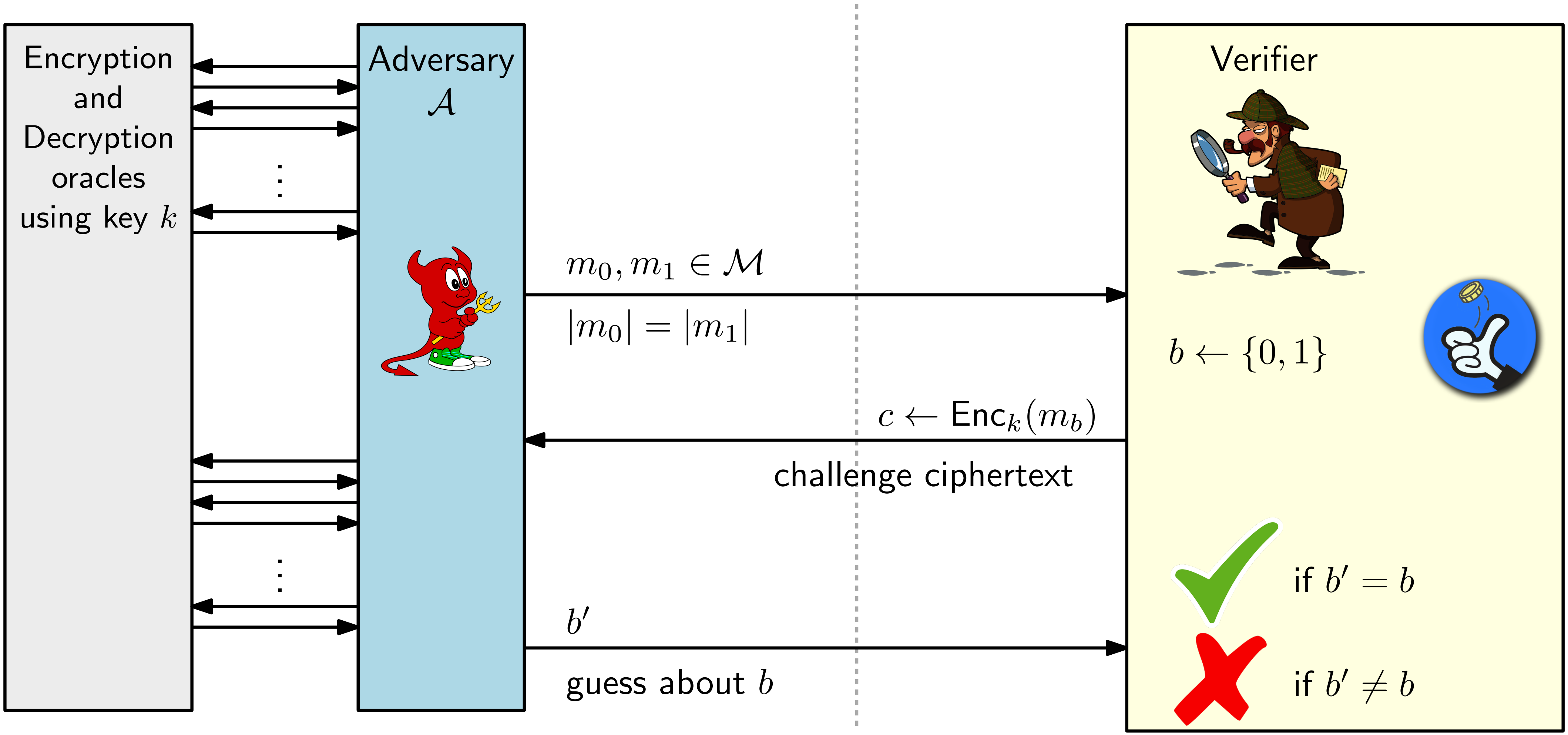
The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated



The $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}$ experiment

A key $k \leftarrow \mathsf{Gen}(1^n)$ is generated



Definition of CCA security

Definition: *A private key encryption scheme* Π *has indistinguishable encryptions under a chosen-ciphertext attack (is* **CCA-secure***) if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ε *such that:*

$$\Pr[\mathit{PrivK}^{cca}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Is the definition realistic?

The definition models the case in which an attacker can obtain the decryption of **any ciphertext** of its choice (except for the challenge ciphertext)

In most real-world scenarios, the attacker would not have access to a full decryption oracle

Is the definition realistic?

The definition models the case in which an attacker can obtain the decryption of **any ciphertext** of its choice (except for the challenge ciphertext)

In most real-world scenarios, the attacker would not have access to a full decryption oracle

Still, some **partial information** about the deciphered ciphertext might be leaked

Is the definition realistic?

The definition models the case in which an attacker can obtain the decryption of **any ciphertext** of its choice (except for the challenge ciphertext)

In most real-world scenarios, the attacker would not have access to a full decryption oracle

Still, some **partial information** about the deciphered ciphertext might be leaked

- For example, the attacker can observe the subsequent actions of the recipient
- Recall the Midway island example: the US could have sent an encrypted message containing the fragment `AF` and observed Japan's response

Is the definition realistic?

The definition models the case in which an attacker can obtain the decryption of **any ciphertext** of its choice (except for the challenge ciphertext)

In most real-world scenarios, the attacker would not have access to a full decryption oracle

Still, some **partial information** about the deciphered ciphertext might be leaked

- For example, the attacker can observe the subsequent actions of the recipient
- Recall the Midway island example: the US could have sent an encrypted message containing the fragment `AF` and observed Japan's response

We take a **worst-case** approach:

- No assumption on how strong real-world adversaries are
- If an encryption scheme withstands a stronger adversary than real-world ones, security is not compromised

Malleability and CCA-security

If an encryption scheme Π is malleable, then it cannot be CCA-secure

Malleability and CCA-security

If an encryption scheme Π is malleable, then it cannot be CCA-secure

Sketch of the proof:

- Output two messages m_0, m_1 and get the challenge ciphertext c

Malleability and CCA-security

If an encryption scheme Π is malleable, then it cannot be CCA-secure

Sketch of the proof:

- Output two messages m_0, m_1 and get the challenge ciphertext c
- Since Π is malleable, there is a way to change c into a new ciphertext c' that results a predictable change to the plaintext

Malleability and CCA-security

If an encryption scheme Π is malleable, then it cannot be CCA-secure

Sketch of the proof:

- Output two messages m_0, m_1 and get the challenge ciphertext c
- Since Π is malleable, there is a way to change c into a new ciphertext c' that results a predictable change to the plaintext
- Submit c' to the decryption oracle and get the corresponding plaintext m'

Malleability and CCA-security

If an encryption scheme Π is malleable, then it cannot be CCA-secure

Sketch of the proof:

- Output two messages m_0, m_1 and get the challenge ciphertext c
- Since Π is malleable, there is a way to change c into a new ciphertext c' that results a predictable change to the plaintext
- Submit c' to the decryption oracle and get the corresponding plaintext m'
- The plaintext m' matches the modified version of some message m_i (i.e., the version of m_i in which the expected change took place)

Malleability and CCA-security

If an encryption scheme Π is malleable, then it cannot be CCA-secure

Sketch of the proof:

- Output two messages m_0, m_1 and get the challenge ciphertext c
- Since Π is malleable, there is a way to change c into a new ciphertext c' that results a predictable change to the plaintext
- Submit c' to the decryption oracle and get the corresponding plaintext m'
- The plaintext m' matches the modified version of some message m_i (i.e., the version of m_i in which the expected change took place)
- Output the guess $b' = i$

Malleability and CCA-security

If an encryption scheme Π is malleable, then it cannot be CCA-secure

Sketch of the proof:

- Output two messages m_0, m_1 and get the challenge ciphertext c
- Since Π is malleable, there is a way to change c into a new ciphertext c' that results a predictable change to the plaintext
- Submit c' to the decryption oracle and get the corresponding plaintext m'
- The plaintext m' matches the modified version of some message m_i (i.e., the version of m_i in which the expected change took place)
- Output the guess $b' = i$

This adversary wins $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage!

Malleability and CCA-security

If an encryption scheme Π is malleable, then it cannot be CCA-secure

Sketch of the proof:

- Output two messages m_0, m_1 and get the challenge ciphertext c
- Since Π is malleable, there is a way to change c into a new ciphertext c' that results a predictable change to the plaintext
- Submit c' to the decryption oracle and get the corresponding plaintext m'
- The plaintext m' matches the modified version of some message m_i (i.e., the version of m_i in which the expected change took place)
- Output the guess $b' = i$

This adversary wins $\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage!

By taking the contrapositive: CAA-security **implies** non-malleability!

Padding Oracle Attacks

We show a scenario in which:

- The encryption scheme is malleable
- Only a single bit of information is leaked

Padding Oracle Attacks

We show a scenario in which:

- The encryption scheme is malleable
- Only a single bit of information is leaked
- We can exploit this to gain knowledge of **the entire plaintext**

Padding Oracle Attacks

We show a scenario in which:

- The encryption scheme is malleable
- Only a single bit of information is leaked
- We can exploit this to gain knowledge of **the entire plaintext**
- This scenario happens in the real world...

  ...In fact, this attack has been used against SSL!

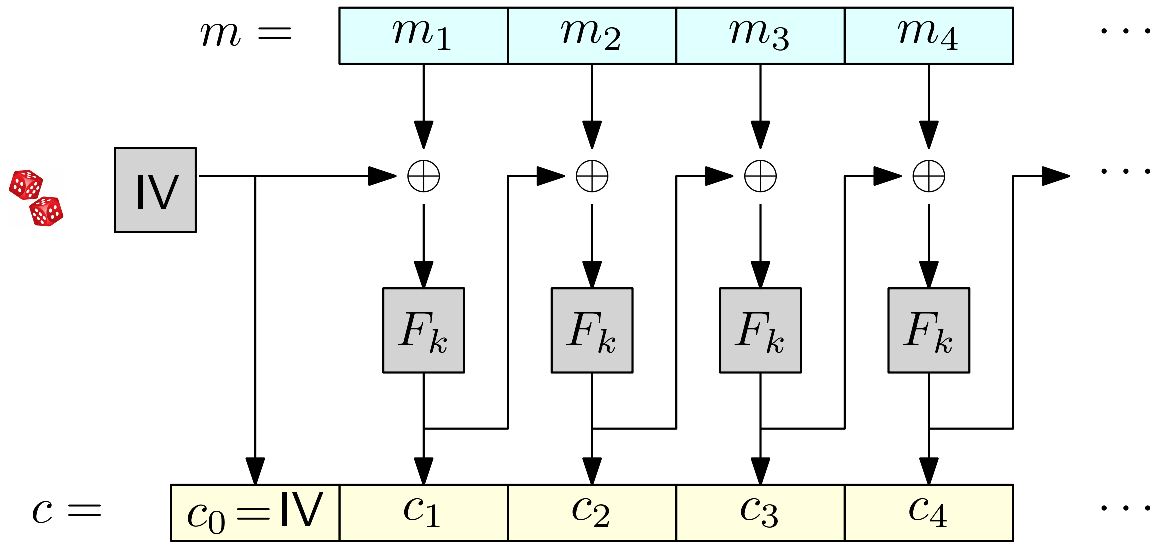
Padding Oracle Attacks

We show a scenario in which:

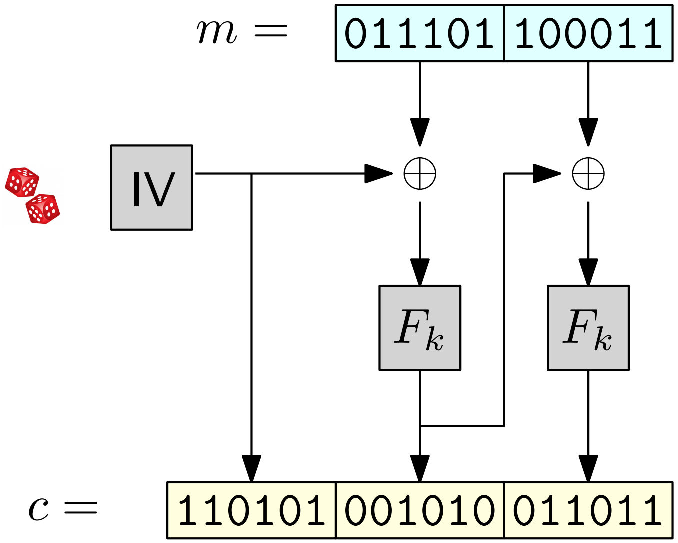
- The encryption scheme is malleable
- Only a single bit of information is leaked
- We can exploit this to gain knowledge of **the entire plaintext**
- This scenario happens in the real world...

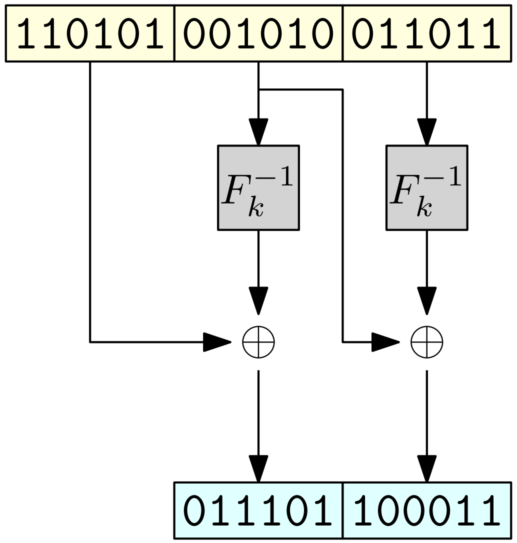
  ...In fact, this attack has been used against SSL!

Reminder: block ciphers in CBC mode

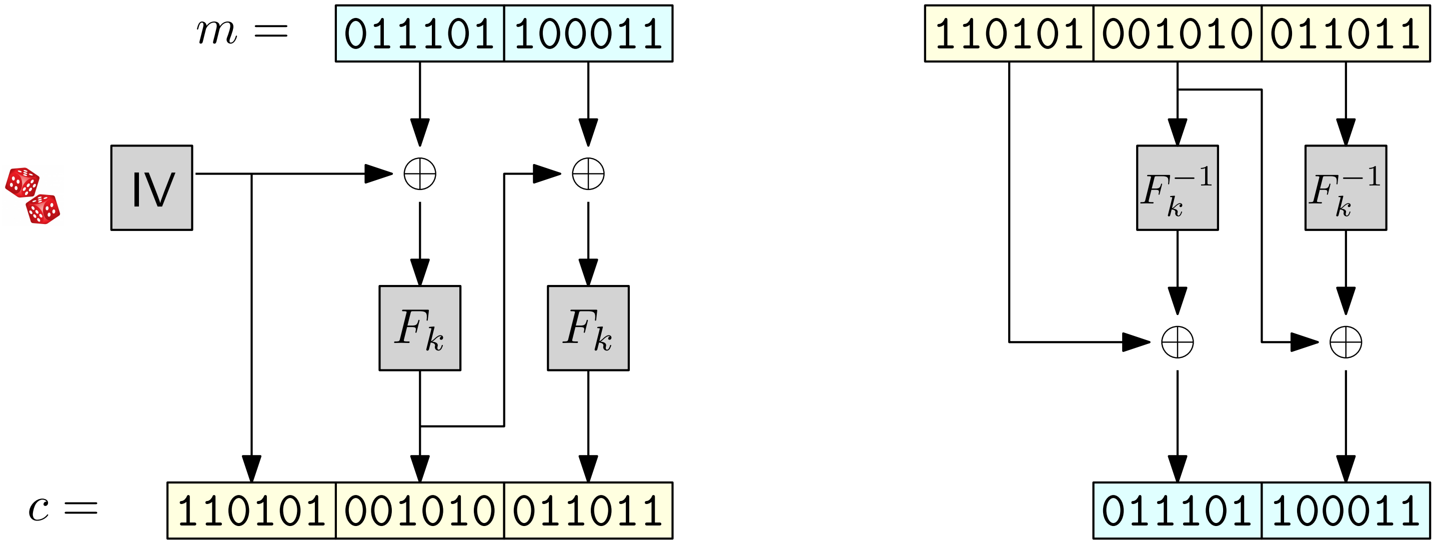


Malleability of CBC mode



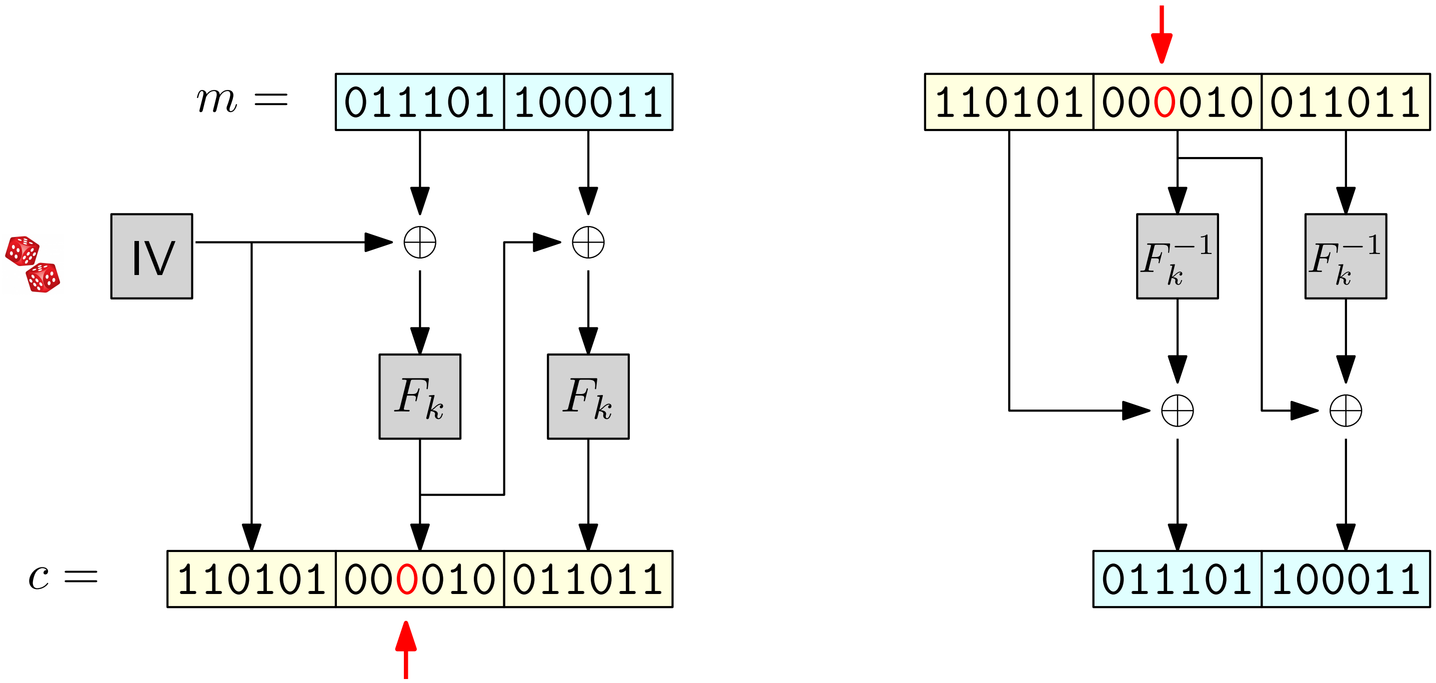


Malleability of CBC mode



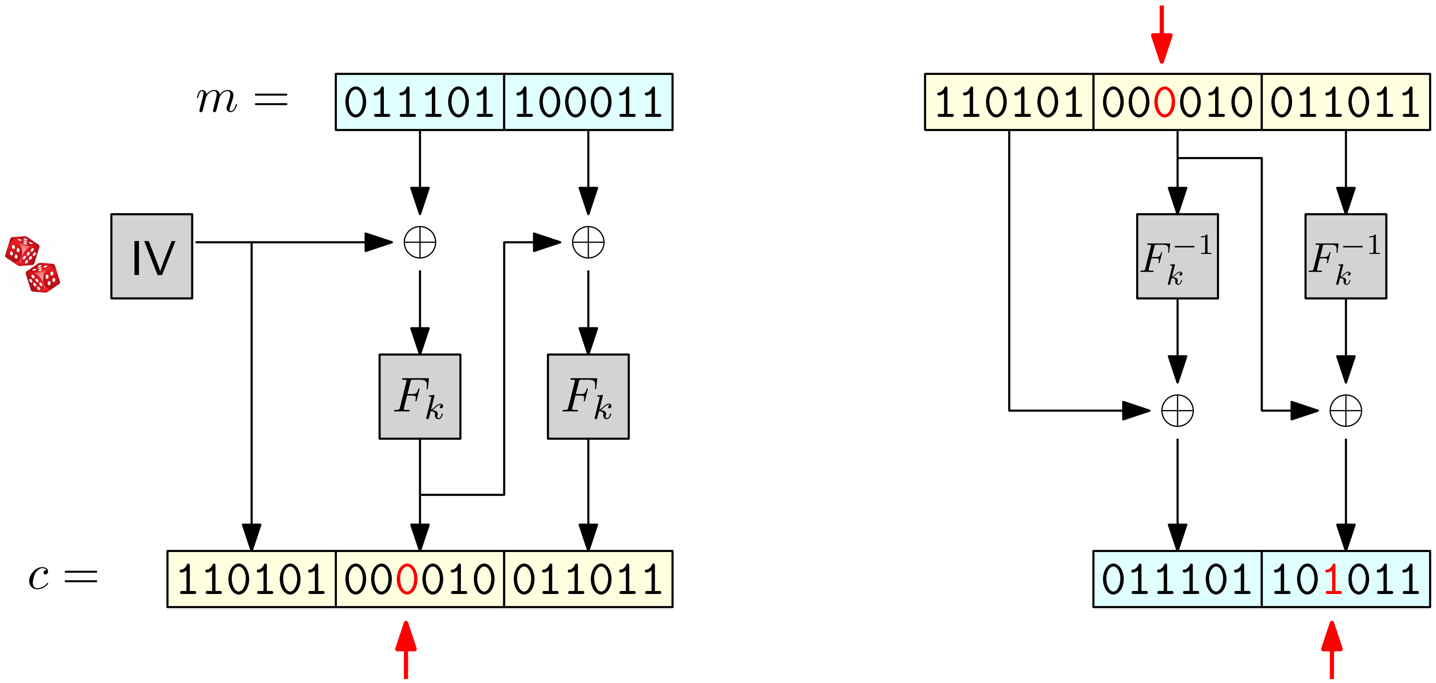
If we flip the generic j-th bit in the i-th block c_i of the ciphertext. . .

Malleability of CBC mode



If we flip the generic j-th bit in the i-th block c_i of the ciphertext...

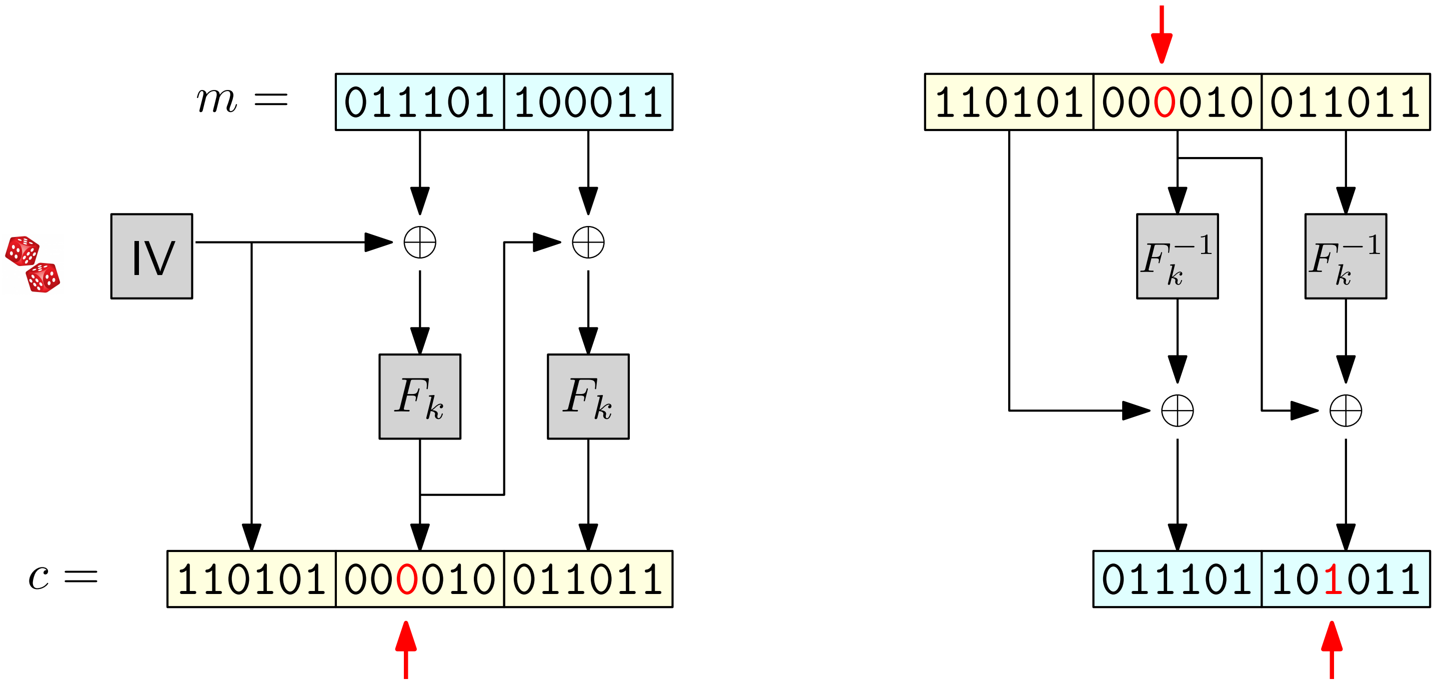
Malleability of CBC mode



If we flip the generic j-th bit in the i-th block c_i of the ciphertext...

