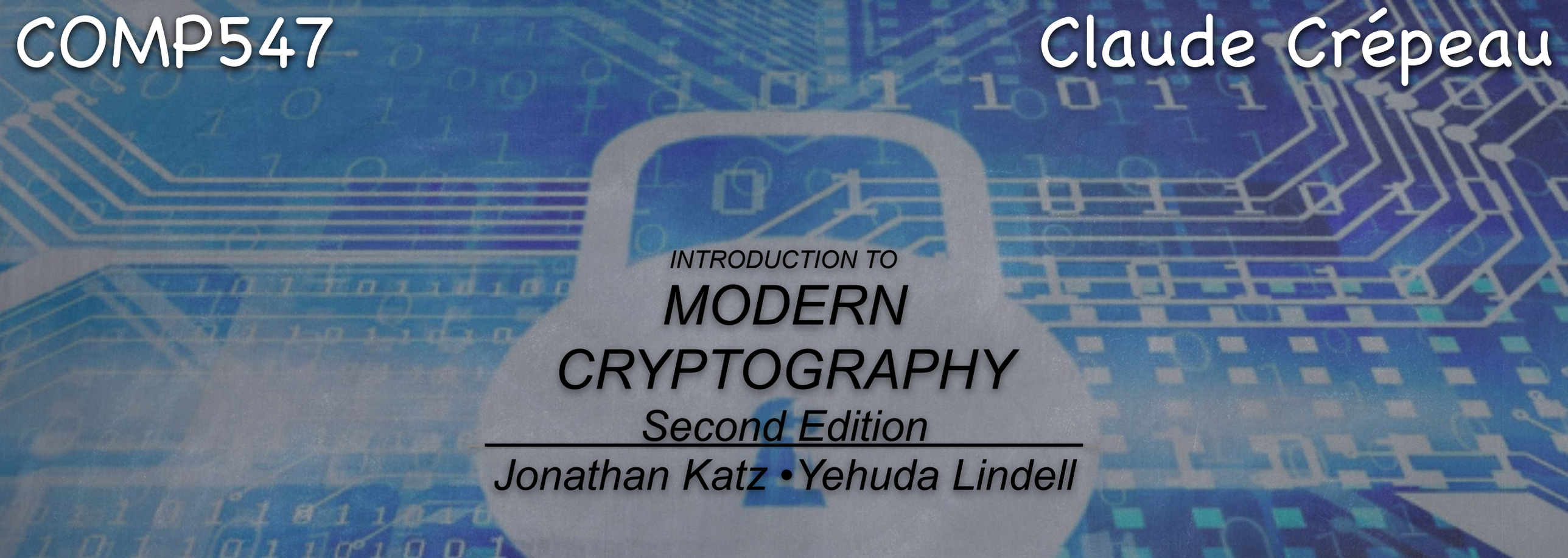
COMP547 Claude Crépeau



Chapter 4 : Message Authentication Codes

Message Authentication Codes and Collision-Resistant Hash Functions

~~4.1 Message Integrity~~
4.6 Information Theoretical Authentication
4.2 Message Authentication Codes – Definitions
4.3 Constructing Secure Message Authentication Codes
4.4 CBC-MAC
4.5 Authenticated Encryption
4.8 Constructing CCA-Secure Encryption Schemes
4.9 Obtaining Privacy and Message Authentication

4.2 Definition – Message Authentication Codes

The aim of a message authentication code is to prevent an adversary from modifying a message sent by one party to another, without the parties detecting that a modification has been made.

This is only possible if the communicating parties have some secret that the adversary does not know (otherwise nothing can prevent an adversary from impersonating the party sending the message).

Definition – Message Authentication Codes

Here, we will continue to consider the private-key setting where the parties share the same secret key.

The syntax of a **MAC**

- Before formally defining security of a message authentication code (**MAC**), we first define what a **MAC** is and how it is used.

- Two users who wish to communicate in an authenticated manner begin by generating and sharing a secret key $\boldsymbol{k} \in \mathcal{K}$ in advance of their communication.

- When one party wants to send a message $\boldsymbol{m} \in M$ to the other, she computes a **MAC** tag (or simply a tag) $\boldsymbol{t} \in \mathcal{T}$ based on the message and the shared key, and sends the message $\boldsymbol{m}$ along with the tag $\boldsymbol{t}$ to the other party.

The syntax of a MAC

- The tag is computed using a *tag-generation algorithm* that will be denoted by **Mac**; rephrasing what we have already said, the sender of a message m computes $t \leftarrow \mathbf{Mac}_k(m)$ and transmits (m, t) to the receiver.

- Upon receiving (m,t), the second party *verifies* whether t is a valid tag on the message m (with respect to the shared key) or not.

- This is done by running a *verification algorithm* **Vrfy** that takes as input the shared key as well as a message m and a tag t, and indicates whether the given tag is valid.

Chapter 2/4
Perfect Message Authentication Codes -- Definitions of Chap. 4 in the style of Chap. 2

The syntax of a MAC

DEFINITION 4.0 *A* message authentication code (*or* **MAC**) *is a tuple of algorithms* **(Gen, Mac, Vrfy)** *such that :*

1. The key-generation algorithm **Gen** *outputs a* $\boldsymbol{k} \in \mathcal{K}$.

2. The tag-generation algorithm **Mac** *takes as input a key* $\boldsymbol{k} \in \mathcal{K}$ *and a message* $\boldsymbol{m} \in M$, *and outputs a tag* $\boldsymbol{t} \in \mathcal{T}$. *Since this algorithm may be randomized, we write this as* $\boldsymbol{t} \leftarrow \mathbf{Mac}_k(\boldsymbol{m})$.

(Not in book) NIB

The syntax of a MAC

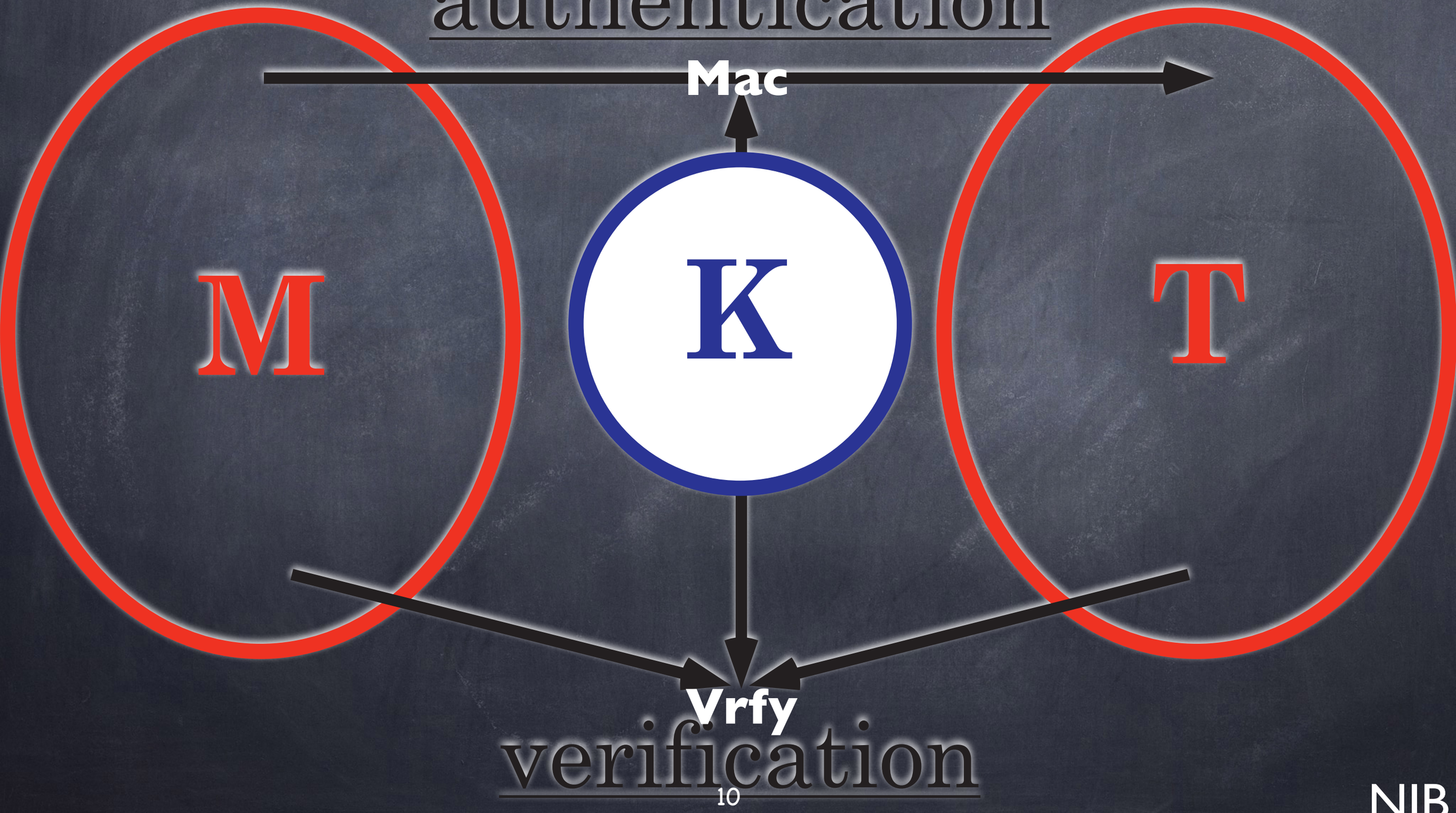
2. *The* tag-generation algorithm **Mac** *takes as input a key* $\boldsymbol{k} \in \mathcal{K}$ *and a message* $\boldsymbol{m} \in M$*, and outputs a tag* $\boldsymbol{t} \in \mathcal{T}$*. Since this algorithm may be randomized, we write this as* $\boldsymbol{t} \leftarrow \mathbf{Mac}_k(\boldsymbol{m})$.

3. *The* verification algorithm **Vrfy** *takes as input a key* $\boldsymbol{k} \in \mathcal{K}$*, a message* $\boldsymbol{m} \in M$*, and a tag* $\boldsymbol{t} \in \mathcal{T}$*. It outputs a bit* $\boldsymbol{b}$*, with* $\boldsymbol{b} = \mathbf{1}$ *meaning* valid *and* $\boldsymbol{b} = \mathbf{0}$ *meaning* invalid. *We assume without loss of generality that* **Vrfy** *is deterministic, and so write this as* $\boldsymbol{b} := \mathbf{Vrfy}_k(\boldsymbol{m}, \boldsymbol{t})$.

NIB

symmetric authentication

authentication



NIB

The syntax of a MAC

It is required that for every key $k \in \mathcal{K}$ *output by* **Gen**, *and every* $m \in M$, *it holds that*

$$\mathbf{Vrfy}_k(m, \mathbf{Mac}_k(m)) = 1 \ .$$

NIB

Security of MAC

We now define the notion of security for message authentication codes.

The intuitive idea behind the definition of security is that no adversary should be able to generate a valid tag on any "new" message that was not previously sent (and authenticated) by one of the communicating parties.

NIB

Security of MAC

To formalize this notion we have to define both the adversary's power as well as what is considered a "break".

As usual, we consider all adversaries and so the real question with regard to the power of the adversary is how we model the adversary's interaction with the communicating parties.

In the setting of message authentication, an adversary observing the communication between the honest parties will be able to see all the messages sent by these parties along with their corresponding **MAC** tags.

NIB

Security of MAC

- The adversary may also be able to influence the *content* of these messages, either indirectly, or directly.

- As an example of the latter, consider the case where the adversary is the personal assistant of one of the parties and has significant control over what messages this party sends.

Security of MAC

To model the above possibilities, we allow the adversary to request **MAC** tags for *any* messages of its choice.

Formally, we give the adversary access to a ***MAC*** *oracle* $\mathbf{Mac}_k(\cdot)$; the adversary can submit any message $m \in M$ that it likes to this oracle, and is given in return a tag $t \leftarrow \mathbf{Mac}_k(m)$.

Security of MAC

We will consider it a "break" of the scheme if the adversary is able to output any message ***m*** along with a tag ***t*** such that:

(1) ***t*** is a valid tag on the message ***m*** i.e., $\mathbf{Vrfy}_k(m, t)=1$

(2) the adversary had not previously requested a **MAC** tag on the message ***m*** (i.e., from its oracle).

Security of MAC

- Adversarial success in the first condition means that, in the real world, if the adversary were to send (m, t) to one of the honest parties, then this party would be mistakenly fooled into thinking that m originated from the legitimate party (since $\mathbf{Vrfy}_k(m, t) = 1$).

