COMS BC3262: Introduction to Cryptography

Lecture 9: CCA-Security

February 18, 2026

Logistics

Office hours:

- **Eysa**: Today 3-5 (Normally Mondays 3-5), Milstein 512
- **Mark**: Normally Tuesdays 6:30-8:30, but he is traveling these next two weeks.
    - *No office hours Tuesday, Feb 17 or Tuesday, Feb 24*
    - *Mark's next two office hours are tentatively set for Sunday via Zoom, time TBD*

PS2 is due tomorrow

Each late day is 10% off, can be submitted up to 3 days late

Today's Lecture

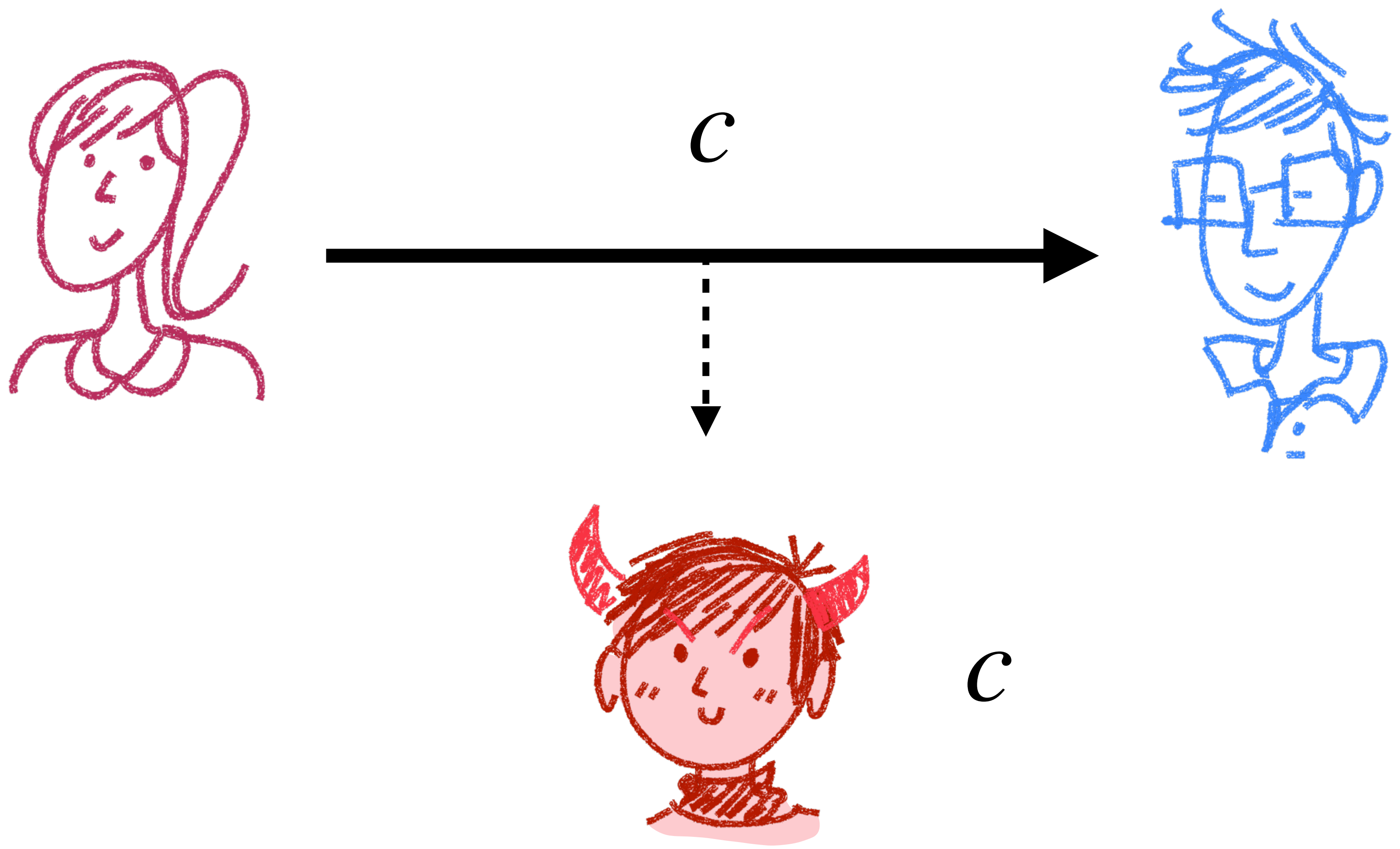
- Message Authentication Codes (MACs)
  - Arbitrary-length MACs
- Secrecy Against an Active Adversary
  - CCA-Security
  - Padding Oracle Attack
- Authenticated Encryption

Message Authenticity

Encryption vs Authentication

Orthogonal aspects; in general, one does not guarantee the other

- **Encryption** focuses on data **secrecy**
    - Hiding the contents of the message from an adversary
- **Authentication** focuses on data **integrity**
    - Assuring a receiver that a message has not been modified



Message Authentication Codes (MACs)

Syntax: Three algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$

- **Key generation** algorithm Gen takes input 1^n and outputs a key k
- **Tag generation** algorithm Mac takes a key k and a message $m \in \{0,1\}^*$ and outputs a tag $t \in \{0,1\}^*$
- **Verification** algorithm Verify takes a key k, a message m, a tag t, and outputs a bit b

m

$t \leftarrow \mathsf{Mac}_k(m)$

$\xrightarrow{(m,t)}$

$b = \mathsf{Verify}_k(m,t)$

Message Authentication Codes (MACs)

Syntax: Three algorithms (Gen, Mac, Verify)

- **Key generation** algorithm Gen takes input 1^n and outputs a key k
- **Tag generation** algorithm Mac takes a key k and a message $m \in \{0,1\}^*$ and outputs a tag $t \in \{0,1\}^*$
- **Verification** algorithm Verify takes a key k, a message m, a tag t, and outputs a bit b

Correctness: $\forall n, \forall k$ output by Gen(1^n), $\forall m \in \{0,1\}^*$, $\forall t$ output by Mac$_k(m)$,

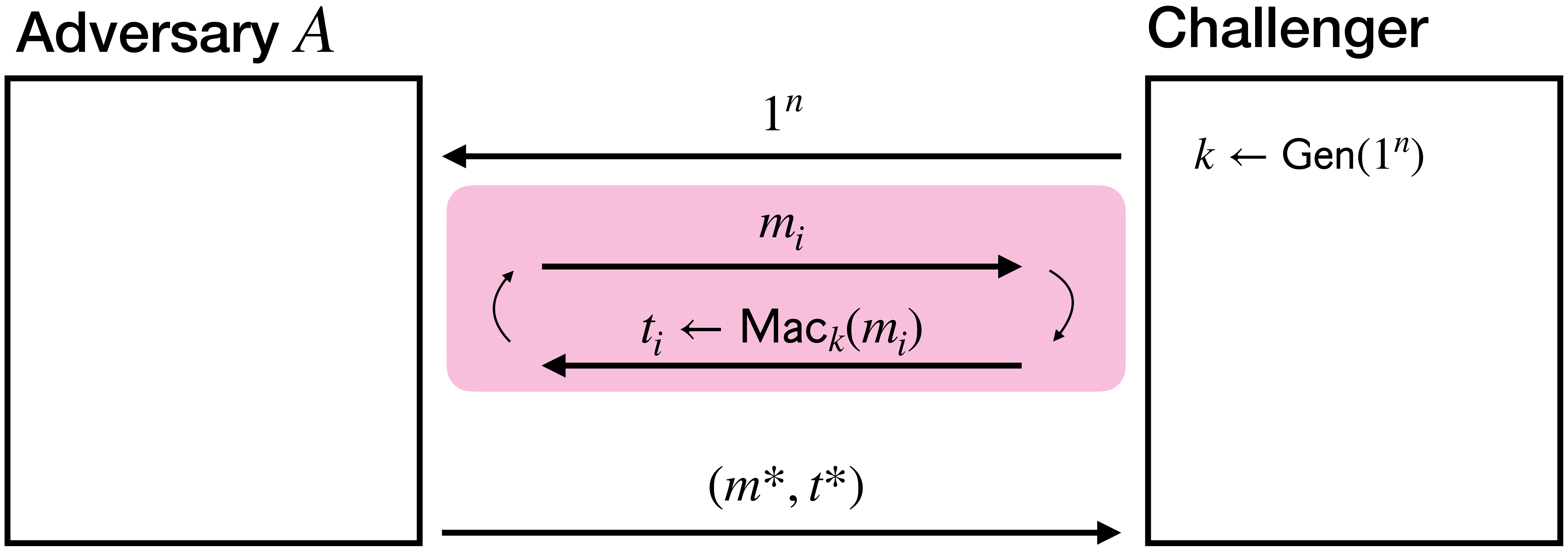
Verify$_k(m, t) = 1$

Canonical Verification:
If Mac algorithm is deterministic, then Verify just checks if
Mac$_k(m) = t$

$b =$ Verify$_k(m, t)$

MAC Security

Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$. We define $\mathsf{MacForge}_{\mathscr{A},\Pi}(n)$ as follows

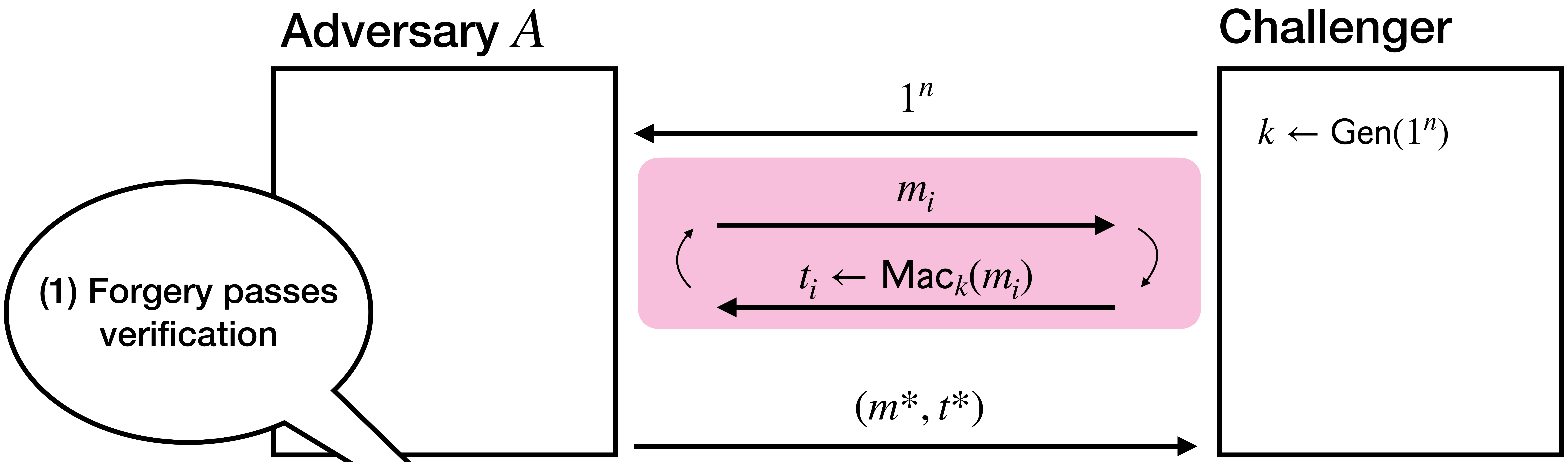


We say the adversary succeeds ($\mathsf{MacForge}_{\mathscr{A},\Pi}(n) = 1$) if:

1. $\mathsf{Verify}_k(m^*, t^*) = 1$
2. $m^* \neq m_i$ for all queried m_i

MAC Security

Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$. We define $\mathsf{MacForge}_{\mathscr{A},\Pi}(n)$ as follows



