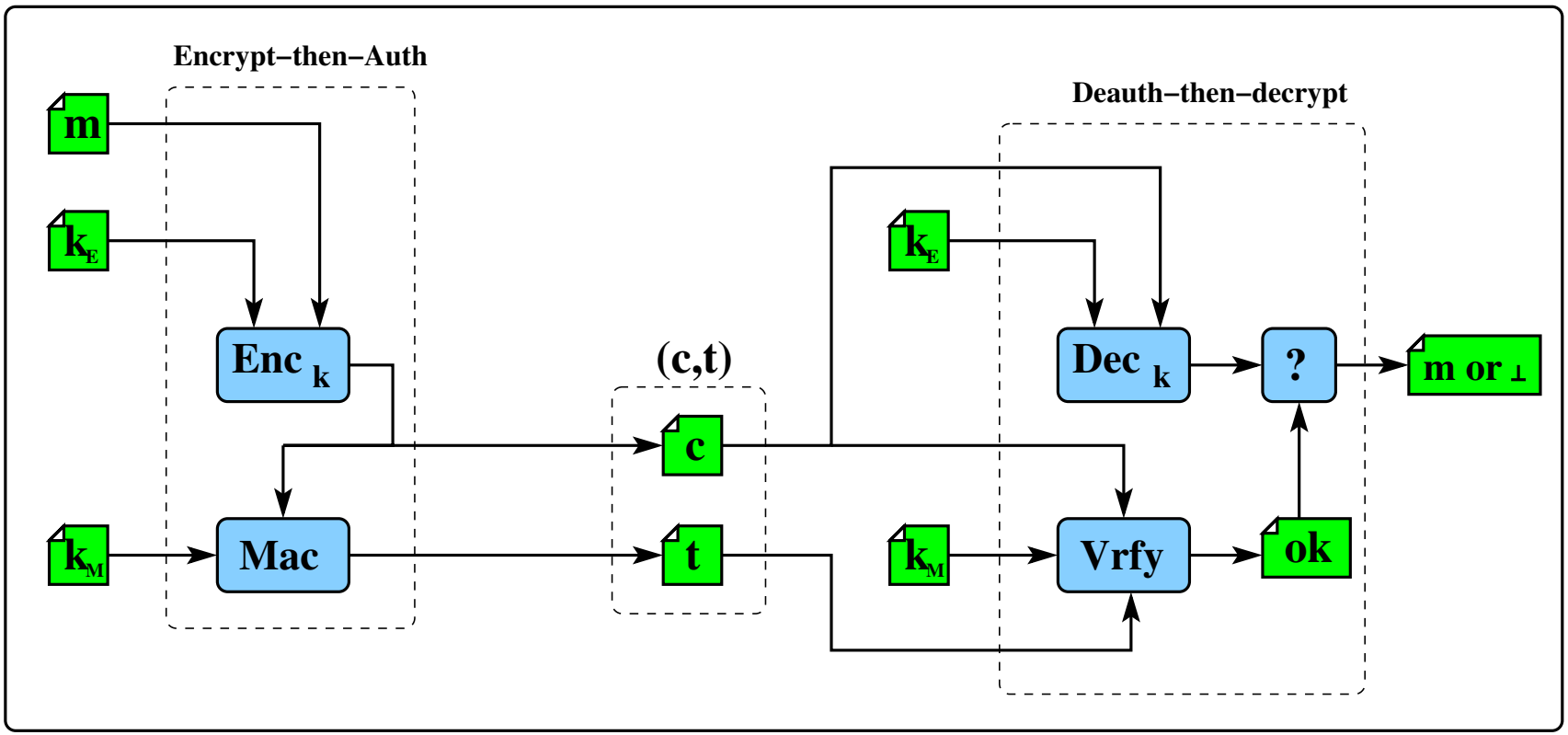
CS4236 Cryptography Theory and Practice Topic 6 - Mac and Authenticated-Encryption

Hugh Anderson

National University of Singapore
School of Computing

September, 2022

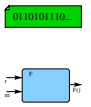
Authenticated encryption...



Outline

The story so far ... where are we?

The last 5 weeks: weekly steps we have taken



1. We had a historical/contextual introduction in Session 1.
2. We progressed from the traditional view of perfect secrecy, through to a game/experiment view: *perfect* indistinguishability.
3. *Perfectly* indistinguishable was relaxed to give *computationally* indistinguishable. An EAV-Secure system was constructed with a PRG.
4. The notion of CPA-secure was developed, where the adversary had access to an encryption oracle. CPA-secure was achieved with a PRF.
5. Modes were described, and CCA-Security was outlined. The “padding oracle” example was described, motivating CCA.

So - all of this was all about secrecy/encryption, the C in CIA, and (in particular) secret key encryption schemes/systems, and their properties. The progress was slow, because I wanted to emphasize the structure of the mathematics used to formalize the systems (systems, properties expressed as probabilistic adversarial games, constructions, and proofs).

This week - we move on to a new system - the MAC.

Message Authentication Codes

From CIA: Confidentiality vs Authenticity...