- The second condition is required because it is always possible for the adversary to just copy a message and **MAC** tag that was previously sent by the legitimate parties (and, of course, these would be accepted).

Security of MAC

- Such an adversarial attack is called a *replay attack* and is not considered a "break" of the message authentication code.

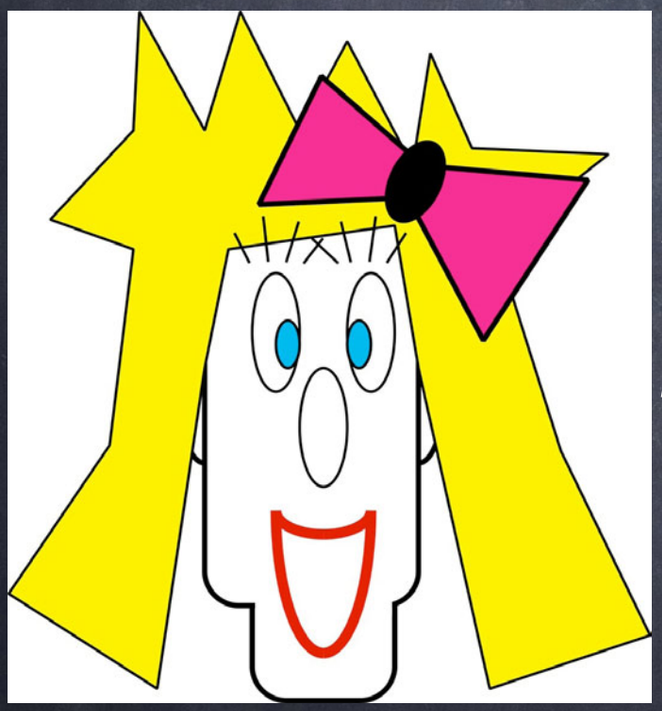
- This does *not* mean that replay attacks are not a security concern; they are, and we will have more to say about this further on.

Security of **MAC**

- A **MAC** satisfying the level of security specified above is said to be *existentially unforgeable under an adaptive chosen-message attack*.

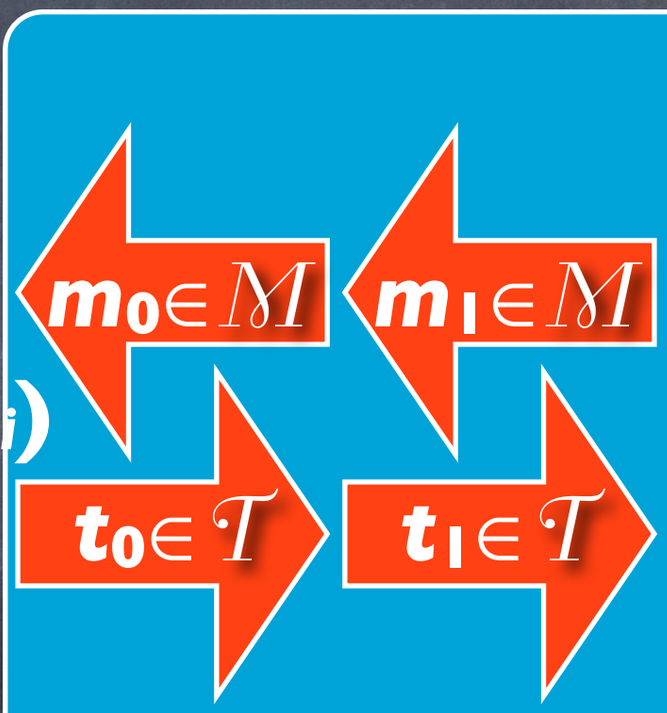
- "Existential unforgeability" refers to the fact that the adversary must not be able to forge a valid tag on *any* message, and "adaptive chosen-message attack" refers to the fact that the adversary is able to obtain **MAC** tags on any message it likes, where these messages may be chosen adaptively during its attack.

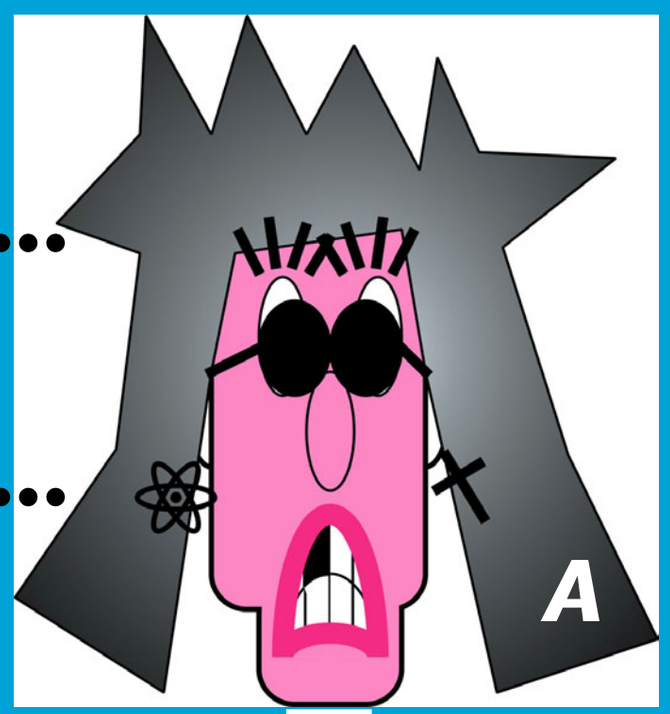
$\mathbf{Mac\text{-}Forge}^{aCPA}_{A,\Pi}$



$k \leftarrow \mathbf{Gen}$

$t_i \leftarrow \mathbf{Mac}_k(m_i)$





existential unforgeability

$\mathbf{Pr}[\,\mathbf{Vrfy}_k(m,t)=1\,] = 1/|\mathcal{T}|$

m,t

$m \neq m_i$

Perfect Security of MAC

Message authentication experiment $\mathbf{Mac\text{-}Forge}^{\mathrm{aCPA}}_{A,\Pi}$:

1. A random key $\boldsymbol{k}$ *is generated by running* **Gen**.

2. The adversary $\boldsymbol{A}$ *is given oracle access to* $\mathbf{Mac}_k(\cdot)$.
The adversary eventually outputs a pair $(\boldsymbol{m}, \boldsymbol{t})$.
Let Q *denote the set of all queries that* $\boldsymbol{A}$ *asked to its oracle.*

3. The output of the experiment is defined to be **1** *iff*

(1) $\mathbf{Vrfy}_k(\boldsymbol{m}, \boldsymbol{t}) = 1$ (2) $\boldsymbol{m} \notin Q$.

NIB

Perfect Security of MAC

DEFINITION 4.00 *A message authentication code* **Π = (Gen, Mac, Vrfy)** *is* existentially unforgeable under an adaptive chosen-message attack, *or just* secure, *if for all adversaries* ***A*** *:*

$$\mathbf{Pr}[\mathbf{Mac\text{-}Forge}^{\mathbf{aCPA}}_{\mathbf{A},\Pi} = 1] = 1/|\mathcal{T}|.$$

NIB

Is the definition too strong?

- The above definition is rather strong, in two respects.

- First, the adversary is allowed to request a **MAC** tag for *any* message of its choice.

- Second, the adversary is considered to have "broken" the scheme if it can output a valid tag on *any* previously-unauthenticated message.

Is the definition too strong?

One might object that both of these components of the definition are unrealistic and overly strong: in "real-world" usage of a **MAC**, the honest parties would only authenticate "meaningful" messages, and similarly it should only be considered a breach of security if the adversary can forge a valid tag on a "meaningful" message.

Why not tailor the definition to capture this ?

Is the definition too strong?

The crucial point is that what constitutes a meaningful message is entirely *application-dependent*.

While some applications of a **MAC** may only ever authenticate English-text messages, other applications may authenticate spreadsheet files, others database entries, and others raw data.

Protocols may also be designed where *anything* will be authenticated — in fact, certain protocols for entity authentication do exactly this.

Is the definition too strong?

- By making the definition of security for **MACs** as strong as possible, we ensure that secure **MACs** are broadly applicable for a wide range of purposes, without having to worry about compatibility of the **MAC** with the semantics of the application.

Replay attacks

- We emphasize that the previous definition, and message authentication codes in general, offer no protection against *replay attacks* in which a previously-sent message (and its **MAC** tag) are replayed to one of the honest parties.
- Nevertheless, replay attacks are a serious concern.

Replay attacks

Despite the real threat due to replay attacks, a **MAC** inherently *cannot* protect against such attacks since the definition of a **MAC** (**Definitions 4.0** & **4.1**) does not incorporate any notion of *state* into the verification algorithm (and so every time a valid pair (m, t) is presented to the verification algorithm, it will always output **1**).

Rather, protection against replay attacks — if such protection is necessary at all — is left to some higher-level application.

Replay attacks

- The reason the definition of a **MAC** is structured this way is, once again, because we are unwilling to assume any semantics regarding applications that use **MAC**s;

- In particular, the decision as to whether or not a replayed message should be treated as "valid" is considered to be entirely application-dependent.

Replay attacks

Two common techniques for preventing replay attacks involve the use of *sequence numbers* or *time-stamps*.

The basic idea of the first approach is that each message ***m*** is assigned a sequence number ***i***, and the **MAC** tag is computed over the concatenated message $i \| m$.

It is assumed here that the sender always assigns a unique sequence number to each message, and that the receiver keeps track of which sequence numbers it has already seen.

Replay attacks

- Now, any successful replay of a message ***m*** will have to forge a valid **MAC** tag on a new *concatenated* message $i' \| m$, where i' (thus $i' \| m$) has never been used before.

- This is ruled out by the security of the **MAC**.

Replay attacks

A disadvantage of using sequence numbers is that the receiver must store a list of all previous sequence numbers it has received.

To alleviate this, time-stamps are sometimes used to similar effect.

Here, the sender essentially appends the current time to the message (say, to the nearest ms) rather than a sequence number.

Replay attacks

- When the receiver obtains a message, it checks whether the included time-stamp is within some acceptable window of the current time.

- This method has certain drawbacks as well, including the need for the sender and receiver to maintain closely-synchronized clocks, and the possibility that a replay attack can still take place as long as it is done quickly enough.

Chapter 2/4
Perfect Message Authentication Codes -- Definitions of Chap. 4 in the style of Chap. 2

Perfect Security of MAC

Message authentication experiment $\mathbf{Mac\text{-}Forge}^{\mathbf{1\text{-}time}}_{A,\Pi}$:

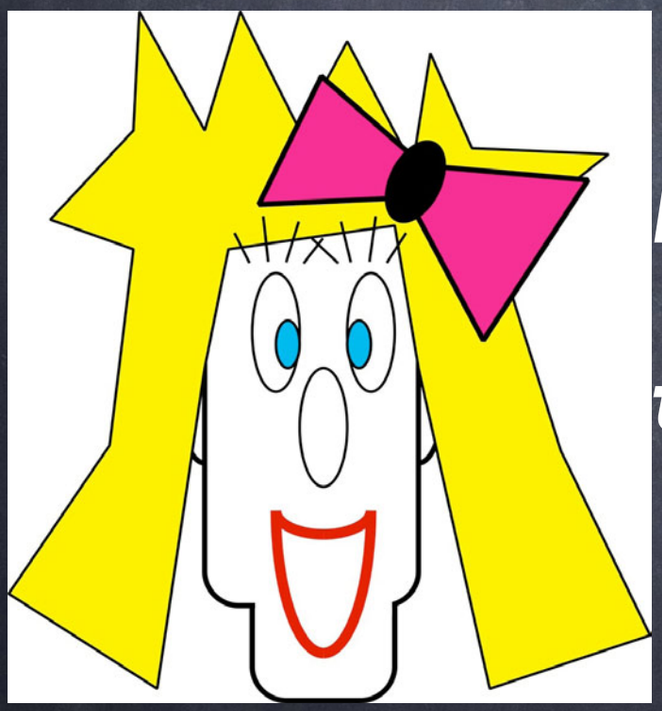
1. A random key $\boldsymbol{k}$ *is generated by running* **Gen**.

2. The adversary $\boldsymbol{A}$ *is given oracle access to* $\mathbf{Mac}_k(\cdot)$.
The adversary eventually outputs a pair $(\boldsymbol{m}, \boldsymbol{t})$.
Let Q *denote the set of all queries that* $\boldsymbol{A}$ *asked to its oracle.*

3. The output of the experiment is defined to be **1** *if and only if*

(1) $\mathbf{Vrfy}_k(\boldsymbol{m}, \boldsymbol{t}) = \mathbf{1}$ (2) $\boldsymbol{m} \notin Q$ (3) $|Q| \leq \mathbf{1}$.