...this causes a flip in the j-th bit of the $(i+1)$-th block m_{i+1} of the plaintext after decryption

Malleability of CBC mode



If we flip the generic j-th bit in the i-th block c_i of the ciphertext...

...this causes a flip in the j-th bit of the $(i+1)$-th block m_{i+1} of the plaintext after decryption

In general, if we XOR the i-th block of the ciphertext with Δ, this causes the $(i+1)$-th block of the plaintext to be XOR-ed with Δ after decryption

Message padding and PKCS #7

- In general, the length $|m|$ of the plaintext m might not be a multiple of the block length ℓ
- The message needs to be **padded** to a multiple of ℓ before encryption

Message padding and PKCS #7

- In general, the length $|m|$ of the plaintext m might not be a multiple of the block length ℓ
- The message needs to be **padded** to a multiple of ℓ before encryption
- One popular padding scheme is defined by the PKCS #7 standard

Message padding and PKCS #7

- In general, the length $|m|$ of the plaintext m might not be a multiple of the block length ℓ
- The message needs to be **padded** to a multiple of ℓ before encryption
- One popular padding scheme is defined by the PKCS #7 standard
- We will work in bytes instead of bits. Assume $\ell < 256$ (bytes)
- If b bytes are missing for $|m|$ to reach to next multiple of ℓ, append b bytes, each with value b to m

Message padding and PKCS #7

- In general, the length $|m|$ of the plaintext m might not be a multiple of the block length ℓ
- The message needs to be **padded** to a multiple of ℓ before encryption
- One popular padding scheme is defined by the PKCS #7 standard
- We will work in bytes instead of bits. Assume $\ell < 256$ (bytes)
- If b bytes are missing for $|m|$ to reach to next multiple of ℓ, append b bytes, each with value b to m

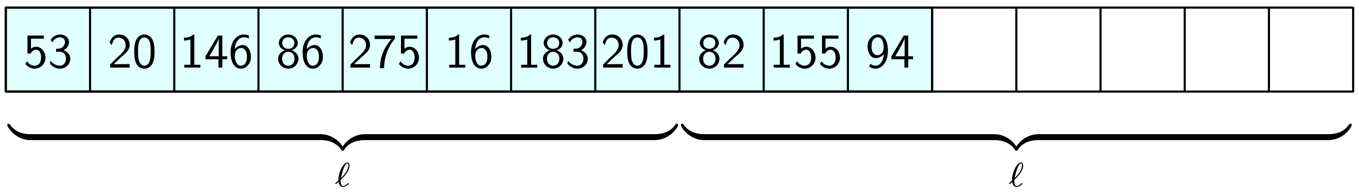
  Note that if $|m|$ is already a multiple of ℓ then $b = \ell$ and a new block is appended
  (this ensures unambiguous decoding)

Message padding and PKCS #7

- In general, the length $|m|$ of the plaintext m might not be a multiple of the block length ℓ
- The message needs to be **padded** to a multiple of ℓ before encryption
- One popular padding scheme is defined by the PKCS #7 standard
- We will work in bytes instead of bits. Assume $\ell < 256$ (bytes)
- If b bytes are missing for $|m|$ to reach to next multiple of ℓ, append b bytes, each with value b to m

  Note that if $|m|$ is already a multiple of ℓ then $b = \ell$ and a new block is appended
  (this ensures unambiguous decoding)

Example with $\ell = 8$:



Message padding and PKCS #7

- In general, the length $|m|$ of the plaintext m might not be a multiple of the block length ℓ
- The message needs to be **padded** to a multiple of ℓ before encryption
- One popular padding scheme is defined by the PKCS #7 standard
- We will work in bytes instead of bits. Assume $\ell < 256$ (bytes)
- If b bytes are missing for $|m|$ to reach to next multiple of ℓ, append b bytes, each with value b to m

  Note that if $|m|$ is already a multiple of ℓ then $b = \ell$ and a new block is appended
  (this ensures unambiguous decoding)

Example with $\ell = 8$:

53	20	146	86	275	16	183	201	82	155	94	5	5	5	5	5

(first 8 bytes: ℓ; last 8 bytes: ℓ)

Message padding and PKCS #7

- In general, the length $|m|$ of the plaintext m might not be a multiple of the block length ℓ
- The message needs to be **padded** to a multiple of ℓ before encryption
- One popular padding scheme is defined by the PKCS #7 standard
- We will work in bytes instead of bits. Assume $\ell < 256$ (bytes)
- If b bytes are missing for $|m|$ to reach to next multiple of ℓ, append b bytes, each with value b to m

  Note that if $|m|$ is already a multiple of ℓ then $b = \ell$ and a new block is appended
  (this ensures unambiguous decoding)

Example with $\ell = 8$:

53	20	146	86	275	16	183	201	82	155	94	31	199	77	42	1
ℓ								ℓ							

Message padding and PKCS #7

- In general, the length $|m|$ of the plaintext m might not be a multiple of the block length ℓ
- The message needs to be **padded** to a multiple of ℓ before encryption
- One popular padding scheme is defined by the PKCS #7 standard
- We will work in bytes instead of bits. Assume $\ell < 256$ (bytes)
- If b bytes are missing for $|m|$ to reach to next multiple of ℓ, append b bytes, each with value b to m

  Note that if $|m|$ is already a multiple of ℓ then $b = \ell$ and a new block is appended
  (this ensures unambiguous decoding)

Example with $\ell = 8$:

53	20	146	86	275	16	183	201	82	155	94	31	199	77	42	1	8	8	8	8	8	8	8	8
ℓ								ℓ								ℓ							

Decryption of a PKCS #7 padded message

When a ciphertext c is received:

- Use the block cipher in CBC mode to decrypt c and obtain the padded plaintext m'

Decryption of a PKCS #7 padded message

When a ciphertext c is received:

- Use the block cipher in CBC mode to decrypt c and obtain the padded plaintext m'
- Look at the last byte b of m'
- If $b = 0$ or $b > \ell$, return "bad padding error"

Decryption of a PKCS #7 padded message

When a ciphertext c is received:

- Use the block cipher in CBC mode to decrypt c and obtain the padded plaintext m'
- Look at the last byte b of m'
- If $b = 0$ or $b > \ell$, return "bad padding error"
- If the last b bytes of m' are not all equal to b, return "bad padding error"
- Strip the last b bytes from m', to obtain the original plaintext m

Decryption of a PKCS #7 padded message

When a ciphertext c is received:

- Use the block cipher in CBC mode to decrypt c and obtain the padded plaintext m'
- Look at the last byte b of m'
- If $b = 0$ or $b > \ell$, return "bad padding error"
- If the last b bytes of m' are not all equal to b, return "bad padding error"
- Strip the last b bytes from m', to obtain the original plaintext m
- Process m (application dependent)

Decryption of a PKCS #7 padded message

When a ciphertext c is received:

- Use the block cipher in CBC mode to decrypt c and obtain the padded plaintext m'
- Look at the last byte b of m'
- If $b = 0$ or $b > \ell$, return "bad padding error"
- If the last b bytes of m' are not all equal to b, return "bad padding error"
- Strip the last b bytes from m', to obtain the original plaintext m
- Process m (application dependent)

Example with $\ell = 8$:

53	20	146	86	275	16	183	201	82	6	6	6	6	6	6	6

$\underbrace{\qquad\qquad\qquad\qquad}_{\ell}\ \underbrace{\qquad\qquad\qquad\qquad}_{\ell}$

Decryption of a PKCS #7 padded message

When a ciphertext c is received:

- Use the block cipher in CBC mode to decrypt c and obtain the padded plaintext m'
- Look at the last byte b of m'
- If $b = 0$ or $b > \ell$, return "bad padding error"
- If the last b bytes of m' are not all equal to b, return "bad padding error"
- Strip the last b bytes from m', to obtain the original plaintext m
- Process m (application dependent)

Example with $\ell = 8$:

53	20	146	86	275	16	183	201	82	6

Padding Oracles

- Often the "bad padding error" results in different behavior of the protocol than a correctly padded but invalid plaintext

Padding Oracles

- Often the “bad padding error” results in different behavior of the protocol than a correctly padded but invalid plaintext

  E.g., the “bad padding” error is sent to the client

Padding Oracles

- Often the “bad padding error” results in different behavior of the protocol than a correctly padded but invalid plaintext

  E.g., the “bad padding” error is sent to the client

- This was the case with SSL

Padding Oracles

- Often the "bad padding error" results in different behavior of the protocol than a correctly padded but invalid plaintext

  E.g., the "bad padding" error is sent to the client

- This was the case with SSL

- Even when there is no explicit difference in the protocol behavior, bad padding errors can sometimes still be detected

  E.g., due to differences in timing

Padding Oracles

- Often the "bad padding error" results in different behavior of the protocol than a correctly padded but invalid plaintext

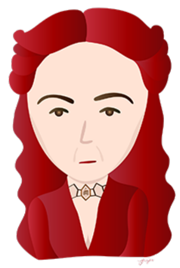
  E.g., the "bad padding" error is sent to the client

- This was the case with SSL

- Even when there is no explicit difference in the protocol behavior, bad padding errors can sometimes still be detected

  E.g., due to differences in timing

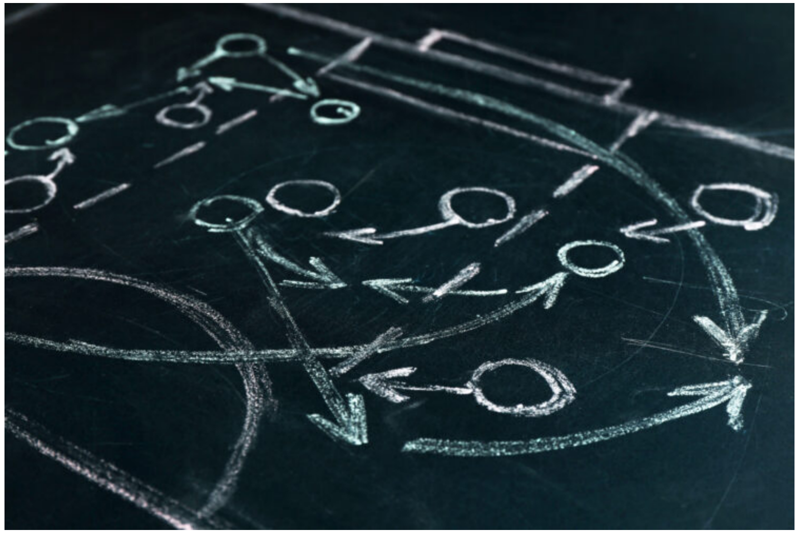
We model the ability of the adversary to tell whether the padding of (the plaintext corresponding to a) ciphertext is valid, with a **padding oracle**



Exploiting a Padding Oracle

Attack plan:

1) Figure out how long is the padding



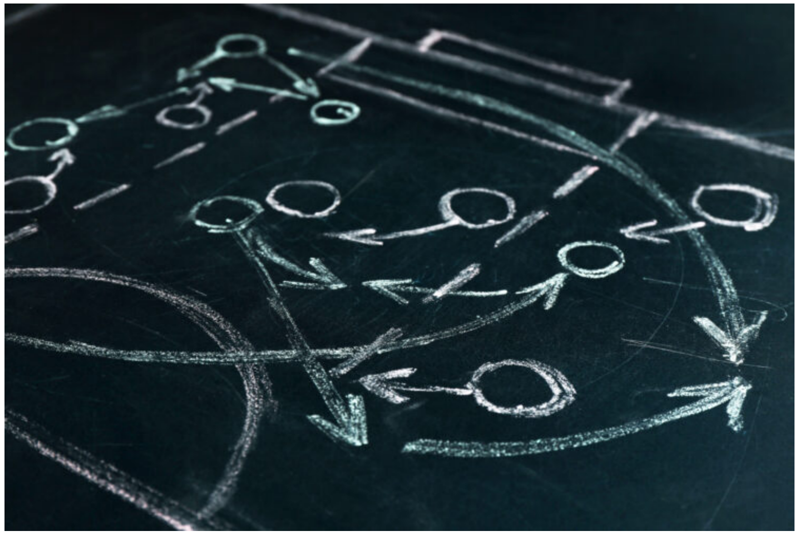
Exploiting a Padding Oracle

Attack plan:

1) Figure out how long is the padding

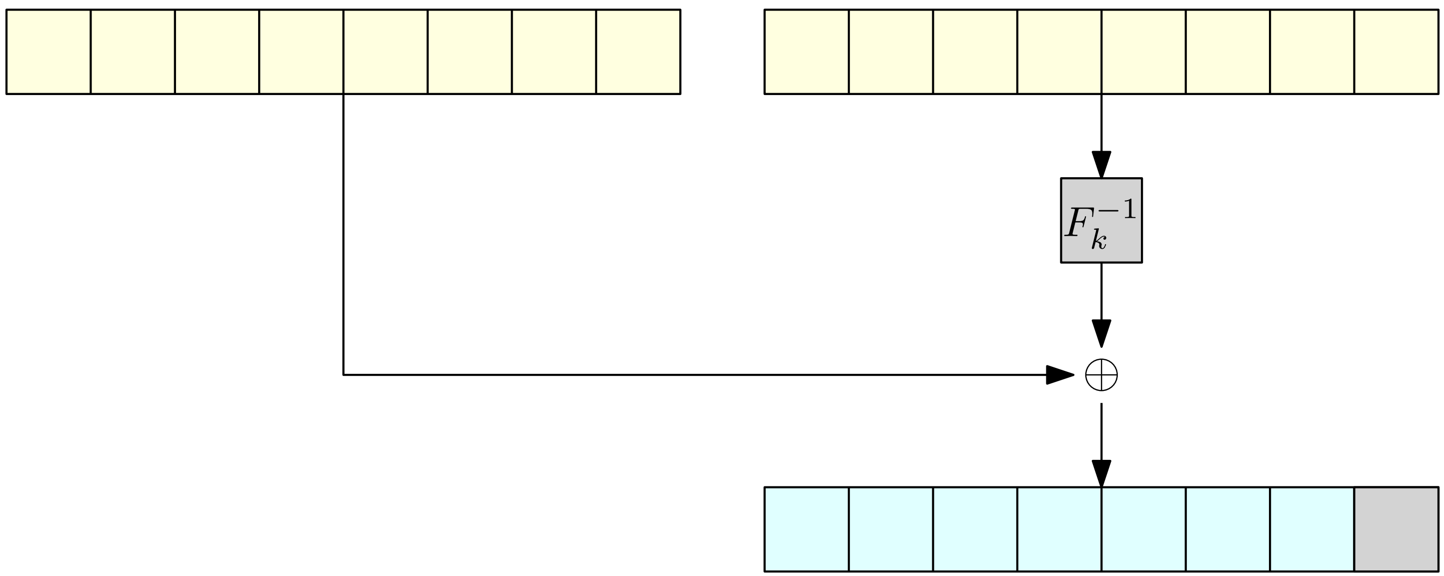
2) Repeat the following until the whole plaintext is recovered:

- Extend the knowledge of the last i bytes of the plaintext (initially $i = 0$) to the last $i + 1$ bytes of the plaintext



Determining the length of the padding

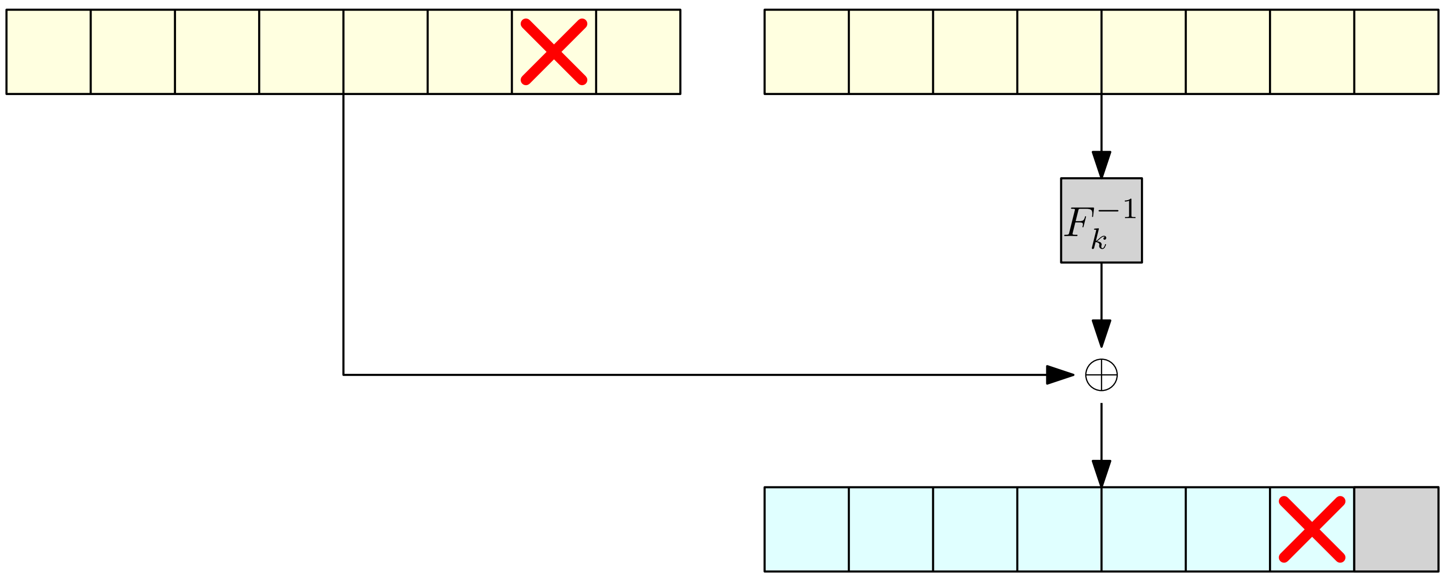
Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding



Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

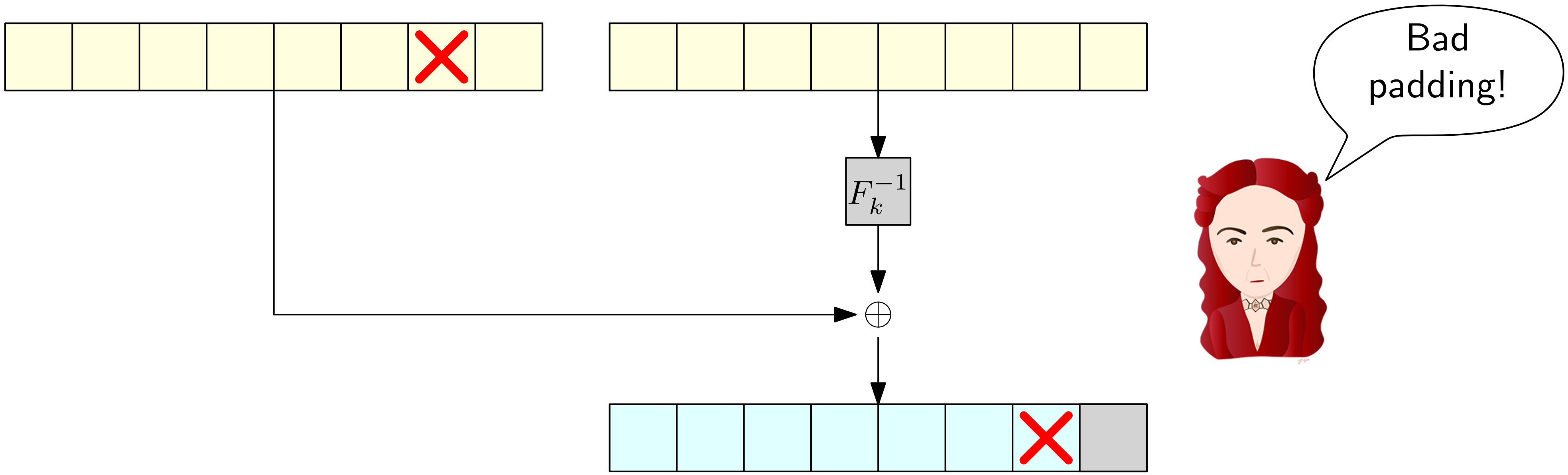
- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext



Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

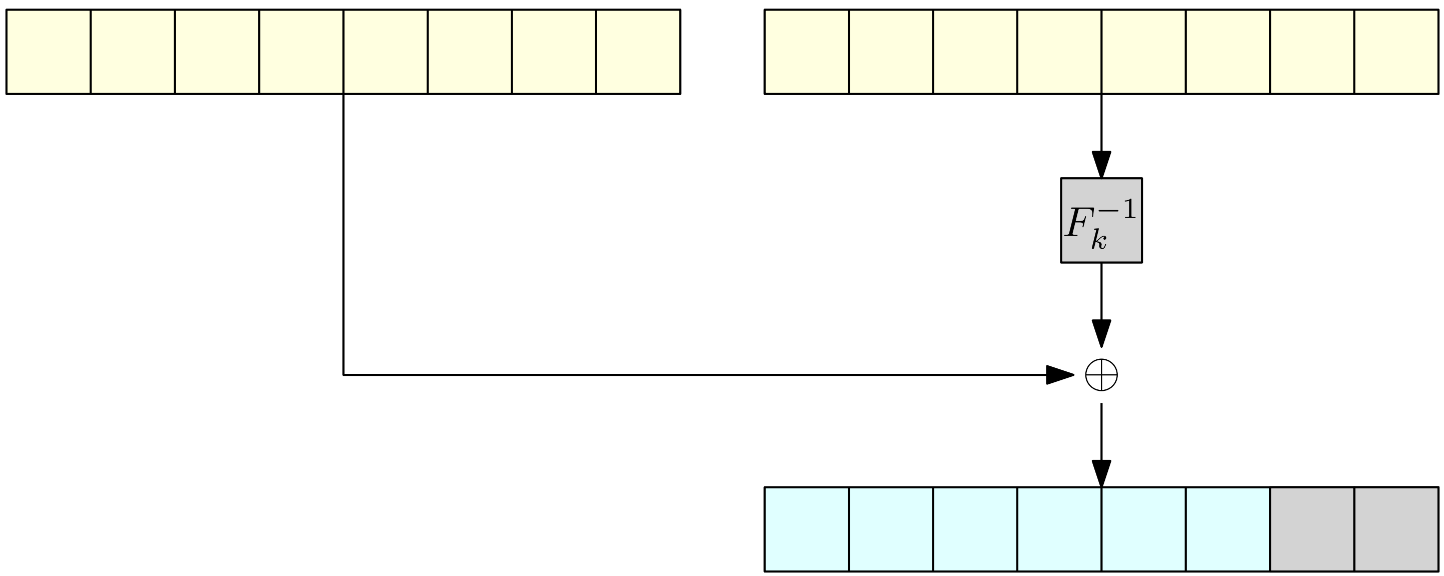
- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding

