# 6CCS3CIS Cryptography, second lecture

*Ngoc Khanh Nguyen and Eamonn W. Postlethwaite*

October 9, 2025

# Programme

- formal definitions of SKE and MAC
- Kerckhoffs' principle(s),
- formal security definitions for SKE and MAC

# SKE **formally**

A triple of algorithms $\mathsf{SKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ with associated key, message and ciphertext spaces $\mathcal{K}, \mathcal{M}, \mathcal{C}$.

- KGen is a randomised algorithm that samples and outputs a secret key $sk \in \mathcal{K}$,
  $$sk \leftarrow \mathsf{KGen}$$
- Enc is a potentially randomised algorithm that takes as input $sk \in \mathcal{K}$ and $m \in \mathcal{M}$ and outputs $c \in \mathcal{C}$,
  $$c \leftarrow \mathsf{Enc}(sk, m) \text{ if randomised, else } c = \mathsf{Enc}(sk, m)$$
- Dec is a deterministic algorithm that takes as input $sk \in \mathcal{K}$ and $c \in \mathcal{C}$ and outputs either $m \in \mathcal{M}$ or a special failure symbol $\perp$.
  $$m = \mathsf{Dec}(sk, c) \text{ or } \perp = \mathsf{Dec}(sk, c)$$

# SKE **formally (II)**

What does it mean for a function to be randomised? Implicitly some randomness[1] is given as an extra input.

[1] Perhaps think of `os.urandom` in Python or `getrandom` in Rust.

# SKE **formally (II)**

What does it mean for a function to be randomised? Implicitly some randomness[1] is given as an extra input.

We can make the function determinstic by additionally specifying the randomness. I.e. for some randomness space $\mathcal{R} = \{0,1\}^r$ let $\rho \leftarrow \mathcal{R}$ and $c = \mathsf{Enc}(sk, m; \rho)$.

[1] Perhaps think of `os.urandom` in Python or `getrandom` in Rust.

# SKE **formally (II)**

What does it mean for a function to be randomised? Implicitly some randomness[1] is given as an extra input.

We can make the function determinstic by additionally specifying the randomness. I.e. for some randomness space $\mathcal{R} = \{0,1\}^r$ let $\rho \leftarrow \mathcal{R}$ and $c = \mathsf{Enc}(sk, m; \rho)$.

We will return to this when we build contemporary SKE. (A fundamental problem with the historical ciphers we discuss shortly is that Enc is deterministic.)

[1] Perhaps think of `os.urandom` in Python or `getrandom` in Rust.

# SKE **correctness**

The idea that a message encrypted to a ciphertext and then decrypted, both computations using the same $sk$, should return the original message.

# SKE **correctness**

The idea that a message encrypted to a ciphertext and then decrypted, both computations using the same $sk$, should return the original message.

$$\text{for all } sk \in \mathcal{K} \text{ and } m \in \mathcal{M},\ \mathsf{Dec}(sk, \mathsf{Enc}(sk, m)) = m.$$

# SKE **correctness**

The idea that a message encrypted to a ciphertext and then decrypted, both computations using the same $sk$, should return the original message.

$$\text{for all } sk \in \mathcal{K} \text{ and } m \in \mathcal{M},\ \mathsf{Dec}(sk, \mathsf{Enc}(sk, m)) = m.$$

Says nothing about security. $\mathsf{Enc}(sk, m) = m$ and $\mathsf{Dec}(sk, c) = c$ is correct but silly.

# MAC **formally**

A triple of algorithms $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vf})$ with associated key, message and tag spaces $\mathcal{K}, \mathcal{M}, \mathcal{T}$.

- KGen is a randomised algorithm that samples and outputs an integrity key $ik \in \mathcal{K}$,
  $ik \leftarrow \mathsf{KGen}$
- Tag is a potentially randomised algorithm that takes as input $ik \in \mathcal{K}$ and $m \in \mathcal{M}$ and outputs $\tau \in \mathcal{T}$,
  $\tau \leftarrow \mathsf{Tag}(ik, m)$ if randomised, else $\tau = \mathsf{Tag}(ik, m)$
- Vf is a deterministic algorithm that takes as input $ik \in \mathcal{K}$, $m \in \mathcal{M}$ and $\tau \in \mathcal{T}$ and outputs either 0 or 1.
  $0 = \mathsf{Vf}(ik, m, \tau)$ or $1 = \mathsf{Vf}(ik, m, \tau)$

# MAC **formally (II)**

We consider MAC with deterministic Tag in this course, that is $\tau = \mathsf{Tag}(ik, m)$.