Mac-Forge$^{I\text{-}time}_{A,\Pi}$

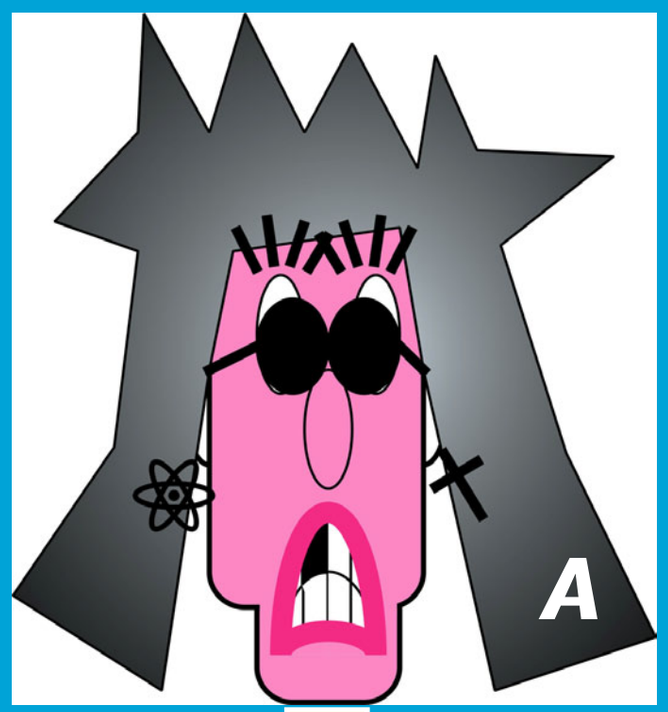


$k \leftarrow \mathbf{Gen}$

$t_0 \leftarrow \mathbf{Mac}_k(m_0)$

$m_0 \in \mathcal{M}$

$t_0 \in \mathcal{T}$



m,t

$m \neq m_0$

existential unforgeability

$\mathbf{Pr}[\,\mathbf{Vrfy}_k(m,t)=1\,] = 1/|\mathcal{T}|$

Perfect Security of MAC

DEFINITION 4.22 *A message authentication code* **Π = (Gen, Mac, Vrfy)** *is* existentially unforgeable under a single chosen-message attack, *or just* secure, *if for all adversaries* ***A*** *:*

$$\mathbf{Pr[Mac\text{-}Forge}^{1\text{-}time}_{A,\Pi} = 1] = 1/|\mathcal{T}|.$$

Existential Unforgeability

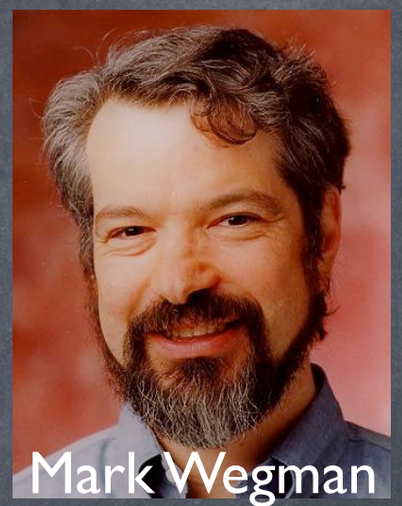
PROPOSITION 4.01 *Let* **Π = (Gen, Mac, Vrfy)** *be an authentication scheme over a message space* $\mathcal{M}$.

Π *is perfectly secret with respect to* ***Definition 4.00*** *if it is perfectly secret with respect to* ***Definition 4.22*** *and a new key is generated for each authentication.*

NIB

The One-Time MAC (Wegman-Carter Authentication)


Larry Carter


Mark Wegman

- In 1981, Carter and Wegman invented the one-time **MAC**.
- For this purpose, they define the useful notion of Strongly Universal$_2$ families of hash functions.

Strongly Universal$_2$

Definition 4.23 (Strongly Universal$_2$)
Let $\mathcal{H}$ be a set of hash functions from $\mathcal{M}$ to $\mathcal{T}$.
$\mathcal{H}$ is Strongly Universal$_2$ if
for all m_1, m_2, distinct elements of $\mathcal{M}$,
for all m_1', m_2', distinct elements of $\mathcal{M}$, and
for all t_1, t_2, t_1', t_2', (not necessarily distinct) elements of $\mathcal{T}$,
we have

$$|\{h \in \mathcal{H} : h(m_1) = t_1, h(m_2) = t_2\}| =$$
$$|\{h \in \mathcal{H} : h(m_1') = t_1', h(m_2') = t_2'\}| = |\mathcal{H}| / |\mathcal{T}|^2$$

Strongly Universal$_2$

Remark: An equivalent definition is the following, $\mathcal{H}$ is Strongly Universal$_2$ if for any ***h*** picked uniformly at random from $\mathcal{H}$ we have that

1. $\forall m, m' \in \mathcal{M}$, $t, t' \in \mathcal{T}$
$\Pr[h(m)=t] = \Pr[h(m')=t'] = 1/|\mathcal{T}|$

2. $\forall m_1 \neq m_2, m_1' \neq m_2' \in \mathcal{T}$, $t_1, t_2, t_1', t_2' \in \mathcal{T}$
$\Pr[h(m_2)=t_2 \mid h(m_1)=t_1] =$
$\Pr[h(m_2')=t_2' \mid h(m_1')=t_1'] = 1/|\mathcal{T}|$.

Strongly Universal$_2$

We first define the following class:

$\mathcal{H}_2 = \{h : \mathcal{F}_q \to \mathcal{F}_q \mid h(m) = am + b$ for some $a, b \in \mathcal{F}_q\}$

here, $M = \mathcal{T} = \mathcal{F}_q$, $|M| = |\mathcal{T}| = q$, $|\mathcal{H}_2| = q^2$.

Theorem 4.26 $\mathcal{H}_2$ is Strongly Universal$_2$.

Strongly Universal$_2$

Proof. Consider $m \neq m'$, and two outputs t,t',

$$\begin{array}{rr} & am + b = t \\ - & am' + b = t' \\ \hline \end{array}$$

$$a(m - m') = (t - t')$$

$$\Leftrightarrow a = (t - t')(m - m')^{-1}$$

($(m - m')^{-1}$ exists and is unique when $m \neq m'$) and so

$$\Leftrightarrow b = t - am = t - (t - t')(m - m')^{-1}m.$$

Strongly Universal$_2$

These values of a and b define a unique h such that $h(m) = t$, $h(m') = t'$. We thus have that

$\forall m \neq m', t, t'$

$|\{ h : h(m) = t,\ h(m') = t' \}| = 1 = |H_2| / |T|^2$

The One-Time MAC (Wegman-Carter Authentication)

One-time **MAC** is defined as follows:

1. Fix a prime power q. Then the message space $\mathcal{M}$, and tag space $\mathcal{T}$ are $\mathbb{F}_q$, while key space $\mathcal{K} = \mathbb{F}_q^2$.

2. The key-generation algorithm **Gen** selects a pair of values a, b uniformly from $\mathcal{K} = \mathbb{F}_q^2$.

The One-Time MAC
(Wegman-Carter Authentication)

3. Authentication **Mac** : given a key $(a, b) \in \mathbb{F}_q^2$ and a message $m \in \mathbb{F}_q$, output $t := am+b$.

4. Verification **Vrfy** : given a key $(a, b) \in \mathbb{F}_q^2$, a message $m \in \mathbb{F}_q$, and a tag $t \in \mathbb{F}_q$, accept iff $t = am+b$.

The One-Time MAC (Wegman-Carter Authentication)

Before discussing the security of the one-time **MAC**, we note that $\mathbf{Vrfy}_k(\mathbf{Mac}_k(m)) = 1$ (a correct authentication scheme).

The one-time **MAC** is perfectly secret because H_2 is Strongly Universal$_2$.

Strongly Universal$_2$

- In $\mathcal{H}_2$, seeing two message-tag pairs completely determines ***a*** and ***b***.

- Therefore, seeing two message-tag pairs enables to determine all further message-tag pairs.

NIB

The One-Time MAC (Wegman-Carter Authentication)

- Unfortunately, the one-time **MAC** authentication scheme has a number of drawbacks.
- *The key is required to be twice as long as the message.*
  - a very long key must be securely stored,
  - the error probability of $1/|M|$ may be a serious overkill. We may be happy with, say, $1/2^{50}$.
- As the name indicates — One-time **MAC** *is only "secure" if keys are used only once.*

NIB

Another Strongly Universal$_2$ Family

In this case, we use the following class:

$\mathcal{H}_M =$

$\{h : (\mathbb{F}_p)^M \to \mathbb{F}_p \mid h(m) = a \odot m + b \bmod p,\ a \in (\mathbb{F}_p)^M,\ b \in \mathbb{F}_p\}$

where $M \subseteq (\mathbb{F}_p)^M, |M| \leq p^M, \mathcal{T} = \mathbb{F}_p, |\mathcal{T}| = p$.

Theorem 4.04 $\mathcal{H}_M$ is Strongly Universal$_2$.

Where $a \odot m$ means $\sum a_i m_i$ over the field $\mathbb{F}_p$.

NIB

Perfect Security of MAC

Assume we wish to authenticate (with error probability at most ϵ) at most ℓ messages from the set $M = (\mathbb{F}_p)^M$, and let $M' = (\mathbb{F}_p)^{M+\log_p \ell}$, for a prime $p \geq 1/\epsilon$.

Let $\mathcal{H}_{M+\log_p \ell}$ be a Strongly Universal$_2$ set of functions from M' to $T = \mathbb{F}_p$ as previously.

To each message in M that we send, we append a unique index number i from $(\mathbb{F}_p)^{\log_p \ell}$, and interpret the result as an element of M'. (The purpose of these indices is to detect replay attacks as well as deletion and insertion of messages).

NIB

The One-Time MAC (Wegman-Carter Authentication)

<u>One-time **MAC** is defined as follows:</u>

1. Fix $\mathcal{M} = (\mathbb{F}_p)^{M+\log_p \ell}$. The tag space $\mathcal{T} = \mathbb{F}_p$, while the key space $\mathcal{K} = (\mathbb{F}_p)^{M+\log_p \ell} \times (\mathbb{F}_p)^{\ell}$.

2. The key-generation algorithm **Gen** selects uniformly a list of values $a_1, a_2, \ldots, a_{M+\log_p \ell}, b_1, b_2, \ldots, b_\ell$ from $\mathcal{K}$.

NIB

The One-Time MAC (Wegman-Carter Authentication)

3. Authentication **Mac** :
given a key (a, b_i) and $m_i' = i \Vert m_i \in (\mathbb{F}_p)^{M+\log_p \ell}$,
output $t_i := a \odot m_i' + b_i \mod p$.

4. Verification **Vrfy** : given a key (a, b_i), with a message $m_i' \in (\mathbb{F}_p)^{M+\log_p \ell}$, and a tag $t_i \in \mathbb{F}_p$, output **1** iff

$$t_i = a \odot m_i' + b_i \mod p.$$

NIB

Existential Unforgeability

PROPOSITION 4.01 *Let* Π_ℓ = **(Gen, Mac, Vrfy)** *be the authentication scheme as above. Then* Π_ℓ *is secure with respect to* ***Definition 4.00*** *as long as* $|Q| < \ell$.