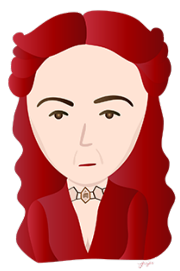


Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding

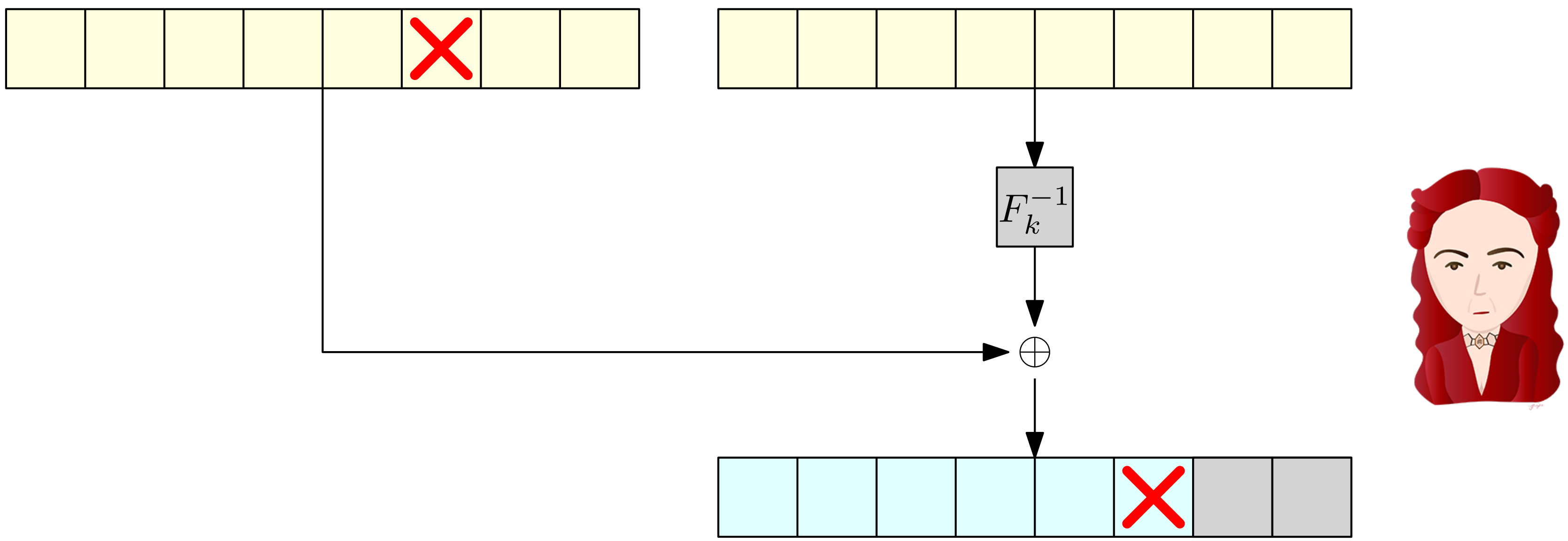




Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

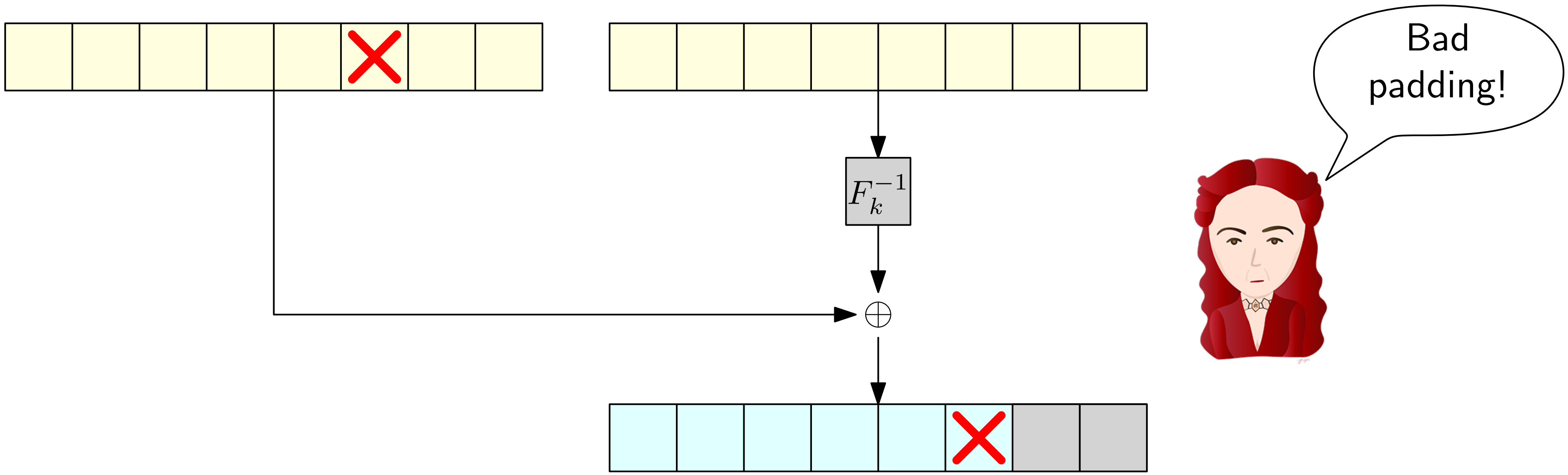
- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding



Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

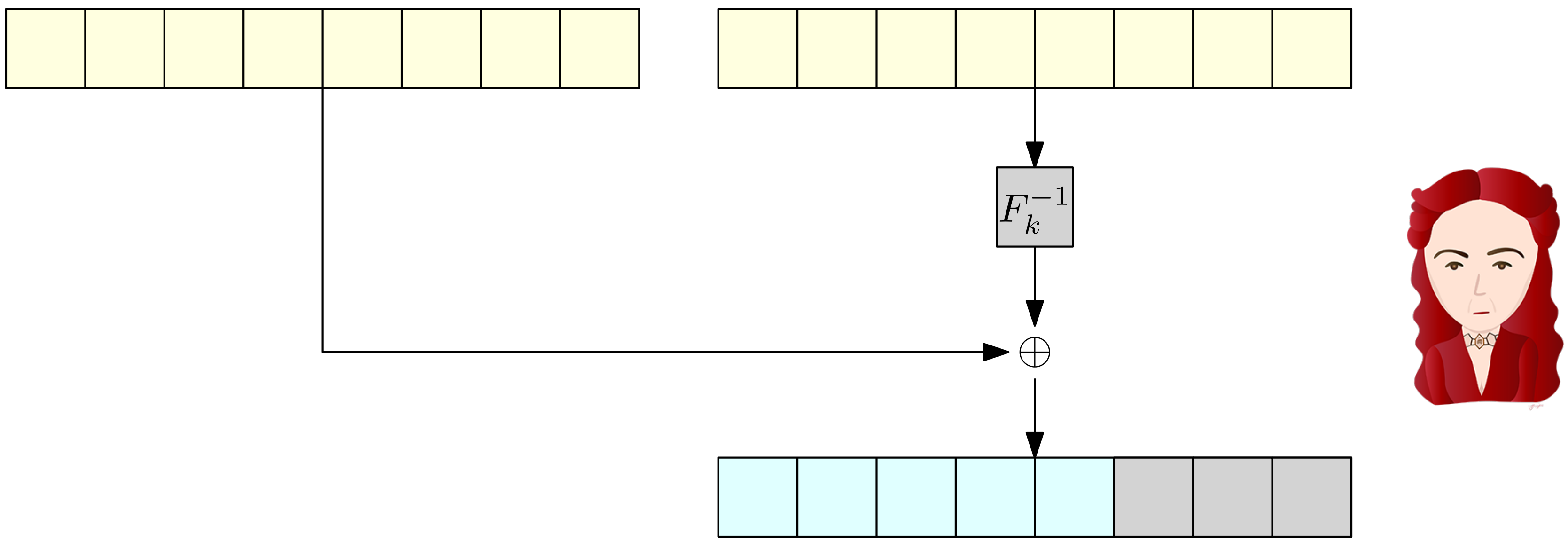
- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding



Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

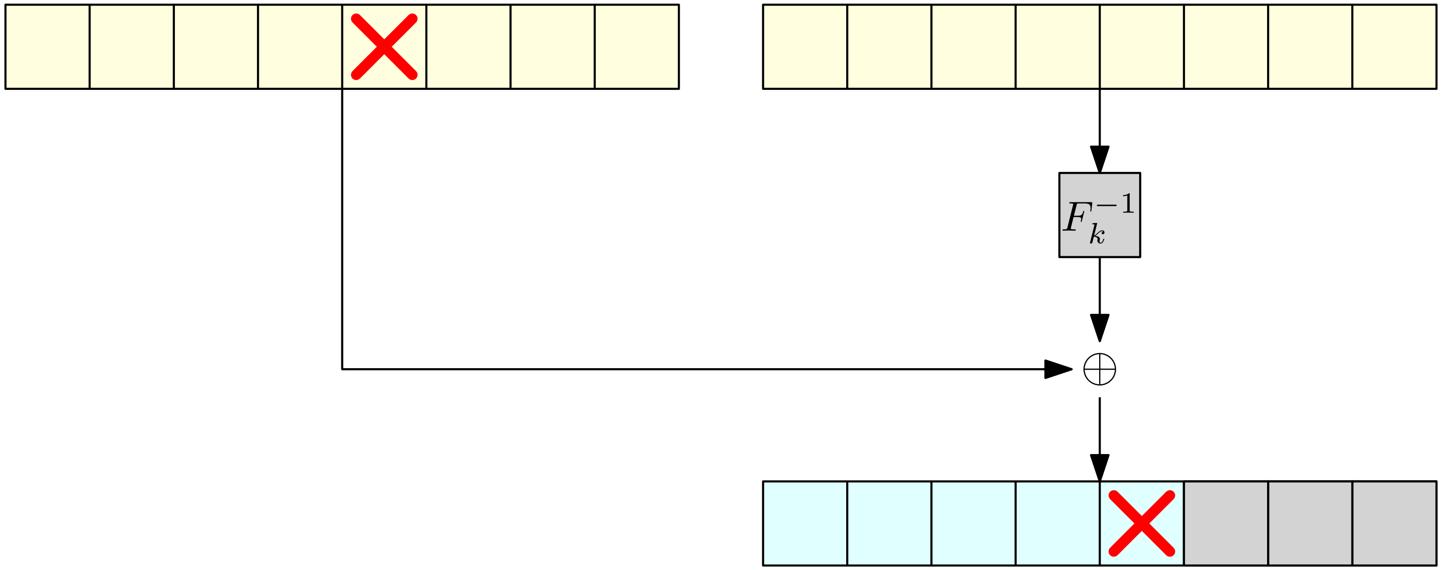
- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding

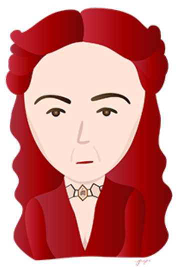


Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding

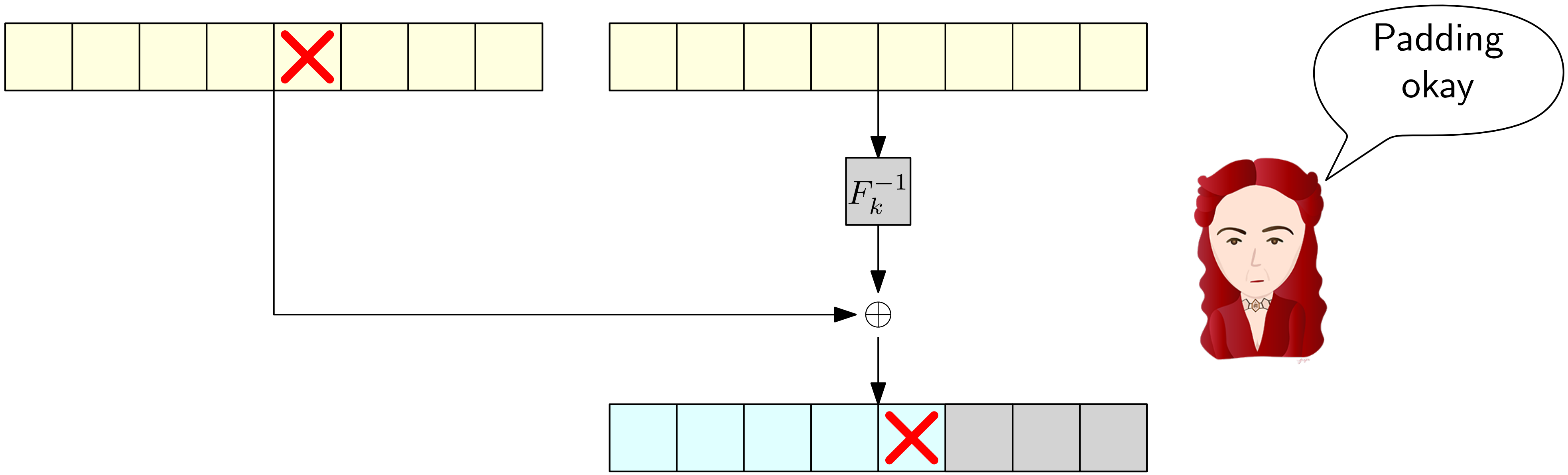




Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

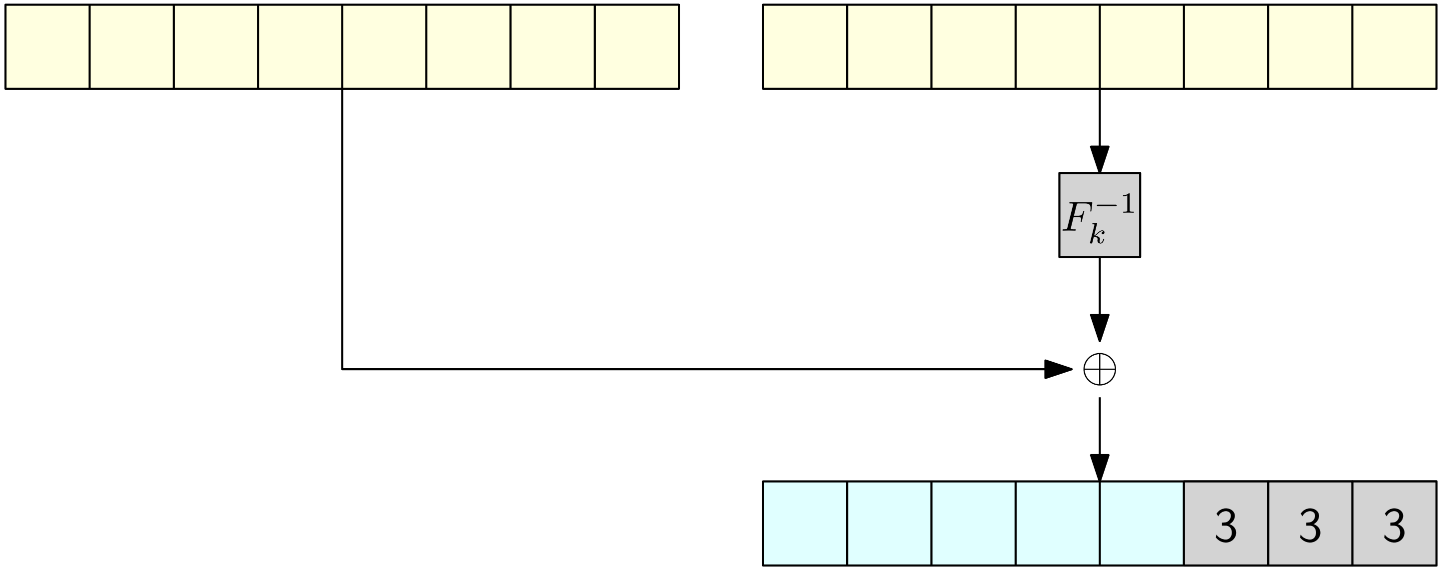
- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding
- Otherwise the j-th byte of the last block is the last byte of the message. Break the loop.

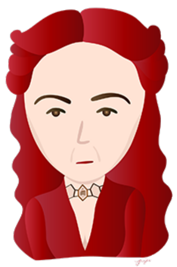


Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding
- Otherwise the j-th byte of the last block is the last byte of the message. Break the loop.

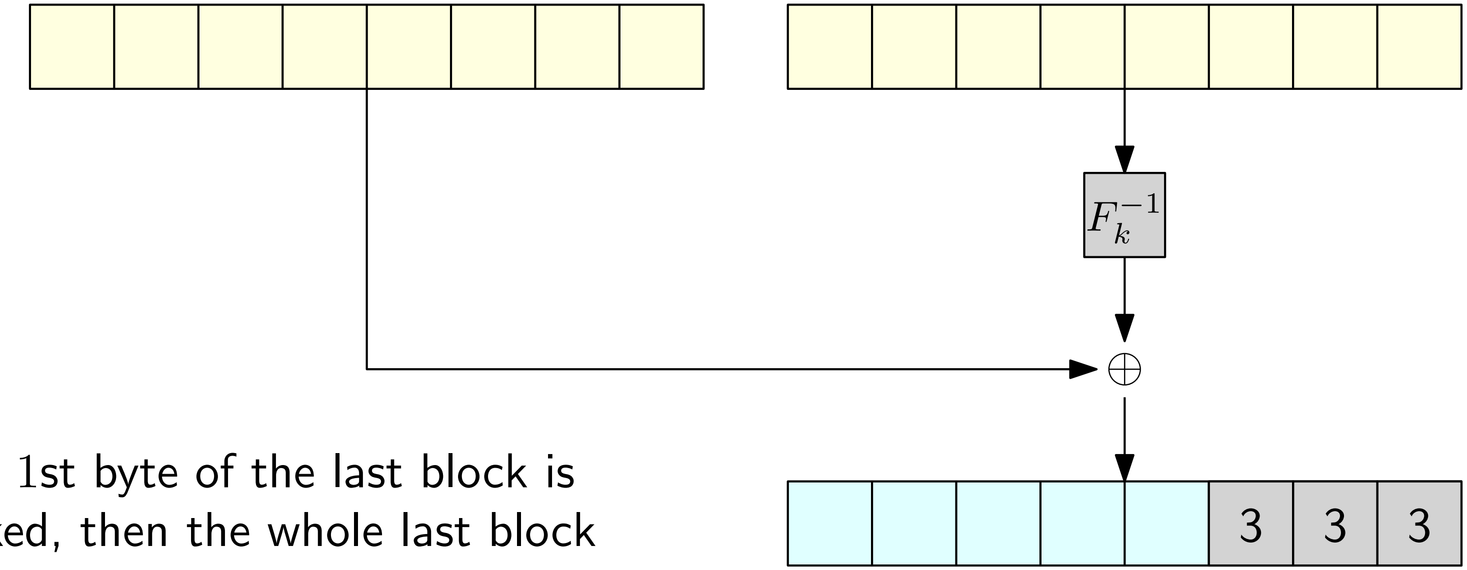


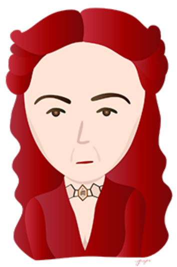


Determining the length of the padding

Idea: for $j = \ell - 1, \ell - 2, \ldots, 1$ determine whether the j-th byte in the last block is part of the padding

- Change the j-th byte in the one-to-last block of the ciphertext
- This alters the j-th byte in the last block of the plaintext
- If "bad padding error" is returned, then j-th byte is checked, and hence it is part of the padding
- Otherwise the j-th byte of the last block is the last byte of the message. Break the loop.

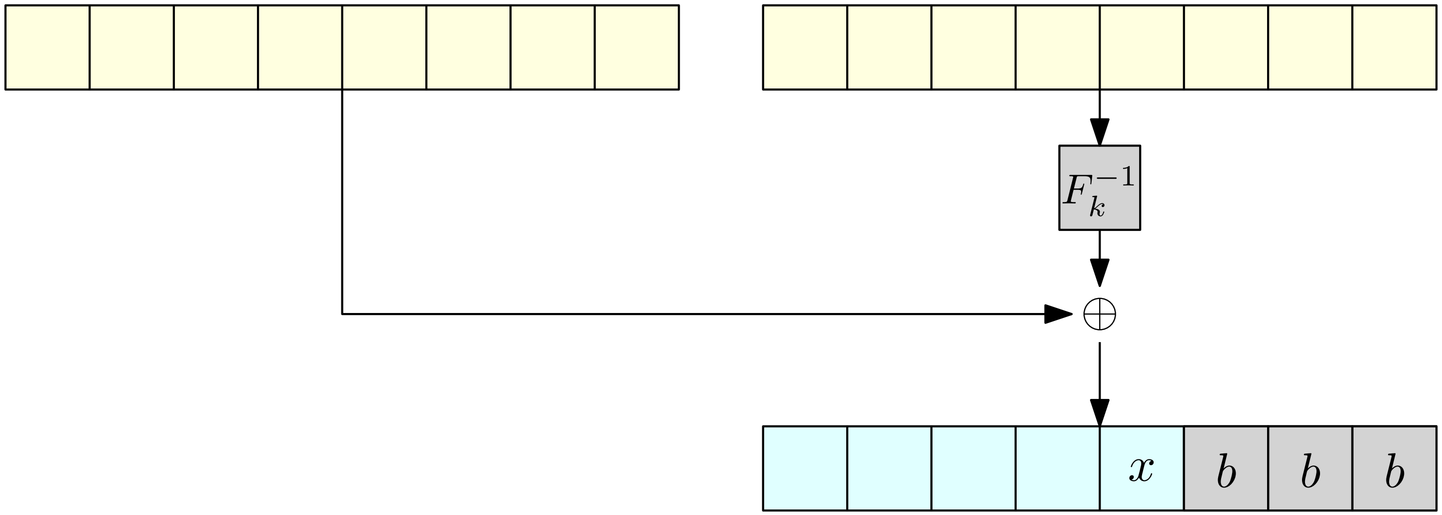


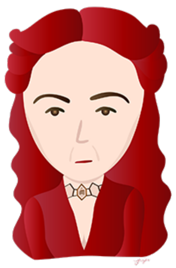


If the 1st byte of the last block is checked, then the whole last block consists of padding

Recovering the last byte of the plaintext

We now know that the last b bytes of the plaintext all have value b. We want to recover the last byte x of the (unpadded) message

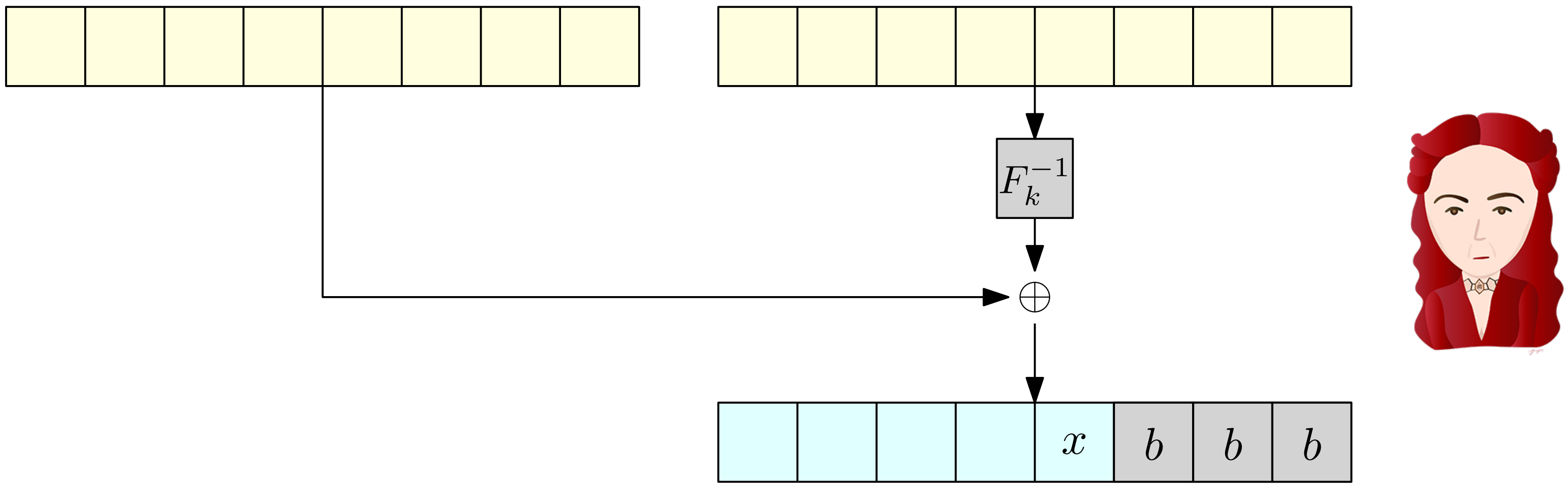




Recovering the last byte of the plaintext

We now know that the last b bytes of the plaintext all have value b. We want to recover the last byte x of the (unpadded) message