# MAC **formally (II)**

We consider MAC with deterministic Tag in this course, that is $\tau = \mathsf{Tag}(ik, m)$.

In this case, we may consider 'canonical verification' without loss of generality. Namely $\mathsf{Vf}(ik, m, \tau)$ does

- compute $\tau' = \mathsf{Tag}(ik, m)$,
- output 1 when $\tau = \tau'$, else output 0.

# MAC **formally (II)**

We consider MAC with deterministic Tag in this course, that is $\tau = \mathsf{Tag}(ik, m)$.

In this case, we may consider 'canonical verification' without loss of generality. Namely $\mathsf{Vf}(ik, m, \tau)$ does

- compute $\tau' = \mathsf{Tag}(ik, m)$,
- output 1 when $\tau = \tau'$, else output 0.

(Note, canonical verification requires a deterministic Tag. Why?)

# MAC **correctness**

The idea that a tag computed for a message given an integrity key should return 1 (be valid) when verified using the same integrity key.

# MAC **correctness**

The idea that a tag computed for a message given an integrity key should return 1 (be valid) when verified using the same integrity key.

$$\text{for all } ik \in \mathcal{K} \text{ and } m \in \mathcal{M},\ \mathsf{Vf}(ik, m, \mathsf{Tag}(ik, m)) = 1.$$

# MAC **correctness**

The idea that a tag computed for a message given an integrity key should return 1 (be valid) when verified using the same integrity key.

$$\text{for all } ik \in \mathcal{K} \text{ and } m \in \mathcal{M},\ \mathsf{Vf}(ik, m, \mathsf{Tag}(ik, m)) = 1.$$

Says nothing about security. Vf that always outputs one is correct but silly.

## What are spaces $\mathcal{K}, \mathcal{M}, \mathcal{C}, \mathcal{T}$?

Formally these spaces are just 'sets', mathematical objects that are collections of things without repeats. Pictures of cats, binary strings, a supernova, whatever you like...

This level of abstraction is sometimes unhelpful, and often in cryptography they are strings from a particular alphabet.

## What are spaces $\mathcal{K}, \mathcal{M}, \mathcal{C}, \mathcal{T}$?

Formally these spaces are just 'sets', mathematical objects that are collections of things without repeats. Pictures of cats, binary strings, a supernova, whatever you like...

This level of abstraction is sometimes unhelpful, and often in cryptography they are strings from a particular alphabet.

For example, call $\alpha = \{0, 1\}$ the 'binary alphabet' then

$$\alpha^0 = \{\varepsilon\},\ \alpha^1 = \{0, 1\},\ \alpha^2 = \{00, 01, 10, 11\},\ \alpha^* = \bigcup_{i \in \mathbb{N}} \alpha^i.$$

## What are spaces $\mathcal{K}, \mathcal{M}, \mathcal{C}, \mathcal{T}$?

Formally these spaces are just 'sets', mathematical objects that are collections of things without repeats. Pictures of cats, binary strings, a supernova, whatever you like...

This level of abstraction is sometimes unhelpful, and often in cryptography they are strings from a particular alphabet.

For example, call $\alpha = \{0, 1\}$ the 'binary alphabet' then

$$\alpha^0 = \{\varepsilon\},\ \alpha^1 = \{0, 1\},\ \alpha^2 = \{00, 01, 10, 11\},\ \alpha^* = \bigcup_{i \in \mathbb{N}} \alpha^i.$$

That is, the empty string, bit strings of length one (a bit), bit strings of length two and the 'Kleene star', i.e. bit strings of any length. (Here $\mathbb{N} = \{0, 1, 2, \dots\}$ are the natural numbers.)