NIB

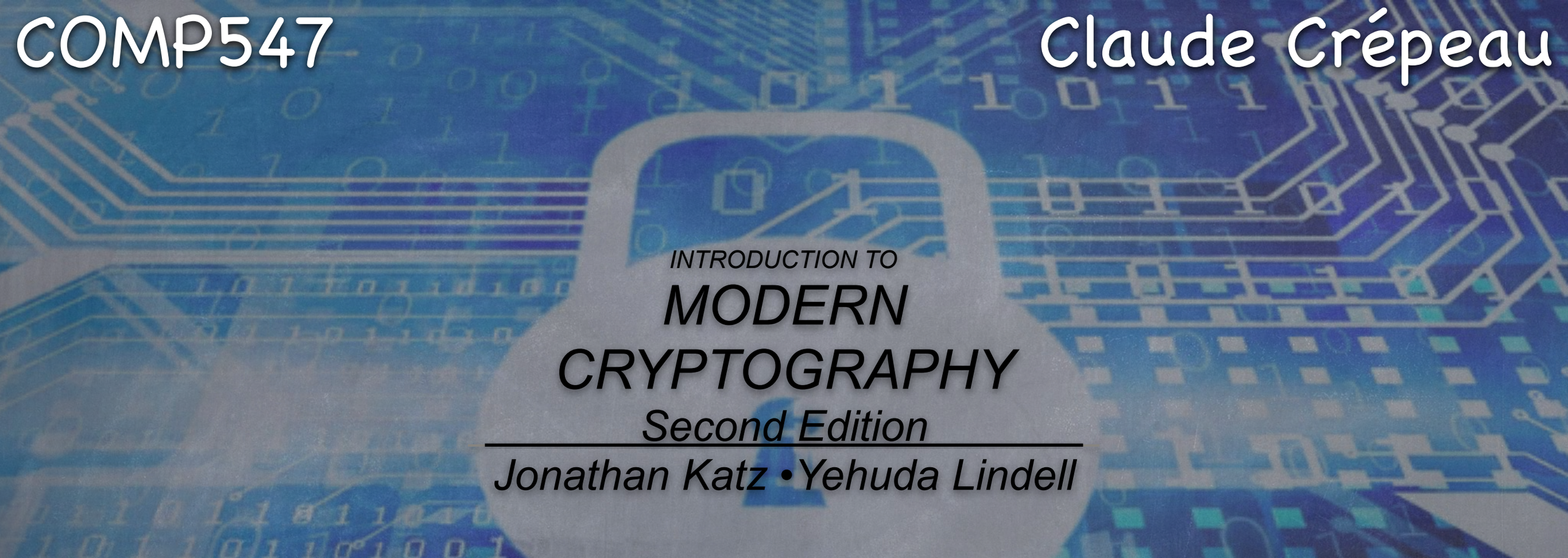
The One-Time MAC (Wegman-Carter Authentication)

- *The key required (per authentication) is of size =*
  tag-size + $(^{\textbf{message-size}}/_{\ell})$ *p-digits* =
  $\mathbf{1} + {}^{M+\log_p \ell}/_{\ell} \sim \mathbf{O(1)}$ *p-digit for large* $\ell \sim M$.
- A rather long key must be securely stored, but is much more efficient than one-time pad.
- If we are flexible on the error probability ($\leq 2\epsilon$), much more efficient families exist.

NIB

TASKS / SECURITY	Encryption	Authentication	Identification	Quantum
Symmetric Informational	Miller-Vernam One-Time PAD	Wegman-Carter Universal Hash	Simple Solutions	Quantum Key Distribution
Symmetric Computational	from PRBG from PRFG DES, AES, etc	from PRBG from PRFG DES, AES, etc	from PRBG from PRFG DES, AES, etc	Quantum Attacks, Q-Safety
Asymmetric Computational	RSA, ElGammal, Blum-Goldwasser	RSA, DSA, etc	Guilloux-Quisquater, Schnor, etc	Quantum Attacks, Q-Safety

DONE	IN PROGRESS	TO DO	GIVE UP



Chapter 4 : Message Authentication Codes

The syntax of a MAC

DEFINITION 4.1 *A* message authentication code (*or* **MAC**) *is a tuple of* ***PPT*** *algorithms* **(Gen, Mac, Vrfy)** *such that:*

1. *The* key-generation algorithm **Gen** *takes as input the security parameter* $\mathbf{1}^n$ *and outputs a key* k *with* $|k| \geq n$.

The syntax of a MAC

2. *The* tag-generation algorithm **Mac** *takes as input a key* k *and a message* $m \in \{0,1\}^*$*, and outputs a tag* t*. Since this algorithm may be randomized, we write this as* $t \leftarrow \mathbf{Mac}_k(m)$.

3. *The* verification algorithm **Vrfy** *takes as input a key* k*, a message* m*, and a tag* t*. It outputs a bit* b*, with* $b = 1$ *meaning* valid *and* $b = 0$ *meaning* invalid. *We assume without loss of generality that* **Vrfy** *is deterministic, and so write this as* $b := \mathbf{Vrfy}_k(m, t)$.

The syntax of a MAC

It is required that for every n*, every key* k *output by* $\mathbf{Gen}(1^n)$*, and every* $m \in \{0,1\}^*$*, it holds that*

$$\mathbf{Vrfy}_k(m, \mathbf{Mac}_k(m)) = 1 \ .$$

If $(\mathbf{Gen}, \mathbf{Mac}, \mathbf{Vrfy})$ *is such that for every* k *output by* $\mathbf{Gen}(1^n)$*, algorithm* $\mathbf{Mac}_k$ *is only defined for messages* $m \in \{0,1\}^{\ell(n)}$ *(and* $\mathbf{Vrfy}_k$ *outputs* 0 *for any* $m \notin \{0,1\}^{\ell(n)}$*), then we say that it is a* fixed-length **MAC** *for length* $\ell(n)$.

Security of MAC

We now define the notion of computational security for message authentication codes.

The intuitive idea behind the definition of security is that no **PPT** adversary should be able to generate a valid tag on any "new" message that was not previously sent (and authenticated) by one of the communicating parties.

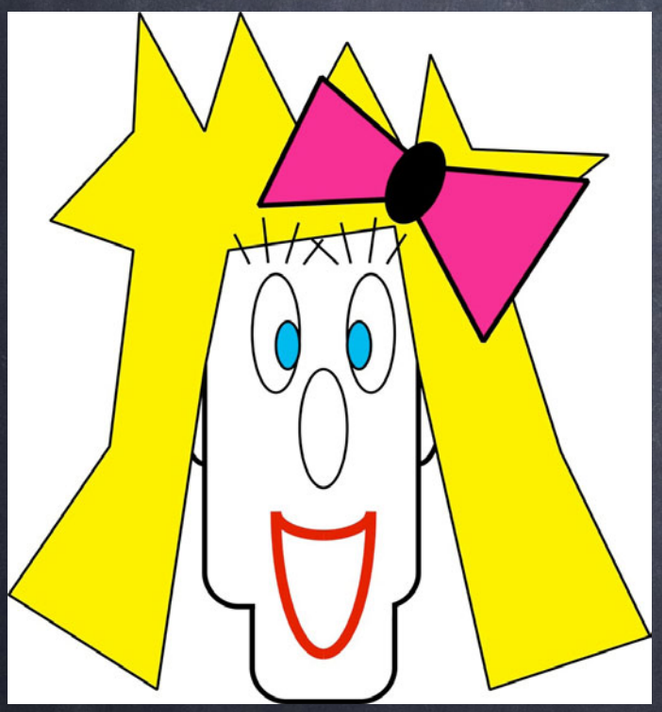
Security of MAC

To formalize this notion we have to define both the adversary's power as well as what is considered a "break".

As usual, we consider only **PPT** adversaries and so the real question with regard to the power of the adversary is how we model the adversary's interaction with the communicating parties.

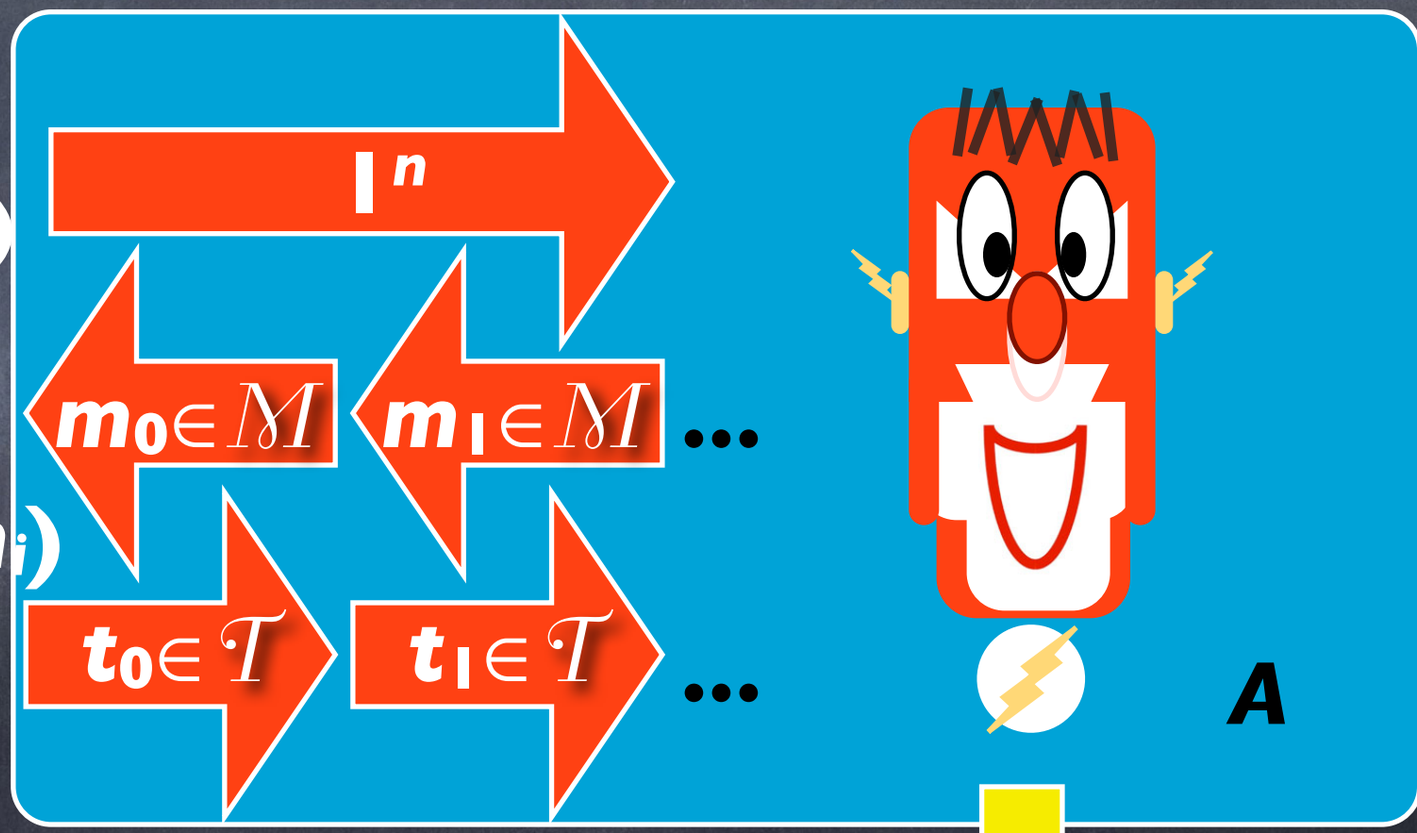
In the setting of message authentication, an adversary observing the communication between the honest parties will be able to see all the messages sent by these parties along with their corresponding **MAC** tags.

$\mathbf{Mac\text{-}Forge}_{A,\Pi}(n)$



$k \leftarrow \mathbf{Gen}(1^n)$

$t_i \leftarrow \mathbf{Mac}_k(m_i)$



existential unforgeability

$\Pr[\mathbf{Vrfy}_k(m, t)=1] \leq \mathbf{negl}(n)$

m,t

$m \neq m_i$

Security of MAC

Message authentication experiment
Mac-forge$_{A,\Pi}(n)$:

1. *A random key* ***k*** *is generated by running* **Gen(1^n)**.

2. *The adversary* ***A*** *is given* 1^n *and oracle access to* **Mac$_k(\cdot)$**.
The adversary eventually outputs a pair (m, t).
Let Q *denote the set of all queries that* ***A*** *asked to its oracle.*

3. *The output of the experiment is defined to be* **1** *iff*

(1) **Vrfy$_k(m, t) = 1$** *and* (2) $m \notin Q$.

Security of MAC

DEFINITION 4.2 *A message authentication code* **Π = (Gen, Mac, Vrfy)** *is* existentially unforgeable under an adaptive chosen-message attack, *or just* secure, *if for all* **PPT** *adversaries* ***A****, there exists a negligible function* **negl** *s.t. :*

$$\mathbf{Pr[\,Mac\text{-}forge}_{A,\Pi}(n) = 1\,] \leq \mathbf{negl}(n).$$

4.4 Constructing Secure MACs

Pseudorandom functions are a natural tool for constructing secure message authentication codes.

Intuitively, if the **MAC** tag ***t*** is obtained by applying a pseudorandom function to the message ***m***, then forging a tag on a previously-unauthenticated message requires the adversary to guess the value of the pseudorandom function at a "new" point (i.e., message).

Constructing Secure MACs

- Now, the probability of guessing the value of a *random* function on a new point is $\mathbf{2^{-n}}$ (when the output length of the function is *n*).

- It follows that the probability of guessing such a value for a *pseudorandom* function can be only negligibly greater.

Constructing Secure MACs

CONSTRUCTION 4.5

Let F be a pseudorandom function. Define a fixed-length MAC for messages of length n as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, output the tag $t := F_k(m)$. (If $|m| \neq |k|$ then output nothing.)
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^n$, and a tag $t \in \{0,1\}^n$, output 1 if and only if $t \stackrel{?}{=} F_k(m)$. (If $|m| \neq |k|$, then output 0.)