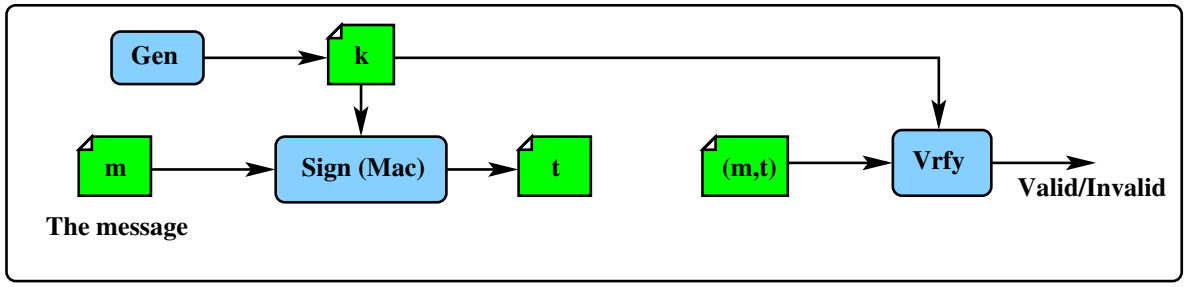
Many developers/programmers are not aware that “encryption” does not necessarily provide “authenticity”, and that their applications require “authenticity” instead of confidentiality. This misleads them to employ encryption (instead of authentication primitives) in their application.

A good example is with stream ciphers. It may be easy to modify the ciphertext. Even CBC mode can be modified as we saw. Crypto primitives for authenticity are MAC (private key) and signature (public key).

A general comment...

It is considered weak to just encrypt in an attempt to provide authenticity. So here we are looking at a formal description of the security requirements on a MAC. As the book says, the right tool for the task!

The MAC, Message Authentication Code



Comments on MAC - $(\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$

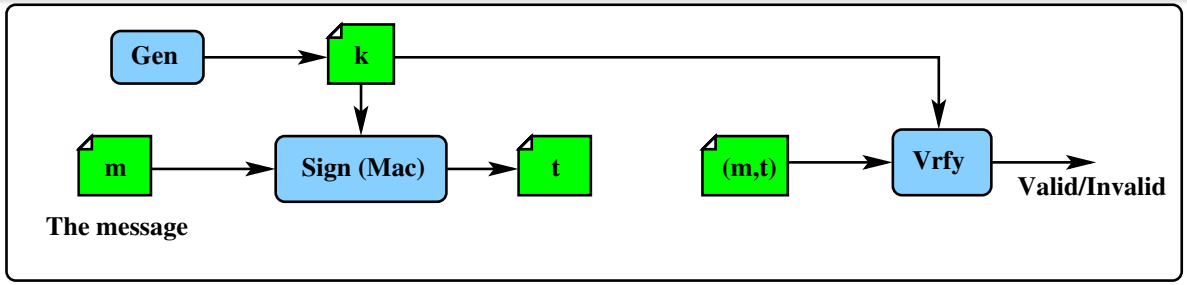
t is called the authentication tag, or MAC. Vrfy is deterministic. Sign/Mac can be probabilistic or deterministic.

The concern is with forgery: without knowing k, an adversary should be unable to get a valid pair of (m, t). Issues such as the prevention of a replay attack is handled at another "protocol" layer and not a concern here.

In most MAC designs, Mac is deterministic. Thus Vrfy can be done by repeating the computation of the MAC and comparing. Such a design is termed Canonical verification (pg 111):

$$\mathrm{Vrfy}_k(m, t) \equiv (\mathrm{Mac}_k(m) = t)$$

System/scheme definition



Definition 4.1. Mac $(\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$

$\mathrm{Gen}(1^n)$: On input 1^n, output a truly random key k with size $|k| \geq |n|$.

$\mathrm{Mac}_k(m)$: (Tag generation) Input a message m and key k. We will write this as $t \leftarrow \mathrm{Mac}_k(m)$ (as Mac may be randomised).

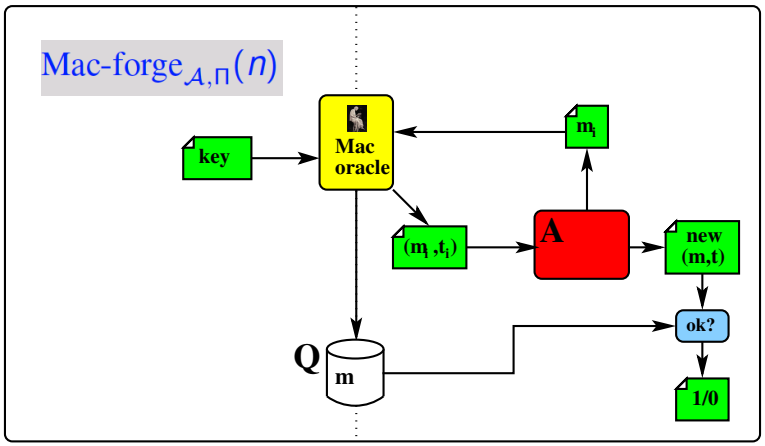
$\mathrm{Vrfy}_k(m, t)$: Verifies and outputs valid (1) or invalid (0). We will write this as $b := \mathrm{Vrfy}_k(m, t)$.

See pg 111 for details.

How to pose properties as a game...

An adversary has an oracle $O = \mathrm{Mac}_k(\cdot)$ that will generate a valid pair (m, t) of any query message m. The first security requirement is easier/weaker - can the adversary forge any message pair (m, t) where the message m was not seen before. This is called existential forgery under a chosen message.