We say the adversary succeeds (MacFor

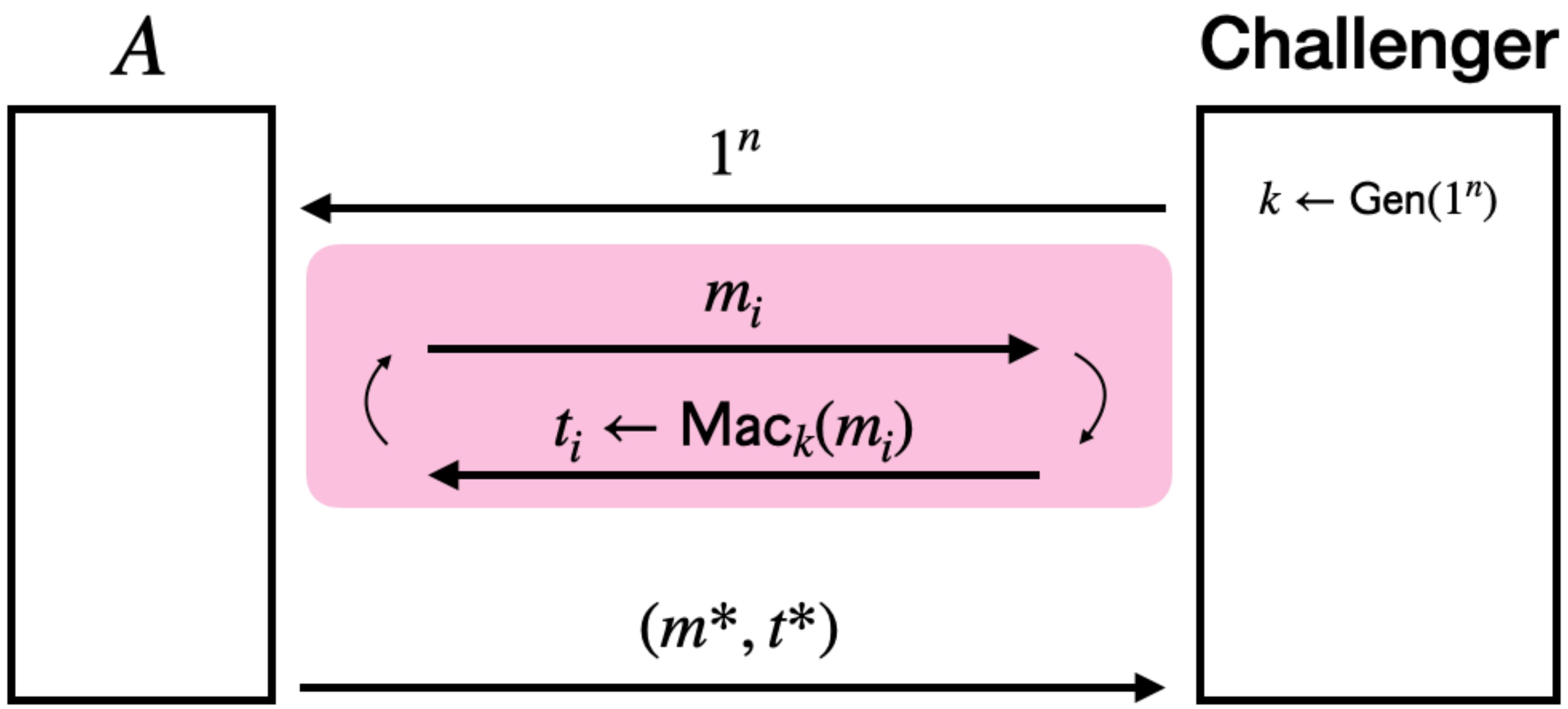
1. $\mathsf{Verify}_k(m^*, t^*) = 1$
2. $m^* \neq m_i$ for all queried m_i

(2) Forgery is on a new message

MAC Security

Definition: A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is existentially unforgeable under an adaptive chosen-message attack if for every PPT adversary A there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{MacForge}_{\mathscr{A},\Pi}(n) = 1] \leq \mathsf{negl}(n)$$



$\mathsf{MacForge}_{\mathscr{A},\Pi}(n) = 1$ if:

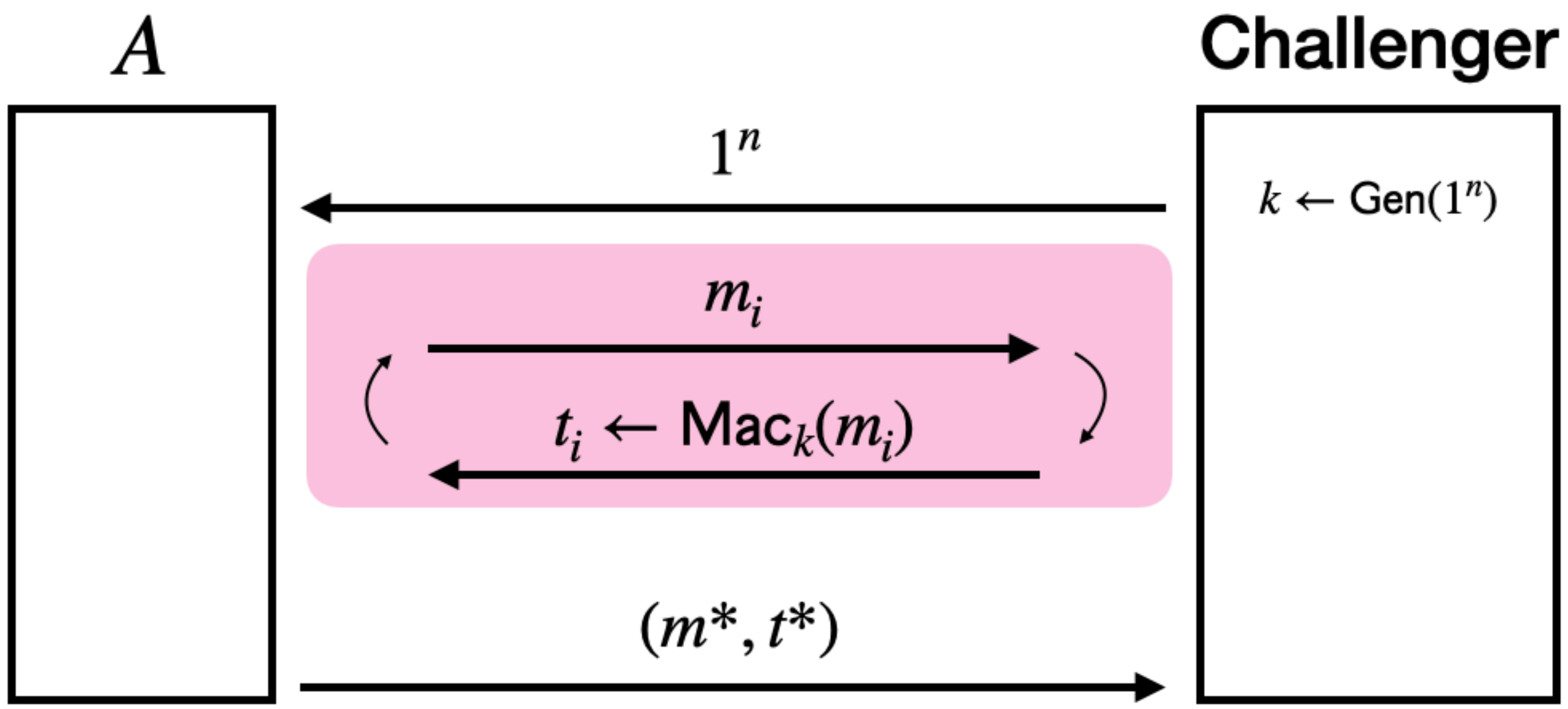
1. $\mathsf{Verify}_k(m^*, t^*) = 1$
2. $m^* \neq m_i$ for all queried m_i

And is 0 otherwise

Strong MAC Security

Definition: A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is strongly secure if for every PPT adversary A there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{MacSForge}_{\mathscr{A},\Pi}(n) = 1] \leq \mathsf{negl}(n)$$



$\mathsf{MacSForge}_{\mathscr{A},\Pi}(n) = 1$ if:

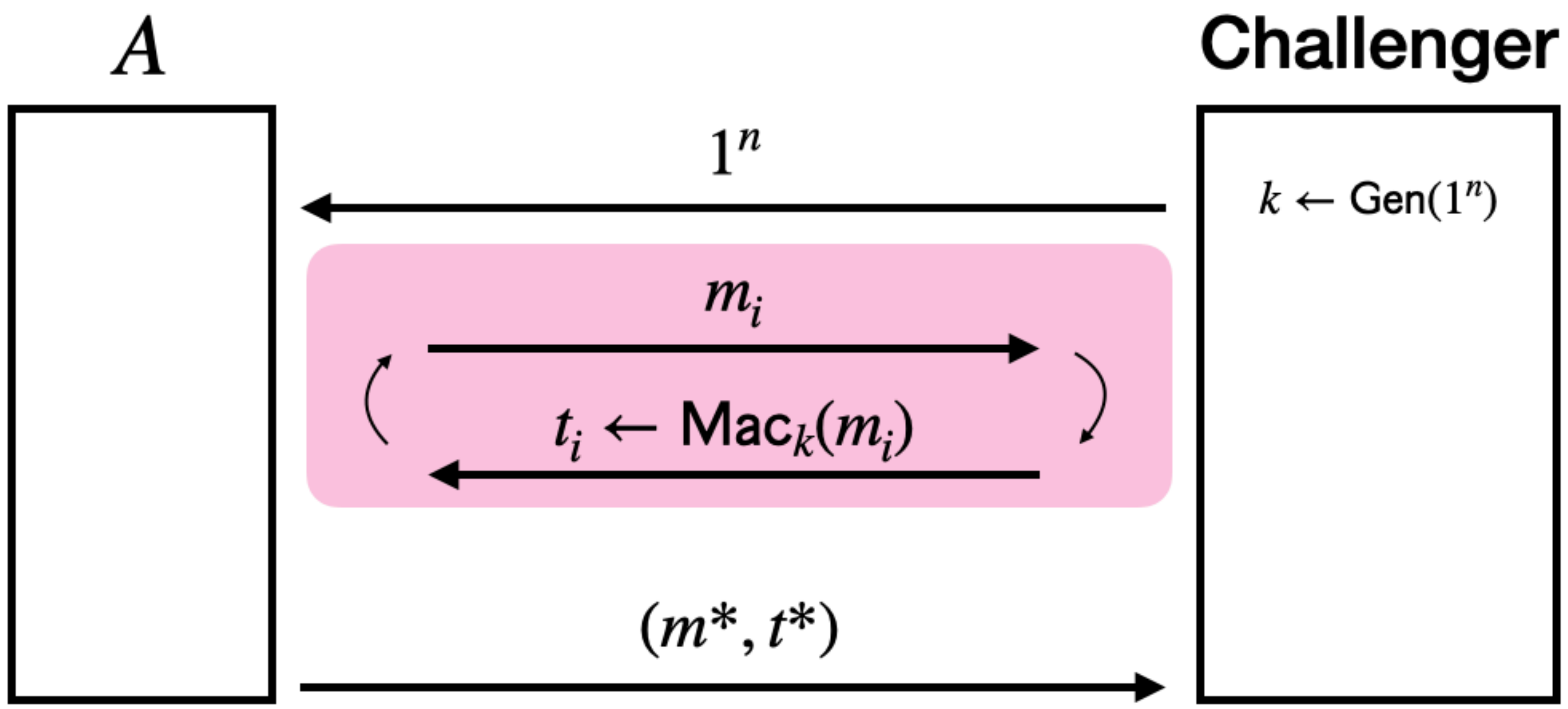
1. $\mathsf{Verify}_k(m^*, t^*) = 1$
2. $(m^*, t^*) \neq (m_i, t_i)$ for all queried m_i and response t_i

And is 0 otherwise

Strong MAC Security

Definition: A MAC $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is strongly secure if for every PPT adversary A there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr[\mathsf{MacSForge}_{\mathscr{A},\Pi}(n) = 1] \leq \mathsf{negl}(n)$$



$\mathsf{MacSForge}_{\mathscr{A},\Pi}(n) = 1$ if:

1. $\mathsf{Verify}_k(m^*, t^*) = 1$
2. $(m^*, t^*) \neq (m_i, t_i)$ for all queried m_i and response t_i

And is 0 otherwise

Theorem: A secure MAC that uses canonical verification is a strong MAC

A Fixed-Length MAC

Let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a PRF

We can construct a MAC as follows:

- **Key generation:** On input 1^n, output a randomly sampled $k \leftarrow \{0,1\}^n$
- **Tag generation:** On input $k \in \{0,1\}^n$ and $m \in \{0,1\}^n$, output $t = F_k(m)$
- **Verification:** On input $k \in \{0,1\}^n$, $m \in \{0,1\}^n$, and $t \in \{0,1\}^n$, output 1 if $t = F_k(m)$ and 0 otherwise

Theorem: If F is a PRF, then the above MAC scheme is secure for messages of length n

Arbitrary-Length MACs

Authenticating Arbitrary-Length Messages

m	$=$	m_1	m_2		$\ldots$		$\ldots$		m_d

Suppose we had a $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ for fixed-length messages.