A fixed-length MAC from any pseudorandom function.

Constructing Secure MACs

THEOREM 4.6 *If F is a pseudorandom function, then* ***Construction 4.5*** *is a fixed-length* ***MAC*** *for messages of length* ***n*** *that is existentially unforgeable under an adaptive chosen-message attack.*

Extension to Variable-Length Messages

CONSTRUCTION 4.7

Let $\Pi' = (\mathsf{Mac}', \mathsf{Vrfy}')$ be a fixed-length MAC for messages of length n. Define a MAC as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ of (nonzero) length $\ell < 2^{n/4}$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Choose a uniform identifier $r \in \{0,1\}^{n/4}$.

  For $i = 1, \ldots, d$, compute $t_i \leftarrow \mathsf{Mac}'_k(r \| \ell \| i \| m_i)$, where i, ℓ are encoded as strings of length $n/4$.[†] Output the tag $t := \langle r, t_1, \ldots, t_d \rangle$.

- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ of length $\ell < 2^{n/4}$, and a tag $t = \langle r, t_1, \ldots, t_{d'} \rangle$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Output 1 if and only if $d' = d$ and $\mathsf{Vrfy}'_k(r \| \ell \| i \| m_i, t_i) = 1$ for $1 \leq i \leq d$.

[†] Note that i and ℓ can be encoded using $n/4$ bits because $i, \ell < 2^{n/4}$.

Extension to Variable-Length Messages

THEOREM 4.8 *If* **Π′** *is a secure fixed-length* ***MAC*** *for messages of length* ***n****, then* ***Construction 4.7*** *is a* ***MAC*** *that is existentially unforgeable under an adaptive chosen-message attack.*

PROOF Sketch

- The intuition is that as long as Π' is secure, an adversary cannot introduce a new block with a valid tag.
- Furthermore, the extra information included in each block prevents the various attacks (dropping blocks, re-ordering blocks, etc.) sketched earlier.
- We stress that showing that known attacks are thwarted is far from a proof of security.

4.4 CBC-MAC

CONSTRUCTION 4.11

Let F be a pseudorandom function, and fix a length function $\ell > 0$. The basic CBC-MAC construction is as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message m of length $\ell(n) \cdot n$, do the following (we set $\ell = \ell(n)$ in what follows):

  1. Parse m as $m = m_1, \ldots, m_\ell$ where each m_i is of length n.
  2. Set $t_0 := 0^n$. Then, for $i = 1$ to ℓ:
     Set $t_i := F_k(t_{i-1} \oplus m_i)$.

  Output t_ℓ as the tag.

- Vrfy: on input a key $k \in \{0,1\}^n$, a message m, and a tag t, do: If m is not of length $\ell(n) \cdot n$ then output 0. Otherwise, output 1 if and only if $t \stackrel{?}{=} \mathsf{Mac}_k(m)$.

Basic CBC-MAC (for fixed-length messages).

CBC-MAC

THEOREM 4.12 *Let ℓ be a polynomial. If* **F** *is a pseudorandom function, then* **Construction 4.11** *is a fixed-length* **MAC** *for messages of length* $\ell(n)\cdot n$ *that is existentially unforgeable under an adaptive chosen-message attack.*

CBC-MAC

- The proof of **Theorem 4.12** is very involved.
- We stress that even though **Construction 4.11** can be extended in the obvious way to handle messages whose length is an arbitrary multiple of ***n***, the construction is only secure when the length of the messages being authenticated is fixed.
- That is, if an adversary is able to obtain **MAC** tags for messages of varying lengths, then the scheme is no longer secure.

CBC-MAC vs. CBC-mode encryption

There are two differences between the basic **CBC-MAC** and the **CBC** mode of encryption:

1. **CBC**-mode encryption uses a *random* ***IV*** and this is crucial for obtaining security.

In contrast, **CBC-MAC** uses no ***IV*** (or the fixed value $\boldsymbol{IV = 0^n}$) and this is also crucial for obtaining security.

Specifically, **CBC-MAC** using a random **IV** is not secure.

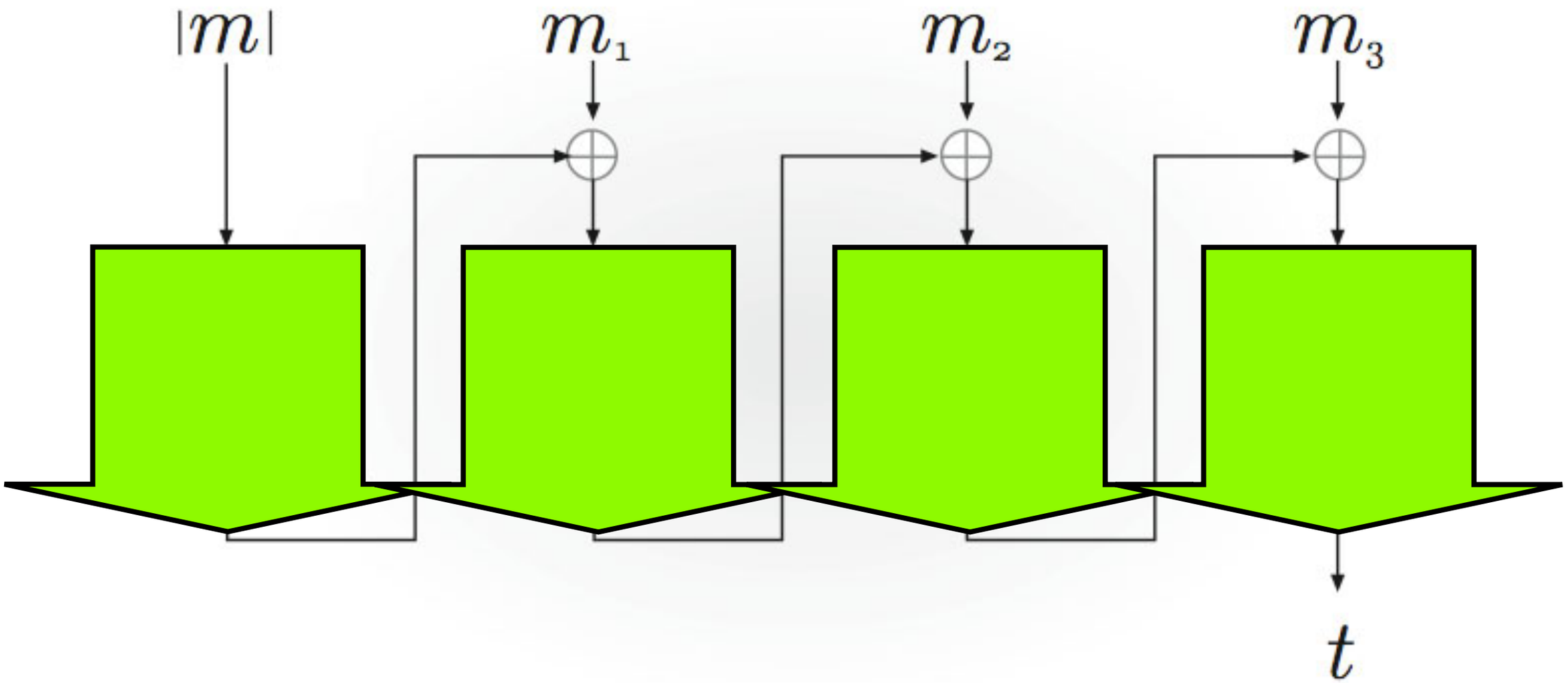
CBC-MAC vs. CBC-mode Encryption

2. In **CBC**-mode encryption all blocks t_i (c_i) are output by the encryption algorithm as part of the ciphertext, whereas in **CBC-MAC** only the final block is output.

This may seem to be a technical difference resulting from the fact that, for the case of encryption, all blocks must be output in order to enable decryption, whereas for a **MAC** this is simply not necessary and so is not done.

However, if **CBC-MAC** is modified to output all blocks then it is no longer secure.

Secure CBC-MAC for variable-length message



Secure CBC-MAC for variable-length message

- In order to obtain a secure version of **CBC-MAC** for variable-length messages, **Construction 4.11** must be modified.
- This can be done in a number of ways. Two possible options that can be proven secure are:

Secure CBC-MAC for variable-length message

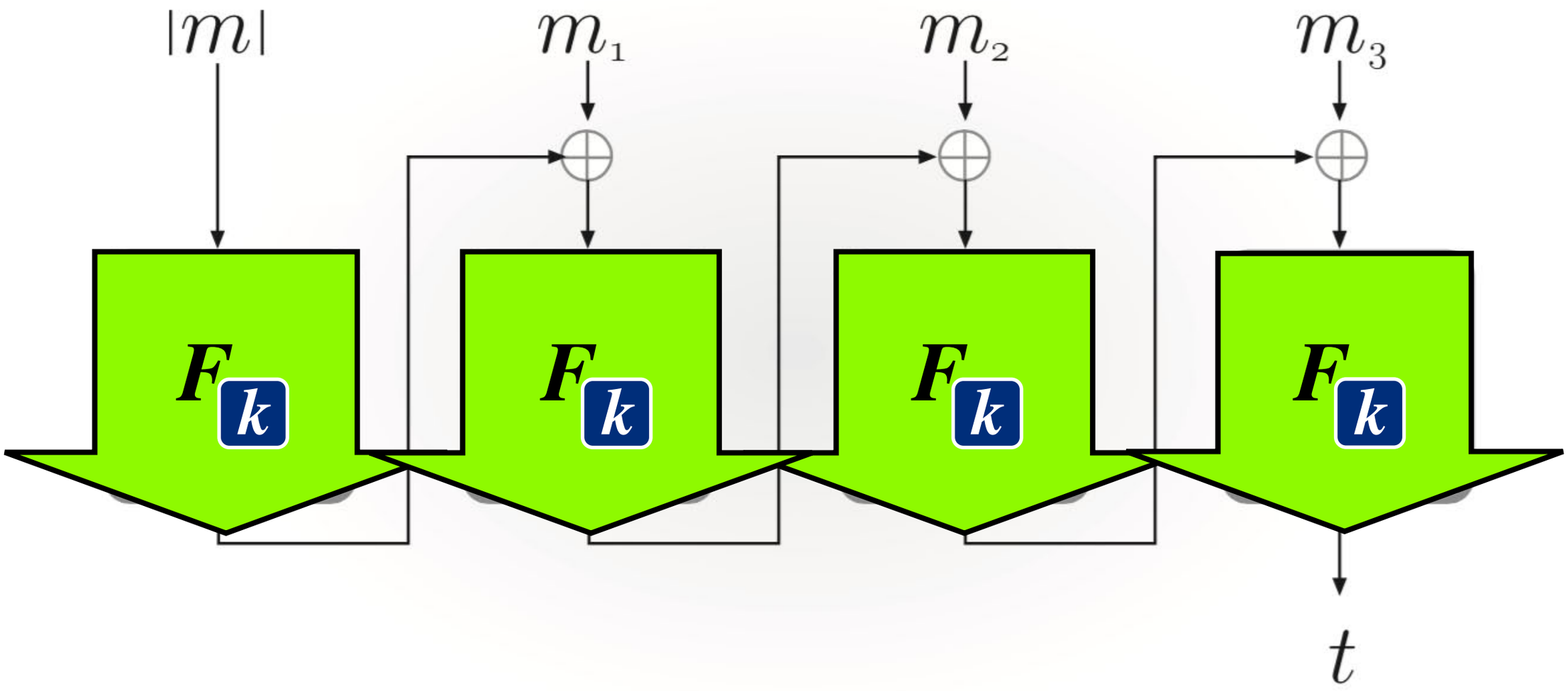


FIGURE 4.2: A variant of CBC-MAC secure for authenticating arbitrary-length messages.

Secure CBC-MAC for variable-length message

1. *Prepend* the message with its length $|m|$ (encoded as an n-bit string), and then compute the basic **CBC-MAC** on the resulting message.

(This is shown in **Figure 4.2**.)

We stress that appending the block length to the *end* of the message is *not* secure.

1/2

Secure CBC-MAC for variable-length message

2. Change the scheme so that key generation chooses two independent keys $k_1 \leftarrow \{0,1\}^n$ and $k_2 \leftarrow \{0,1\}^n$.

Then, to authenticate a message m first compute the basic **CBC-MAC** of m using k_1 and let t be the result; output the tag $t' := F_{k_2}(t)$.