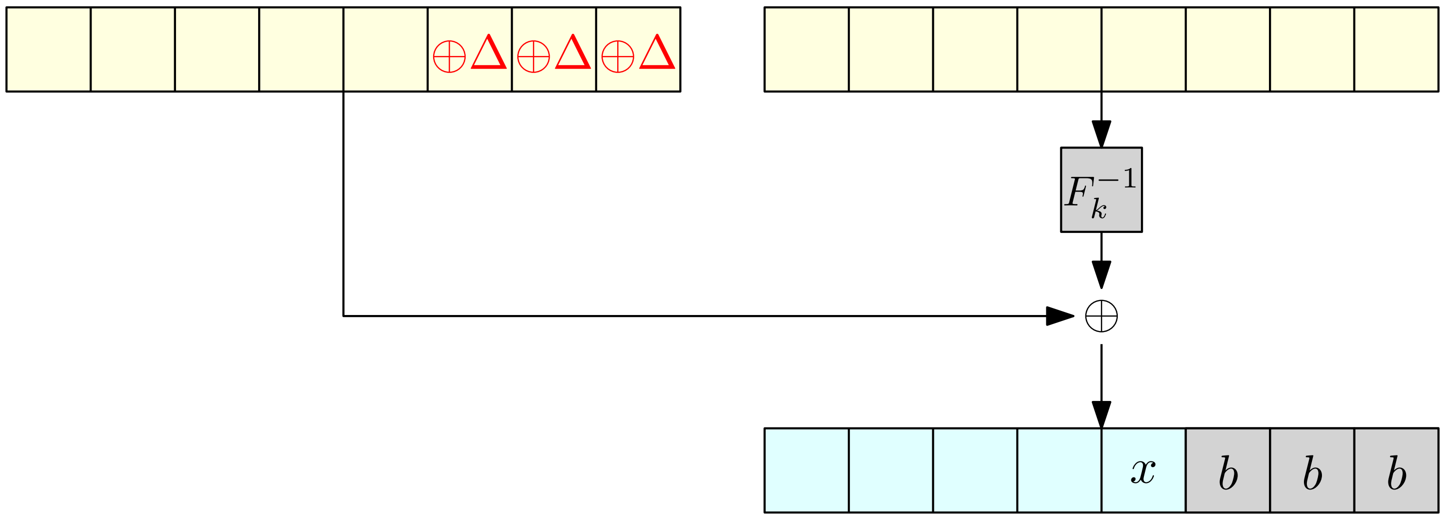
Suppose, for simplicity, that the last block does not consist entirely of padding

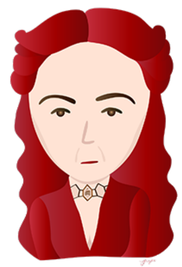


Recovering the last byte of the plaintext

We now know that the last b bytes of the plaintext all have value b. We want to recover the last byte x of the (unpadded) message

- XOR the last b bytes of the one-to-last block of the ciphertext with $\Delta = b \oplus (b+1)$

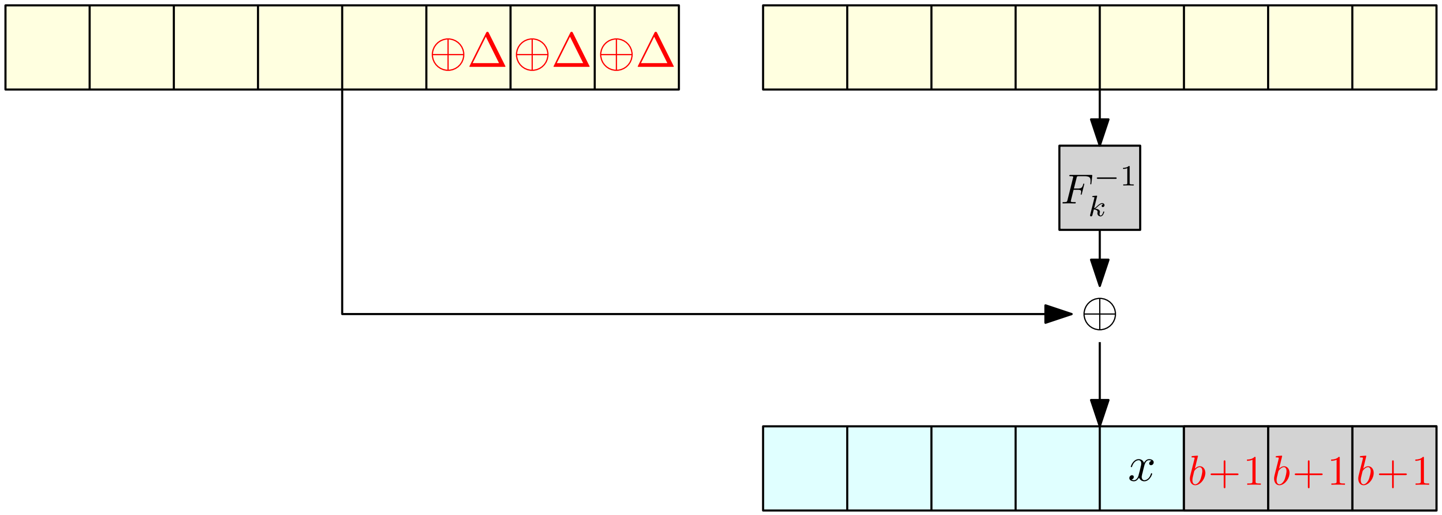


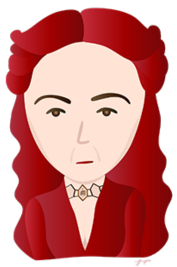


Recovering the last byte of the plaintext

We now know that the last b bytes of the plaintext all have value b. We want to recover the last byte x of the (unpadded) message

- XOR the last b bytes of the one-to-last block of the ciphertext with $\Delta = b \oplus (b+1)$
- This changes the last b blocks of the plaintext to from b to $b \oplus b \oplus (b+1) = b+1$

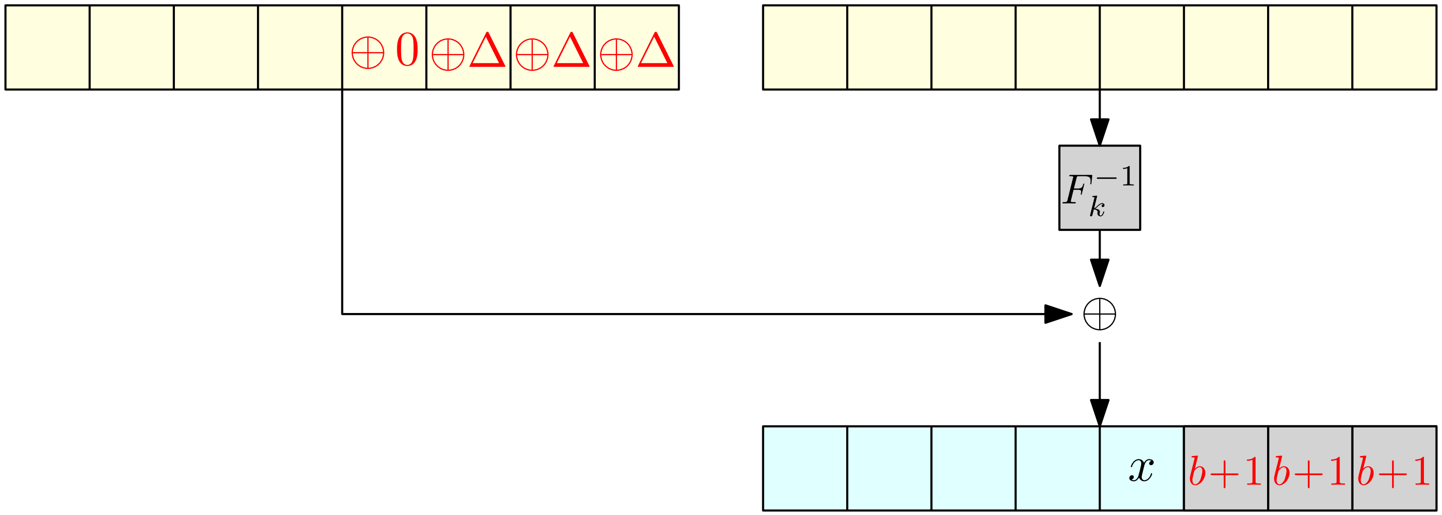


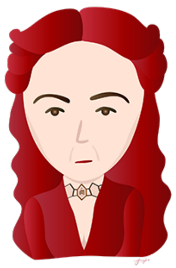


Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i

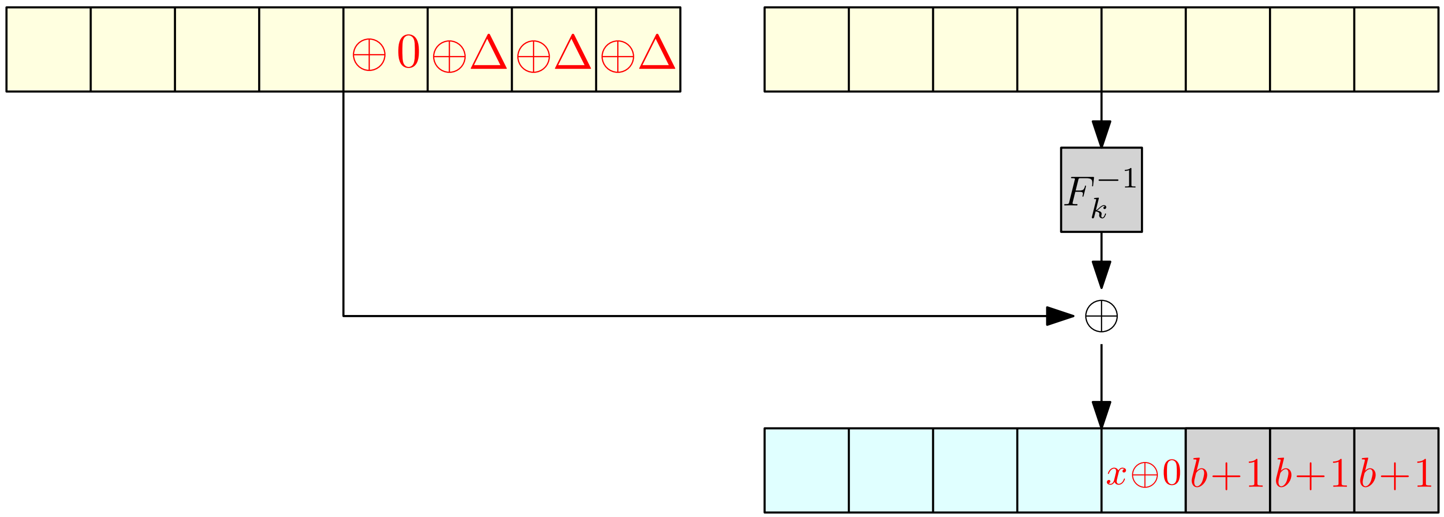


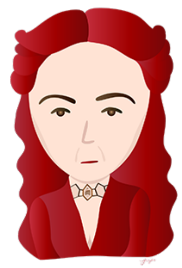


Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$

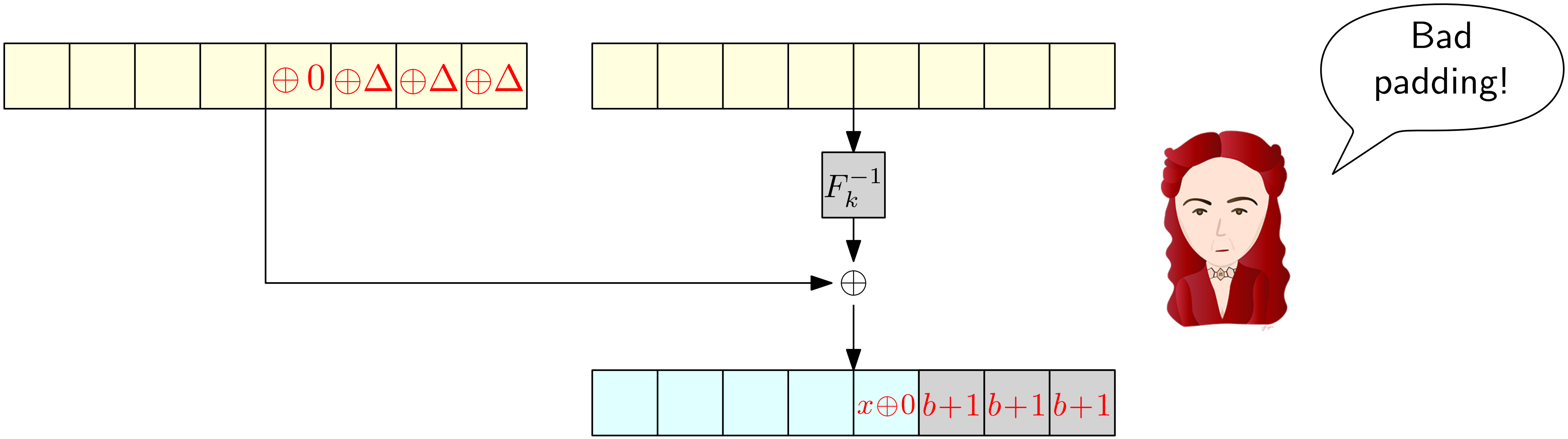




Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

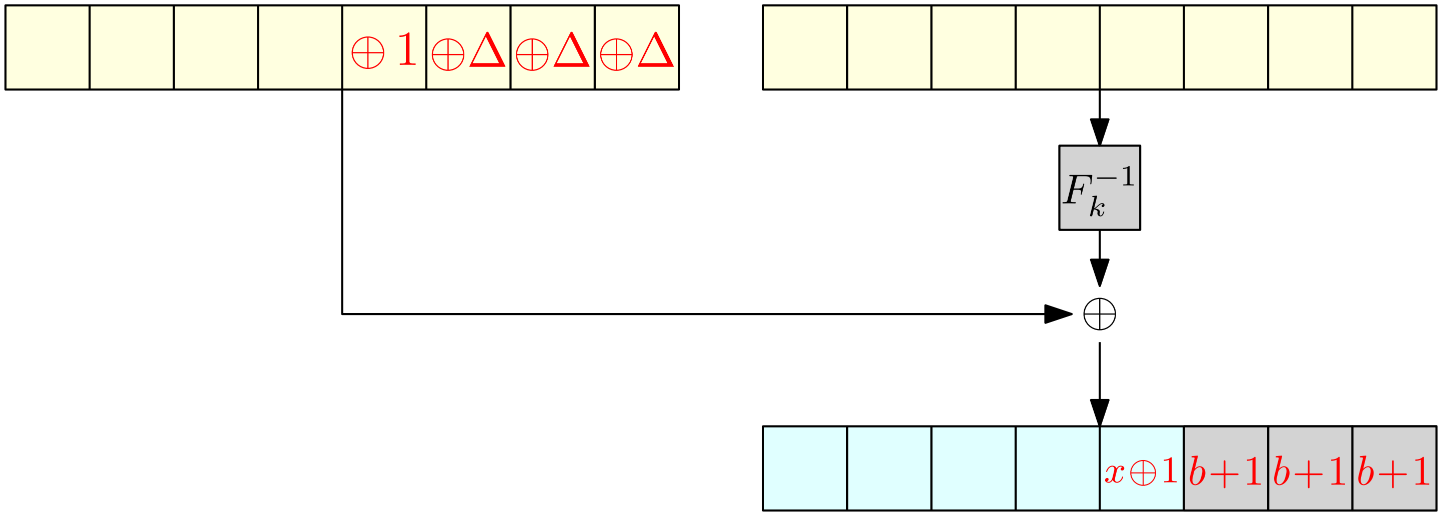
- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b+1$ and we continue with the next value of i

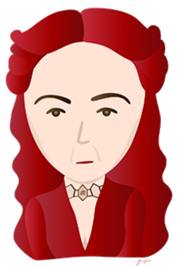


Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b+1$ and we continue with the next value of i

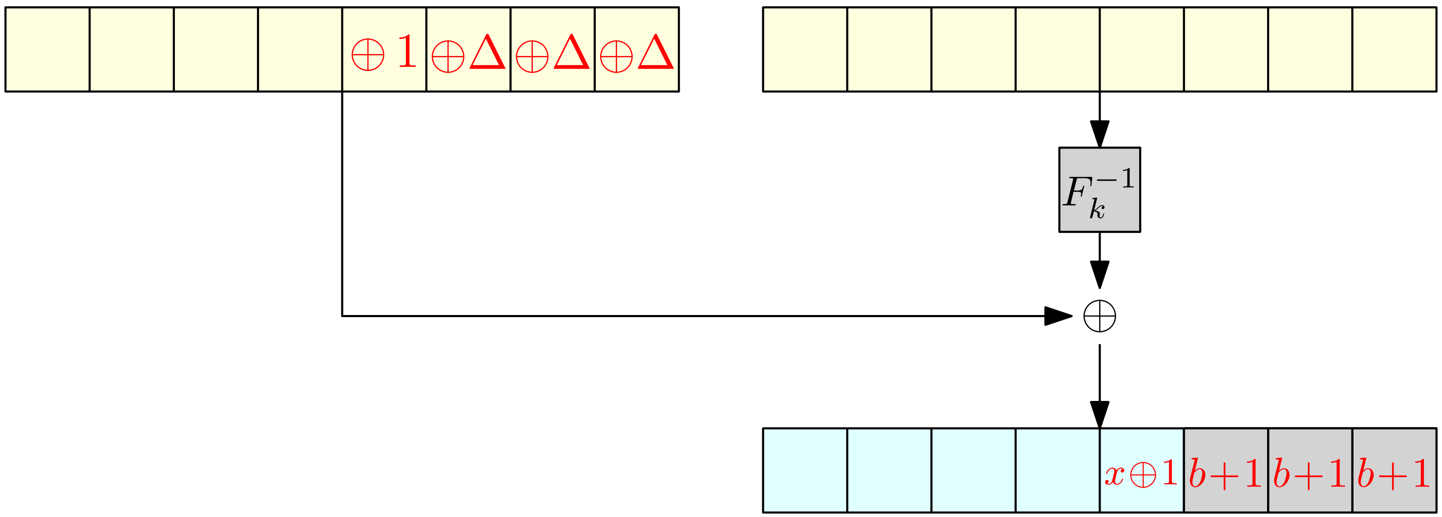




Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

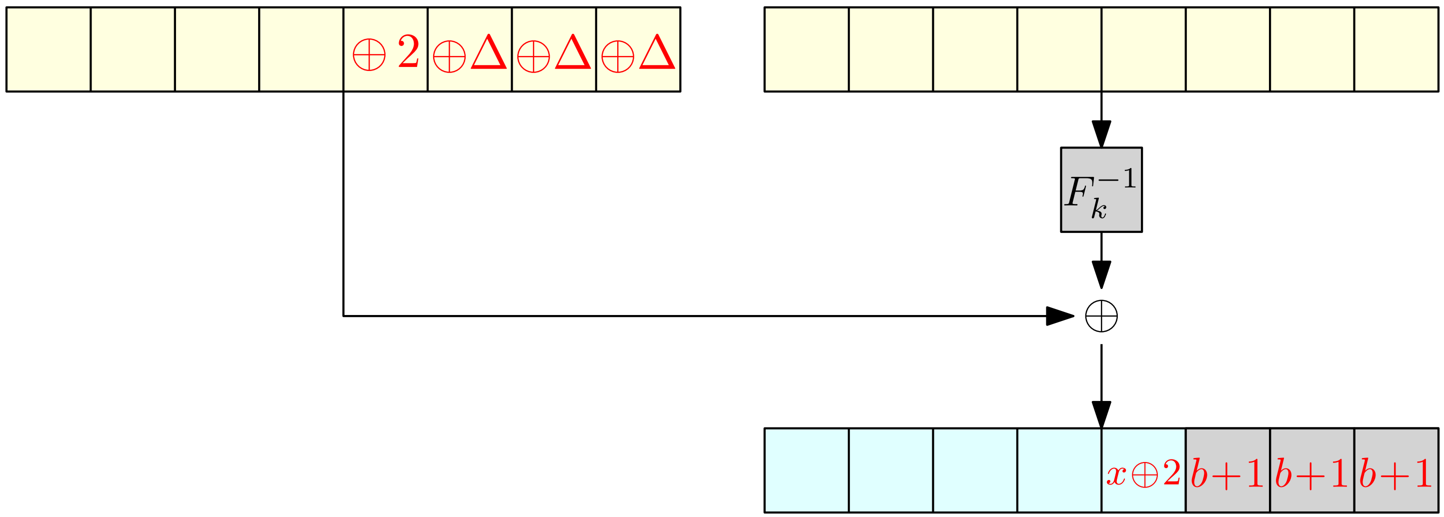
- XOR the $(b + 1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b + 1$ and we continue with the next value of i

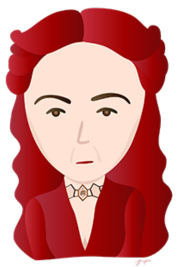


Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b+1$ and we continue with the next value of i

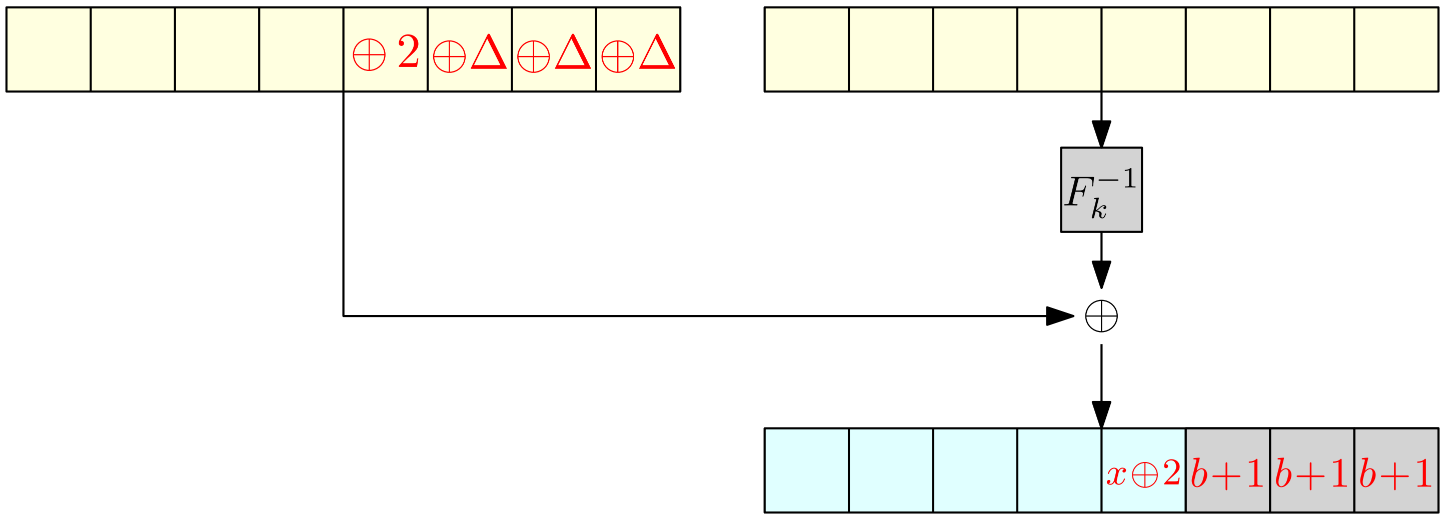




Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

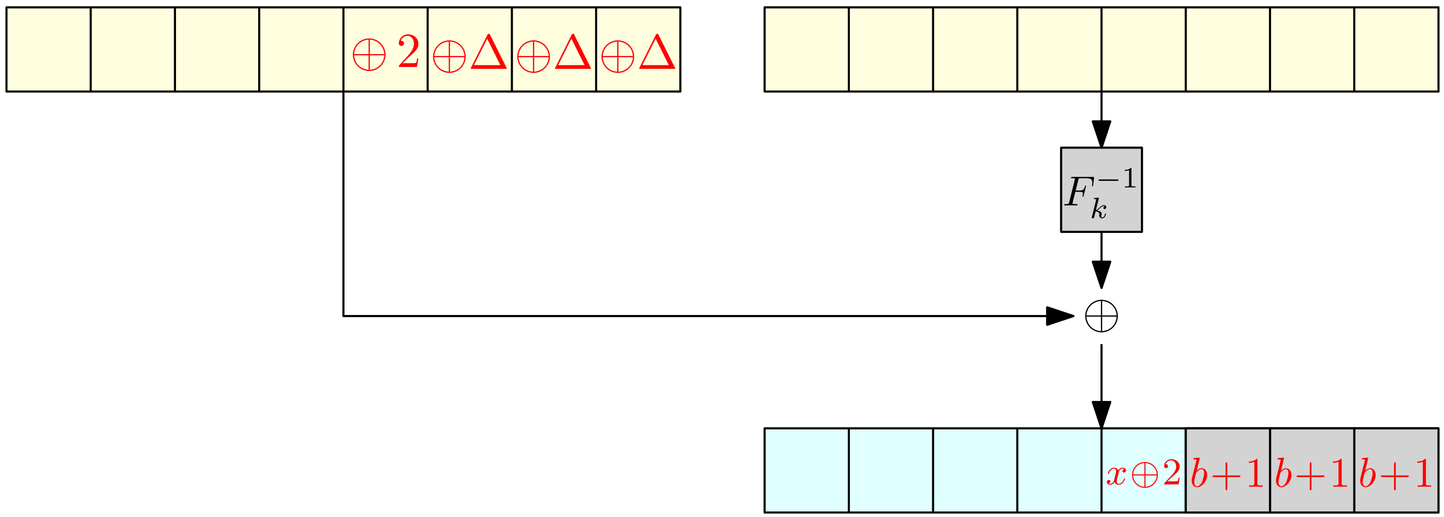
- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b+1$ and we continue with the next value of i
- If the new ciphertext causes no error, then $x \oplus i = b+1$



Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

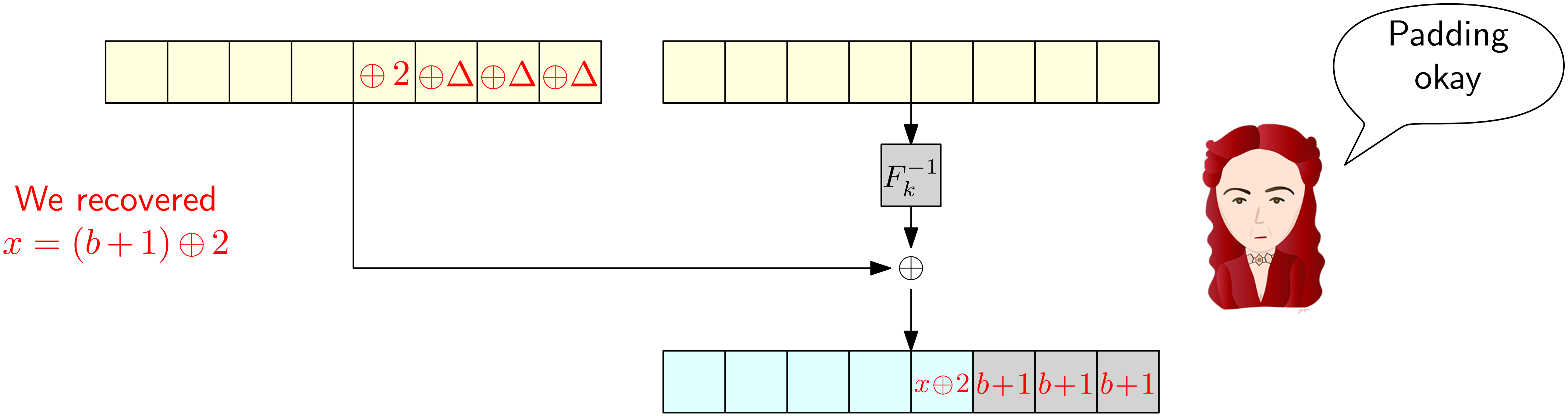
- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b+1$ and we continue with the next value of i
- If the new ciphertext causes no error, then $x \oplus i = b+1 \implies x = (b+1) \oplus i$



Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b+1$ and we continue with the next value of i
- If the new ciphertext causes no error, then $x \oplus i = b+1 \implies x = (b+1) \oplus i$

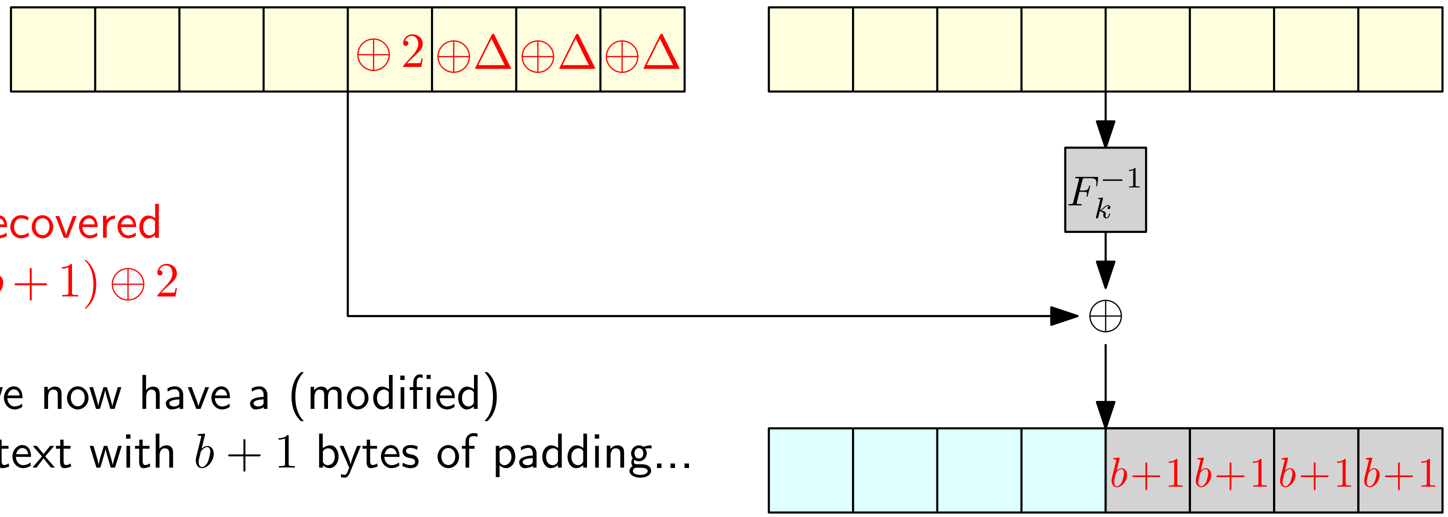


Recovering the last byte of the plaintext

For all possible values $i \in \{0, \dots, 255\}$

- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b+1$ and we continue with the next value of i
- If the new ciphertext causes no error, then $x \oplus i = b+1 \implies x = (b+1) \oplus i$

We recovered
$x = (b+1) \oplus 2$

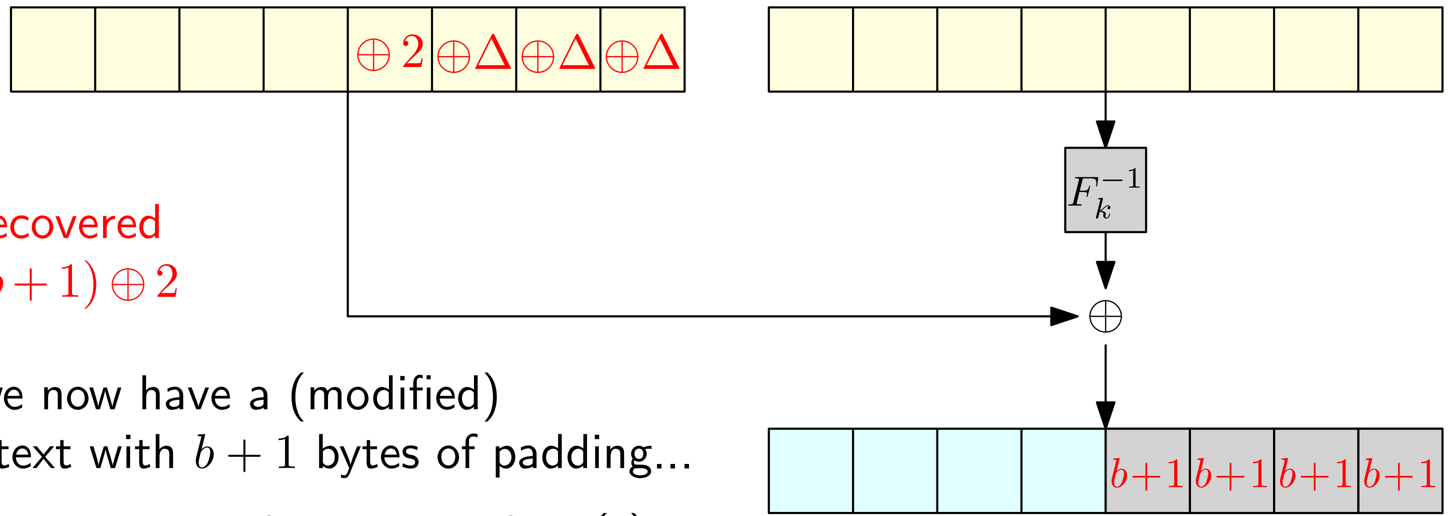


And we now have a (modified) ciphertext with $b+1$ bytes of padding...

Recovering the last byte of the plaintext

For all possible values $i \in \{0, \ldots, 255\}$

- XOR the $(b+1)$-to-last byte of the one-to-last block of the ciphertext with i
- This changes x to $x \oplus i$
- If the new ciphertext results in a "bad padding error" then $x \oplus i \neq b+1$ and we continue with the next value of i
- If the new ciphertext causes no error, then $x \oplus i = b+1 \implies x = (b+1) \oplus i$



We recovered
$x = (b+1) \oplus 2$

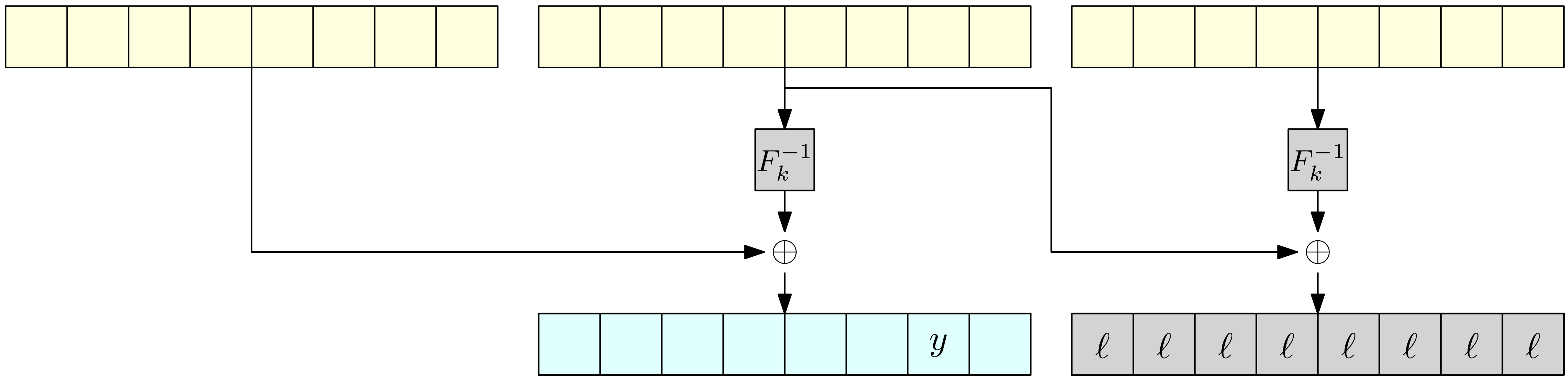
And we now have a (modified) ciphertext with $b+1$ bytes of padding...

Repeat to recover the previous byte(s)...

Recovering the previous block

What do we do if/when the last block of the ciphertext consists entirely of padding?

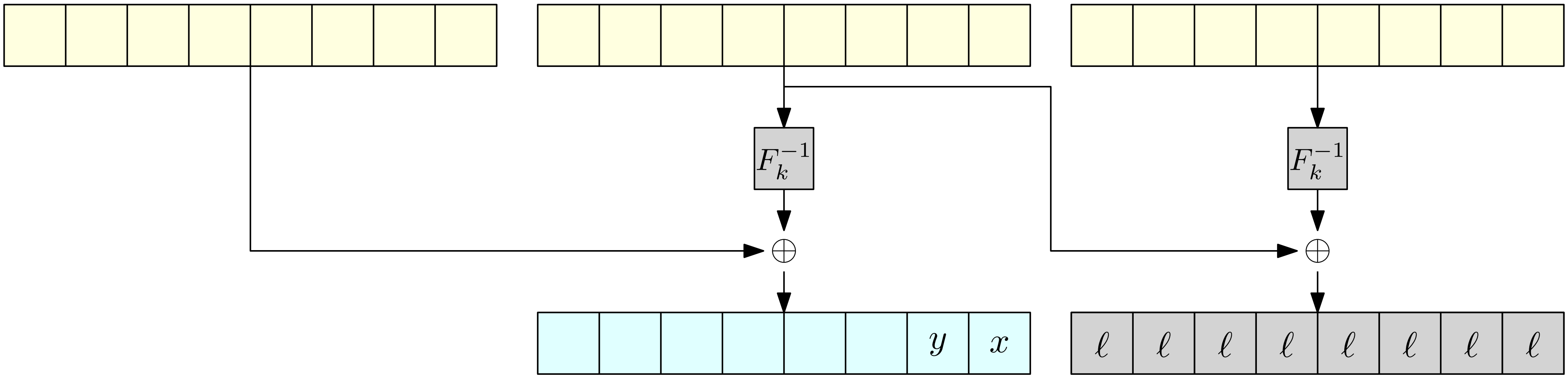
- This can happen on the original ciphertext, or on one of the modified ciphertext after the last block of plaintext has been recovered



Recovering the previous block

What do we do if/when the last block of the ciphertext consists entirely of padding?

- This can happen on the original ciphertext, or on one of the modified ciphertext after the last block of plaintext has been recovered
- We just need to recover the last byte x of the one-to-last block of the plaintext, and then the previous strategy applies

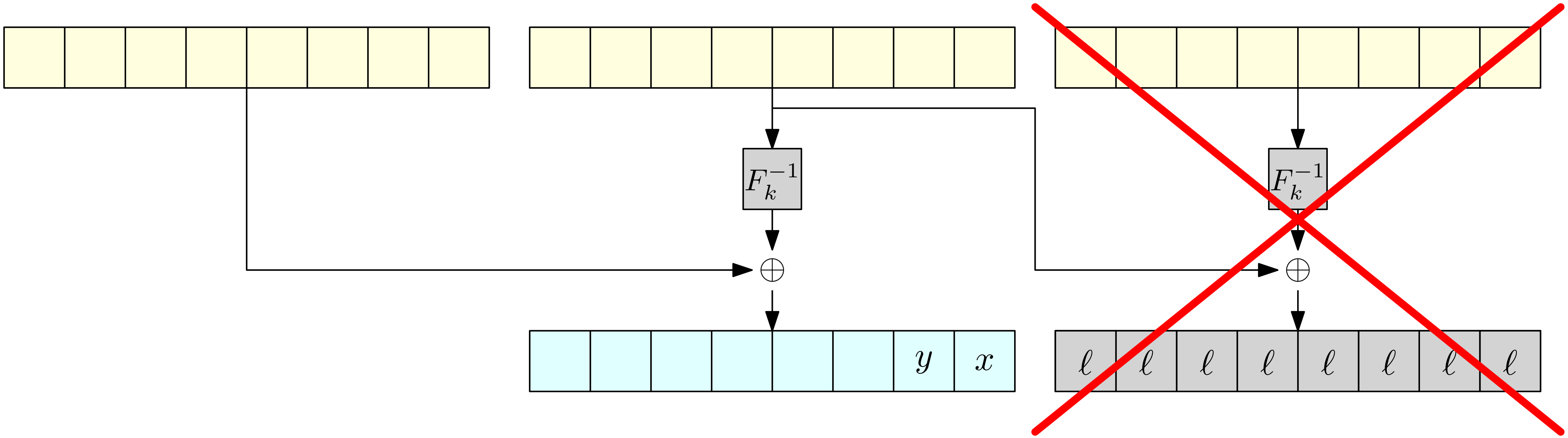


Recovering the previous block

What do we do if/when the last block of the ciphertext consists entirely of padding?

- This can happen on the original ciphertext, or on one of the modified ciphertext after the last block of plaintext has been recovered
- We just need to recover the last byte x of the one-to-last block of the plaintext, and then the previous strategy applies

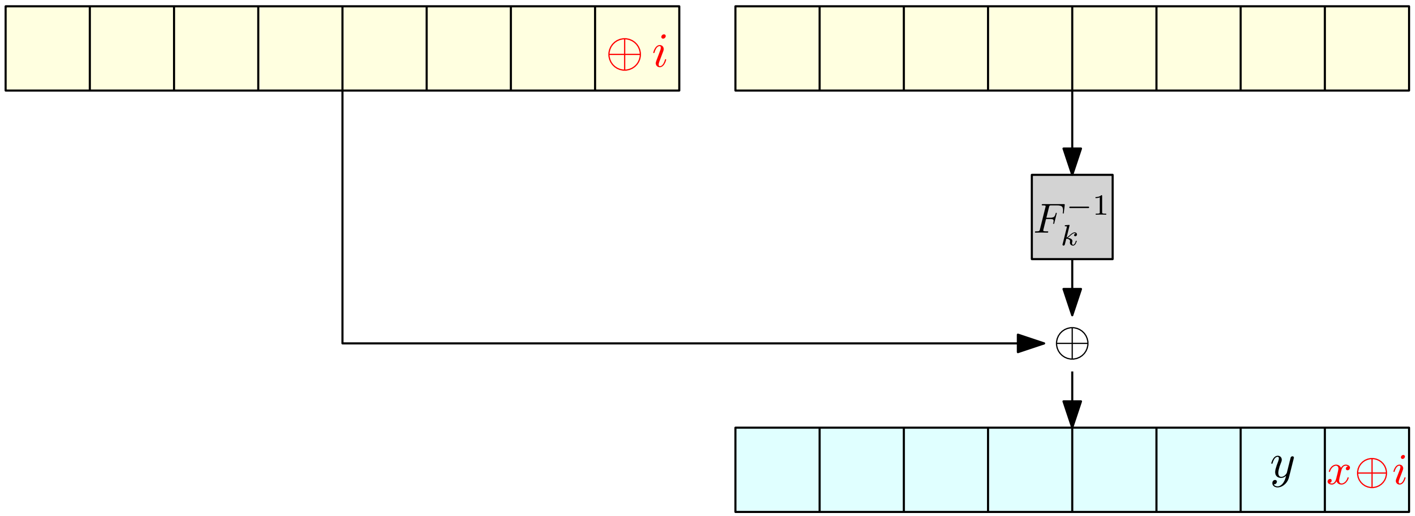
Permanently drop the last block of the ciphertext



Recovering the previous block

Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

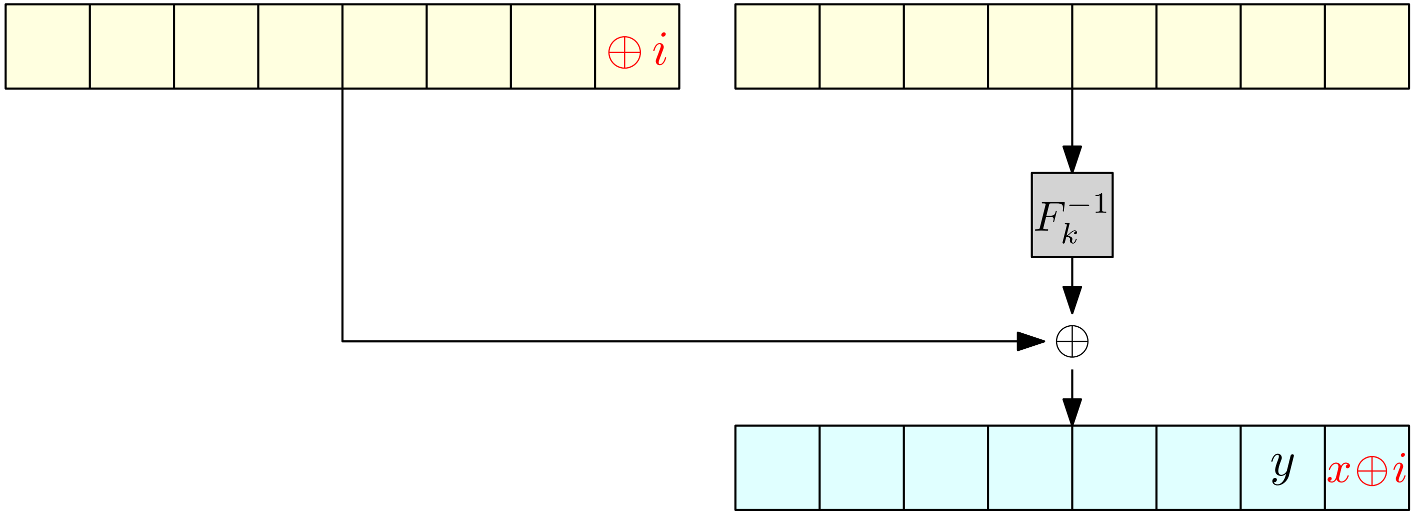
- For every $i \in \{0, \ldots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)



Recovering the previous block

Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

- For every $i \in \{0, \ldots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)
- For which values of i do we obtain a valid padding?

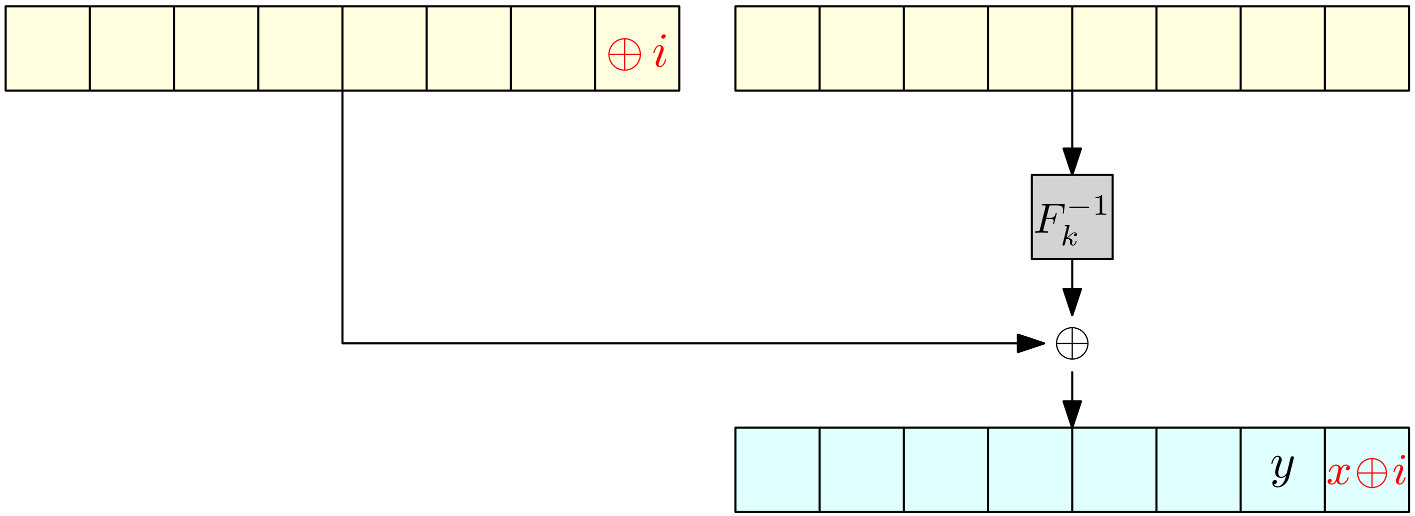


Recovering the previous block

Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

- For every $i \in \{0, \ldots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)
- For which values of i do we obtain a valid padding?