## What are spaces $\mathcal{K}, \mathcal{MC}, \mathcal{T}$? (II)

We may also consider the English alphabet $\alpha = \{\mathsf{a}, \ldots, \mathsf{z}\}$. (Usually historical ciphers consider alphabets in the traditional sense of the word, and contemporary cryptography the binary alphabet.)

## What are spaces $\mathcal{K}, \mathcal{MC}, \mathcal{T}$? (II)

We may also consider the English alphabet $\alpha = \{\mathsf{a}, \ldots, \mathsf{z}\}$. (Usually historical ciphers consider alphabets in the traditional sense of the word, and contemporary cryptography the binary alphabet.)

Consider a SKE and let $\alpha_{\mathcal{K}}, \alpha_{\mathcal{M}}$ and $\alpha_{\mathcal{C}}$ be the alphabets for the key space $\mathcal{K}$, the message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ respectively. Two common examples might be

# What are spaces $\mathcal{K}, \mathcal{MC}, \mathcal{T}$? (II)

We may also consider the English alphabet $\alpha = \{\mathsf{a}, \ldots, \mathsf{z}\}$. (Usually historical ciphers consider alphabets in the traditional sense of the word, and contemporary cryptography the binary alphabet.)

Consider a SKE and let $\alpha_{\mathcal{K}}, \alpha_{\mathcal{M}}$ and $\alpha_{\mathcal{C}}$ be the alphabets for the key space $\mathcal{K}$, the message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ respectively. Two common examples might be

- $\alpha_{\mathcal{K}} = \alpha_{\mathcal{M}} = \alpha_{\mathcal{C}} = \{\mathsf{a}, \ldots, \mathsf{z}\}$ with $\mathcal{K} = \alpha_{\mathcal{K}}^{n}$, $\mathcal{M} = \alpha_{\mathcal{M}}^{*}$ and $\mathcal{C} = \alpha_{\mathcal{C}}^{*}$,
- $\alpha_{\mathcal{K}} = \alpha_{\mathcal{M}} = \{0,1\}$, $\alpha_{\mathcal{C}} = \{0,1\}^{\ell}$ with $\mathcal{K} = \alpha_{\mathcal{K}}^{n}$, $\mathcal{M} = \alpha_{\mathcal{M}}^{*}$ and $\mathcal{C} = \alpha_{\mathcal{C}}^{*}$

## What are spaces $\mathcal{K}, \mathcal{MC}, \mathcal{T}$? (II)

We may also consider the English alphabet $\alpha = \{\mathsf{a}, \ldots, \mathsf{z}\}$. (Usually historical ciphers consider alphabets in the traditional sense of the word, and contemporary cryptography the binary alphabet.)

Consider a SKE and let $\alpha_{\mathcal{K}}, \alpha_{\mathcal{M}}$ and $\alpha_{\mathcal{C}}$ be the alphabets for the key space $\mathcal{K}$, the message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ respectively. Two common examples might be

- $\alpha_{\mathcal{K}} = \alpha_{\mathcal{M}} = \alpha_{\mathcal{C}} = \{\mathsf{a}, \ldots, \mathsf{z}\}$ with $\mathcal{K} = \alpha_{\mathcal{K}}^n$, $\mathcal{M} = \alpha_{\mathcal{M}}^*$ and $\mathcal{C} = \alpha_{\mathcal{C}}^*$,
- $\alpha_{\mathcal{K}} = \alpha_{\mathcal{M}} = \{0,1\}$, $\alpha_{\mathcal{C}} = \{0,1\}^\ell$ with $\mathcal{K} = \alpha_{\mathcal{K}}^n$, $\mathcal{M} = \alpha_{\mathcal{M}}^*$ and $\mathcal{C} = \alpha_{\mathcal{C}}^*$

(Keep these in mind and try and spot them throughout the course!)

# Enc **and** Dec **as functions**

Let $\mathcal{M} = \{m_1, \ldots, m_s\}$ and $\mathcal{C} = \{c_1, \ldots, c_t\}$ for $t \geq s$,[2] and fix a $sk \in \mathcal{K}$. (We may write $\mathrm{Enc}_{sk}(\cdot)$ and $\mathrm{Dec}_{sk}(\cdot)$ instead of $\mathrm{Enc}(sk, \cdot)$ and $\mathrm{Dec}(sk, \cdot)$ in this case.)