Can we construct a $(\widehat{\mathsf{Gen}}, \widehat{\mathsf{Mac}}, \widehat{\mathsf{Verify}})$ for arbitrary-length messages?

Authenticating Arbitrary-Length Messages

m	$=$	m_1	m_2		$\dots$		$\dots$		m_d

Suppose we had a $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ for fixed-length messages.

Can we construct a $(\hat{\mathsf{Gen}}, \hat{\mathsf{Mac}}, \hat{\mathsf{Verify}})$ for arbitrary-length messages?

Idea 1: Authenticate each block on its own

$$\hat{\mathsf{Mac}}_k(m_1 || m_2 || \dots || m_d) = \mathsf{Mac}_k(m_1) || \dots || \mathsf{Mac}_k(m_d)$$

Authenticating Arbitrary-Length Messages

$$m \quad = \quad \boxed{m_1} \boxed{m_2} \boxed{} \boxed{\dots} \boxed{} \boxed{\dots} \boxed{} \boxed{m_d}$$

Suppose we had a $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ for fixed-length messages.

Can we construct a $(\hat{\mathsf{Gen}}, \hat{\mathsf{Mac}}, \hat{\mathsf{Verify}})$ for arbitrary-length messages?

Idea 2: Authenticate each block on its own with an index?

$$\hat{\mathsf{Mac}}_k(m_1 || m_2 || \dots || m_d) = \mathsf{Mac}_k(1 || m_1) || \dots || \mathsf{Mac}_k(d || m_d)$$

Authenticating Arbitrary-Length Messages

$$m \quad = \begin{array}{|c|c|c|c|c|c|c|c|} \hline m_1 & m_2 & & \dots & & \dots & & m_d \\ \hline \end{array}$$

Suppose we had a $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ for fixed-length messages.

Can we construct a $(\hat{\mathsf{Gen}}, \hat{\mathsf{Mac}}, \hat{\mathsf{Verify}})$ for arbitrary-length messages?

Idea 3: Authenticate each block on its own with an index and length?

$$\hat{\mathsf{Mac}}_k(m_1 || m_2 || \dots || m_d) = \mathsf{Mac}_k(1 || d || m_1) || \dots || \mathsf{Mac}_k(d || d || m_d)$$

Authenticating Arbitrary-Length Messages

m	$=$	m_1	m_2		$\ldots$		$\ldots$		m_d

Suppose we had a $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ for fixed-length messages.

Can we construct a $(\hat{\mathsf{Gen}}, \hat{\mathsf{Mac}}, \hat{\mathsf{Verify}})$ for arbitrary-length messages?

Idea 4: Idea 3 but also with a randomly sampled r

$\hat{\mathsf{Mac}}_k(m_1 || m_2 || \ldots || m_d) = \mathsf{Mac}_k(r || 1 || d || m_1) || \ldots || \mathsf{Mac}_k(r || d || d || m_d)$
where r is sampled randomly

Authenticating Arbitrary-Length Messages

$$m \quad = \begin{array}{|c|c|c|c|c|c|c|c|}\hline m_1 & m_2 & & \ldots & & \ldots & & m_d \\ \hline\end{array}$$

Suppose we had a $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ for fixed-length messages.

Can we construct a $(\hat{\mathsf{Gen}}, \hat{\mathsf{Mac}}, \hat{\mathsf{Verify}})$ for arbitrary-length messages?

Idea 4: Idea 3 but also with a randomly sampled r

$\hat{\mathsf{Mac}}_k(m_1 || m_2 || \ldots || m_d) = \mathsf{Mac}_k(r || 1 || d || m_1) || \ldots || \mathsf{Mac}_k(r || d || d || m_d)$ where r is sampled randomly

Theorem: If $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$ is a secure MAC for fixed-length messages, then $(\hat{\mathsf{Gen}}, \hat{\mathsf{Mac}}, \hat{\mathsf{Verify}})$ is a secure MAC for arbitrary-length messages

Smaller Tags: CBC-MAC

$$m = \boxed{m_1 \mid m_2 \mid \quad \mid \ldots \mid \quad \mid \ldots \mid \quad \mid m_d}$$

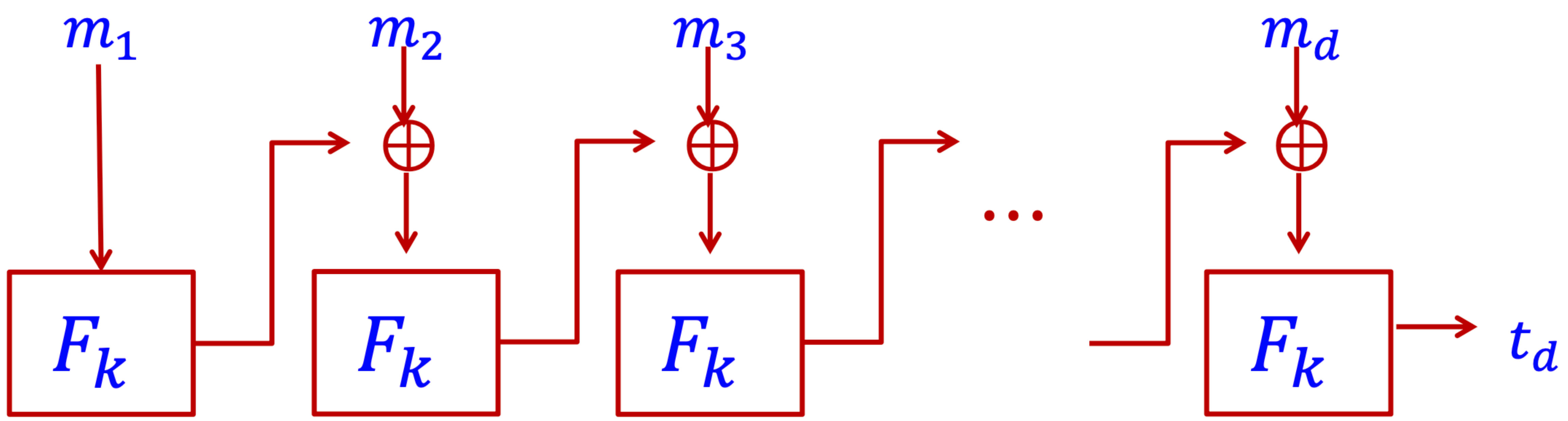
CBC-MAC (for fixed length $d \cdot n$):

Mac$_k(m) = t_d$ where

$t_0 = 0^n$

$t_i = F_k(t_{i-1} \oplus m_i)$ for $i = 1, \ldots, d$

and F_k is a length-preserving PRF



Theorem: For every polynomial $\ell(\cdot)$, CBC-MAC is a secure MAC for messages of length $\ell(n)$

Smaller Tags: CBC-MAC

$$m = \begin{array}{|c|c|c|c|c|c|c|c|}\hline m_1 & m_2 & & \dots & & \dots & & m_d \\ \hline \end{array}$$

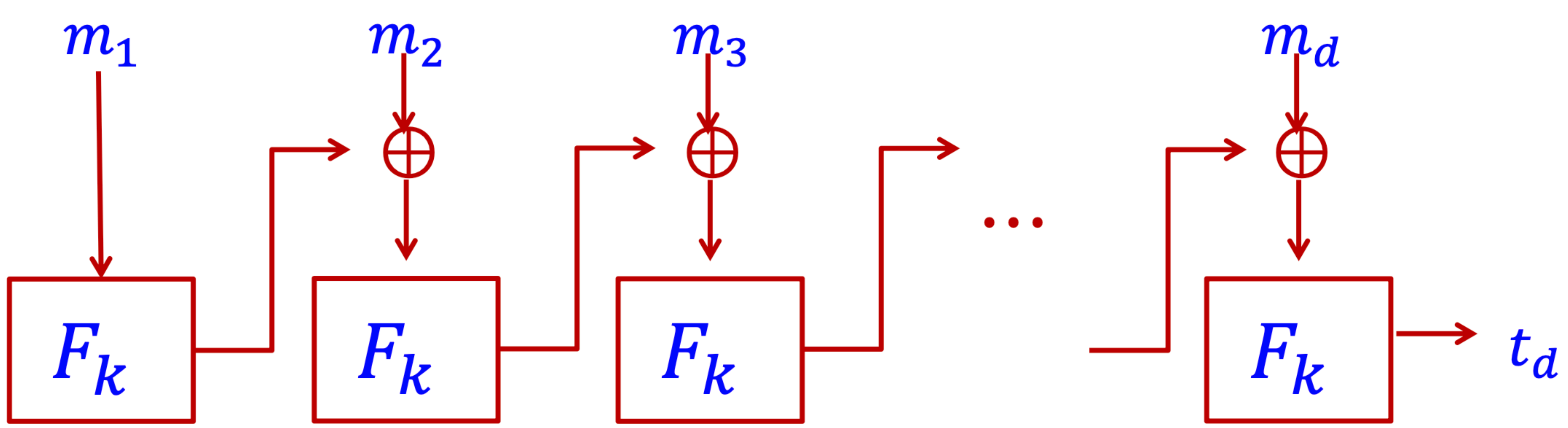
CBC-MAC (for fixed length $d \cdot n$):

$\text{Mac}_k(m) = t_d$ where

$t_0 = 0^n$

$t_i = F_k(t_{i-1} \oplus m_i)$ for $i = 1, \ldots, d$

and F_k is a length-preserving PRF



Theorem: For every polynomial $\ell(\cdot)$, CBC-MAC is a secure MAC for messages of length $\ell(n)$

For arbitrary length: Message length is appended to the front and used as the first block