  When $x \oplus i = 1 \ldots$

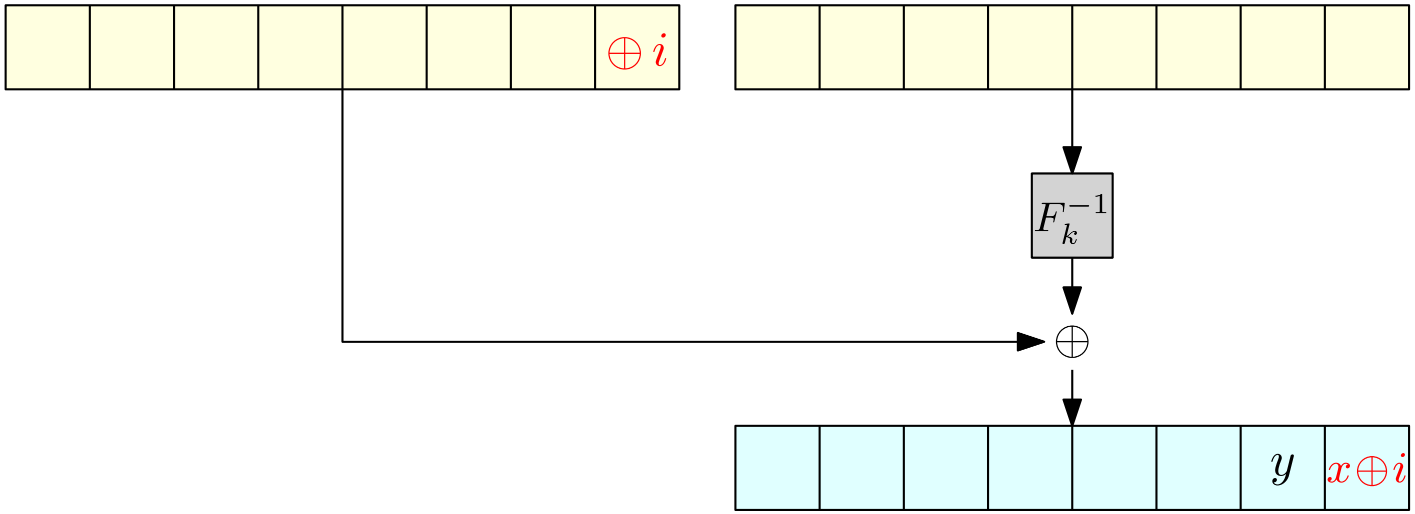


Recovering the previous block

Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

- For every $i \in \{0, \ldots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)
- For which values of i do we obtain a valid padding?

When $x \oplus i = 1 \ldots$ and (possibly) when $x \oplus i = y$ and the last y bytes of the plaintext are y



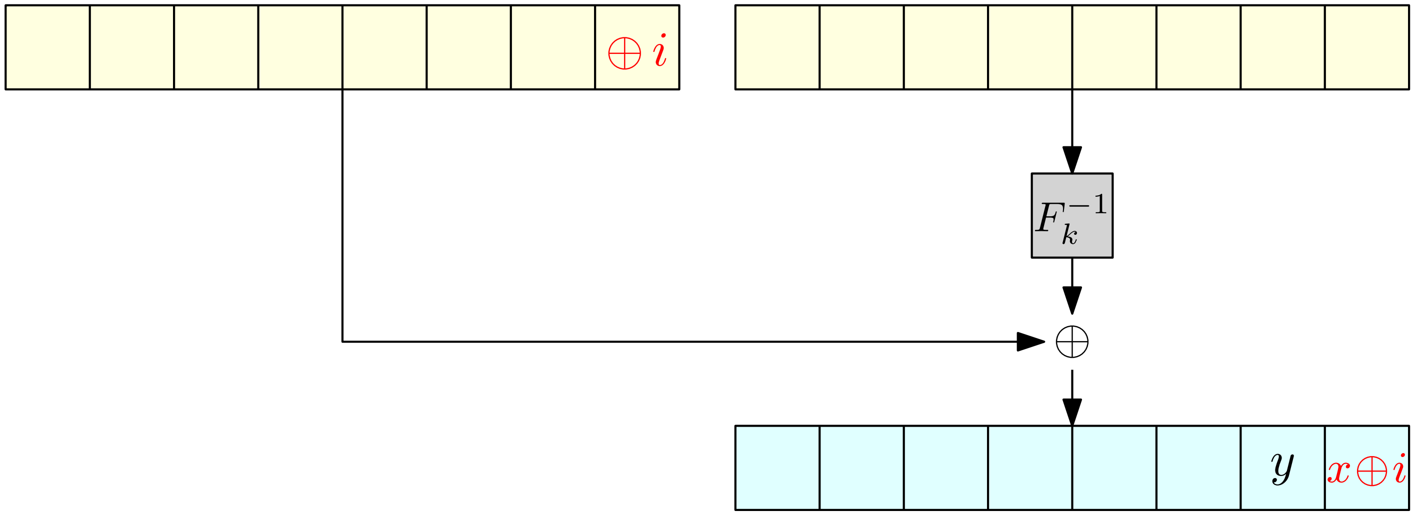
Recovering the previous block

Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

- For every $i \in \{0, \ldots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)
- For which values of i do we obtain a valid padding?

  When $x \oplus i = 1 \ldots$ and (possibly) when $x \oplus i = y$ and the last y bytes of the plaintext are y

- At most 2 candidates for i.



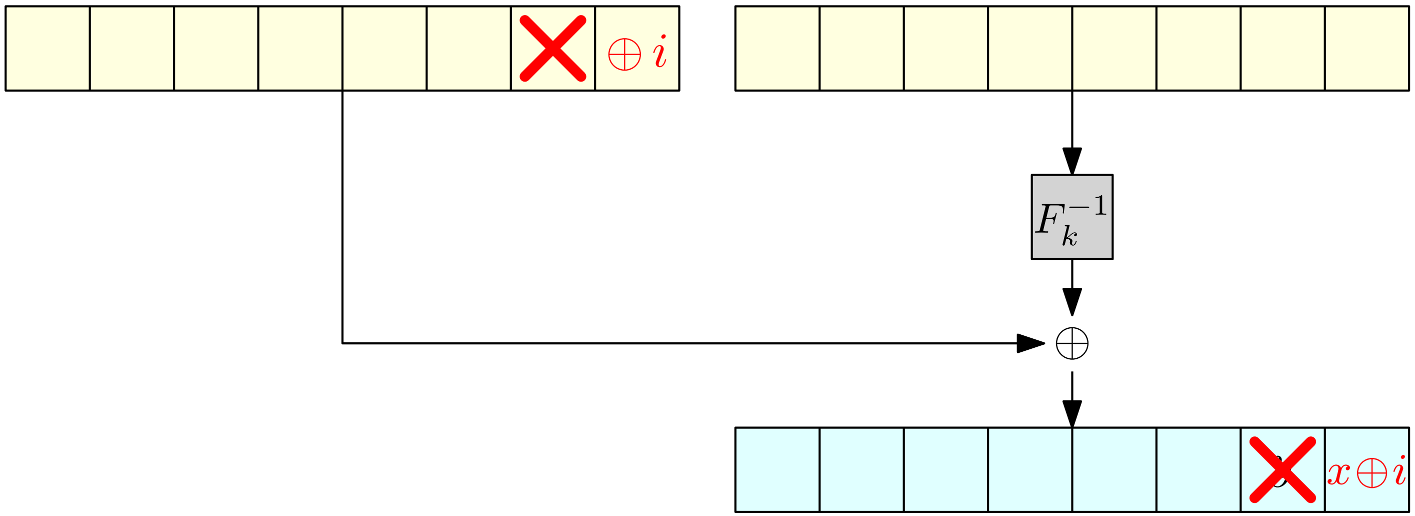
Recovering the previous block

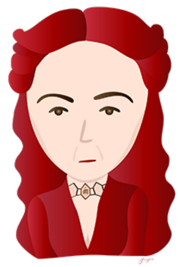
Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

- For every $i \in \{0, \ldots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)
- For which values of i do we obtain a valid padding?

  When $x \oplus i = 1 \ldots$ and (possibly) when $x \oplus i = y$ and the last y bytes of the plaintext are y

- At most 2 candidates for i. Find the correct one by altering the previous byte of the plaintext





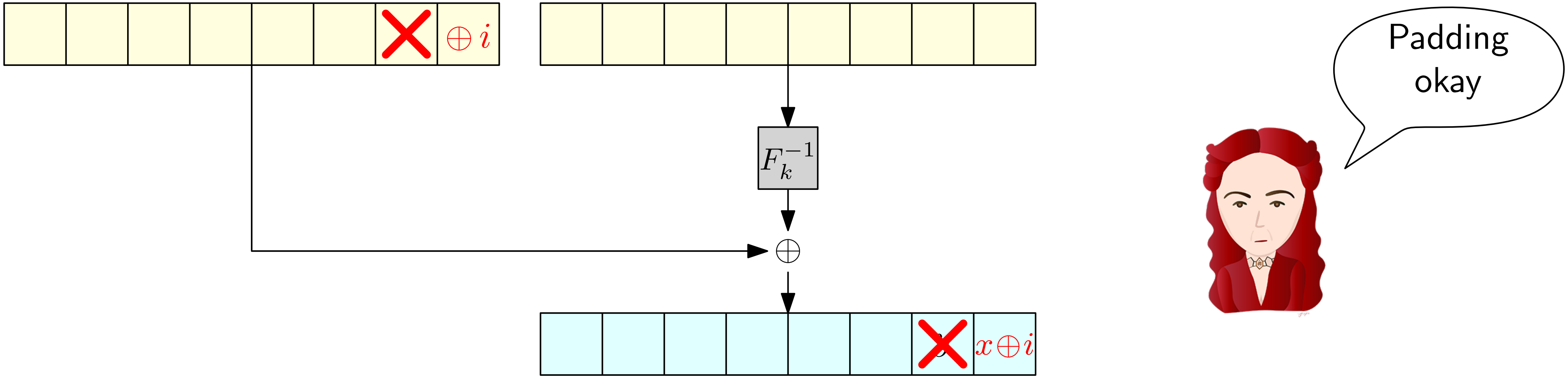
Recovering the previous block

Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

- For every $i \in \{0, \dots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)
- For which values of i do we obtain a valid padding?

  When $x \oplus i = 1 \dots$ and (possibly) when $x \oplus i = y$ and the last y bytes of the plaintext are y

- At most 2 candidates for i. Find the correct one by altering the previous byte of the plaintext



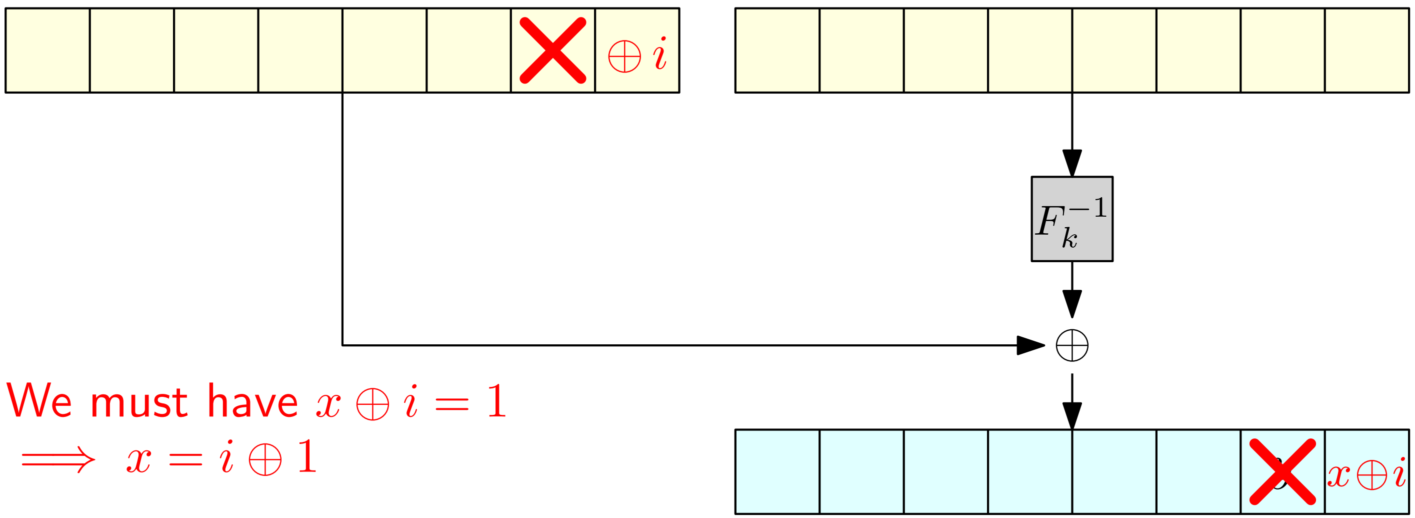
Recovering the previous block

Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

- For every $i \in \{0, \ldots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)
- For which values of i do we obtain a valid padding?

  When $x \oplus i = 1 \ldots$ and (possibly) when $x \oplus i = y$ and the last y bytes of the plaintext are y

- At most 2 candidates for i. Find the correct one by altering the previous byte of the plaintext



We must have $x \oplus i = 1$
$\implies x = i \oplus 1$

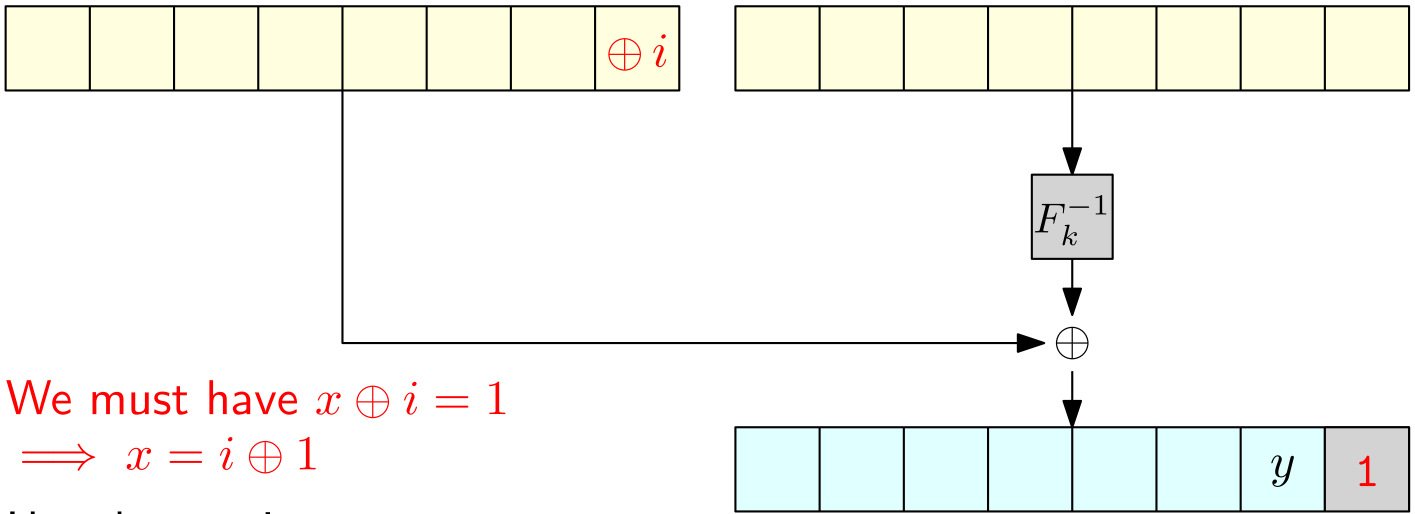
Recovering the previous block

Try to transform x into a 1 (which is a valid 1-byte padding of the remaining ciphertext)

- For every $i \in \{0, \ldots, 255\}$, XOR x with i (by XOR-ing the previous block of the ciphertext)
- For which values of i do we obtain a valid padding?

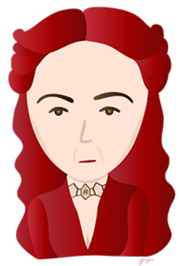
  When $x \oplus i = 1 \ldots$ and (possibly) when $x \oplus i = y$ and the last y bytes of the plaintext are y

- At most 2 candidates for i. Find the correct one by altering the previous byte of the plaintext



We must have $x \oplus i = 1$
$\implies x = i \oplus 1$

Use the previous strategy to recover the rest of the block



Padding Oracle Attack: Complexity?

- At most ℓ attempts to learn the length of the padding
- At most 257 attempts to learn a byte of the ciphertext
    - $\leq 256 + 1$ attempts to learn the last byte of a block
    - ≤ 256 attempts to learn each of the other bytes

(can be slightly improved)

Padding Oracle Attack: Complexity?

- At most ℓ attempts to learn the length of the padding
- At most 257 attempts to learn a byte of the ciphertext
  - $\leq 256 + 1$ attempts to learn the last byte of a block
  - ≤ 256 attempts to learn each of the other bytes

(can be slightly improved)

At most $\ell + 257 \cdot |m|$ decryption attempts

Linear in $|m|$ and ℓ!

CCA Security and Authenticated Encryption

- Chosen-ciphertext attacks are not just a theoretical threat

  E.g., padding oracle attacks against SSL, IPSec, Steam...

CCA Security and Authenticated Encryption

- Chosen-ciphertext attacks are not just a theoretical threat

  E.g., padding oracle attacks against SSL, IPSec, Steam...

- We need CCA-secure encryption schemes!

- None of the encryption schemes we have seen so far are CCA-secure

CCA Security and Authenticated Encryption

- Chosen-ciphertext attacks are not just a theoretical threat

  E.g., padding oracle attacks against SSL, IPSec, Steam...

- We need CCA-secure encryption schemes!

- None of the encryption schemes we have seen so far are CCA-secure

- Fortunately we can build CCA-secure encryption schemes from CPA-secure encryption schemes

CCA Security and Authenticated Encryption

- Chosen-ciphertext attacks are not just a theoretical threat

  E.g., padding oracle attacks against SSL, IPSec, Steam...

- We need CCA-secure encryption schemes!
- None of the encryption schemes we have seen so far are CCA-secure
- Fortunately we can build CCA-secure encryption schemes from CPA-secure encryption schemes
- In fact, we are going to achieve an even stronger security guarantee:

Authenticated Encryption

Authenticated Encryption

- We know how to achieve secrecy against passive adversaries
- We know how to achieve integrity against active adversaries

Authenticated Encryption

- We know how to achieve secrecy against passive adversaries   Stream ciphers, block ciphers
- We know how to achieve integrity against active adversaries

Authenticated Encryption

- We know how to achieve secrecy against passive adversaries — Stream ciphers, block ciphers
- We know how to achieve integrity against active adversaries — MACs

Authenticated Encryption

- We know how to achieve secrecy against passive adversaries — Stream ciphers, block ciphers
- We know how to achieve integrity against active adversaries — MACs

What if we want **both**, **against active adversaries**?

Authenticated Encryption

- We know how to achieve secrecy against passive adversaries — Stream ciphers, block ciphers
- We know how to achieve integrity against active adversaries — MACs

What if we want **both**, **against active adversaries**?

- **Secrecy requirement:** CCA-security

  Intuition: The adversary cannot efficiently learn anything about the plaintext even if it can tamper with the ciphertext
  (except for a negligible probability)

Authenticated Encryption

- We know how to achieve secrecy against passive adversaries — Stream ciphers, block ciphers
- We know how to achieve integrity against active adversaries — MACs

What if we want **both**, **against active adversaries**?

- **Secrecy requirement:** CCA-security

  Intuition: The adversary cannot efficiently learn anything about the plaintext even if it can tamper with the ciphertext
  (except for a negligible probability)

- **Integrity requirement:** unforgeability

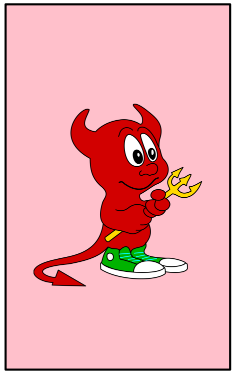
  Intuition: The adversary cannot efficiently provide any valid ciphertext
  (unless it corresponds to a message that was already encrypted by the honest parties)

The Unforgeability Experiment (adapted for AE)

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. We name the following experiment $\mathsf{Enc\text{-}forge}_{\mathcal{A},\Pi}(n)$:

- A key k is generated using $\mathsf{Gen}(1^n)$

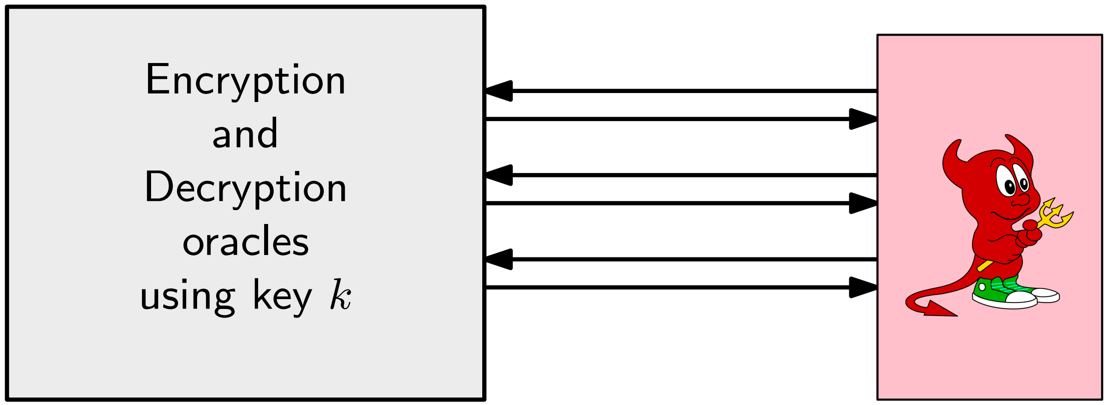
Encryption
and
Decryption
oracles
using key k



The Unforgeability Experiment (adapted for AE)

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. We name the following experiment $\mathsf{Enc\text{-}forge}_{\mathcal{A},\Pi}(n)$:

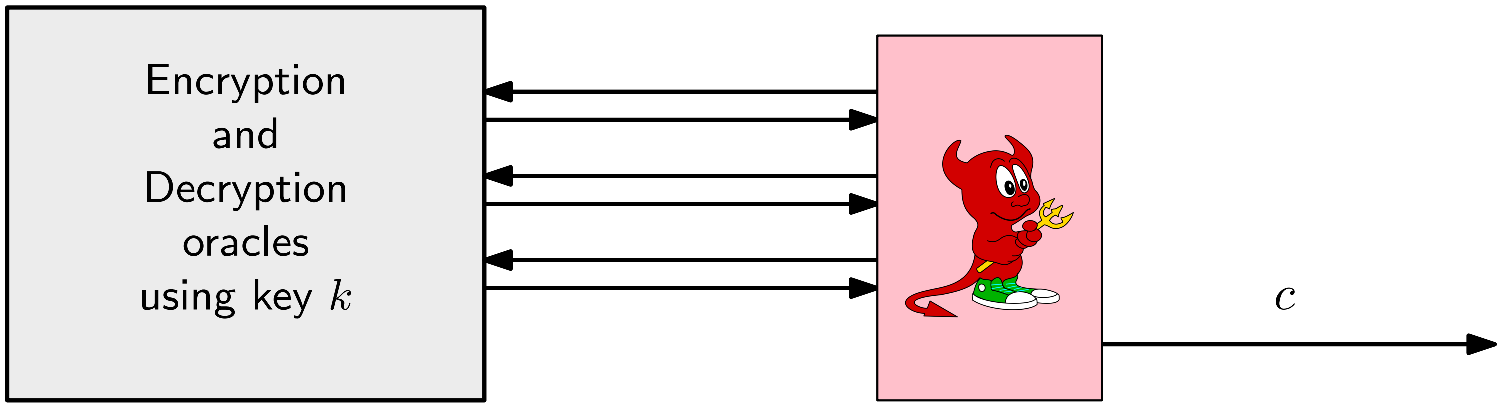
- A key k is generated using $\mathsf{Gen}(1^n)$
- The adversary $\mathcal{A}$ can interact with an encryption oracle providing access to $\mathsf{Enc}_k(\cdot)$ and with a decryption oracle providing access to $\mathsf{Dec}_k(\cdot)$



The Unforgeability Experiment (adapted for AE)

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. We name the following experiment $\mathsf{Enc\text{-}forge}_{\mathcal{A},\Pi}(n)$:

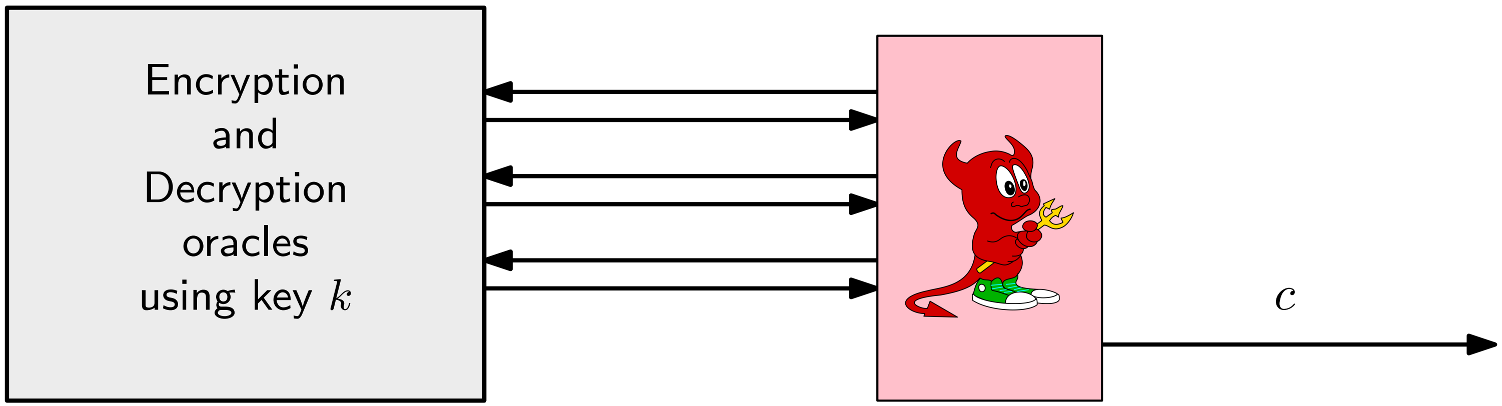
- A key k is generated using $\mathsf{Gen}(1^n)$
- The adversary $\mathcal{A}$ can interact with an encryption oracle providing access to $\mathsf{Enc}_k(\cdot)$ and with a decryption oracle providing access to $\mathsf{Dec}_k(\cdot)$
- The adversary outputs a ciphertext c



The Unforgeability Experiment (adapted for AE)

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. We name the following experiment $\mathsf{Enc\text{-}forge}_{\mathcal{A},\Pi}(n)$:

- A key k is generated using $\mathsf{Gen}(1^n)$
- The adversary $\mathcal{A}$ can interact with an encryption oracle providing access to $\mathsf{Enc}_k(\cdot)$ and with a decryption oracle providing access to $\mathsf{Dec}_k(\cdot)$
- The adversary outputs a ciphertext c
- Let $m \leftarrow \mathsf{Dec}_k(c)$
- The outcome of the experiment is 1 if $m \neq \perp$ and the adversary never queried the encryption oracle with m. Otherwise the outcome is 0.