2/2

4.5 Obtaining Privacy **and** Authentication

- In **Chapter 3**, we studied how it is possible to encrypt messages and thereby obtain *privacy*.
- In **Chapter 4**, we have shown how message authentication codes can be used to guarantee *data authenticity* or *integrity*.
- Often, however, we need both privacy and message integrity.

Privacy only vs. privacy and message integrity

- It is best practice to *always encrypt and authenticate by default*;
- Encryption alone should not be used unless there are compelling reasons to do so (such as implementations on severely resource-constrained devices) and, even then, only if one is absolutely sure that no damage can be caused by undetected modification of the data.

Security requirements

Let Π_E = **(Gen$_E$, Enc, Dec)** be an arbitrary encryption scheme and Π_M = **(Gen$_M$, Mac, Vrfy)** be a message authentication code.

An Encryption *scheme*

Π'=(Gen′, EncMac′, Dec′)

derived as a combination of Π_E and Π_M is a tuple of algorithms that operate as follows:

Security requirements

- The *key-generation algorithm* **Gen′** takes input 1^n, and runs $\mathbf{Gen}_E(1^n)$ and $\mathbf{Gen}_M(1^n)$ to obtain keys k_E and k_M (resp.). The key is $k=(k_E, k_M)$.

- The *Encryption* **EncMac′** takes as input the key k and a message m and outputs a value c that is derived by applying some combination of $\mathbf{Enc}_{k_E}(\cdot)$ and $\mathbf{Mac}_{k_M}(\cdot)$.

- The *decryption algorithm* **Dec′** takes as input the key k and a transmitted value c, and applies some combination of $\mathbf{Dec}_{k_E}(\cdot)$ and $\mathbf{Vrfy}_{k_M}(\cdot)$.

- The output of **Dec′** is either a plaintext m or a special symbol $\perp$ that indicates an error (of authentication).

Correctness requirements

The correctness requirement is that for every n, every pair of keys (k_E, k_M) output by $\mathbf{Gen}'(1^n)$, and every value $m \in \{0,1\}^*$,

$$\mathbf{Dec}'_{k_E,k_M}(\mathbf{EncMac}'_{k_E,k_M}(m)) = m.$$

Π' actually satisfies the syntax of a private-key encryption scheme.

Security requirements

Unforgeable-Encryption experiment $\mathbf{Enc\text{-}forge}_{A,\Pi'}(n)$:

1. A random key $k = (k_E, k_M)$ *is generated by running* $\mathbf{Gen}'(1^n)$.

2. The adversary A *is given input* 1^n *and oracle access to the algorithm* $\mathbf{Enc}'_k(\cdot)$. *The adversary eventually outputs* c. *Let* Q *denote the set of all queries that* A *asked to its oracle.*

3. Let $m := \mathbf{Dec}'_k(c)$. *The output of the experiment is defined to be* 1 *iff* (1) $m \neq \perp$ (2) $m \notin Q$.

Security requirements

DEFINITION 4.16 *An Encryption scheme* **Π′** *achieves* Unforgeability *if for all* ***PPT*** *adversaries* ***A****, there exists a negligible function* **negl** *such that:*

$$\mathbf{Pr[\,Enc\text{-}forge}_{A,\Pi'}(n) = 1\,] \leq \mathbf{negl}(n).$$

Security requirements

DEFINITION 4.17 *A tuple* **(Gen′, EncMac′, Dec′)** *is* an Authenticated Encryption scheme *if it is* *Unforgeable & **CCA**-secure.*

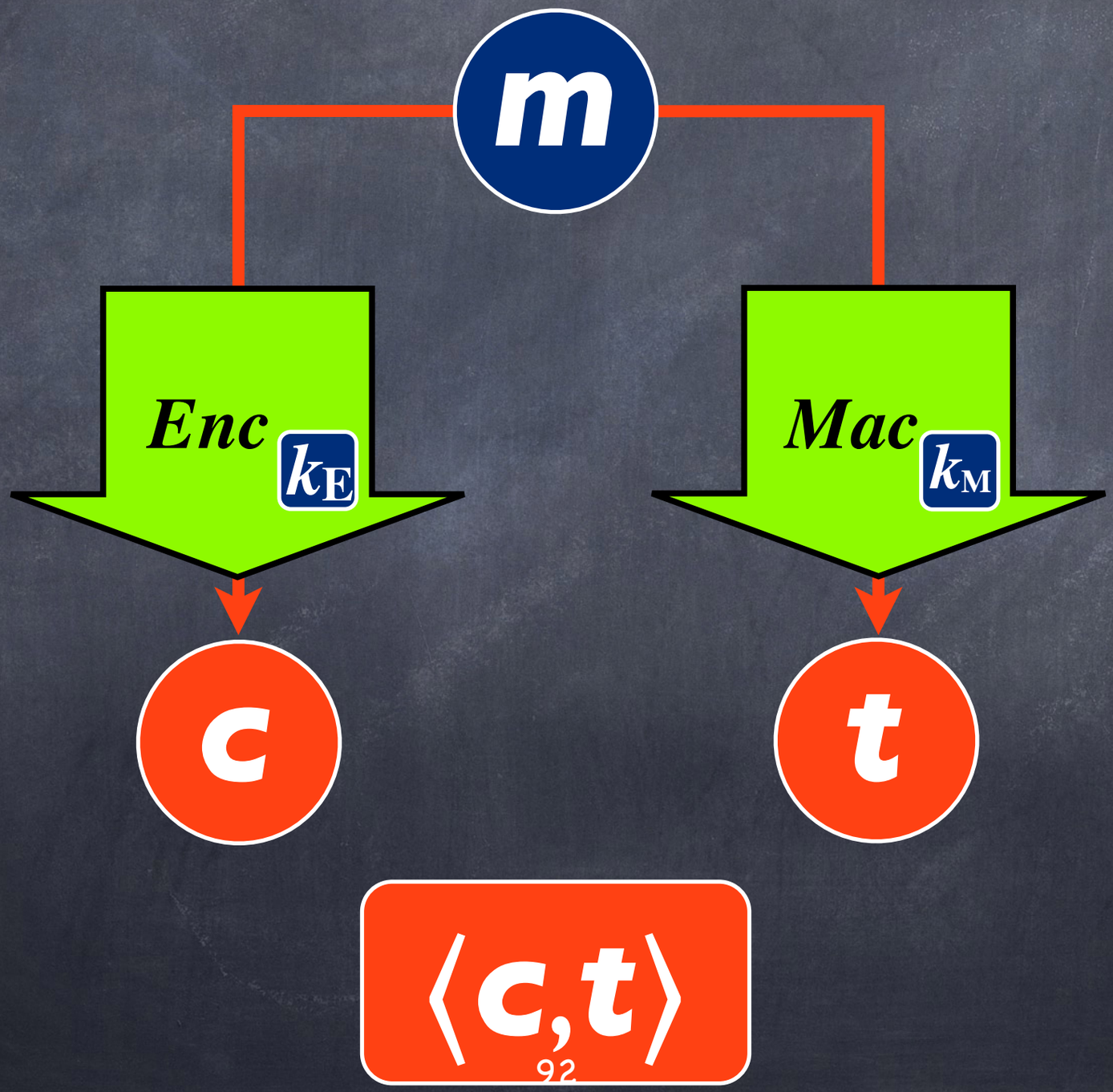
Obtaining Privacy and Authentication

There are three common approaches to combining encryption and message authentication. Let k_E be an encryption key, and k_M be a **MAC** key.

A common mistake is to use the *same* key for both encryption and authentication.

This should never be done, as independent keys should always be used for independent applications (unless a specific proof of security when using the same key is known).

Encrypt-and-authenticate



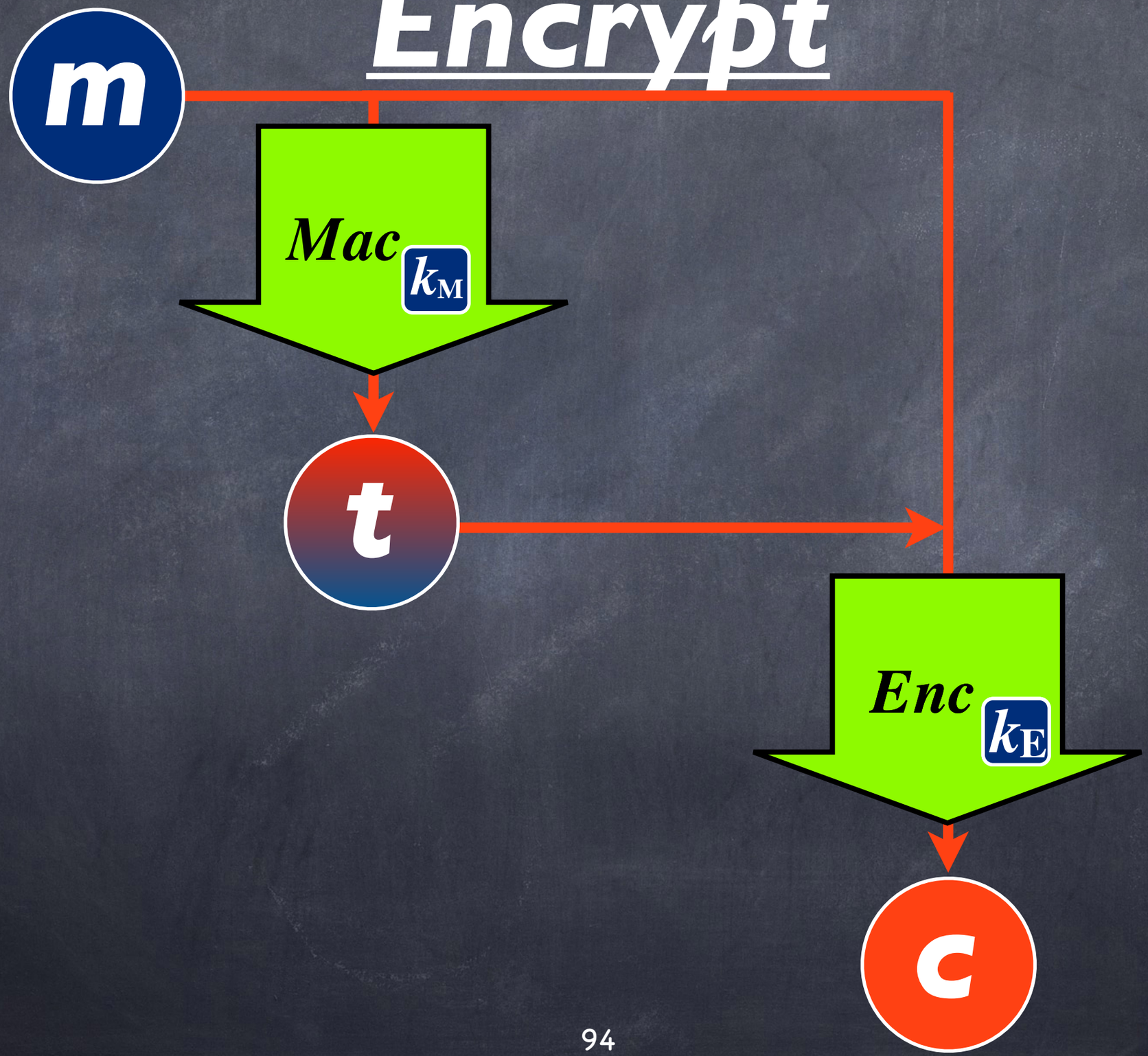
Obtaining Privacy and Authentication

1. *Encrypt-and-authenticate*: In this method, encryption and message authentication are computed independently. That is, given a plaintext message m, the sender transmits $\langle c, t \rangle$ where:

$$c \leftarrow \mathbf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathbf{Mac}_{k_M}(m).$$

The receiver decrypts c to recover m, and then verifies the tag t. If $\mathbf{Vrfy}_{k_M}(m, t) = 1$, the receiver outputs m; otherwise, it outputs $\perp$.