[2] In problem sheet two you will show that Enc must be injective and therefore that $t \geq s$ is necessary.

# Enc and Dec as functions

Let $\mathcal{M} = \{m_1, \ldots, m_s\}$ and $\mathcal{C} = \{c_1, \ldots, c_t\}$ for $t \geq s$,[2] and fix a $sk \in \mathcal{K}$. (We may write $\mathrm{Enc}_{sk}(\cdot)$ and $\mathrm{Dec}_{sk}(\cdot)$ instead of $\mathrm{Enc}(sk, \cdot)$ and $\mathrm{Dec}(sk, \cdot)$ in this case.)

Then per our definition of SKE, $\mathrm{Enc}_{sk}$: $\mathcal{M} \to \mathcal{C}$ and $\mathrm{Dec}_{sk}$: $\mathcal{C} \to \mathcal{M} \cup \{\perp\}$.

[2] In problem sheet two you will show that Enc must be injective and therefore that $t \geq s$ is necessary.

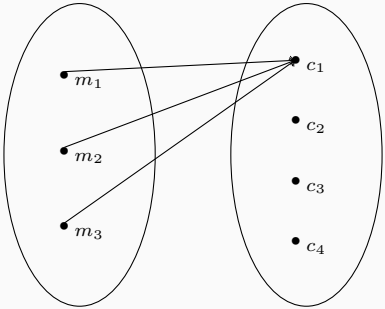
# Enc and Dec as functions (II)

Below we examine $\mathrm{Enc}_{sk}$ as a function.[3]

[3] See problem sheet two for a reminder of the definition of injective and bijective.

# Enc and Dec as functions (II)

Below we examine $\mathrm{Enc}_{sk}$ as a function.[3]

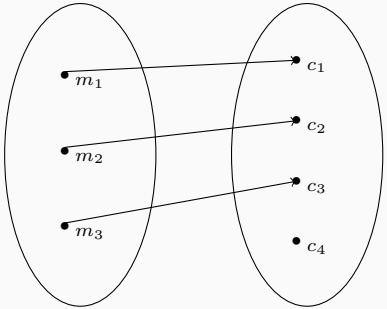



- The number of functions is $t^s$, here $4^3$,

[3] See problem sheet two for a reminder of the definition of injective and bijective.

# Enc and Dec as functions (II)

Below we examine $\mathrm{Enc}_{sk}$ as a function.[3]

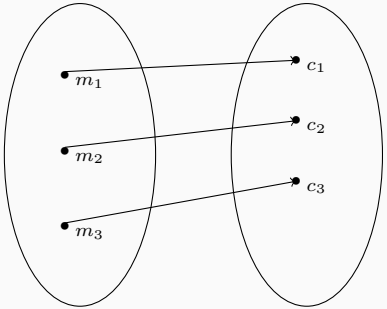



- The number of functions is $t^s$, here $4^3$,
- The number of injective functions is $\binom{t}{s} \cdot s!$, here $\binom{4}{3} \cdot 3!$,

[3] See problem sheet two for a reminder of the definition of injective and bijective.

# Enc and Dec as functions (II)

Below we examine $\mathsf{Enc}_{sk}$ as a function.[3]




- The number of functions is $t^s$, here $4^3$,
- The number of injective functions is $\binom{t}{s} \cdot s!$, here $\binom{4}{3} \cdot 3!$,
- If $t = s$, so that an injective function is a bijection, we have $\binom{t}{s} = 1$ and the number of injective functions is $s!$, here $3!$.

[3] See problem sheet two for a reminder of the definition of injective and bijective.

# Kerckhoffs' principle(s)

From a military manual,[4] and considered an early example of cryptographic thought.

- the system must be substantially, if not mathematically, indecipherable;
- **the system must not require secrecy and can be stolen by the enemy without causing trouble**;
- it must be easy to communicate and retain the key without the aid of written notes, and easy to modify the key at the discretion of the correspondents;
- the system ought to be compatible with telegraph communication;
- the system must be portable, and its use must not require more than one person;
- given the circumstances in which the system is applied, it must be easy to use and must neither stress the mind or require the knowledge of a long series of rules.