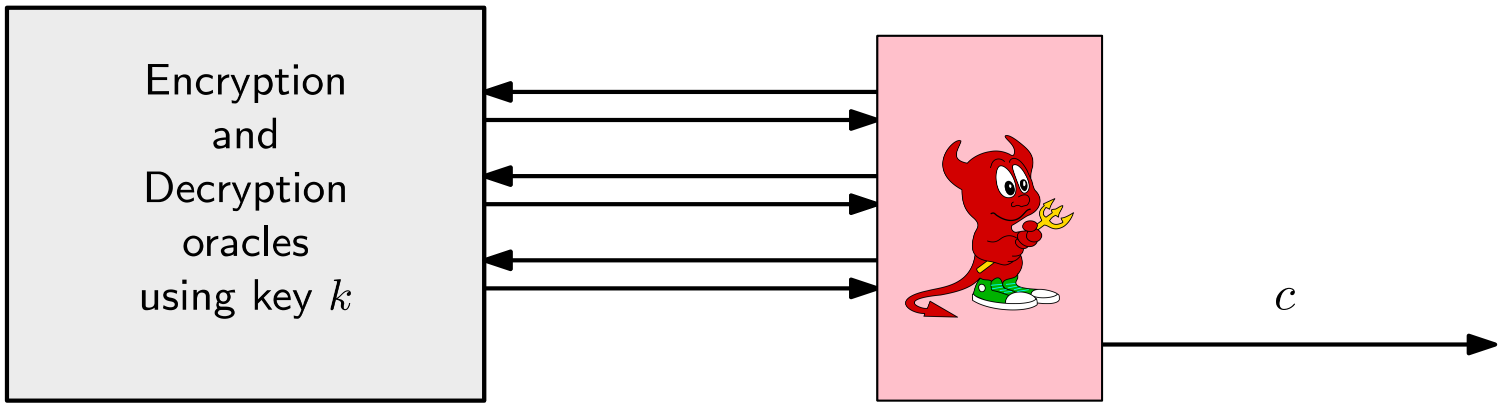
The Unforgeability Experiment (adapted for AE)

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme. We name the following experiment $\mathsf{Enc\text{-}forge}_{\mathcal{A},\Pi}(n)$:

- A key k is generated using $\mathsf{Gen}(1^n)$

The key is kept secret from $\mathcal{A}$

- The adversary $\mathcal{A}$ can interact with an encryption oracle providing access to $\mathsf{Enc}_k(\cdot)$ and with a decryption oracle providing access to $\mathsf{Dec}_k(\cdot)$
- The adversary outputs a ciphertext c
- Let $m \leftarrow \mathsf{Dec}_k(c)$
- The outcome of the experiment is 1 if $m \neq \perp$ and the adversary never queried the encryption oracle with m. Otherwise the outcome is 0.



Definition of Authenticated Encryption

Definition: *A private key encryption scheme* Π *is* **unforgeable** *if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ε *such that:*

$$\Pr[\textit{Enc-forge}_{\mathcal{A},\Pi}(n) = 1] \leq \varepsilon(n)$$

Definition of Authenticated Encryption

Definition: *A private key encryption scheme* Π *is* **unforgeable** *if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ε *such that:*

$$\Pr[\textit{Enc-forge}_{\mathcal{A},\Pi}(n) = 1] \leq \varepsilon(n)$$

Definition: *A private-key encryption scheme is an* **authenticated encryption (AE)** *scheme if it is CCA-secure and unforgeable.*

There is also an equivalent definition of authenticated encryption based on a single experiment. See the textbook if interested.

Definition of Authenticated Encryption

Definition: *A private key encryption scheme* Π *is* **unforgeable** *if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ε *such that:*

$$\Pr[\textit{Enc-forge}_{\mathcal{A},\Pi}(n) = 1] \leq \varepsilon(n)$$

Definition: *A private-key encryption scheme is an* **authenticated encryption (AE)** *scheme if it is CCA-secure and unforgeable.*

Notice that AE $\implies$ **CCA-security**

There is also an equivalent definition of authenticated encryption based on a single experiment. See the textbook if interested.

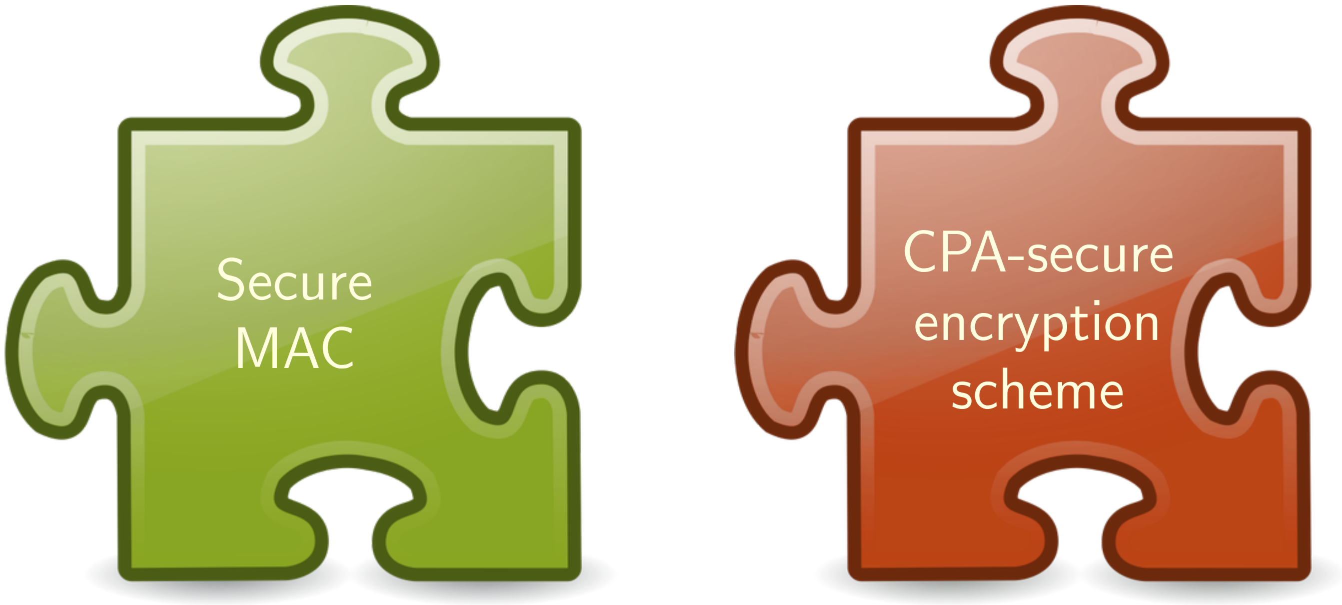
Modular Construction of Authenticated Encryption schemes

Can we design an Authenticated Encryption scheme in a modular way?

Modular Construction of Authenticated Encryption schemes

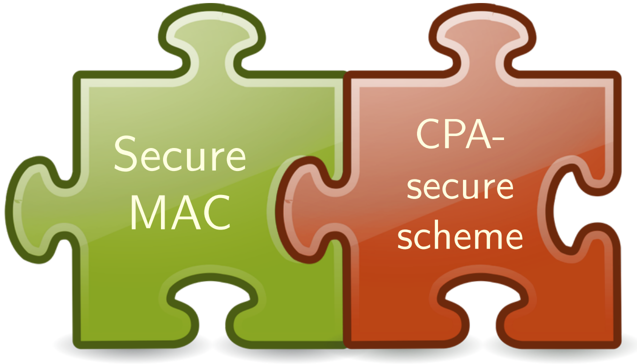
Can we design an Authenticated Encryption scheme in a modular way?

- Pick any secure MAC
- Pick any CPA-secure encryption scheme
- Combine them (somehow)



Combining MACs and CPA-secure encryption schemes

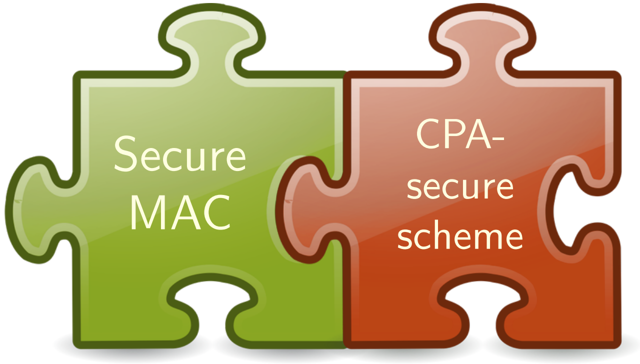
How do we combine MACs with CPA-secure encryption schemes?



Ideas?

Combining MACs and CPA-secure encryption schemes

How do we combine MACs with CPA-secure encryption schemes?



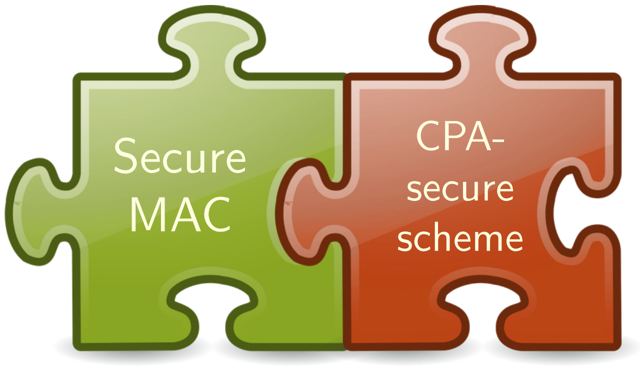
Ideas?

Three natural choices:

- Encrypt **and** Authenticate

Combining MACs and CPA-secure encryption schemes

How do we combine MACs with CPA-secure encryption schemes?



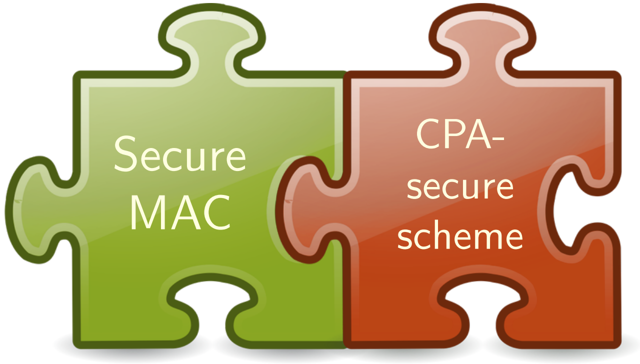
Ideas?

Three natural choices:

- Encrypt **and** Authenticate
- Authenticate **then** Encrypt

Combining MACs and CPA-secure encryption schemes

How do we combine MACs with CPA-secure encryption schemes?



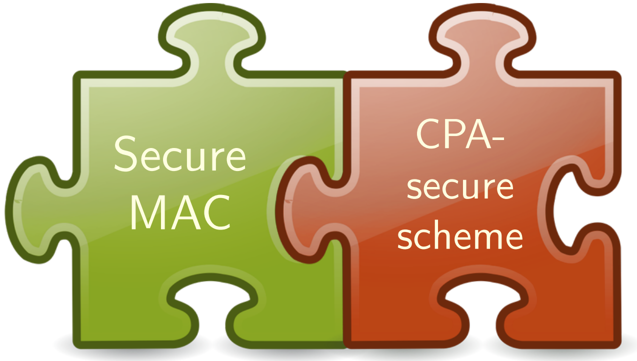
Ideas?

Three natural choices:

- Encrypt **and** Authenticate
- Authenticate **then** Encrypt
- Encrypt **then** Authenticate

Combining MACs and CPA-secure encryption schemes

How do we combine MACs with CPA-secure encryption schemes?



Ideas?

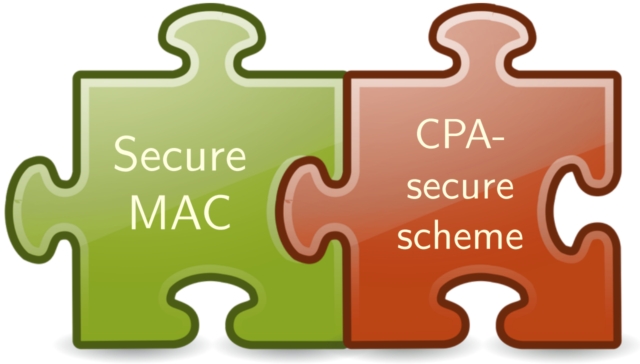
Three natural choices:

- Encrypt **and** Authenticate
- Authenticate **then** Encrypt
- Encrypt **then** Authenticate

How good are these choices?

Combining MACs and CPA-secure encryption schemes

How do we combine MACs with CPA-secure encryption schemes?



Ideas?

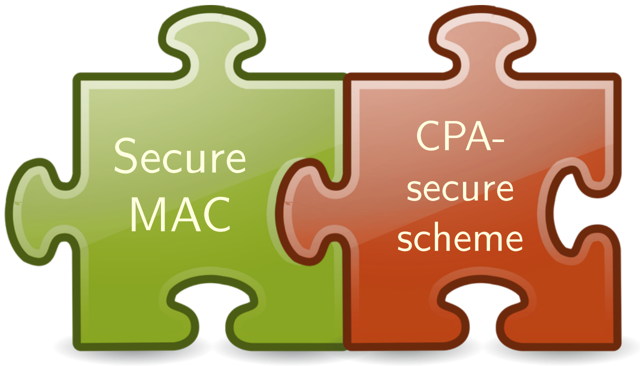
Three natural choices:

- Encrypt **and** Authenticate ☹ Very bad!
- Authenticate **then** Encrypt
- Encrypt **then** Authenticate

How good are these choices?

Combining MACs and CPA-secure encryption schemes

How do we combine MACs with CPA-secure encryption schemes?



Ideas?

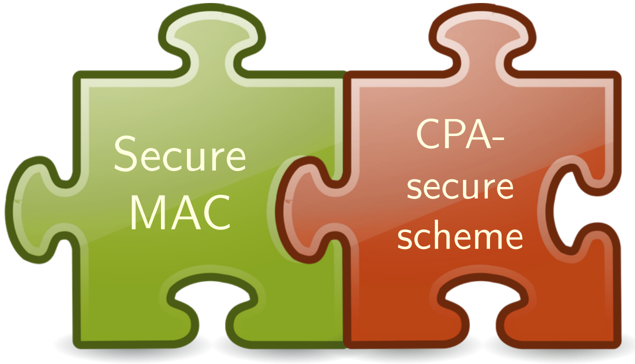
Three natural choices:

- Encrypt **and** Authenticate ☹ Very bad!
- Authenticate **then** Encrypt 😐 Still bad
- Encrypt **then** Authenticate

How good are these choices?

Combining MACs and CPA-secure encryption schemes

How do we combine MACs with CPA-secure encryption schemes?



Ideas?

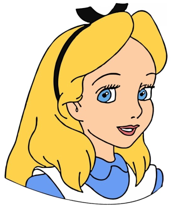
Three natural choices:

- Encrypt **and** Authenticate ☹ Very bad!
- Authenticate **then** Encrypt 😐 Still bad
- Encrypt **then** Authenticate ☺ Good!

How good are these choices?

Encrypt and Authenticate

Pick two **independent** keys k_1 and k_2 for encryption and MAC, respectively

k_1, k_2 

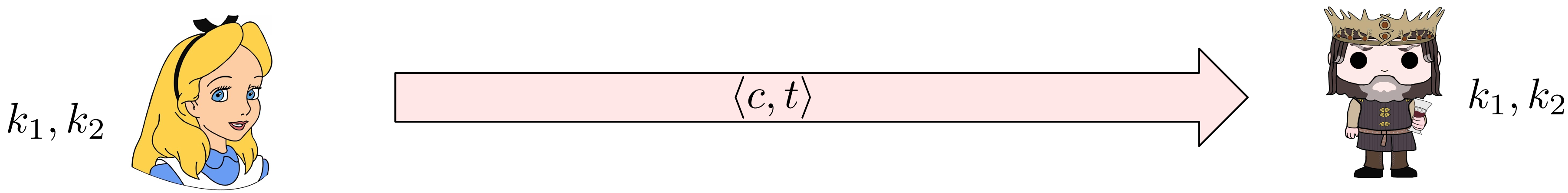
 k_1, k_2

Encrypt and Authenticate

Pick two **independent** keys k_1 and k_2 for encryption and MAC, respectively

Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- Return the ciphertext $\langle c, t \rangle$



Encrypt and Authenticate

Pick two **independent** keys k_1 and k_2 for encryption and MAC, respectively

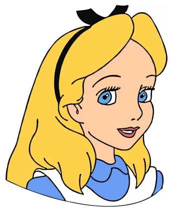
Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- Return the ciphertext $\langle c, t \rangle$

Decrypting $\langle c, t \rangle$:

- $m \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
  - Return m
- Otherwise return $\perp$

k_1, k_2



$\langle c, t \rangle$

k_1, k_2

Encrypt and Authenticate

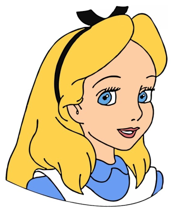
Pick two **independent** keys k_1 and k_2 for encryption and MAC, respectively

Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- Return the ciphertext $\langle c, t \rangle$

Decrypting $\langle c, t \rangle$:

- $m \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
  - Return m
- Otherwise return $\perp$

k_1, k_2 

$\langle c, t \rangle$ →

 k_1, k_2

Problems?

Encrypt and Authenticate

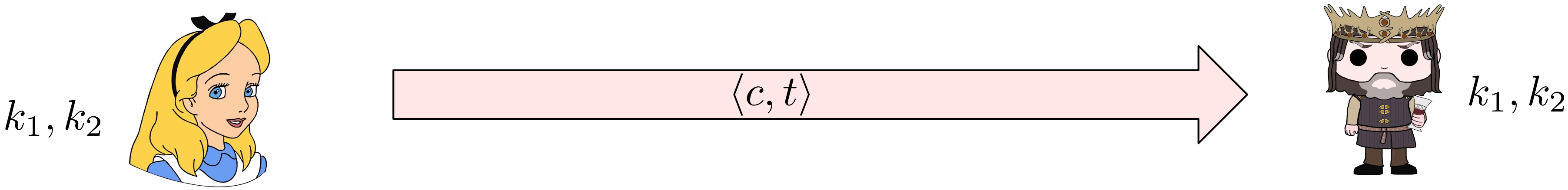
Pick two **independent** keys k_1 and k_2 for encryption and MAC, respectively

Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- Return the ciphertext $\langle c, t \rangle$

Decrypting $\langle c, t \rangle$:

- $m \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
  - Return m
- Otherwise return $\perp$



Problems?

- The tag t is not required to hide any information about m
- Consider the tag obtained by concatenating the first bit of the message with $\mathsf{Mac}_{k_2}(m)$

Encrypt and Authenticate

Pick two **independent** keys k_1 and k_2 for encryption and MAC, respectively

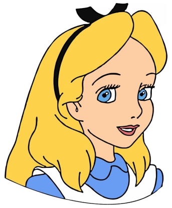
Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- Return the ciphertext $\langle c, t \rangle$

Decrypting $\langle c, t \rangle$:

- $m \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
  - Return m
- Otherwise return $\perp$

k_1, k_2



$\langle c, t \rangle$

k_1, k_2

Problems?

- The tag t is not required to hide any information about m
- Consider the tag obtained by concatenating the first bit of the message with $\mathsf{Mac}_{k_2}(m)$
- If the Mac function is deterministic, this scheme leaks whether the same message is encrypted twice

Encrypt and Authenticate

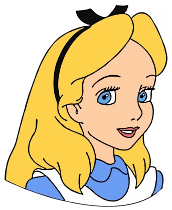
Pick two **independent** keys k_1 and k_2 for encryption and MAC, respectively

Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- Return the ciphertext $\langle c, t \rangle$

Decrypting $\langle c, t \rangle$:

- $m \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
  - Return m
- Otherwise return $\perp$

k_1, k_2  $\langle c, t \rangle$ →  k_1, k_2

Problems?

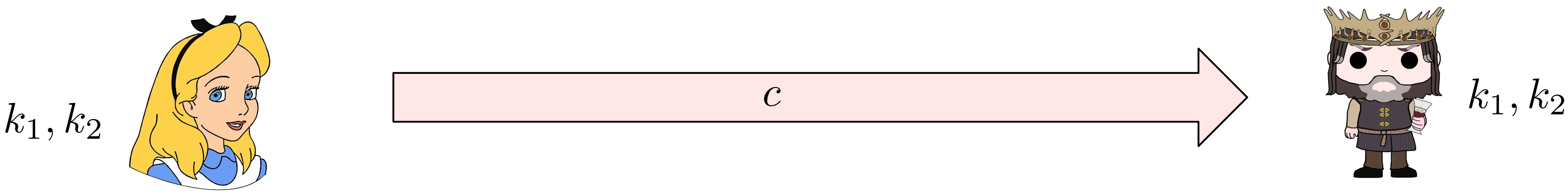
- The tag t is not required to hide any information about m
- Consider the tag obtained by concatenating the first bit of the message with $\mathsf{Mac}_{k_2}(m)$
- If the Mac function is deterministic, this scheme leaks whether the same message is encrypted twice

This scheme is not even CPA-secure!