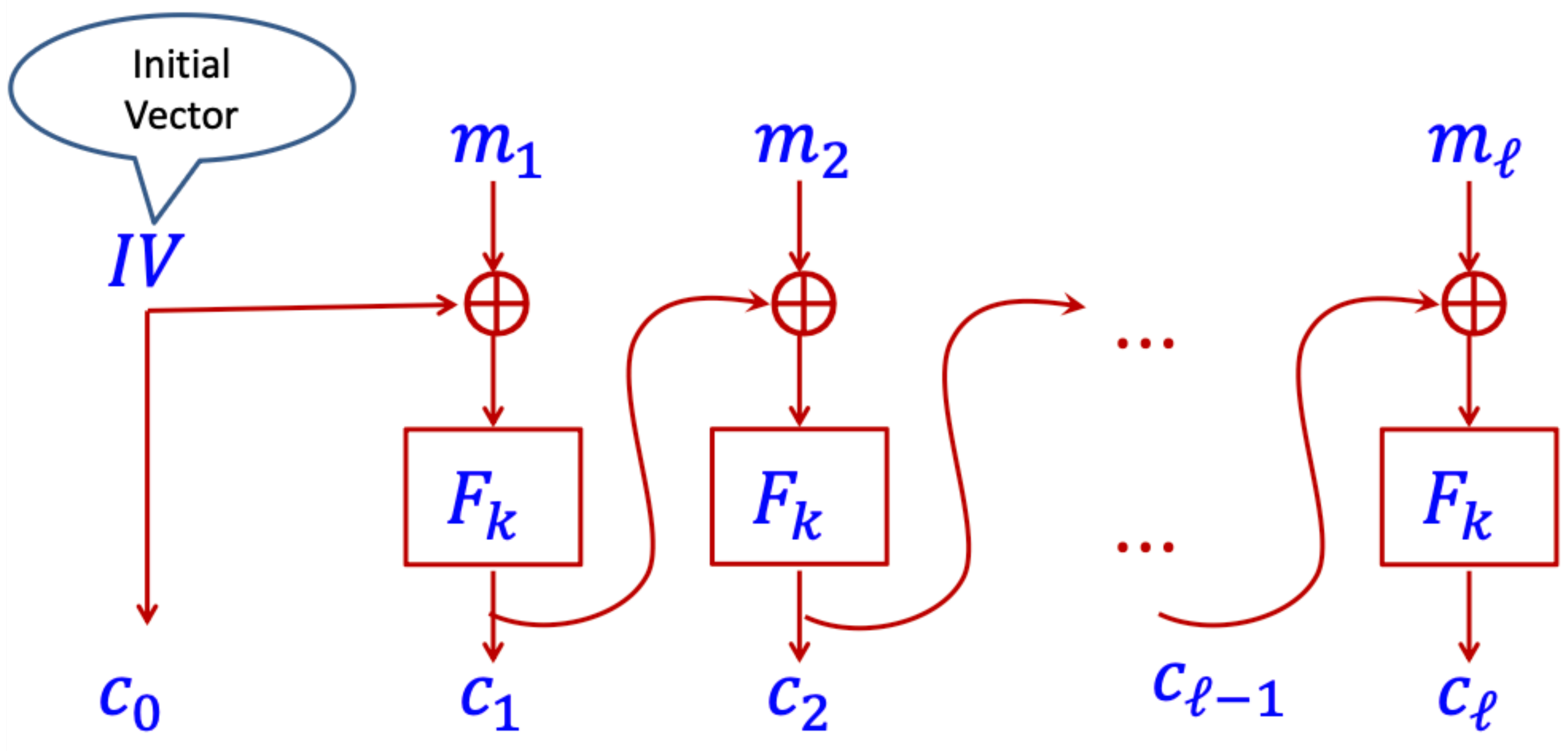
CBC-mode Encryption vs CBC-MAC

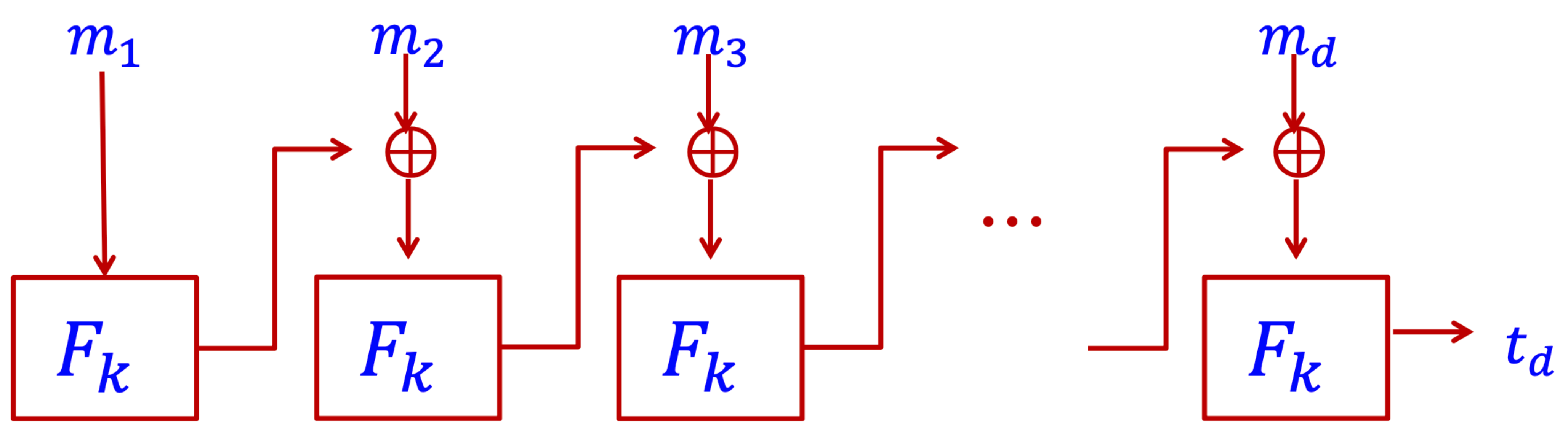
CBC-mode Encryption:

- Requires a random IV (insecure if IV is not random)
- Outputs each PRF output (block outputs are required for decryption)

CBC-MAC:

- No IV (random IV is insecure)
- Only outputs the last block (outputting all PRF outputs is insecure)

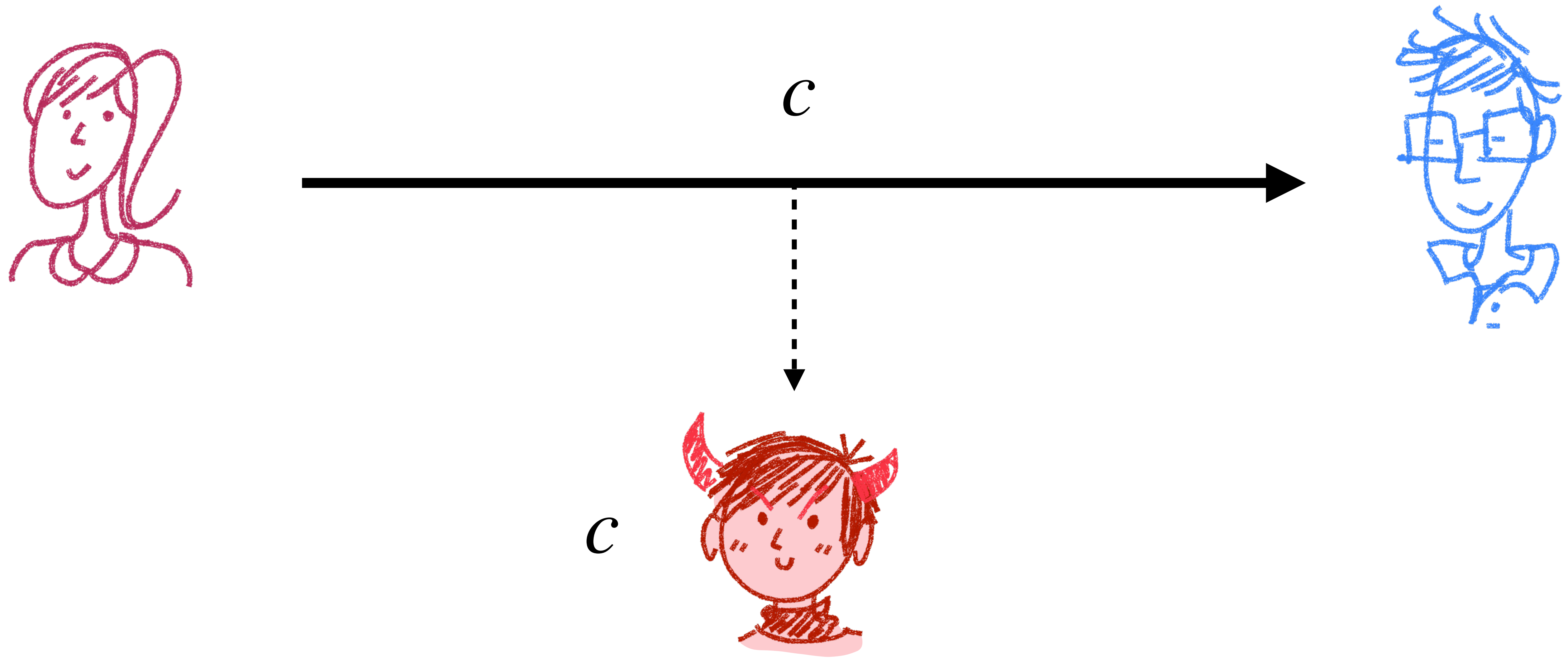




CCA-Security

Active Attacks

- So far our security defs for encryption only consider passive adversaries
  - Can listen and influence messages (see encryptions of messages of its choice)
- What about an active adversary?



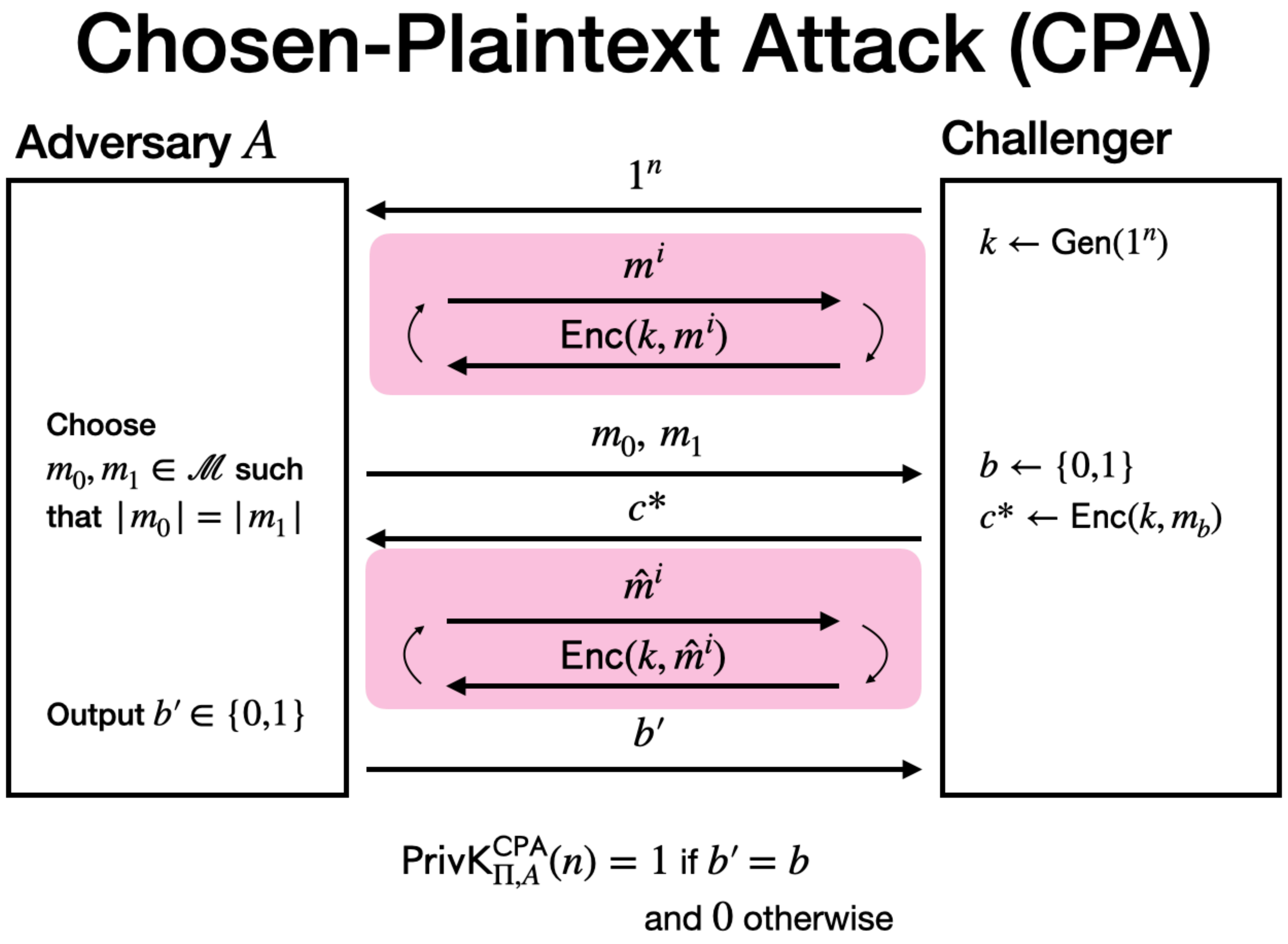
Active Attacks

- What if an active adversary sends a modified ciphertext c' to Bob?
  - Bob can decrypt the ciphertext and may behave differently based on what was decrypted
  - Adversary may witness the behavior and infer something about c'!