Security requirement: Existential forgery



The first game: Mac-forge$_{\mathcal{A},\Pi}$ (Secure MAC)

1. Generate key $k = \text{Gen}(1^n)$.
2. Adversary has oracle access. Adversary outputs (m, t). Let $\mathcal{Q}$ be all the messages the adversary had sent to the oracle in this experiment.
3. Adversary wins (output of experiment is 1) iff (m, t) is valid and $m \notin \mathcal{Q}$.

Definition 4.2 (pg 113) Existential unforgeability

We say that Π is existentially unforgeable under an adaptive chosen-message attack, or simply secure, iff for any $\mathcal{A}$, there is a negl, s.t.

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

Security requirement: Strong Mac

The second game: $\text{Mac-sforge}_{\mathcal{A},\Pi}$ (Strong MAC)

In the previous experiment, an adversary wins when $m \notin \mathcal{Q}$. In cases where signing is probabilistic, it may be possible to give a (m, t) even if $m \in \mathcal{Q}$, but the t hasn't seen before.

The above motivates the definition of Strongly Secure mac. Here, the second winning condition is $(m, t) \notin \mathcal{Q}'$ where $\mathcal{Q}'$ is the set of (m', t'), queries and corresponding tags already sent to the oracle.

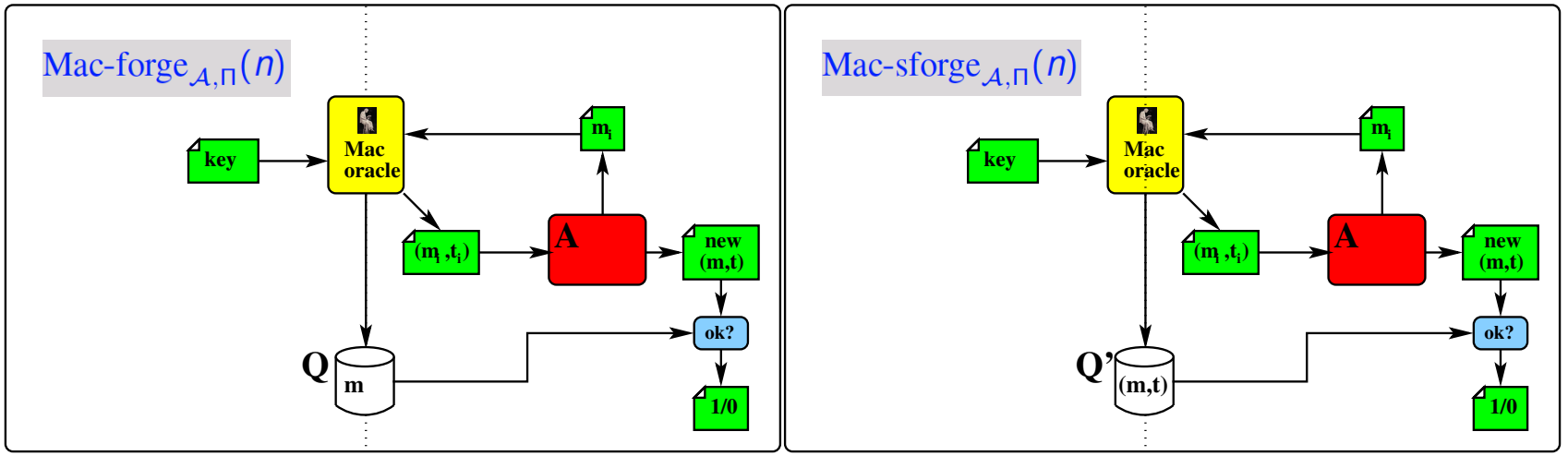
Definition 4.3 (pg 114) - strongly secure, or strong MAC

Here, Π is strongly secure iff for any PPT adversary $\mathcal{A}$, there is a negl, s.t.

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

If a scheme uses canonical verification and is secure, then it is also a strong MAC (it is a special case). Canonical verification implies deterministic $\text{Mac}_k()$ – thus, you cannot find two valid pairs $(m, t), (m, t')$ where $t \neq t'$. See Proposition 4.4.