Authenticate then Encrypt

Encrypting m:

- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- $c \leftarrow \mathsf{Enc}_{k_1}(m \,\|\, t)$
- Return the ciphertext c



Authenticate then Encrypt

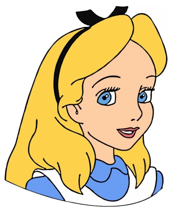
Encrypting m:

- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- $c \leftarrow \mathsf{Enc}_{k_1}(m \,\|\, t)$
- Return the ciphertext c

Decrypting c:

- $m \,\|\, t \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
    - Return m
- Otherwise return $\perp$

k_1, k_2



c

k_1, k_2

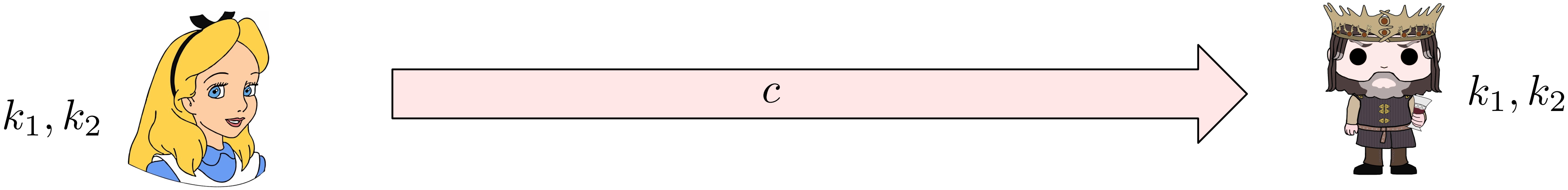
Authenticate then Encrypt

Encrypting m:

- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- $c \leftarrow \mathsf{Enc}_{k_1}(m \,\|\, t)$
- Return the ciphertext c

Decrypting c:

- $m \,\|\, t \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
  - Return m
- Otherwise return $\perp$



Problems?

Authenticate then Encrypt

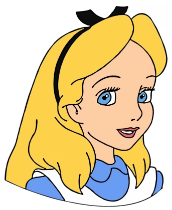
Encrypting m:

- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- $c \leftarrow \mathsf{Enc}_{k_1}(m \,\|\, t)$
- Return the ciphertext c

Decrypting c:

- $m \,\|\, t \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
  - Return m
- Otherwise return $\perp$

k_1, k_2



c →

k_1, k_2

Problems?

- If encryption requires padding and the padding is wrong, an error can be raised by $\mathsf{Dec}_{k_1}(c)$

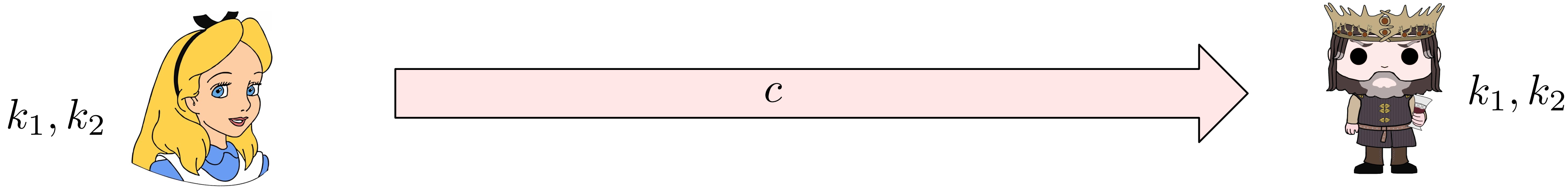
Authenticate then Encrypt

Encrypting m:

- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- $c \leftarrow \mathsf{Enc}_{k_1}(m \,\|\, t)$
- Return the ciphertext c

Decrypting c:

- $m \,\|\, t \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
  - Return m
- Otherwise return $\perp$



Problems?

- If encryption requires padding and the padding is wrong, an error can be raised by $\mathsf{Dec}_{k_1}(c)$
- If the encryption scheme is malleable and the padding error is leaked to the user, the adversary has a **padding oracle**

Authenticate then Encrypt

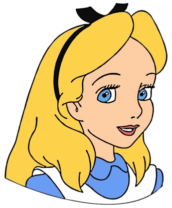
Encrypting m:

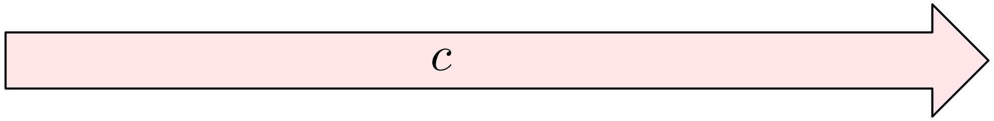
- $t \leftarrow \mathsf{Mac}_{k_2}(m)$
- $c \leftarrow \mathsf{Enc}_{k_1}(m \,\|\, t)$
- Return the ciphertext c

Decrypting c:

- $m \,\|\, t \leftarrow \mathsf{Dec}_{k_1}(c)$
- If $\mathsf{Vrfy}_{k_2}(m, t) = 1$:
    - Return m
- Otherwise return $\perp$

k_1, k_2





k_1, k_2

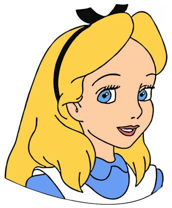
Problems?

- If encryption requires padding and the padding is wrong, an error can be raised by $\mathsf{Dec}_{k_1}(c)$
- If the encryption scheme is malleable and the padding error is leaked to the user, the adversary has a **padding oracle**
- There are other counterexamples that do not rely on padding errors

Encrypt then Authenticate

Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(c)$
- Return the ciphertext $\langle c, t \rangle$

k_1, k_2  $\langle c, t \rangle$ →  k_1, k_2

Encrypt then Authenticate

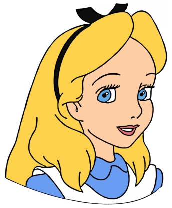
Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(c)$
- Return the ciphertext $\langle c, t \rangle$

Decrypting $\langle c, t \rangle$:

- If $\mathsf{Vrfy}_{k_2}(c, t) = 1$:
  - $m \leftarrow \mathsf{Dec}_{k_1}(c)$
  - Return m
- Otherwise return $\perp$

k_1, k_2



$\langle c, t \rangle$

k_1, k_2

Encrypt then Authenticate

Encrypting m:

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(c)$
- Return the ciphertext $\langle c, t \rangle$

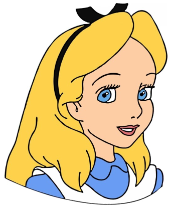
Keys (w. security parameter n):

- $k_1 \leftarrow \mathsf{Gen}_E(1^n)$
- $k_2 \leftarrow \mathsf{Gen}_M(1^n)$
- Return $k_1 \| k_2$

Decrypting $\langle c, t \rangle$:

- If $\mathsf{Vrfy}_{k_2}(c, t) = 1$:
  - $m \leftarrow \mathsf{Dec}_{k_1}(c)$
  - Return m
- Otherwise return $\perp$

k_1, k_2



$\langle c, t \rangle$ →

k_1, k_2

Encrypt then Authenticate

Encrypting m**:**

- $c \leftarrow \mathsf{Enc}_{k_1}(m)$
- $t \leftarrow \mathsf{Mac}_{k_2}(c)$
- Return the ciphertext $\langle c, t \rangle$

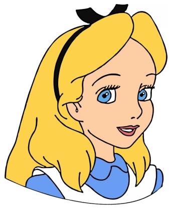
Keys (w. security parameter n):

- $k_1 \leftarrow \mathsf{Gen}_E(1^n)$
- $k_2 \leftarrow \mathsf{Gen}_M(1^n)$
- Return $k_1 \,\|\, k_2$

Decrypting $\langle c, t \rangle$**:**

- If $\mathsf{Vrfy}_{k_2}(c, t) = 1$:
  - $m \leftarrow \mathsf{Dec}_{k_1}(c)$
  - Return m
- Otherwise return $\perp$

k_1, k_2



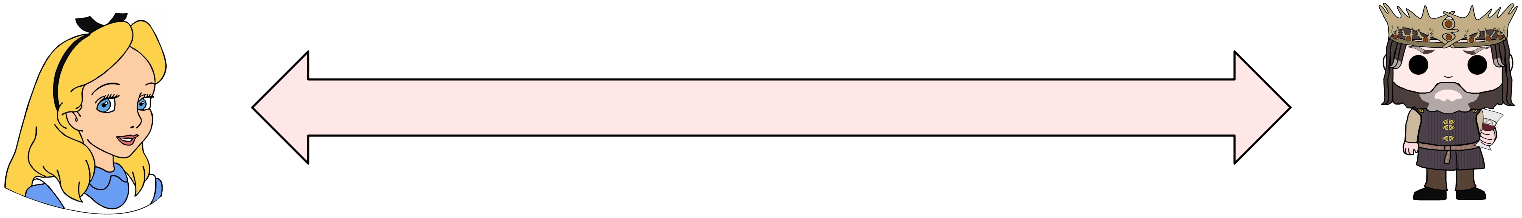
$\langle c, t \rangle$

k_1, k_2

Theorem: If $(\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure private-key encryption scheme, and $(\mathsf{Gen}_M, \mathsf{Mac}, \mathsf{Vrfy})$ is a strongly secure message authentication code, then the above construction is an authenticated encryption scheme.

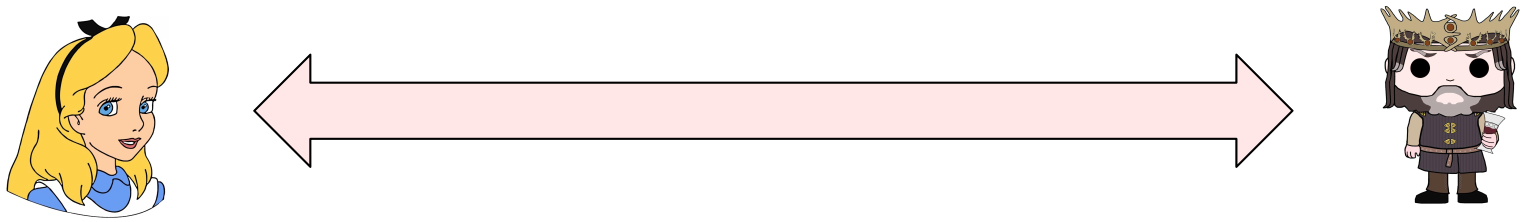
Secure Sessions

Alice and Bob wish to communicate securely (over an insecure channel) over the course of a *communication session* (a period of time over which they maintain state) exchanging multiple messages



Secure Sessions

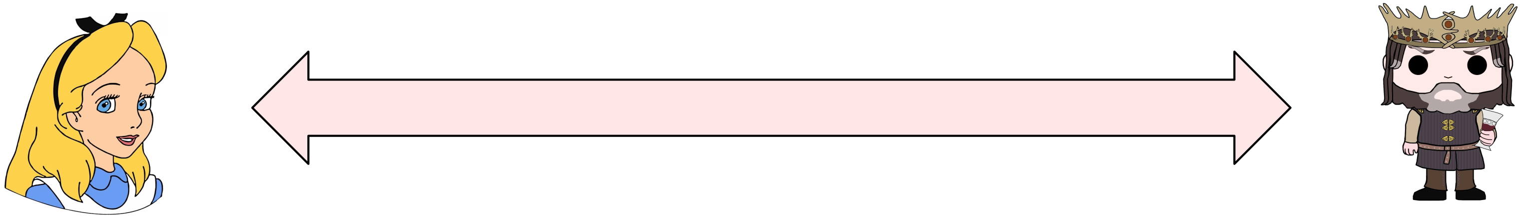
Alice and Bob wish to communicate securely (over an insecure channel) over the course of a *communication session* (a period of time over which they maintain state) exchanging multiple messages



Easy! Just use Authenticated Encryption

Secure Sessions

Alice and Bob wish to communicate securely (over an insecure channel) over the course of a *communication session* (a period of time over which they maintain state) exchanging multiple messages

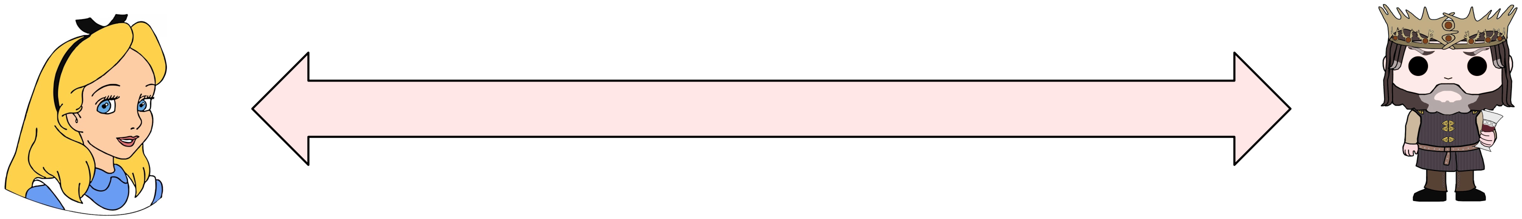


~~Easy! Just use Authenticated Encryption~~

Not so fast...

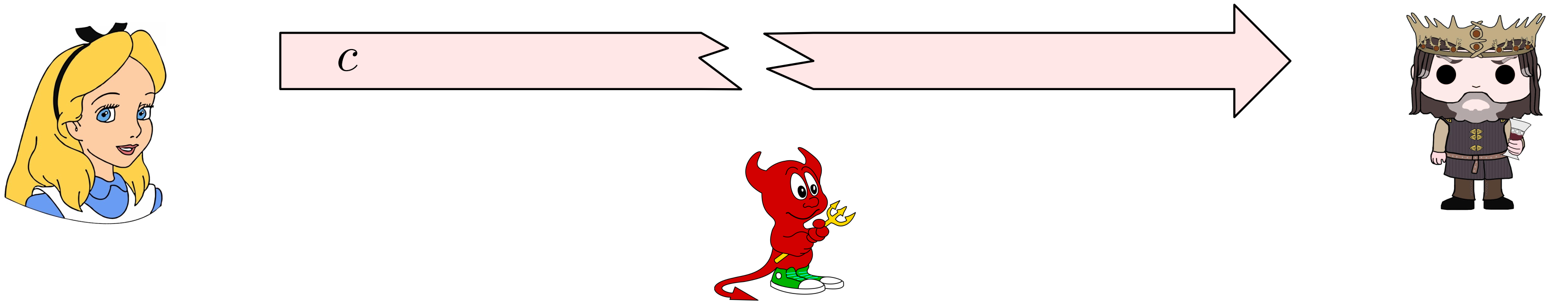
Secure Sessions

Alice and Bob wish to communicate securely (over an insecure channel) over the course of a *communication session* (a period of time over which they maintain state) exchanging multiple messages



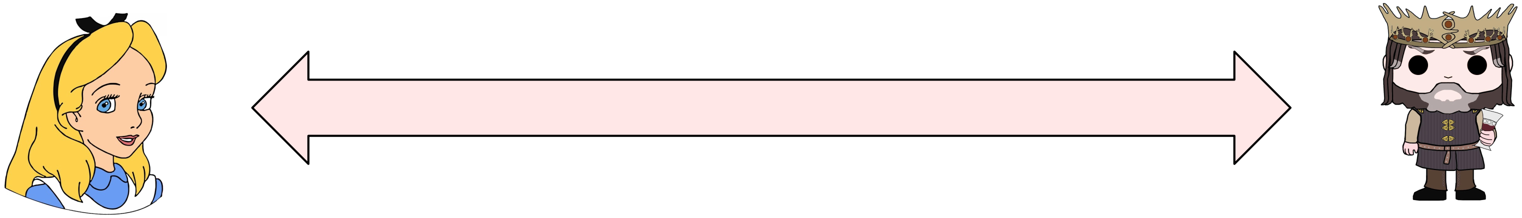
~~Easy! Just use Authenticated Encryption~~ Not so fast...

Message dropping:



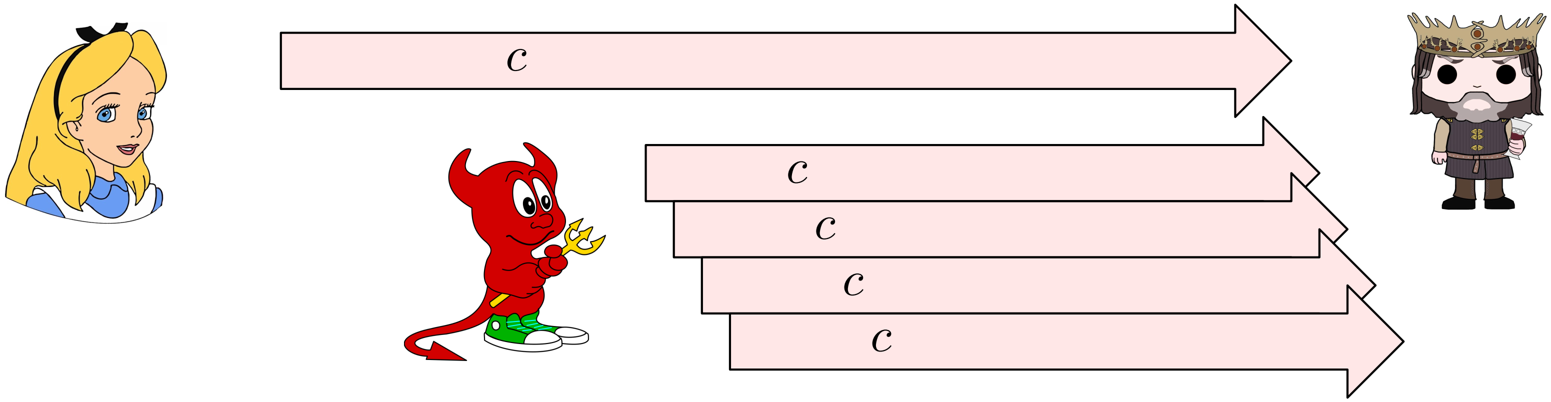
Secure Sessions

Alice and Bob wish to communicate securely (over an insecure channel) over the course of a *communication session* (a period of time over which they maintain state) exchanging multiple messages



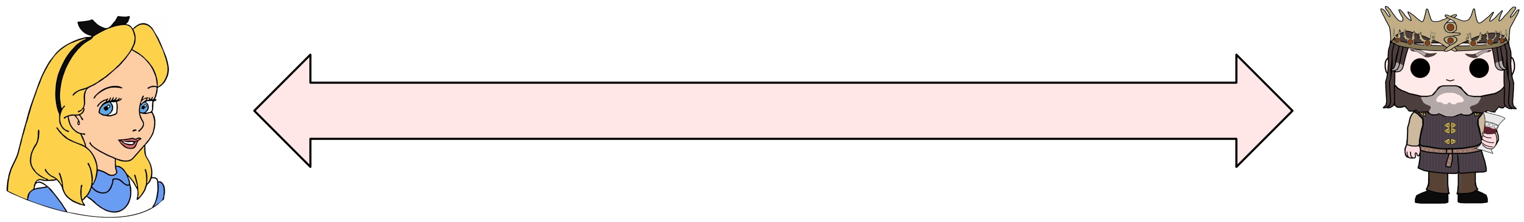
~~Easy! Just use Authenticated Encryption~~ Not so fast...

Replay attack (we have already encountered this attack):



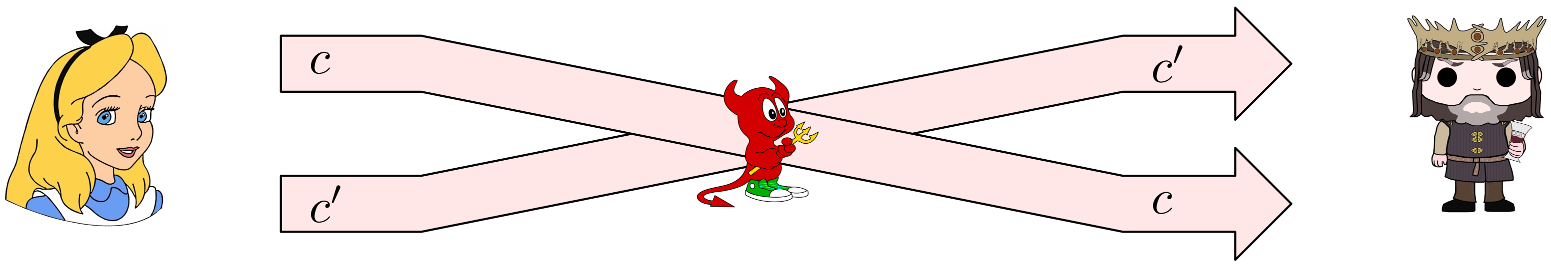
Secure Sessions

Alice and Bob wish to communicate securely (over an insecure channel) over the course of a *communication session* (a period of time over which they maintain state) exchanging multiple messages



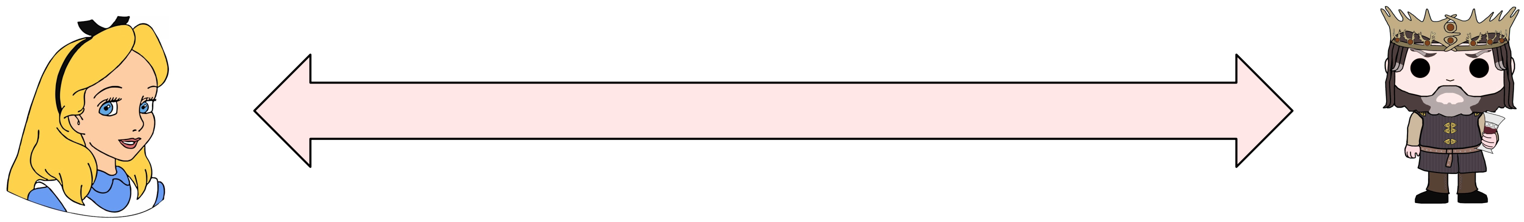
~~Easy! Just use Authenticated Encryption~~ Not so fast...

Re-ordering attack (the adversary reorders messages, not blocks):



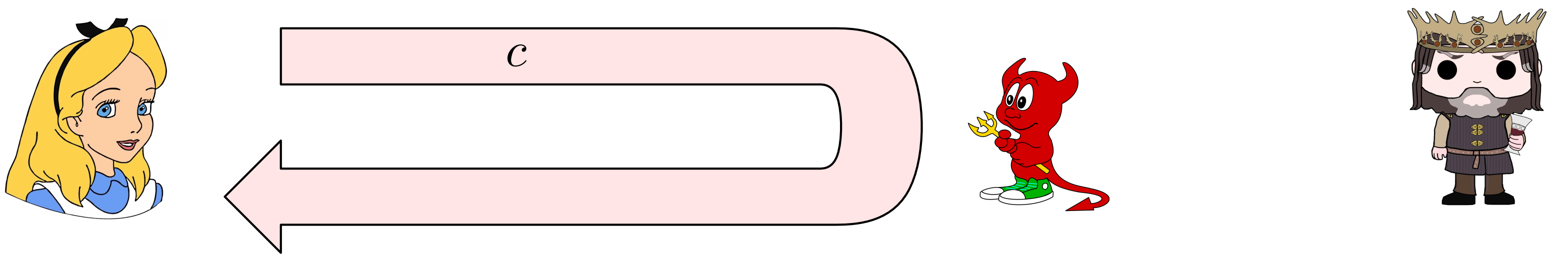
Secure Sessions

Alice and Bob wish to communicate securely (over an insecure channel) over the course of a *communication session* (a period of time over which they maintain state) exchanging multiple messages



~~Easy! Just use Authenticated Encryption~~ Not so fast...

Reflection attack



Secure Sessions

How do we defend from these attacks?

Message dropping, **Replay attacks** and **Re-ordering attacks**:

- Send a counter along with each message
- The recipient checks that the received counters are consecutive numbers
- Message dropping cannot be prevented, but we can at least detect it if a subsequent message reaches the recipient

Secure Sessions

How do we defend from these attacks?

Message dropping, **Replay attacks** and **Re-ordering attacks**:

- Send a counter along with each message
- The recipient checks that the received counters are consecutive numbers
- Message dropping cannot be prevented, but we can at least detect it if a subsequent message reaches the recipient

Reflection attack

- Add a directionality bit d to each message
- E.g., $d = 0$ if the message is sent from Alice to Bob and $d = 1$ if the message is sent from Bob to Alice
- Need to agree on direction. E.g., in a client/server connection we might assign $d = 0$ to the client and $d = 1$ to the server

Secure Sessions