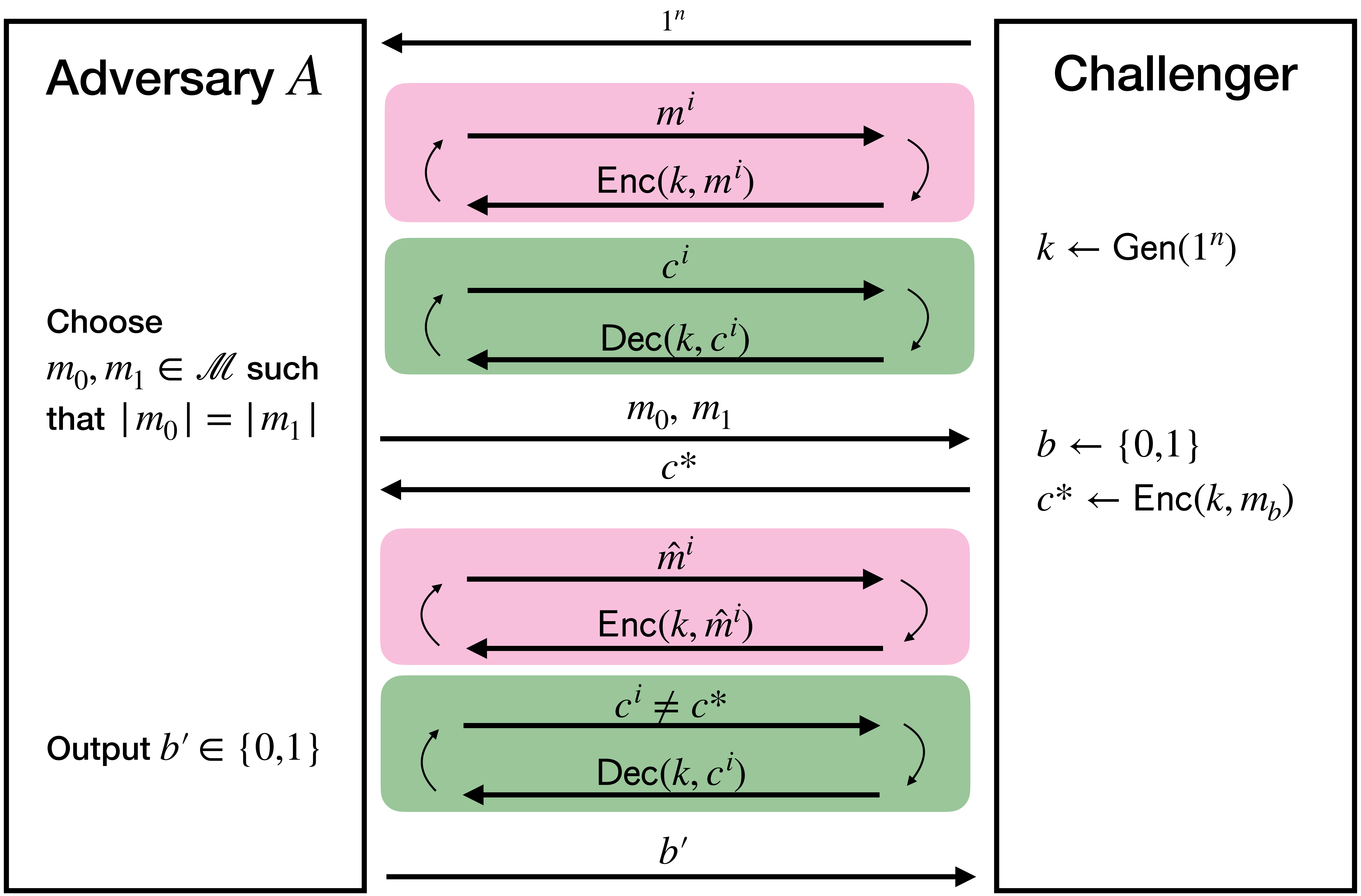
CCA-Security

Chosen-Ciphertext Attack (CCA)

- New notion of security for encryption schemes to capture an active adversary
- Use CPA-security as a basis, but add a decryption oracle
    - Adversary can request decryptions of arbitrary ciphertexts (except c^*)



Chosen-Ciphertext Attack (CCA)



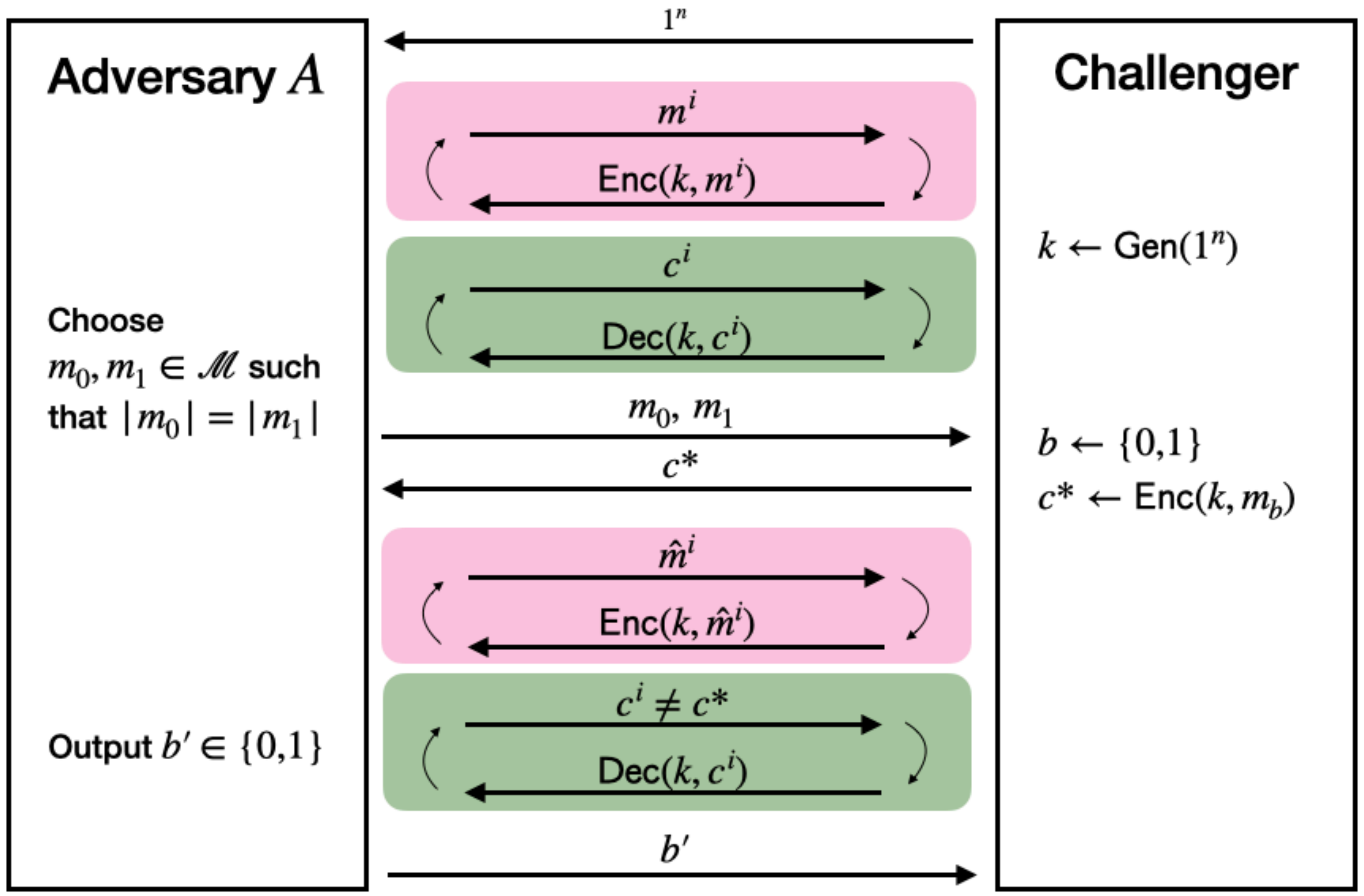
$$\mathsf{PrivK}^{\mathsf{CCA}}_{\Pi,A}(n) = \begin{cases} 1 & b' = b \\ 0 & b' \neq b \end{cases}$$

Chosen-Ciphertext Attack (CCA)

Definition:

Π has indistinguishable encryptions under chosen-ciphertext attack (or CCA-security) if for every PPT adversary A there exists a negligible function $\epsilon(\cdot)$ such that

$$\Pr[\mathsf{PrivK}^{\mathsf{CCA}}_{\Pi,A}(n) = 1] \leq \frac{1}{2} + \epsilon(n)$$



$$\mathsf{PrivK}^{\mathsf{CCA}}_{\Pi,A}(n) = \begin{cases} 1 & b' = b \\ 0 & b' \neq b \end{cases}$$

Notes:

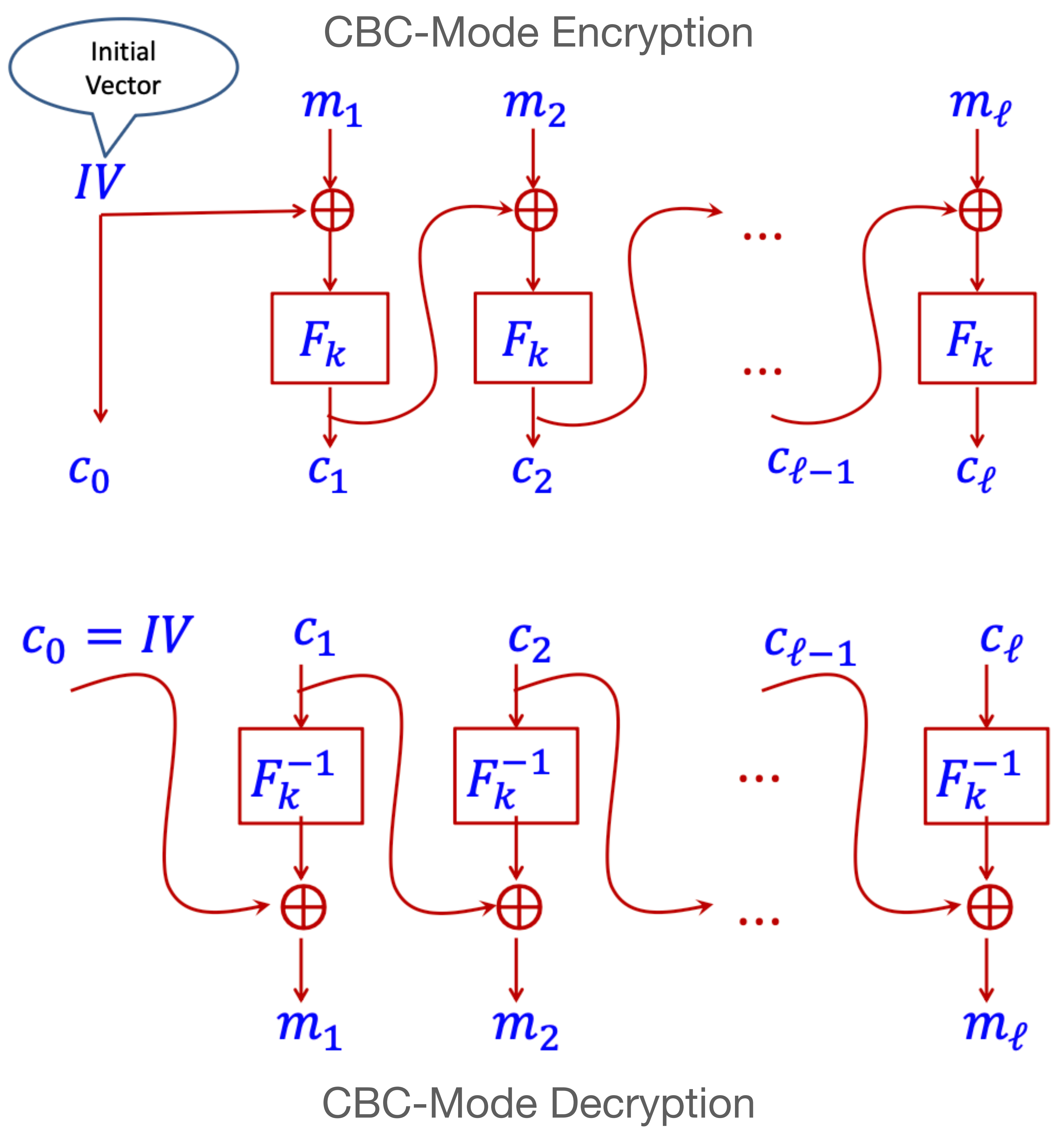
- Sometimes referred to as CCA2
- CCA1 refers to a weaker version where the adversary only access to the decryption oracle before c^* is given

Padding Oracle Attack

- Consider an adversary who can send ciphertexts to Bob to decrypt
  - She may not know the decryption of c' (yet), but she may know he has predefined behavior on malformed messages
    - For example, he send returns an error or ends the connection
- What if she know how to modify ciphertexts to change the underlying message in a predictable way?