[4] Auguste Kerckhoffs, 'La cryptographie militaire', Journal des sciences militaires, vol. IX, pp. 5–38, Jan. 1883, pp. 161–191, Feb. 1883, `https://petitcolas.net/kerckhoffs/`.

# Introducing formal security notions for SKE

Intuition: an adversary without $sk$ should not be able to 'learn' anything 'meaningful' about messages, given the ability to

- observe ciphertext(s),
- choose messages to be encrypted and observe the respective ciphertexts,
- choose messages to be encrypted and (certain) messages to be decrypted and observe the respective ciphertexts and messages.

# Introducing formal security notions for SKE

Intuition: an adversary without $sk$ should not be able to 'learn' anything 'meaningful' about messages, given the ability to

- observe ciphertext(s),
- choose messages to be encrypted and observe the respective ciphertexts,
- choose messages to be encrypted and (certain) messages to be decrypted and observe the respective ciphertexts and messages.

How should we actually reason about this? What do 'learn' and 'meaningful' mean?

# Introducing formal security notions for SKE

Intuition: an adversary without $sk$ should not be able to 'learn' anything 'meaningful' about messages, given the ability to

- observe ciphertext(s),
- choose messages to be encrypted and observe the respective ciphertexts,
- choose messages to be encrypted and (certain) messages to be decrypted and observe the respective ciphertexts and messages.

How should we actually reason about this? What do 'learn' and 'meaningful' mean?

We must specify an adversary's abilities and define what they must not be able to do. We do this via cryptographic 'games'.

# Introducing formal security notions for SKE

Intuition: an adversary without $sk$ should not be able to 'learn' anything 'meaningful' about messages, given the ability to

- observe ciphertext(s),
- choose messages to be encrypted and observe the respective ciphertexts,
- choose messages to be encrypted and (certain) messages to be decrypted and observe the respective ciphertexts and messages.

How should we actually reason about this? What do 'learn' and 'meaningful' mean?

We must specify an adversary's abilities and define what they must not be able to do. We do this via cryptographic 'games'.

In each the algorithms, e.g. $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, are public (known to the adversary) but the secret, e.g. $sk$, is not.

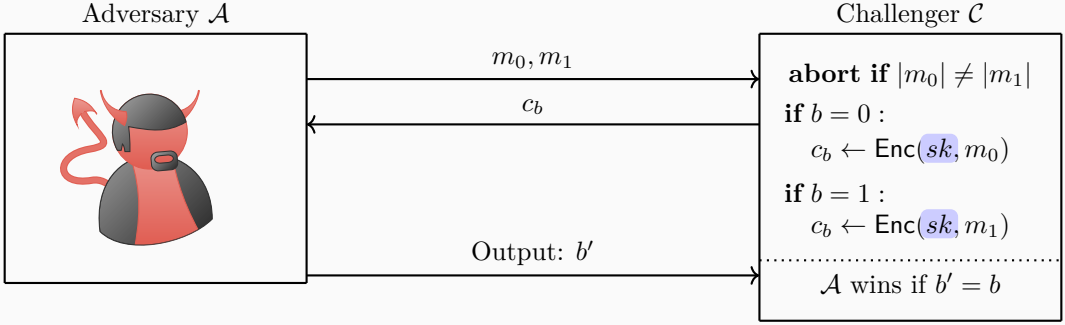
# Introducing formal security notions for SKE = (KGen, Enc, Dec) (IND-EAV)

IND-EAV: INDistinguishability against EAVesdroppers.[5] A challenger $\mathcal{C}$ samples $sk \leftarrow \text{KGen}$ and uniform bit $b \in \{0, 1\}$.

[5] See KL-IMC Section 3.2.1.

# Introducing formal security notions for SKE = (KGen, Enc, Dec) (IND-EAV)

IND-EAV: INDistinguishability against EAVesdroppers.[5] A challenger $\mathcal{C}$ samples $sk \leftarrow \mathsf{KGen}$ and uniform bit $b \in \{0,1\}$.




[5] See KL-IMC Section 3.2.1.