Mac-forge **and** Mac-sforge

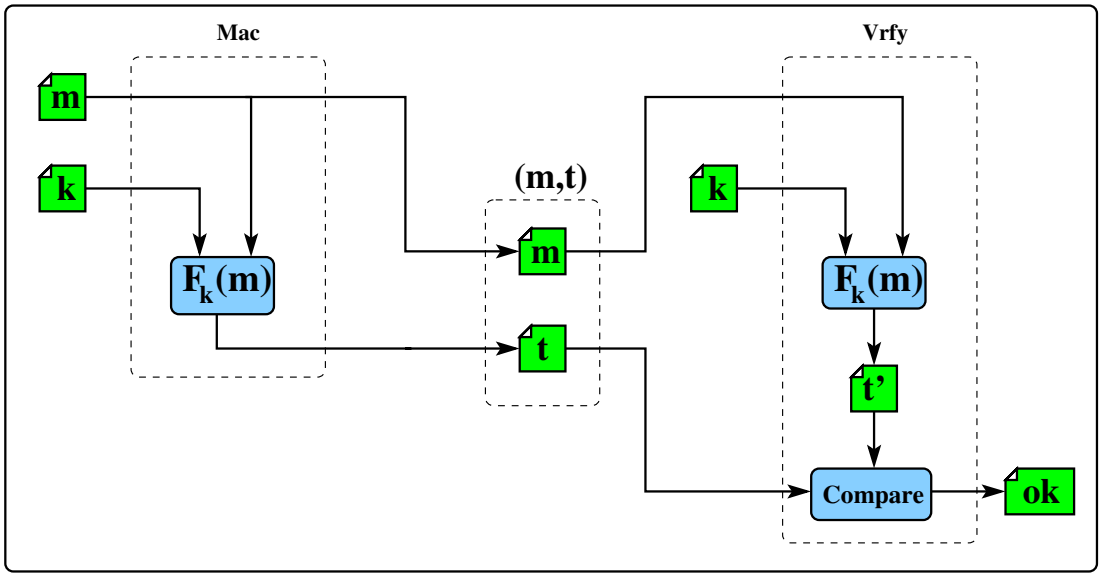


Existential forgery (Secure-MAC) vs Strong-MAC

Mac-Forge: A succeeds iff $m \notin \mathcal{Q}$

Mac-sForge: A succeeds iff $(m, t) \notin \mathcal{Q}'$

Construction of secure fixed size MAC



Construction 4.5 (p117 - Fixed-size input)

Uses a pseudorandom function F_k with key $k := \mathrm{Gen}(1^n)$.

$\mathrm{Mac}_k(m)$: on (same sized) input message m, key k, output $t := F_k(m)$

$\mathrm{Vrfy}_k(m, t)$: on input (m, t), key k outputs 1 (i.e. valid) iff $t = F_k(m)$

The above is dependant on the security of F.

Theorem 4.6

If F_k is a PRF, then construction 4.5 is a secure fixed size MAC.

Overview of Proof

Outline of proof

I found the book proof a bit confusing, as the proof does not clarify that Mac must be deterministic, as it is constructed from a PRF. We compare Π, and a construction $\widetilde{\Pi}$ that uses a uniform function f instead of $F_k(.)$. We want to show:

$$\begin{aligned} F \text{ is secure} &\longrightarrow \Pi \text{ is a secure MAC} \\ F, f \text{ are indistinguishable} &\longrightarrow \forall \mathcal{A}, \Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n) \end{aligned}$$

How to rigorously prove it?

Given any $\mathcal{A}$, we want to evaluate its behaviour, i.e. the probability:

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

1. We first evaluate the behaviour of $\mathcal{A}$ when it faces construction $\widetilde{\Pi}$. We show that the probability the adversary wins with $\widetilde{\Pi}$ is negligible.
2. Next, using the assumption that F is secure, show that the difference of $\mathcal{A}$'s behaviour w.r.t the original Π, and $\widetilde{\Pi}$, is negligible.

Combining the above two statements, we have the result.

Proof detail

Given construction $\widetilde{\Pi}$ and the corresponding experiment

1. As f is uniform, for any $\mathcal{A}$, $\Pr[\text{Mac-forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1] \leq \frac{1}{2^n}$
2. We want to show that for any $\mathcal{A}$, there is a negl, s.t.

$$|\ \overset{(1)}{\Pr[\text{Mac-forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1]} \quad - \quad \overset{(2)}{\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1]}\ | \leq \text{negl}(n) \qquad (3)$$

   and given 1 above, this would imply $\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n)$.

3. Given $\mathcal{A}$, let's build a (reduction style) distinguisher D that distinguishes F from a random function. We can show that, for D, when given an oracle of a truly random function f, its behaviour is the same as (1).

$$\Pr[D^{f(\cdot)}(1^n) = 1] = \Pr[\text{Mac-forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1]$$

   and when given an oracle of F, its behaviour is the same as (2).

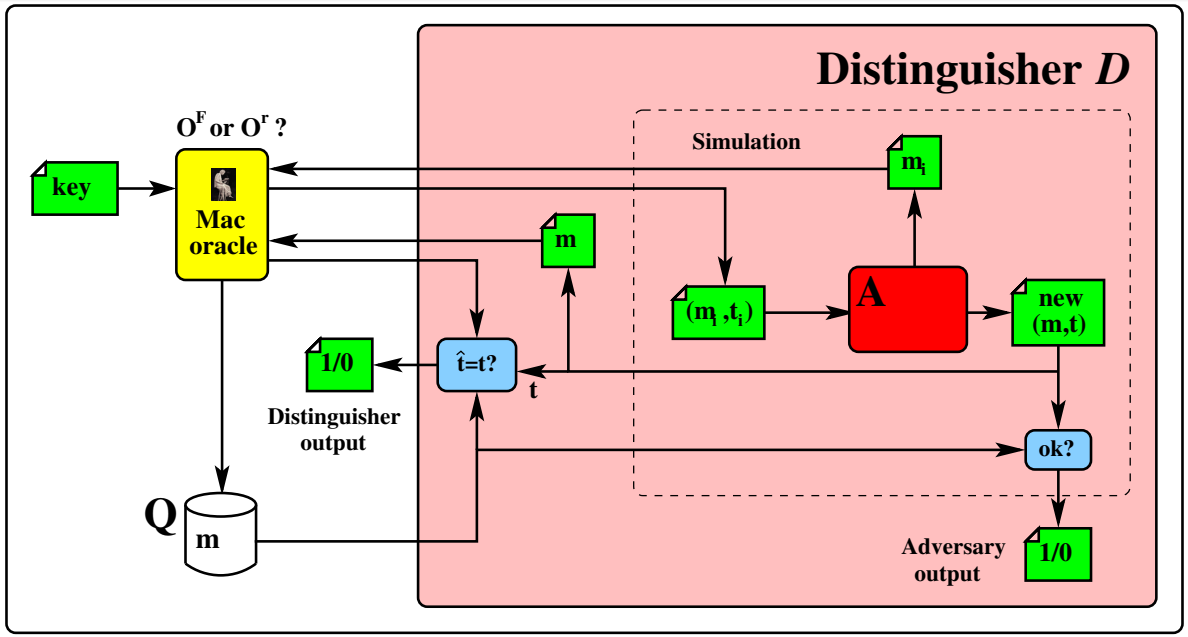
$$\Pr[D^{F_k(\cdot)}(1^n) = 1] = \Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1]$$