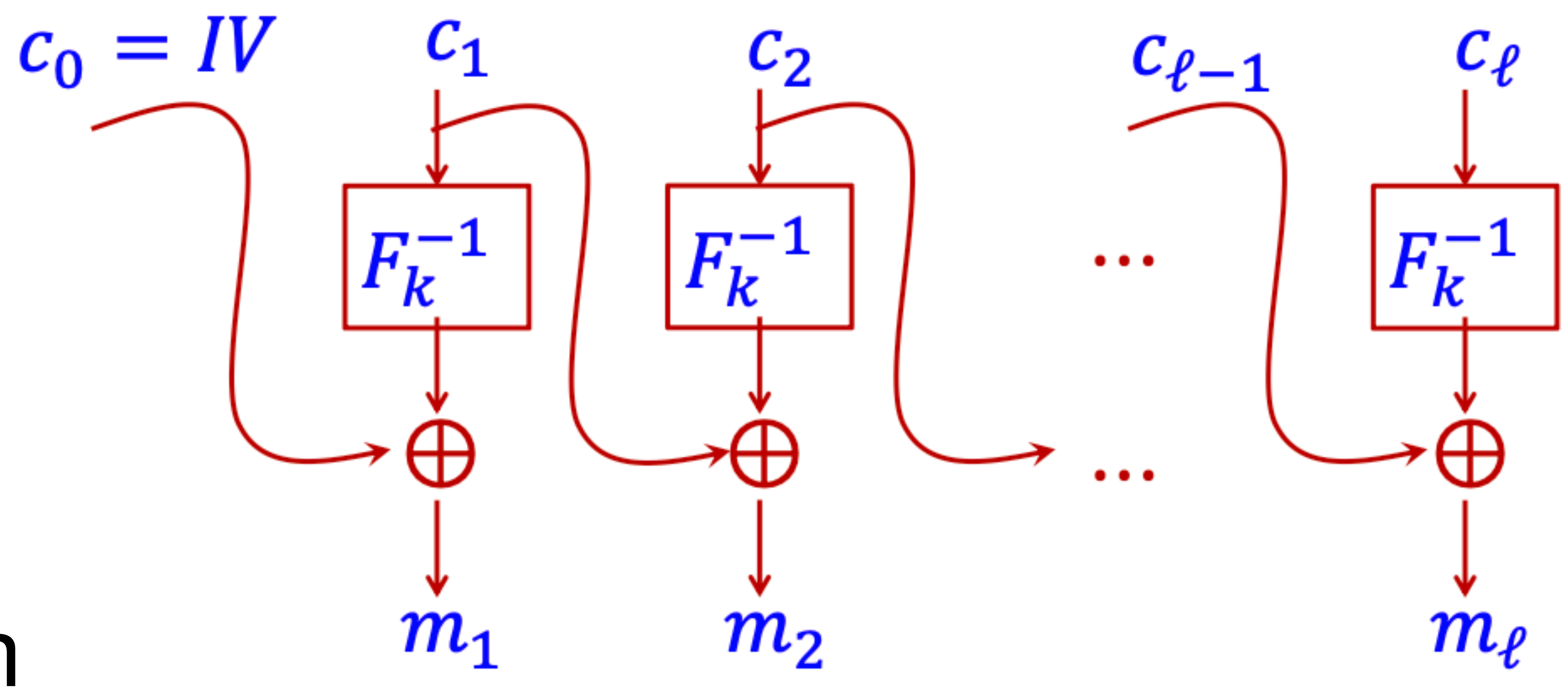
Padding Oracle Attack

- Recall CBC Mode
- Message length is assumed to be a multiple of the block length
  - If not, then padding is needed
- PKCS #7 (Public-Key Cryptographic Standards): write the number of bytes in the padding
  - 1 byte of padding: 0x01
  - 2 bytes of padding: 0x0202
  - 3 bytes of padding: 0x030303



Padding Oracle Attack

- PKCS #7 (Public-Key Cryptographic Standards): write the number of bytes in the padding
  - 1 byte of padding: 0x01
  - 2 bytes of padding: 0x0202
  - 3 bytes of padding: 0x030303

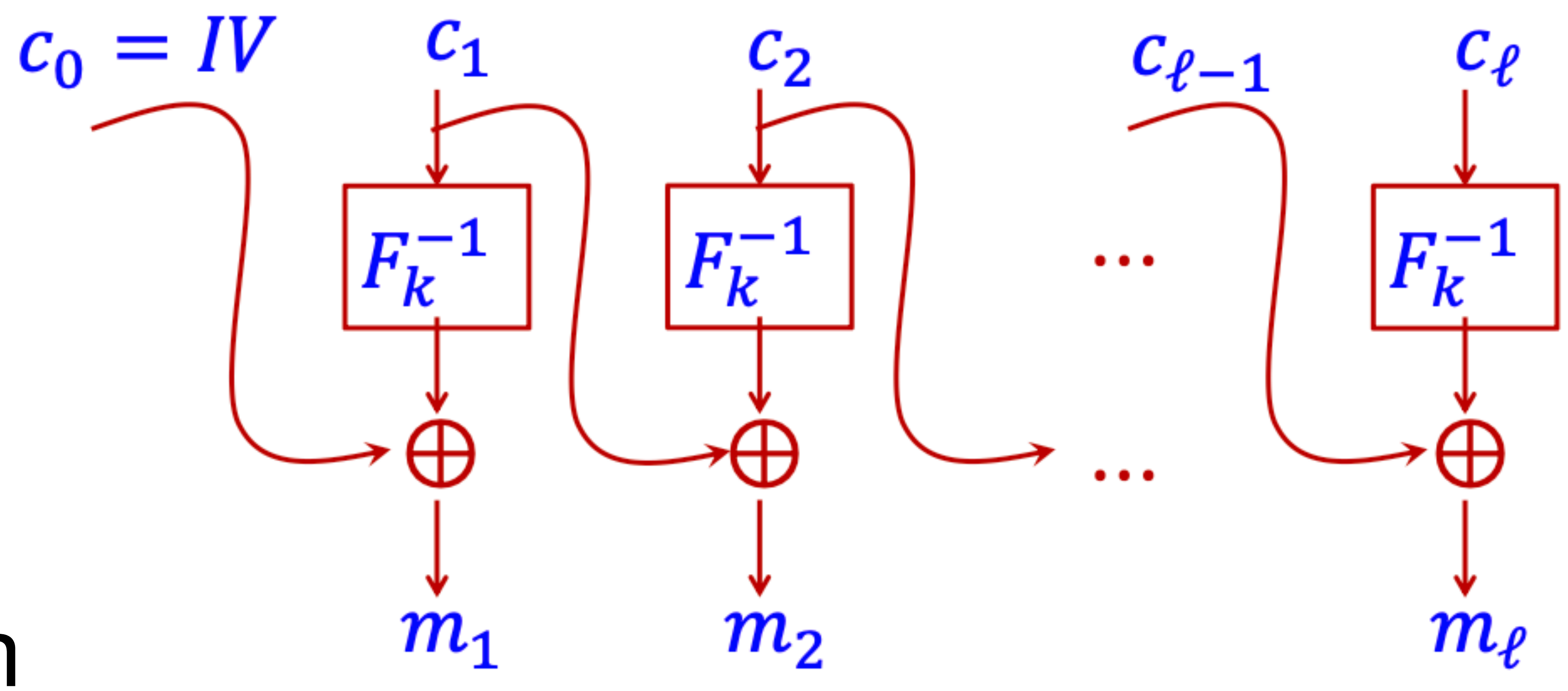


- The last block must be of a particular form or decryption may throw an error!
  - If it's too short, it must end with 0x0b repeated b times
- Last block is computed as

$$m_\ell = F_k^{-1}(c_\ell) \oplus c_{\ell-1}$$

Padding Oracle Attack

- PKCS #7 (Public-Key Cryptographic Standards): write the number of bytes in the padding
  - 1 byte of padding: 0x01
  - 2 bytes of padding: 0x0202
  - 3 bytes of padding: 0x030303



- The last block must be of a particular form or decryption may throw an error!
  - If it's too short, it must end with 0x0b repeated b times

- Last block is computed as

$$m_\ell = F_k^{-1}(c_\ell) \oplus c_{\ell-1}$$

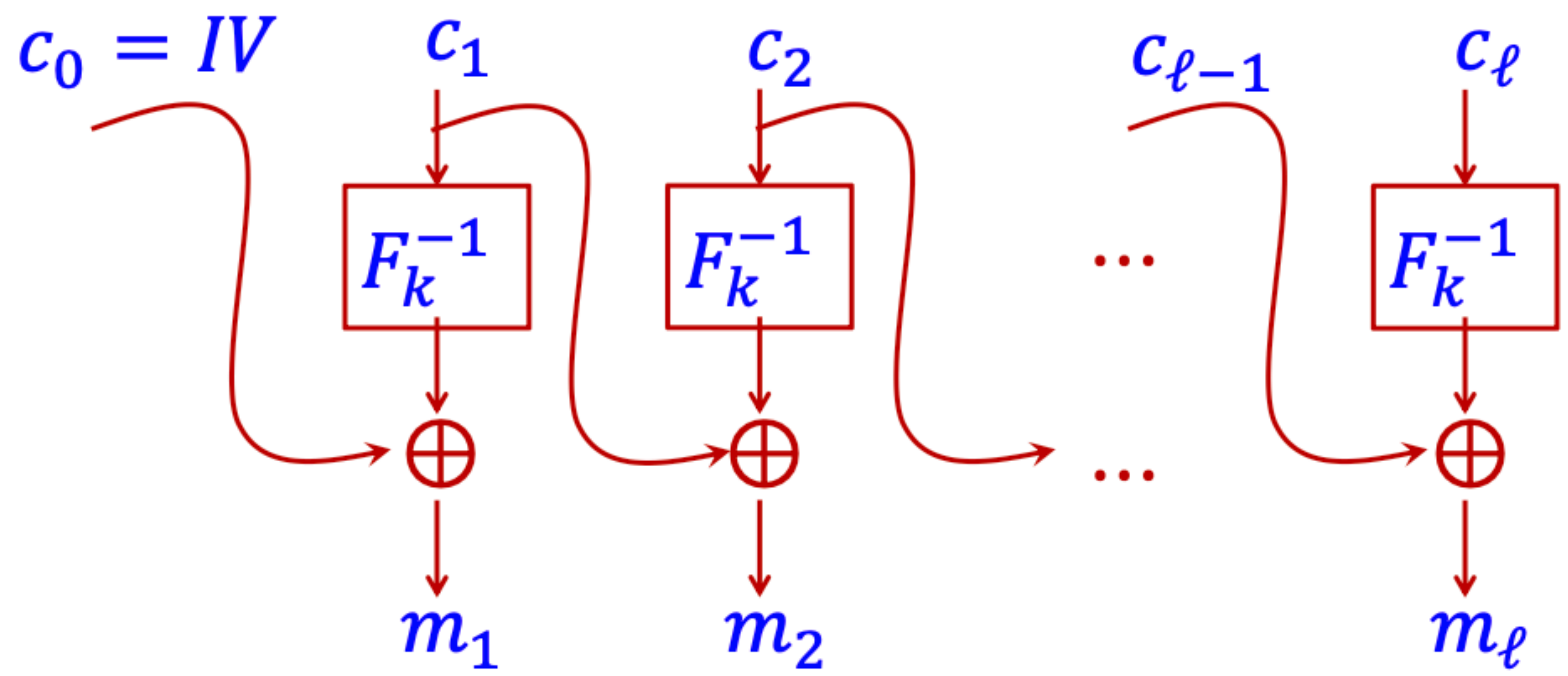
What happens to m_ℓ if we change $c_{\ell-1}$?