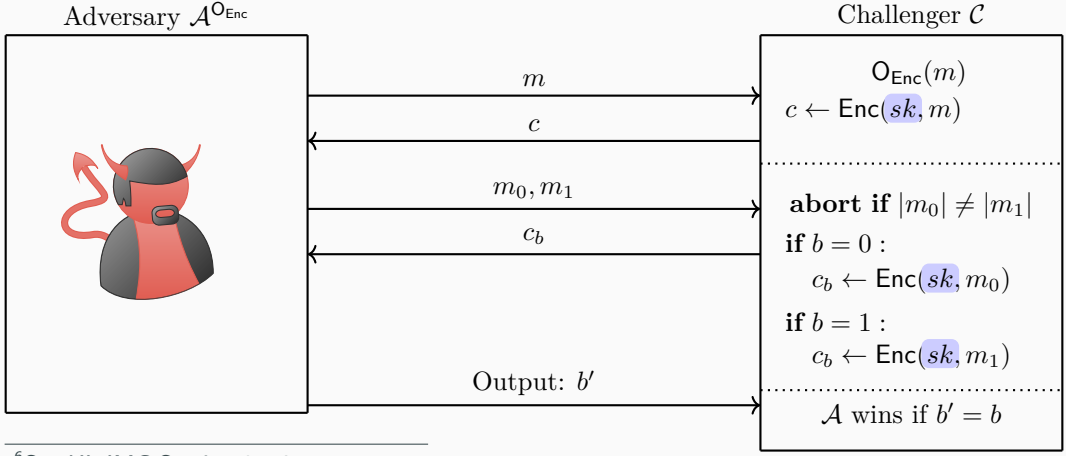
# Introducing formal security notions for SKE $= (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ (IND-CPA)

IND-CPA: INDistinguishability against Chosen Plaintext Attack.[6] A challenger $\mathcal{C}$ samples $sk \leftarrow \mathsf{KGen}$ and uniform bit $b \in \{0, 1\}$.

[6] See KL-IMC Section 3.4.2.

# Introducing formal security notions for SKE = (KGen, Enc, Dec) (IND-CPA)

IND-CPA: INDistinguishability against Chosen Plaintext Attack.[6] A challenger $\mathcal{C}$ samples $sk \leftarrow \mathsf{KGen}$ and uniform bit $b \in \{0,1\}$.




[6] See KL-IMC Section 3.4.2.

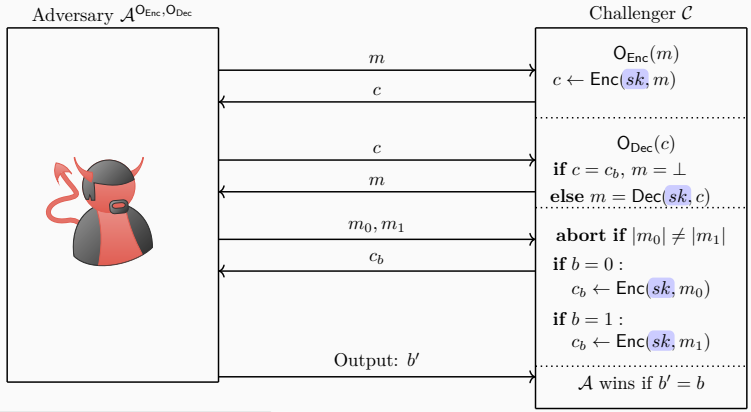
## Introducing formal security notions for SKE $= (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ (IND-CCA)

IND-CCA: INDistinguishability against Chosen Ciphertext Attack.[7] A challenger $\mathcal{C}$ samples $sk \leftarrow \mathsf{KGen}$ and uniform bit $b \in \{0, 1\}$.

[7] See KL-IMC Section 5.1.2.

# Introducing formal security notions for SKE = (KGen, Enc, Dec) (IND-CCA)

IND-CCA: INDistinguishability against Chosen Ciphertext Attack.[7] A challenger $\mathcal{C}$ samples $sk \leftarrow \mathsf{KGen}$ and uniform bit $b \in \{0,1\}$.




[7] See KL-IMC Section 5.1.2.