Authenticate-then-Encrypt



Obtaining Privacy and Authentication

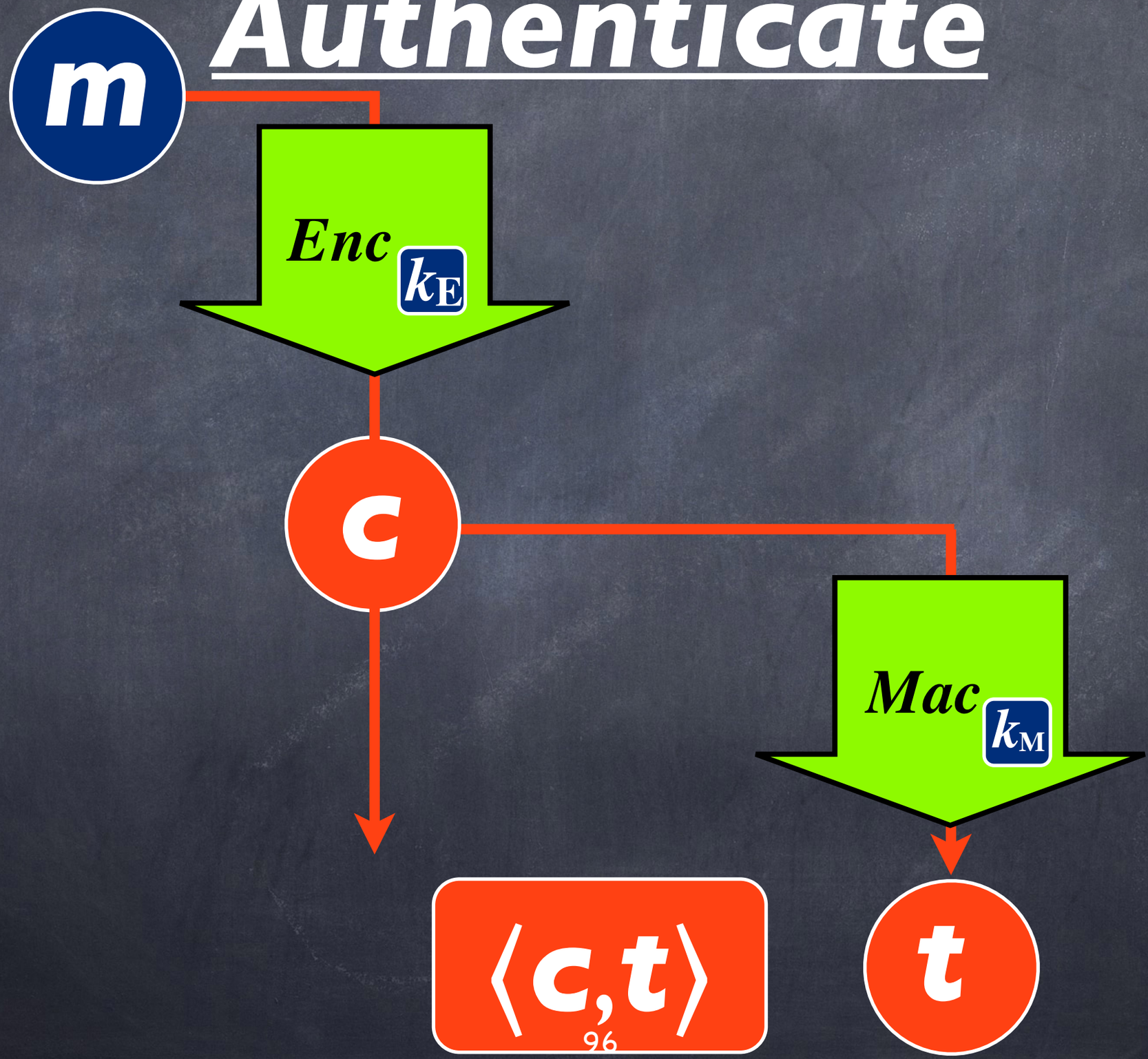
2. *Authenticate-then-encrypt*: Here a **MAC** tag t is first computed, and then the message and tag are encrypted together. That is, the sender transmits c computed as:

$$t \leftarrow \mathbf{Mac}_{k_M}(m) \quad \text{and} \quad c \leftarrow \mathbf{Enc}_{k_E}(m \| t).$$

The authentication tag t is not sent "in the clear", but is instead incorporated into the plaintext that is encrypted.

The receiver decrypts c, and then verifies the tag t on m. As before, if $\mathbf{Vrfy}_{k_M}(m, t) = 1$ the receiver outputs m; otherwise, it outputs $\perp$.

Encrypt-then-Authenticate



Obtaining Privacy and Authentication

3. *Encrypt-then-authenticate*: In this case, the message m is first encrypted and then a **MAC** tag is computed over the encrypted message. That is, the message is the pair $\langle c, t \rangle$ where:

$$c \leftarrow \mathbf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathbf{Mac}_{k_M}(c).$$

The receiver verifies t before decrypting c. As before, if $\mathbf{Vrfy}_{k_M}(m,c) = 1$ the receiver outputs m; otherwise, $\perp$. Observe that this is exactly **Construction 4.19** from the previous section.

Encrypt-and-authenticate

This combination is *not* (necessarily) secure, since it may violate privacy.

To see this, note that a secure **MAC** does not necessarily imply *any* privacy and, specifically, it is possible for the tag of a message to leak the entire message.

Authenticate-then-encrypt

We show that this combination is also not necessarily secure.

We use the following encryption scheme:

Let **Transform(*m*)** be as follows: any **0** in ***m*** is transformed to **00** and any **1** in ***m*** is transformed arbitrarily to **01** or **10**. The inverse transform parses the encoded message as pairs of bits, and then maps **00** to **0**, and **01** or **10** to **1**.
If a **11** is encountered, the result is $\perp$.

Authenticate-then-encrypt

Define $\mathbf{Enc}_k(m) = \mathbf{Enc}'_k(\mathbf{Transform}(m))$, where **Enc′** represents counter mode encryption using a pseudorandom function.

(The important point is it works by generating a new pseudorandom stream for each message to encrypt, and then **XOR**ing the stream with the message.)

Note that **Enc** is **CPA**-secure.

Authenticate-then-encrypt

Consider the following chosen-ciphertext attack: Given a challenge ciphertext

$$\mathbf{c} = \mathbf{Enc}'_{k_1}(\mathbf{Transform}(m \| \mathbf{Mac}_{k_2}(m))),$$

the attacker simply flips the first two bits of the second block of $\mathbf{c}$ (the first being the random IV) and verifies whether the resulting ciphertext is valid.

(Technically, this can be determined via a query to the decryption oracle. The adversary only needs to know if the ciphertext is valid or not, however, and so a weaker oracle would also suffice.)

Authenticate-then-encrypt

If the first bit of the underlying message ***m*** is **1**, then the modified ciphertext will be valid.

This is because if the first bit of ***m*** is **1**, then the first two bits of **Transform(*m*)** are **01** or **10**, and flipping these bits yields another valid encoding of ***m***.

Furthermore, the tag will still be valid since it is applied to ***m*** and not the encoding of ***m***.

Authenticate-then-encrypt

On the other hand, if the first bit of ***m*** is **0** then the modified ciphertext will <u>not</u> be valid since the first two bits of **Transform(*m*)** would be **00** and their complement is **11**.

This attack can be carried out on each bit of ***m*** separately, resulting in complete recovery of the message ***m***.

Obtaining Privacy and Authentication

CONSTRUCTION 4.18

Let $\Pi_E = (\mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme and let $\Pi_M = (\mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code, where in each case key generation is done by simply choosing a uniform n-bit key. Define a private-key encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

- Gen': on input 1^n, choose independent, uniform $k_E, k_M \in \{0,1\}^n$ and output the key (k_E, k_M).
- Enc': on input a key (k_E, k_M) and a plaintext message m, compute $c \leftarrow \mathsf{Enc}_{k_E}(m)$ and $t \leftarrow \mathsf{Mac}_{k_M}(c)$. Output the ciphertext $\langle c, t \rangle$.
- Dec': on input a key (k_E, k_M) and a ciphertext $\langle c, t \rangle$, first check whether $\mathsf{Vrfy}_{k_M}(c, t) \stackrel{?}{=} 1$. If yes, then output $\mathsf{Dec}_{k_E}(c)$; if no, then output $\perp$.

A generic construction of an authenticated encryption scheme.

Constructing Autheticated Encryption Schemes

The construction works in the following way. The sender and receiver share two keys, one for a **CPA**-secure encryption scheme and the other for a message authentication code.

To encrypt a message ***m***, the sender first encrypts it using the **CPA**-secure scheme and then computes a **MAC** tag ***t*** on the resulting ciphertext ***c***; the entire ciphertext is now ‹***c***,***t***›.

Given a ciphertext ‹***c***,***t***›, the recipient verifies validity of the **MAC** tag before decrypting ***c***.

Encrypt-then-authenticate

THEOREM 4.19 *Let* Π_E *be a* ***CPA****-secure private-key encryption scheme, and let* Π_M *be a secure message authentication code (with unique tags). Then the combination* **(Gen′, Enc′, Dec′)** *derived by applying the encrypt-then-authenticate approach to* Π_E, Π_M (***Construction 4.18***) *is an Authenticated Encryption scheme.*

The need for independent keys

We conclude by stressing a basic principle of security and cryptography: *different security goals should always use independent keys.*

That is, if an encryption scheme and a message authentication code are both needed, then independent keys should be used for each one.

The need for independent keys

- In order to illustrate this here, consider what can happen to the encrypt-then-authenticate methodology when the same key ***k*** is used for both encryption and authentication.

- Let ***F*** be a strong pseudorandom permutation.

- It follows that F^{-1} is also a strong pseudorandom permutations.

The need for independent keys

- Define $\mathbf{Enc}_k(m) = F_k(m \| r)$ for $m \in \{0,1\}^{n/2}$ and a random $r \leftarrow \{0,1\}^{n/2}$, and $\mathbf{Mac}_k(c) = F_k^{-1}(c)$.

- It can be shown that the given encryption scheme is **CPA**-secure (in fact, it is even **CCA**-secure), and we know that the given message authentication code is a secure **MAC**.

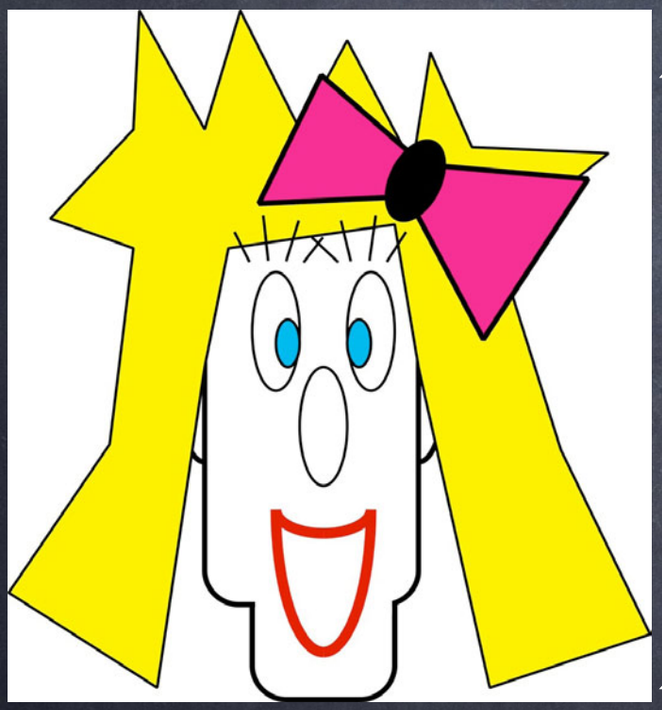
The need for independent keys

- Define $\mathbf{Enc}_k(m) = F_k(m \| r)$ for $m \in \{0,1\}^{n/2}$ and a random $r \leftarrow \{0,1\}^{n/2}$, and $\mathbf{Mac}_k(c) = F_k^{-1}(c)$.