   Now, by assumption that F is pseudorandom,

$$\left|\Pr[D^{f(\cdot)}(1^n) = 1] - \Pr[D^{F_k(\cdot)}(1^n) = 1]\right| \leq \text{negl}(n)$$

■

The D in the proof



Given an $\mathcal{A}$ we construct the following D:

D has access to a MAC oracle. D's goal is to guess whether this oracle uses a uniform function f, or PRF F. Lets assume O^F.

D simulates $\mathcal{A}$. When $\mathcal{A}$ queries its MAC oracle, D queries its oracle and passes the result back to $\mathcal{A}$.
When $\mathcal{A}$ outputs (m, t) at the end, D queries the oracle with m, and gets $\hat{t}$.

The distinguisher D outputs 1 (i.e. it declares that it is F) iff $\hat{t} = t$ and $\mathcal{A}$ never queried for m. Note that D and $\mathcal{A}$ will always output the same value.

Extending to arbitrary length messages

Construction 4.7 (p120 - variable-length input)

Given m, k, and a fixed sized $\Pi' = (\text{Mac}', \text{Vrfy}')$, then we have:

$\text{Mac}_k(m)$**:** The MAC is constructed a block at a time:

- A random $\frac{n}{4}$-bit id r is generated. Let ℓ be the size of m, represented as an $\frac{n}{4}$-bit binary number.
- Message is divided into blocks of size $\frac{n}{4}$-bits. Let b be the number of blocks - $m_1, m_2, \ldots, m_b$
- For each i-th ($\frac{n}{4}$-bit) block, compute the tag $t_i = \text{Mac}'_k(r \mathbin{\|} \ell \mathbin{\|} i \mathbin{\|} m_i)$, where $\|$ is (still) concatenation.

...then output $(r, t_1, t_2, \ldots, t_b)$

$\text{Vrfy}_k(m, t)$**:** Reverse the above. Also check that the length is correct.

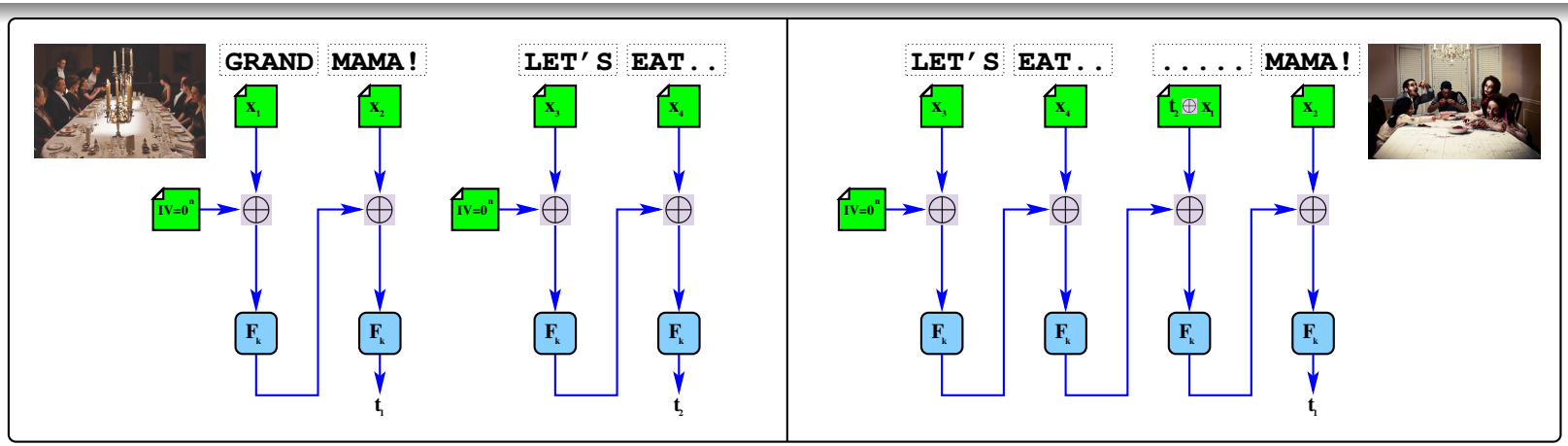
Theorem 4.8

If we are using an underlying secure fixed size MAC of size n, then construction 4.7 is a secure MAC.

Note

The length of m is still not arbitrary. It cannot be longer than $2^{\frac{n}{4}}$.