# Why INDistinguishability?

IND-EAV, IND-CPA and IND-CCA all share IND. Why does this capture ‘not learning anything about messages from ciphertexts?’

# Why INDistinguishability?

IND-EAV, IND-CPA and IND-CCA all share IND. Why does this capture 'not learning anything about messages from ciphertexts?'

Imagine $\mathcal{M} = \{0,1\}^*$ and $\mathcal{A}$ can decide the Hamming weight (number of ones) in $m \in \mathcal{M}$ given $c \leftarrow \mathsf{Enc}(sk, m)$. How does $\mathcal{A}$ win IND-EAV?

# Why INDistinguishability?

IND-EAV, IND-CPA and IND-CCA all share IND. Why does this capture 'not learning anything about messages from ciphertexts?'

Imagine $\mathcal{M} = \{0,1\}^*$ and $\mathcal{A}$ can decide the Hamming weight (number of ones) in $m \in \mathcal{M}$ given $c \leftarrow \mathsf{Enc}(sk, m)$. How does $\mathcal{A}$ win IND-EAV?

Submit $m_0 = 0$ and $m_1 = 1$.

# Why INDistinguishability?

IND-EAV, IND-CPA and IND-CCA all share IND. Why does this capture 'not learning anything about messages from ciphertexts?'

Imagine $\mathcal{M} = \{0,1\}^*$ and $\mathcal{A}$ can decide the Hamming weight (number of ones) in $m \in \mathcal{M}$ given $c \leftarrow \mathsf{Enc}(sk, m)$. How does $\mathcal{A}$ win IND-EAV?

Submit $m_0 = 0$ and $m_1 = 1$.

(If $\mathcal{A}$ requires $\mathsf{O}_{\mathsf{Enc}}$ to learn the Hamming weight $\Rightarrow$ IND-CPA, similar for IND-CCA.)

# IND-$\{$EAV, CPA, CCA$\}$ secure

We say SKE $= (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-$\{$EAV, CPA, CCA$\}$ secure if

- no efficient adversary $\mathcal{A}$,
- wins the IND-$\{$EAV, CPA, CCA$\}$ game against SKE with probability noticeably different from one half.

# IND-$\{$EAV, CPA, CCA$\}$ secure

We say SKE $= (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-$\{$EAV, CPA, CCA$\}$ secure if

- no efficient adversary $\mathcal{A}$,
- wins the IND-$\{$EAV, CPA, CCA$\}$ game against SKE with probability noticeably different from one half.

Why one half?

# IND-{EAV, CPA, CCA} secure

We say SKE $= (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-{EAV, CPA, CCA} secure if

- no efficient adversary $\mathcal{A}$,
- wins the IND-{EAV, CPA, CCA} game against SKE with probability noticeably different from one half.

Why one half?

Because you can always just guess $b$, e.g. flip a coin.

# Introducing formal security notions for MAC

Intuition: an adversary without $ik$ should not be able to forge a tag for a message it has not seen a tag on, given the ability to ask for tags on messages of its choice.

[8] See KL-IMC Section 4.2 paragraph 'Strong unforgeability' and below.

# Introducing formal security notions for MAC

Intuition: an adversary without $ik$ should not be able to forge a tag for a message it has not seen a tag on, given the ability to ask for tags on messages of its choice.

(Like IND-EAV, IND-CPA and IND-CCA for SKE, there are different strengths of security definition for a MAC. We are not concerned with them in this course.[8])

[8] See KL-IMC Section 4.2 paragraph 'Strong unforgeability' and below.