Padding Oracle Attack

- The last block must be of a particular form and is computed as

$$m_\ell = F_k^{-1}(c_\ell) \oplus c_{\ell-1}$$

- Attack is as follows:
  - First find the amount of padding by changing bits of $c_{\ell-1}$ left to right until an error is observed
  - Then modify the lower bits of $c_{\ell-1}$ to change the lower b bits of c_ℓ to 0x(b+1)(b+1)(b+1)…
  - Change the (b+1)th lowest bit of $c_{\ell-1}$ until there is no padding error

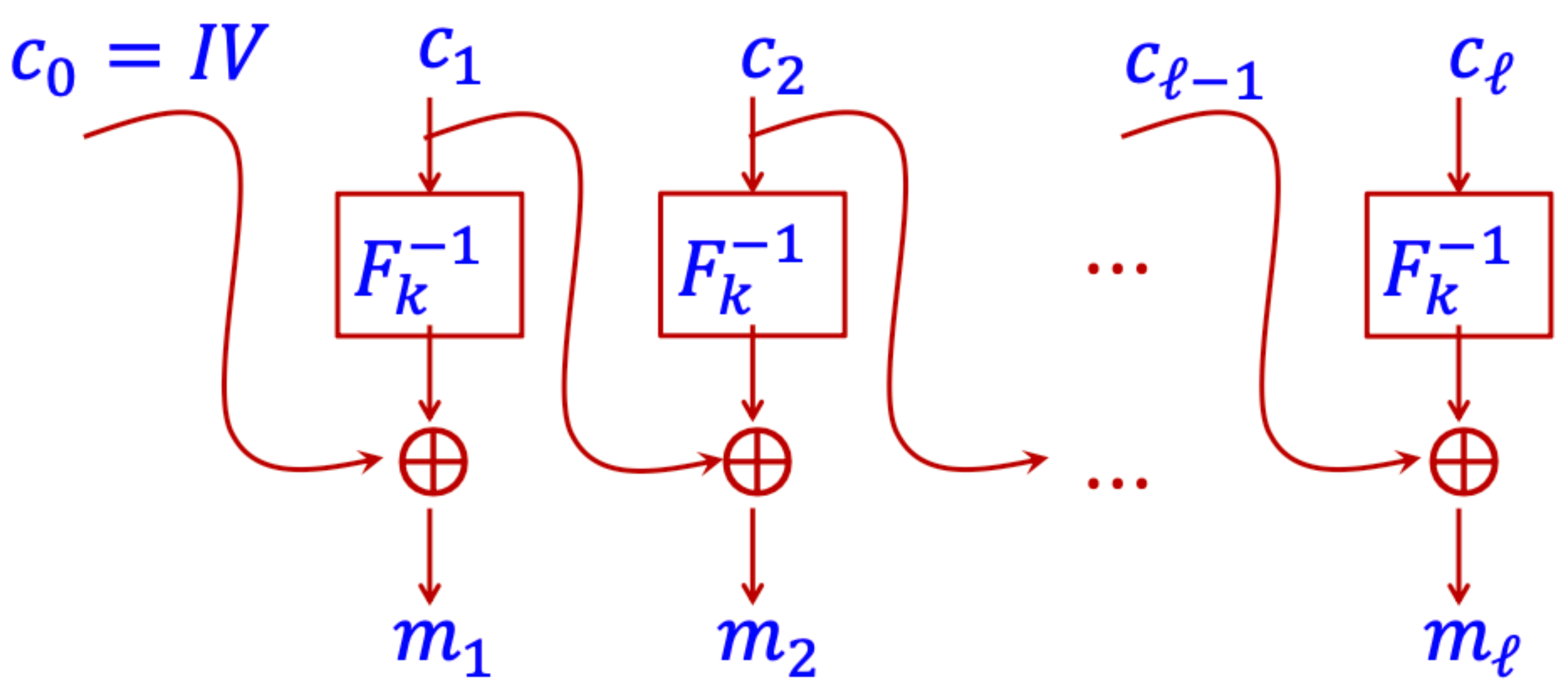


Padding Oracle Attack

- The last block must be of a particular form and is computed as

$$m_\ell = F_k^{-1}(c_\ell) \oplus c_{\ell-1}$$

- Attack is as follows:
  - First find the amount of padding by changing bits of $c_{\ell-1}$ left to right until an error is observed
  - Then modify the lower bits of $c_{\ell-1}$ to change the lower b bits of c_ℓ to 0x(b+1)(b+1)(b+1)…
  - Change the (b+1)th lowest bit of $c_{\ell-1}$ until there is no padding error



This is a special case of CCA-security that was used to break real systems that are only CPA-secure

Constructing CCA-Secure Encryption?

- With the schemes we've seen so far, it's easy to create an encryption of some message $m \oplus \Delta$ given an encryption of m (even if you don't know m)
  - This is known as malleability
- Informally, this malleability makes these schemes *not* CCA-secure
  - To get CCA-security, we need non-malleability
  - (We are not going to get into the weeds about malleability/non-malleability)

A CCA-Secure Encryption Scheme

Let F be a strong PRP. Define $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows.

- Gen(1^n): output a random $k \leftarrow \{0,1\}^n$
- Enc$_k(m)$: Sample $r \leftarrow \{0,1\}^{\ell(n)/2}$ and output $F_k(r || m)$
- Dec$_k(c)$: Compute $x = F_k^{-1}(c)$ and output the right half of x

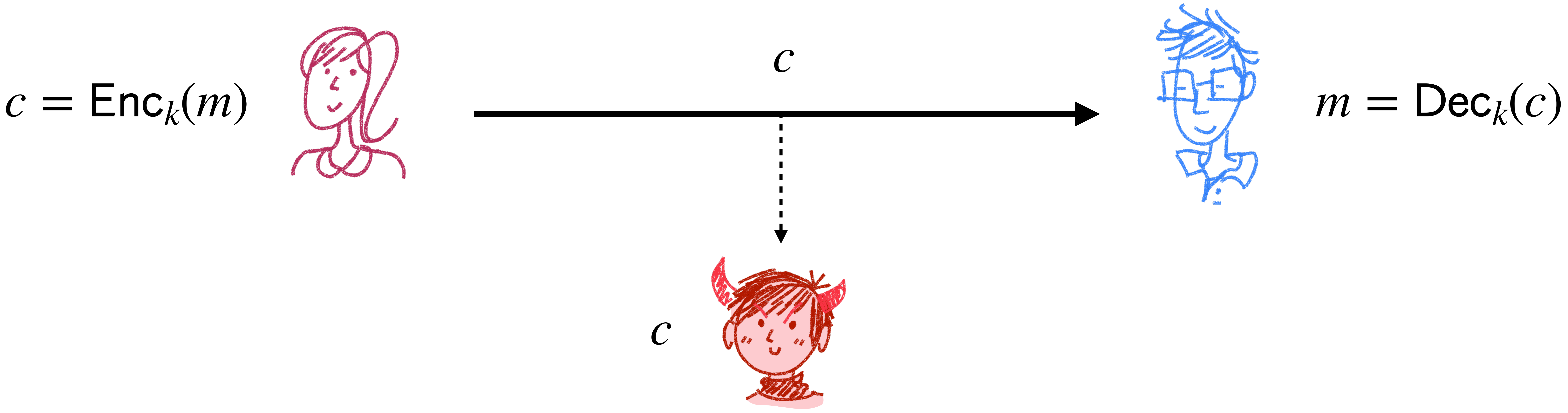
Claim: Π is CCA-secure

Authenticated Encryption

Authenticated Encryption

So far we've talked about secrecy and integrity separately, what if we want both at the same time

- Specifically, CCA-security and unforgeability (assurance that m is not modified)



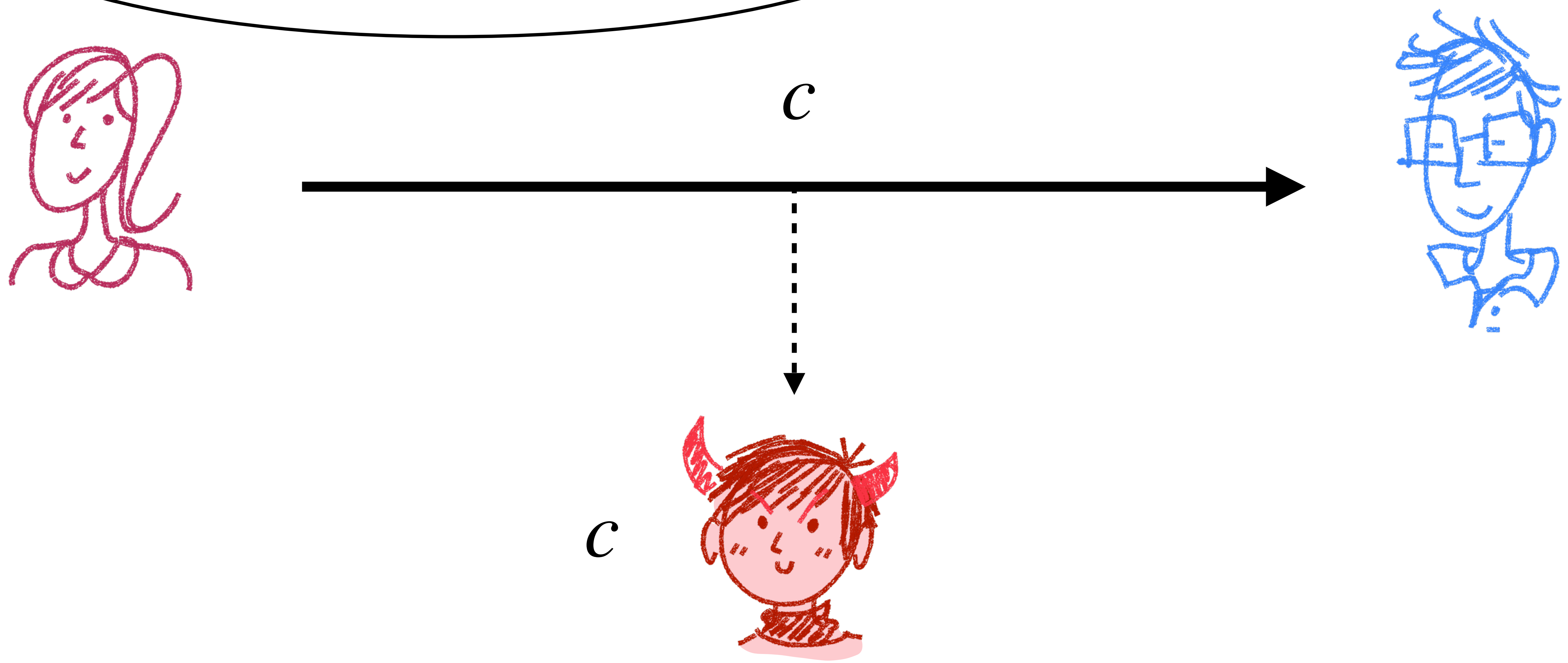
Authenticated Encryption

So far we've talked about secrecy and integrity separately, what if we want both at the same time

- Specifically, CCA-security and unforgeability (assurance that m is not modified)

Analogue of MAC unforgeability for encryption schemes

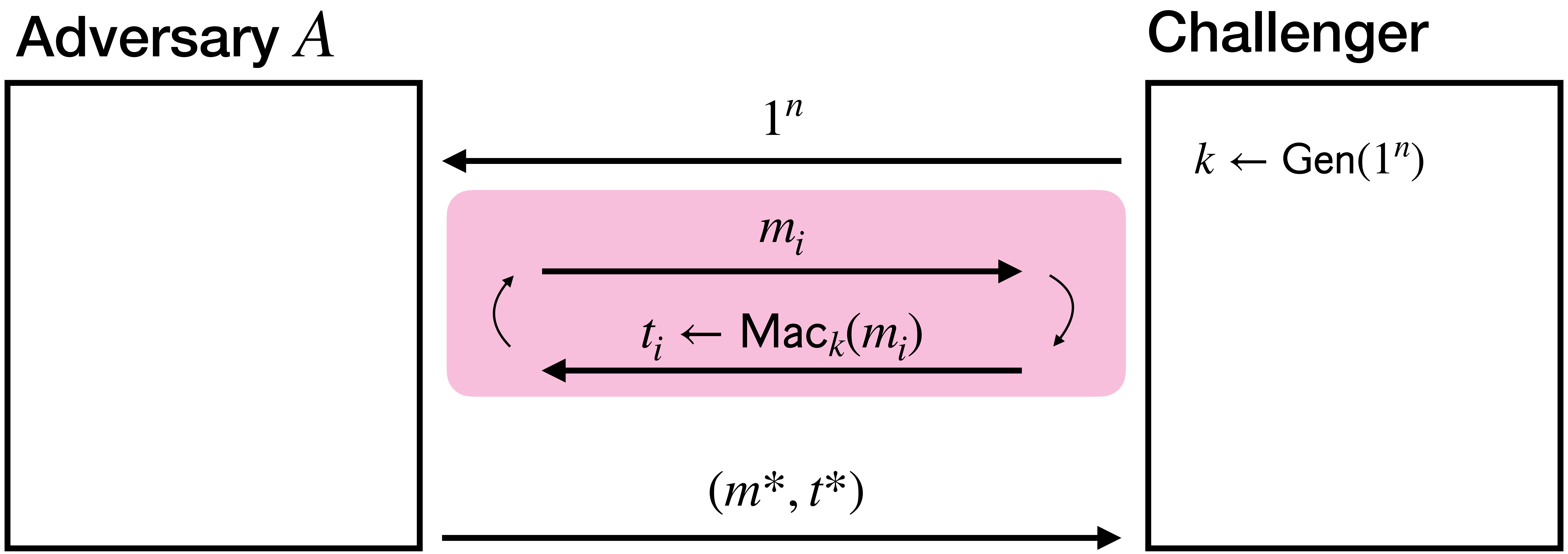
$c = \text{Enc}_k(m)$



$m = \text{Dec}_k(c)$

Recall: MAC Unforgeability

Let $\Pi = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$. We define $\mathsf{MacForge}_{\mathscr{A},\Pi}(n)$ as follows