CBC-MAC: extending (PRF - fixed size) Mac



Construction 4.11 - Fixed-length CBC-MAC (p123)

With key $k \in \{0,1\}^n$, PRF F_k, message length $\ell \times n$, $t_o := 0^n$, then we have:

$\mathrm{Mac}_k(m)$: For $i = 1, ..., \ell$: $t_i := F_k(t_{i-1} \oplus m_i)$. Then output the tag $t = t_\ell$.

$\mathrm{Vrfy}_k(m, t)$: output 1 if and only if $t = \mathrm{Mac}_k(m)$.

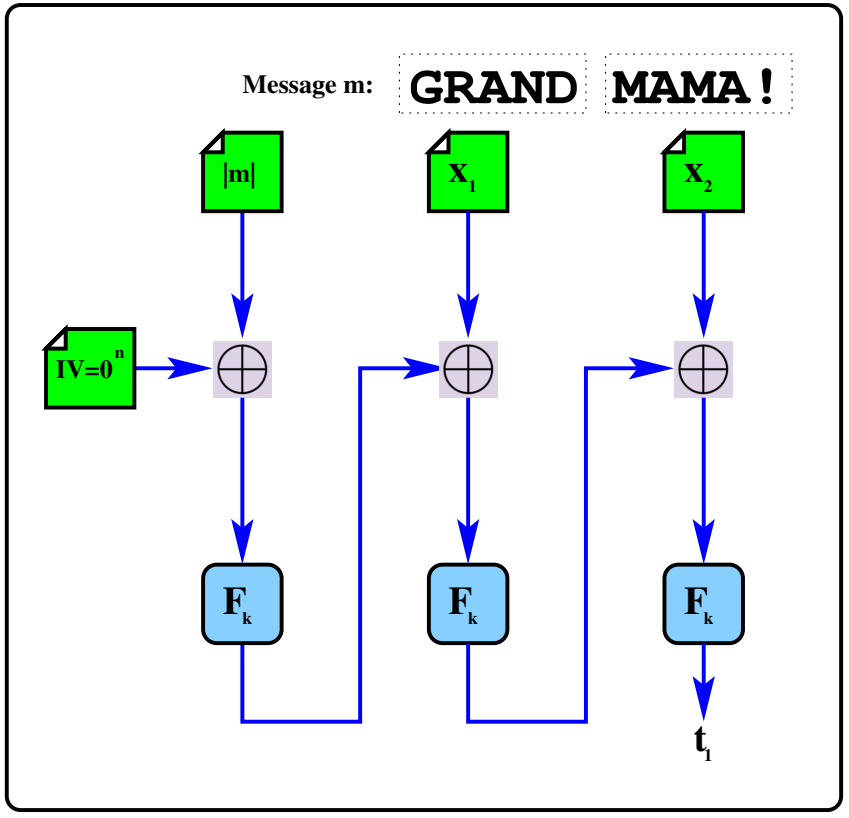
Theorem 4.12

If F_k is PRF, and m fixed length, CBC-MAC is secure.

Concatenation attack possible for arbitrary length m

If the adversary has obtained two pairs (m_1, t_1) and (m_2, t_2) - then m_3 can be easily created with a valid tag t_1 or t_2. In the above case (m_3, t_1) passes Vrfy.

CBC-MAC: Variable-length CBC-MAC (CBC)



Construction 4.11(a) - Variable-length CBC-MAC (p124)

Given key $k \in \{0,1\}^n$, PRF F_k, a message m of length $\ell \times n$, then set $t_o := |m|$, and we have:

$\mathrm{Mac}_k(m)$: For $i = 1, ..., \ell$: $t_i := F_k(t_{i-1} \oplus m_i)$. Then output the tag $t = t_\ell$.

$\mathrm{Vrfy}_k(m, t)$: output 1 if and only if $t = \mathrm{Mac}_k(m)$.

Authenticated encryption

Confidentiality and authentication

Many applications need both confidentiality and authenticity (e.g. TLS/SSL). It would be convenient (and more secure) to have a primitive or standard that achieves both. Makes the job easier for everyone!

Authenticated-encryption. Here, we look at a formulation of authenticated encryption, and a construction.

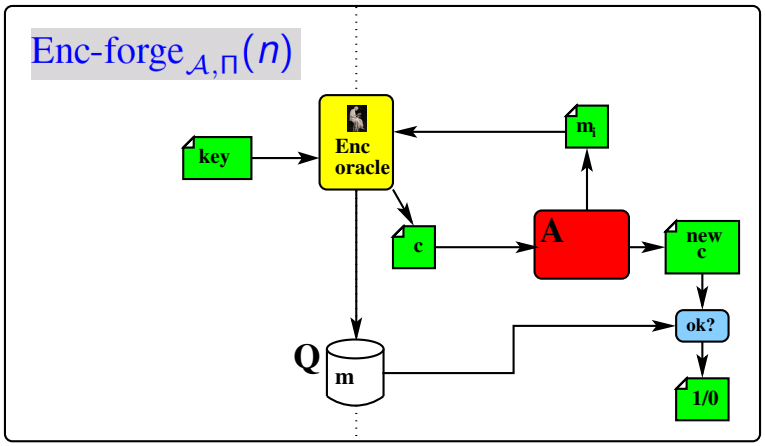
Decryption changes for authenticated systems

An authenticated-encryption scheme is the same as an encryption scheme, except that it has a new type of output for decryption. Here, the decryption algorithm, on input k and a ciphertext c outputs...