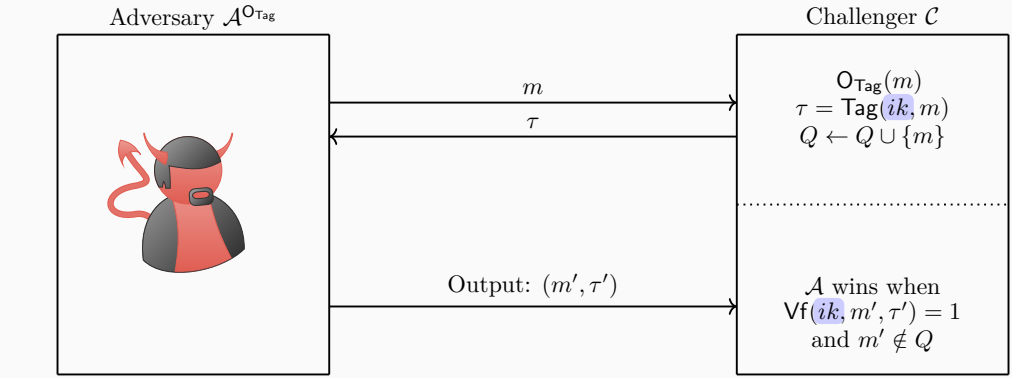
# Introducing formal security notions for MAC (EUF-CMA)

EUF-CMA: Existential UnForgability against Chosen Message Attack.[9] A challenger $\mathcal{C}$ samples $ik \leftarrow \mathsf{KGen}$ and initialises $Q = \emptyset$.

[9] See KL-IMC Section 4.2.

# Introducing formal security notions for MAC (EUF-CMA)

EUF-CMA: Existential UnForgability against Chosen Message Attack.[9] A challenger $\mathcal{C}$ samples $ik \leftarrow \mathsf{KGen}$ and initialises $Q = \emptyset$.




[9] See KL-IMC Section 4.2.

# Why Existential UnForgeability?

Why does Existential UnForgeability capture ‘not being able to forge a tag for a new message’?

# Why Existential UnForgeability?

Why does Existential UnForgeability capture 'not being able to forge a tag for a new message'?

Imagine having seen $(m_1, \tau_1), \ldots, (m_n, \tau_n)$ that $\mathcal{A}$ can forge a tag $\tau'$ for $m' \notin \{m_1, \ldots, m_n\}$. How does $\mathcal{A}$ win EUF-CMA?

# Why Existential UnForgeability?

Why does Existential UnForgeability capture 'not being able to forge a tag for a new message'?

Imagine having seen $(m_1, \tau_1), \ldots, (m_n, \tau_n)$ that $\mathcal{A}$ can forge a tag $\tau'$ for $m' \notin \{m_1, \ldots, m_n\}$. How does $\mathcal{A}$ win EUF-CMA?

Submit $m_1, \ldots, m_n$ to $\mathsf{O}_{\mathsf{Tag}}$ to receive $\tau_1, \ldots, \tau_n$ and then output $(m', \tau')$.

# EUF-CMA secure

We say $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vf})$ is EUF-CMA secure if

- no efficient adversary $\mathcal{A}$,
- wins the EUF-CMA game against MAC with probability noticeably different from zero.

# EUF-CMA secure

We say $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vf})$ is EUF-CMA secure if

- no efficient adversary $\mathcal{A}$,
- wins the EUF-CMA game against MAC with probability noticeably different from zero.

Why zero?

# EUF-CMA secure

We say $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vf})$ is EUF-CMA secure if

- no efficient adversary $\mathcal{A}$,
- wins the EUF-CMA game against MAC with probability noticeably different from zero.

Why zero?

In this game $\mathcal{A}$ tries to 'create' something (a forged tag) rather than distinguish between two possibilities, each occurring with probability one half.

## Which adversaries?

- IND-EAV, IND-CPA, IND-CCA say ‘no efficient adversary should win with probability noticeably different to one half’,
- EUF-CMA says ‘no efficient adversary should win with probability noticeably different to zero’.

[10] See KL-IMC Section 3.1 and especially 3.1.2.

# Which adversaries?

- IND-EAV, IND-CPA, IND-CCA say ‘no efficient adversary should win with probability noticeably different to one half’,
- EUF-CMA says ‘no efficient adversary should win with probability noticeably different to zero’.

Efficient is shorthand for ‘PPT’: probabilistic polynomial time, i.e. a machine that has access to randomness and runs in time polynomial in the size of its input.

Noticeably different to is shorthand for negligible, i.e. any function that is eventually dominated by any inverse polynomial.

(Formal definitions are not required for this course.[10])

[10] See KL-IMC Section 3.1 and especially 3.1.2.

# Thanks

Please as ever bring your questions to the LGT and, better yet, also ask them beforehand in the course forum.