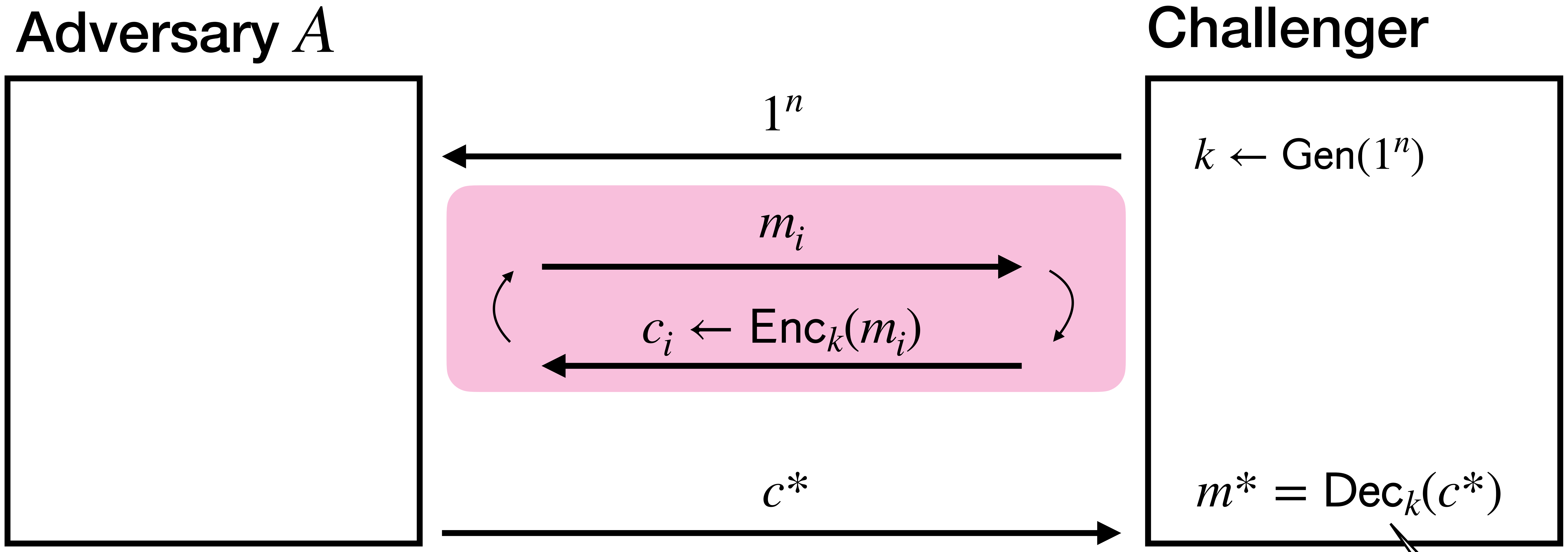


We say the adversary succeeds ($\mathsf{MacForge}_{\mathscr{A},\Pi}(n) = 1$) if:

1. $\mathsf{Verify}_k(m^*, t^*) = 1$
2. $m^* \neq m_i$ for all queried m_i

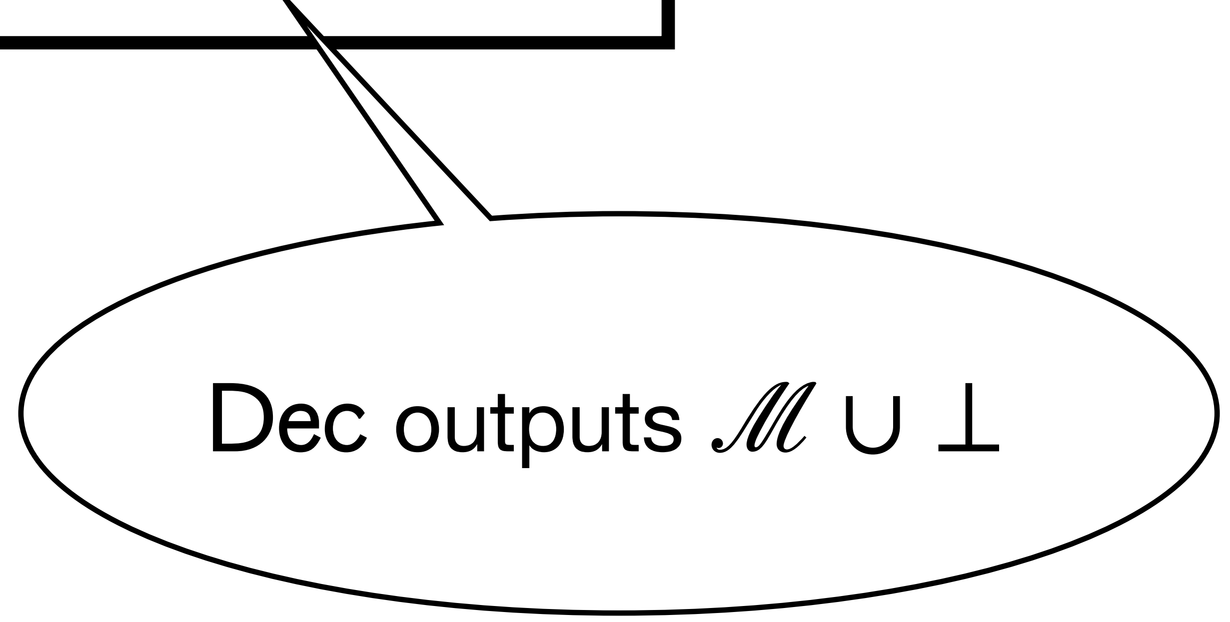
Unforgeability for Encryption

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. We define $\mathsf{EncForge}_{\mathscr{A},\Pi}(n)$ as follows



We say the adversary succeeds ($\mathsf{EncForge}_{\mathscr{A},\Pi}(n) = 1$) if:

1. $m^* \neq \perp$
2. $m^* \neq m_i$ for all queried m_i

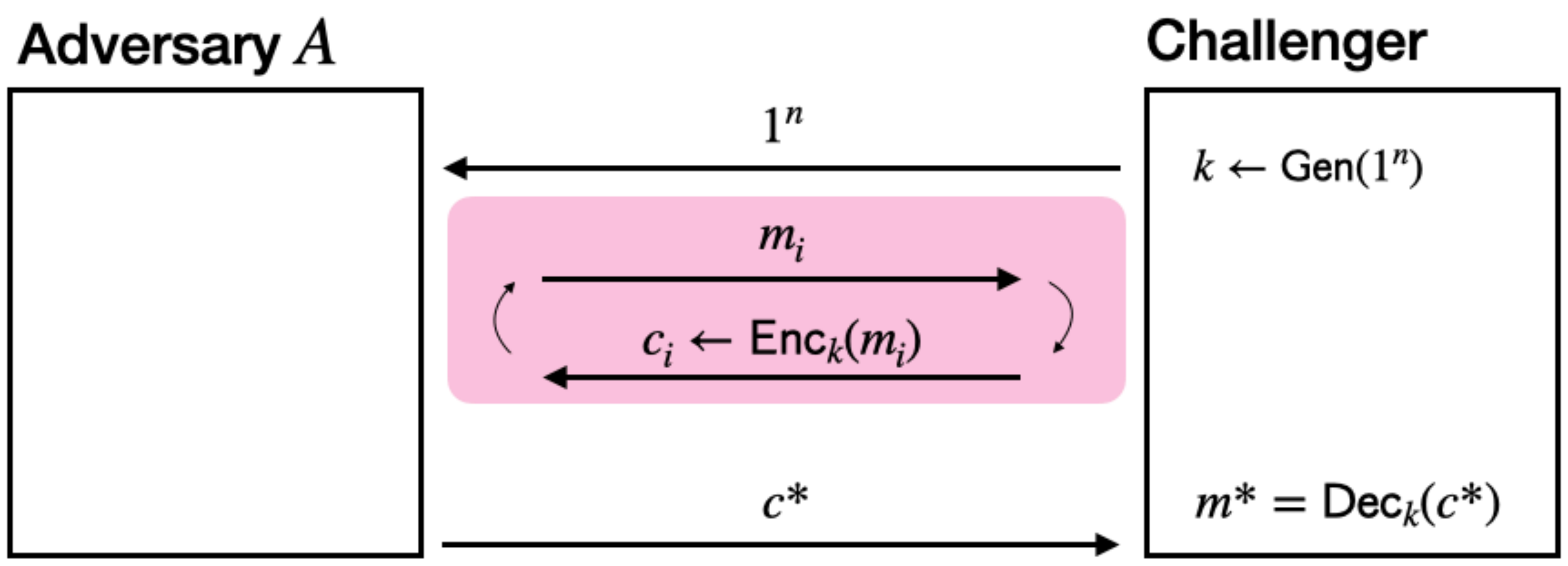


Authenticated Encryption

Definition:

$\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an authenticated encryption scheme (AE) if it is CCA-secure and unforgeable: for every PPT adversary A there exists a negligible function $\epsilon(\cdot)$ such that

$$\Pr[\mathsf{EncForge}_{\Pi,A}(n) = 1] \leq \epsilon(n)$$



$\mathsf{EncForge}_{\mathcal{A},\Pi}(n) = 1$ if:

1. $m^* \neq \perp$
2. $m^* \neq m_i$ for all queried m_i

How to Build Authenticated Encryption?

Encryption + Message Authentication = Authenticated Encryption

Given:

- Encryption scheme $\Pi_E = (\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$
- MAC scheme $\Pi_M = (\mathsf{Gen}_M, \mathsf{Mac}, \mathsf{Verify})$

Goal: Construct an AE scheme $\hat{\Pi} = (\hat{\mathsf{Gen}}, \hat{\mathsf{Enc}}, \hat{\mathsf{Dec}})$

How do we combine the two?

Idea 1: Encrypt-and-Authenticate

$\Pi_M = (\mathsf{Gen}_M, \mathsf{Mac}, \mathsf{Verify})$

$\Pi_E = (\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$

$c = \mathsf{Enc}_{k_E}(m)$
$t = \mathsf{Mac}_{k_M}(m)$

$\xrightarrow{c,\, t}$

$m = \mathsf{Dec}_{k_E}(c)$
Output an error if $\mathsf{Verify}_{k_M}(m, t) \neq 1$.

Idea 2: Authenticate-then-Encrypt

$\Pi_M = (\mathsf{Gen}_M, \mathsf{Mac}, \mathsf{Verify})$

$\Pi_E = (\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$

$t = \mathsf{Mac}_{k_M}(m)$

$c = \mathsf{Enc}_{k_E}(m \,||\, t)$

$\xrightarrow{c}$

$m \,||\, t = \mathsf{Dec}_{k_E}(c)$

Output an error if $\mathsf{Verify}_{k_M}(m, t) \neq 1$.

Idea 3: Encrypt-then-Authenticate

$\Pi_M = (\mathsf{Gen}_M, \mathsf{Mac}, \mathsf{Verify})$

$\Pi_E = (\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$

$c = \mathsf{Enc}_{k_E}(m)$
$t = \mathsf{Mac}_{k_M}(c)$

$$\xrightarrow{\quad c, t \quad}$$

Decryption produces an error if $\mathsf{Verify}_{k_M}(c, t) \neq 1$.

Otherwise, it outputs $\mathsf{Dec}_{k_E}(c)$

Next Time

- Today
  - Arbitrary-Length MACs
  - CCA-Security
  - Authenticated Encryption
- Monday
  - Hash functions