1. the decrypted message m, or
2. $\perp$ if the algorithm decides that the c is being forged.

The formulation consists of two parts, one on authenticity (unforgeable), and one on confidentiality (CCA-secure).

Authenticated encryption - unforgeability



Unforgeable experiment $\text{Enc-forge}_{\mathcal{A},\Pi}$

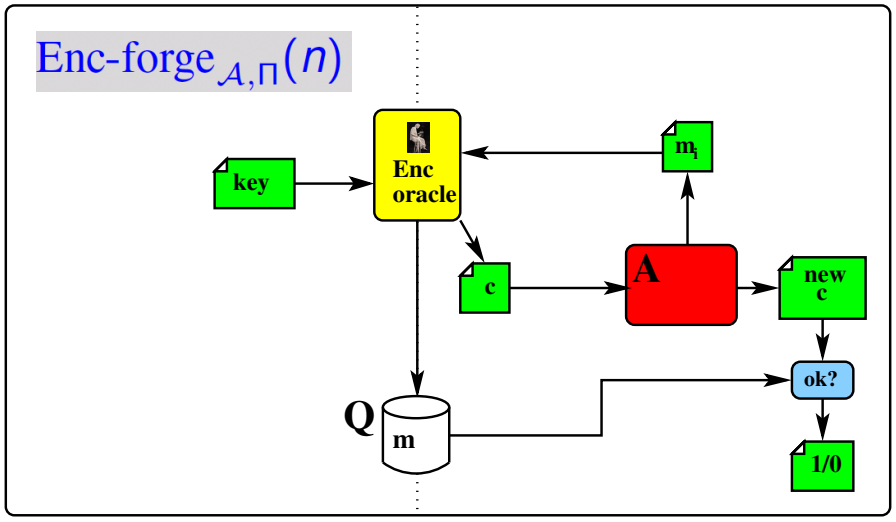
1. Generate key $k = \text{Gen}(1^n)$.
2. Adversary $\mathcal{A}$ has an encryption oracle and outputs ciphertext c.
3. Let $m := \text{Dec}_k(c)$. Let $\mathcal{Q}$ be the set of all queries $\mathcal{A}$ asked. $\mathcal{A}$ wins (output of experiment is 1) iff $m \neq \perp$ and $m \notin \mathcal{Q}$

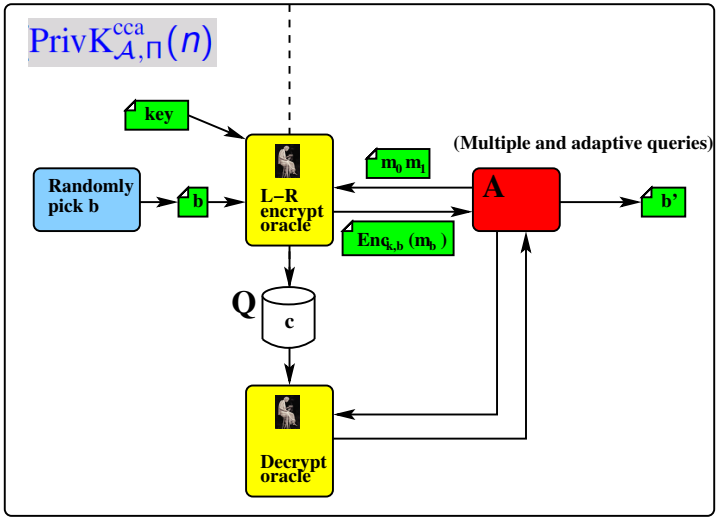
Definition 4.16 (unforgeability)

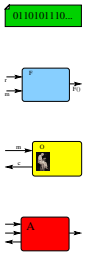
An encryption scheme Π is unforgeable iff for any $\mathcal{A}$, there is a negl, s.t.

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

Authenticated - confidential and unforgeable







Definition 4.17 (for confidentiality as well)

A scheme is an authenticated encryption scheme iff It is CCA-secure and unforgeable.

Why CCA and not CPA?

We allow an adversary to have access to a decryption oracle, since the attack is to find (new) valid ciphertexts and the adversary should be allowed to see multiple copies of valid ciphertexts.

Construction approaches

...from Strong-Mac and CPA-secure

Suppose we have a strongly secure mac, and a CPA-secure encryption scheme. How should we construct an authenticated encryption? Here are three obvious ways, assuming k_1, k_2 and m:

Encrypt-and-authenticate $\mathrm{Enc}_{k_1}(m) \mathbin{+\!\!+} \mathrm{Mac}_{k_2}(m)$

Authenticate-then-Encrypt $\mathrm{Enc}_{k_1}(m \mathbin{+\!\!+} \mathrm{Mac}_{k_2}(m))$

Encrypt-then-Authenticate $\mathrm{Enc}_{k_1}(m) \mathbin{+\!\!+} \mathrm{Mac}_{k_2}(\mathrm{Enc}_{k_1}(m))$