- However, the encrypt-then-authenticate combination applied to the message m with the same key k yields:

$$\mathbf{Enc}_k(m), \mathbf{Mac}_k(\mathbf{Enc}_k(m)) = F_k(m \| r), F_k^{-1}(F_k(m \| r))$$
$$= F_k(m \| r), m \| r.$$

CCA Indistinguishability Experiment: $\mathbf{PrivK}^{cca}_{A,\Pi}(n)$



$k \leftarrow \mathbf{Gen}(1^n)$

$x_i \leftarrow \mathbf{Enc}_k(v_i)$

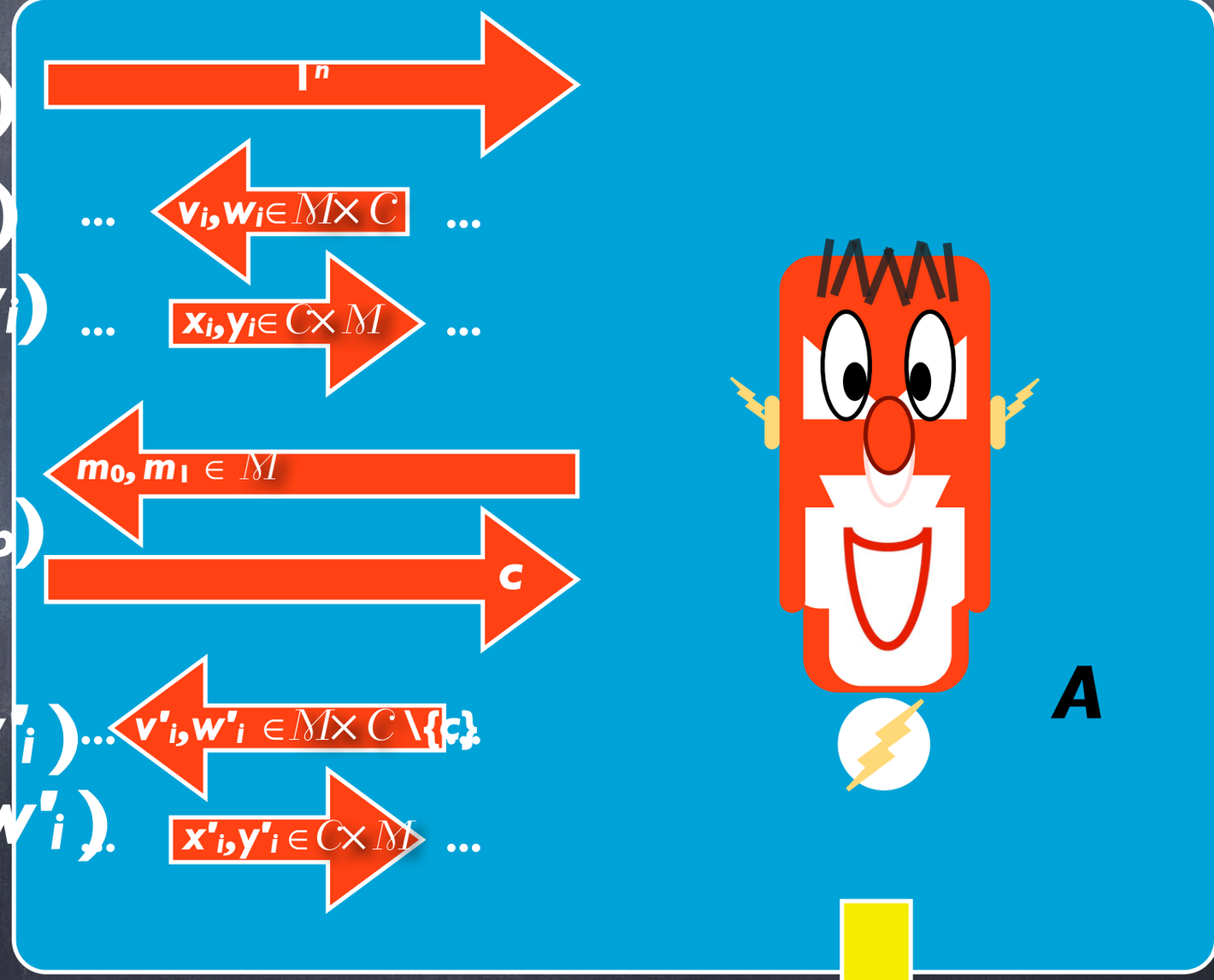
$y_i \leftarrow \mathbf{Dec}_k(w_i)$

$b \leftarrow \{0, 1\}$

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

$x'_i \leftarrow \mathbf{Enc}_k(v'_i)$

$y'_i \leftarrow \mathbf{Dec}_k(w'_i)$



b

b'

computationally secret

$$\Pr[b = b'] \leq \tfrac{1}{2} + \mathrm{negl}(n)$$

3.7 Sec. Against Chosen-Ciphertext Attacks (CCA)

Consider the following experiment for any private-key encryption scheme **Π = (Gen, Enc, Dec)**, adversary ***A***, and value ***n*** for the sec. parameter.

CCA indistinguishability experiment $\mathbf{PrivK}^{cca}_{A,\Pi}(n)$:

1. *A key* ***k*** *is generated by running* $\mathbf{Gen}(1^n)$.

2. *The adversary* ***A*** *is given input* 1^n *and oracle access to* $\mathbf{Enc}_k(\cdot)$ *and* $\mathbf{Dec}_k(\cdot)$. *It outputs a pair of messages* m_0, m_1 *of the same length.*

3.7 Sec. Against Chosen-Ciphertext Attacks (CCA)

3. *A random bit* $b \leftarrow \{0, 1\}$ *is chosen, and then a* challenge *ciphertext* $c \leftarrow \mathrm{Enc}_k(m_b)$ *is computed and given to* A.

4. *The adversary* A *continues to have oracle access to* $\mathrm{Enc}_k(\cdot)$ *and* $\mathrm{Dec}_k(\cdot)$, *but is not allowed to query the latter on the challenge ciphertext itself. Finally,* A *outputs a bit* b'.

5. *The output of the experiment is defined to be* **1** *if* $b' = b$, *and* **0** *otherwise.*

3.7 Sec. Against Chosen-Ciphertext Attacks (CCA)

DEFINITION 3.33 *A private-key encryption scheme* **Π** *has* indistinguishable encryptions under a chosen-ciphertext attack (*or is* **CCA**-secure) *if for all* ***PPT*** *adversaries* ***A*** *there exists a negligible function* **negl** *such that:*

$$\mathbf{Pr}[\,\mathbf{PrivK}^{\mathbf{cca}}_{\mathbf{A},\Pi}(n) = 1\,] \leq \tfrac{1}{2} + \mathbf{negl}(n),$$

where the probability is taken over all random coins used in the experiment.

Security Against Chosen-Ciphertext Attacks (CCA)

THEOREM 4.20

*If Π_E is a **CPA**-secure private-key encryption scheme, and Π_M is a secure **MAC** (with unique tags), then **Construction 4.18** is a **CCA**-secure private-key encryption scheme.*

CCA-security and real-life implications

Construction 4.18 may seem unsatisfying, since it achieves **CCA**-security simply by ensuring that the decryption oracle is useless to the adversary.

Instead of being viewed as a drawback of the construction, this should be viewed as an advantage!

Although there are other ways to achieve **CCA**-security, here the adversary is unable to generate *any* valid ciphertext that was not already created by one of the honest parties.

This means that **Construction 4.18** achieves both privacy *and* message authentication.

Authenticated Encryption vs. CCA-security

- Although we use the same construction for achieving **CCA**-security and Authenticated Encryption, the security goals in each case are different.

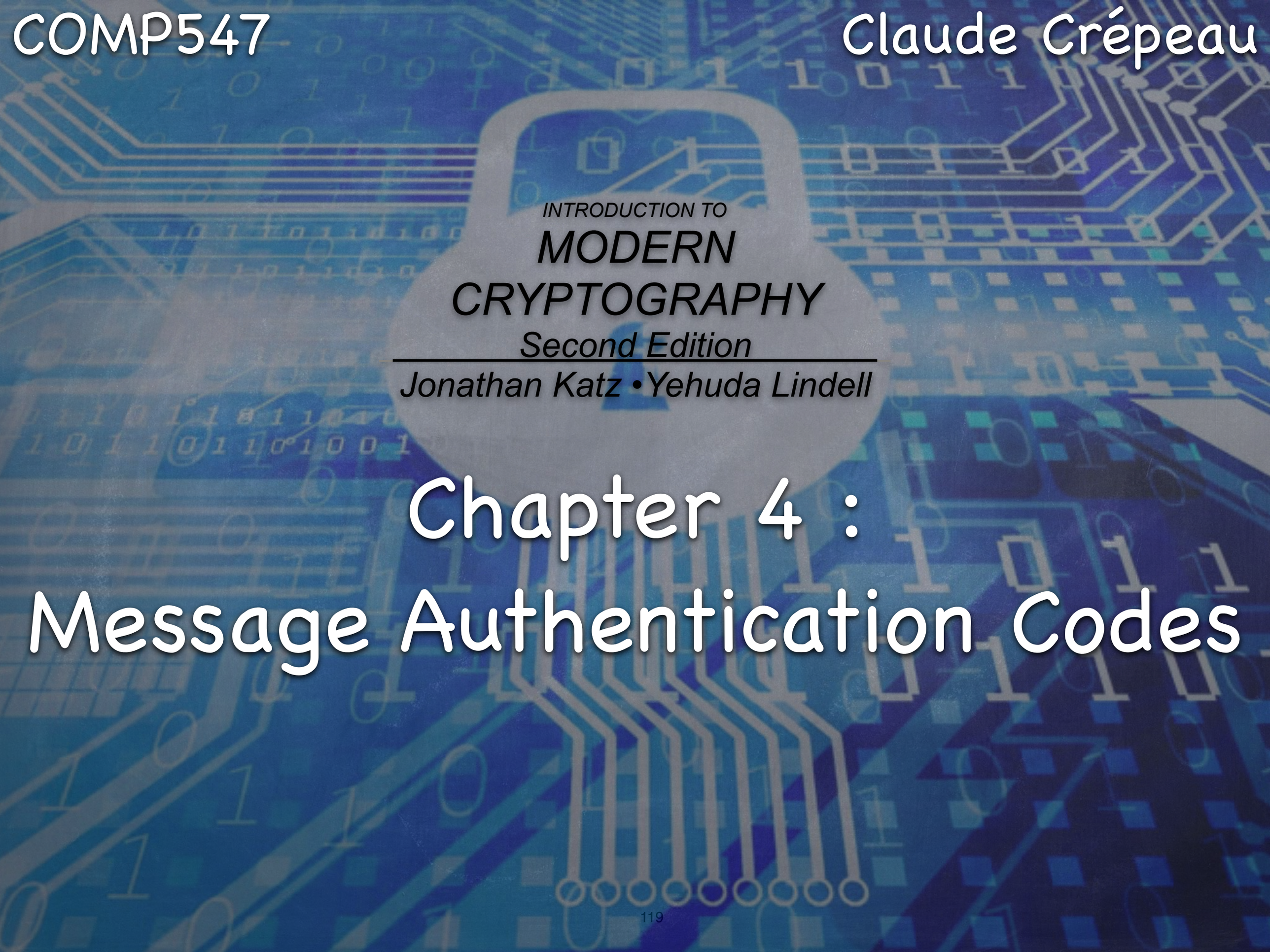
- In the setting of **CCA**-security we are not necessarily interested in obtaining message authentication; rather, we wish to ensure privacy even against a strong adversary who is able to make decryption queries.

Secure message transmission vs. CCA-security

- When considering Authenticated Encryption, in contrast, we are interested in the twin goals of **CCA**-security *and* integrity.

- Clearly, as we have defined it, Authenticated Encryption implies **CCA**-security. The opposite direction is not necessarily true.

COMP547 Claude Crépeau



Chapter 4 : Message Authentication Codes

Encryption from Authentication

- Many governments felt for a long time that Encryption could be used by dishonest people to accomplish some unauthorized activities,
- whereas Authentication was a much safer primitive because it does not a priori provides confidentiality.
- This impression turned out to be wrong as demonstrated by Ron Rivest in 1998.

NIB

Encryption from Authentication

- Suppose Alice and Bob have a secure **MAC** **(Gen, $\mathbf{Mac}_k$, $\mathbf{Vrfy}_k$)** and that Alice wants to send a secret bit ***b*** to Bob.

- She may pick two random messages m_0, m_1, with a random tag $r \neq \mathbf{Mac}_k(m_i)$ and sends :

  $(m_0, \mathbf{Mac}_k(m_0))$ and (m_1, r) to transmit $b = 0$ or
  (m_0, r) and $(m_1, \mathbf{Mac}_k(m_1))$ to transmit $b = 1$.

NIB

Encryption from Authentication

- Bob upon receiving two messages $(m_0, t_0),(m_1, t_1)$ decrypts the bit b as

$b = 0$ if $\mathbf{Vrfy}_k(m_0, t_0)=1$ and
$b = 1$ if $\mathbf{Vrfy}_k(m_1, t_1)=1$.

NIB

Encryption from Authentication

- With probability one, Bob can tell which of the two messages is authenticated properly,
- whereas an adversary who does not know the key ***k*** cannot determine the validity of the two messages...

NIB

TASKS / SECURITY	Encryption	Authentication	Identification	Quantum
Symmetric Informational	Miller-Vernam One-Time PAD	Wegman-Carter Universal Hash	Simple Solutions	Quantum Key Distribution
Symmetric Computational	from PRBG from PRFG DES, AES, etc	from PRBG from PRFG DES, AES, etc	from PRBG from PRFG DES, AES, etc	Quantum Attacks, Q-Safety
Asymmetric Computational	RSA, ElGammal, Blum-Goldwasser	RSA, DSA, etc	Guilloux-Quisquater, Schnor, etc	Quantum Attacks, Q-Safety
	DONE	IN PROGRESS	TO DO	GIVE UP

McGill

Cryptography Research Group

Groupe de recherche en cryptographie