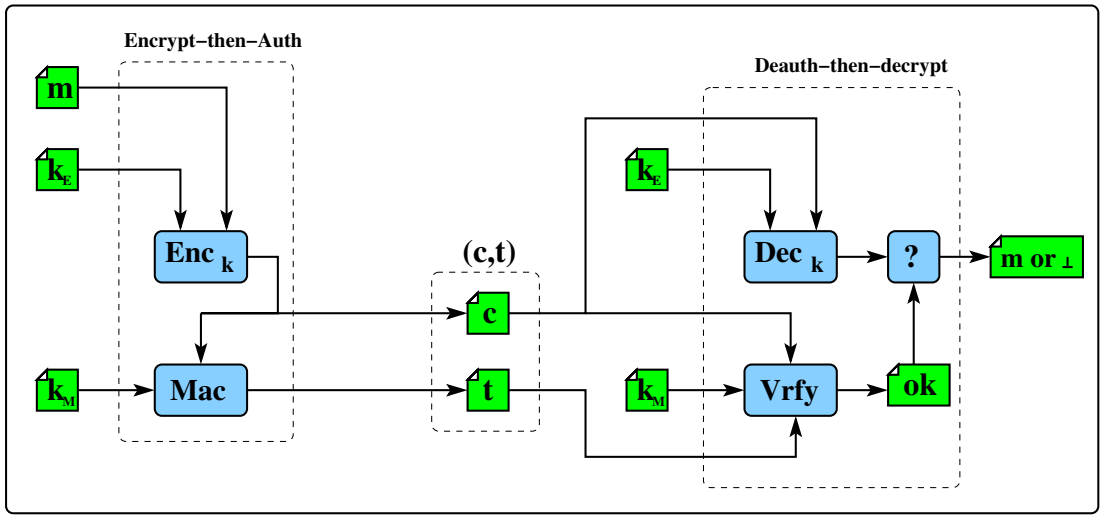
Note that sometimes we say mac instead of authenticate (encrypt-and-mac).

Security of schemes

Encrypt-and-authenticate is not secure. Easy to give counter example. SSH uses this with modifications to make it unforgeable.
Authenticate-then-encrypt also has counter examples. The examples are in a sense contrived. There are systems that adopt authenticate-then-encrypt, e.g. TLS/SSL.
Encrypt-then-authenticate can be shown to be secure. IPSec, SSHv2.

We are going to show that Enc-then-auth is secure.

Construction example



Construction 4.18 (Encrypt-then-authenticate)

Given encryption $\Pi_E = (\mathrm{Gen}', \mathrm{Enc}', \mathrm{Dec}')$ and $\Pi_M = (\mathrm{Mac}, \mathrm{Vrfy})$

$\mathrm{Gen}(1^n)$: output two keys k_E, k_M (important: generate independently)

$\mathrm{Enc}_{\langle k_E, k_M \rangle}(m)$: on input message m, output $\langle c \leftarrow \mathrm{Enc}'_{k_E}(m), t \leftarrow \mathrm{Mac}_{k_M}(c) \rangle$

$\mathrm{Dec}_{\langle k_E, k_M \rangle}(\langle c, t \rangle)$: on input $\langle c, t \rangle$, if $\mathrm{Vrfy}_{k_M}(c, t) = 1$, output $\mathrm{Dec}'_{k_E}(c)$
otherwise output $\perp$

Theorem 4.19

If Π_E is CPA-secure and Π_M is strongly secure, then construction 4.18 (encrypt-then-auth) is an authenticated encryption.

Proof of Theorem 4.19

Overview:

Note that for a particular scheme

$$\text{CPA-secure} \wedge \text{StrongMac} \longrightarrow \text{CCA-secure} \wedge \text{Unforgeable}$$

is equivalent to

$$\text{CPA-secure} \wedge \text{StrongMac} \longrightarrow \text{CCA-secure}$$

AND

$$\text{CPA-secure} \wedge \text{StrongMac} \longrightarrow \text{Unforgeable}$$

Two things to prove - divide and conquer!

Proving $\text{CPA-secure} \wedge \text{StrongMac} \longrightarrow \text{Unforgeable}$ is much easier. The textbook explains in great detail.

Let's focus on $\text{CPA-secure} \wedge \text{StrongMac} \longrightarrow \text{CCA-secure}$.

Proof: CPA-secure $\wedge$ StrongMac $\longrightarrow$ CCA-secure

There are two components to the proof:

1. *(Exploiting Strong Mac - Claim 4.20)* Given an adversary $\mathcal{A}$, we construct an adversary $\mathcal{A}_M$ under the Strong Mac experiment to show that a certain behaviour of $\mathcal{A}$ is similar to $\mathcal{A}_M$: the event that $\mathcal{A}$ submits a valid ciphertext to its decryption oracle, is not far from $\mathcal{A}_M$ winning:

$$\frac{\Pr[\mathcal{A}\text{-ValidQuery}]}{p(n)} \leq \Pr[\mathcal{A}_M\text{-wins}] \leq \text{negl}(n)$$

and so

$$\Pr[\mathcal{A}\text{-ValidQuery}] \leq \text{negl}(n) \qquad (1)$$

2. *(Exploiting CPA-secure - Claim 4.21)* Given the $\mathcal{A}$, we construct an $\mathcal{A}_E$ in the CPA experiment, and show that the event $\mathcal{A}_E$ wins is similar to $\mathcal{A}$ wins the game and yet didn't submit valid ciphertext:

$$\Pr[\mathcal{A}_E\text{-wins}] \geq \Pr[\mathcal{A}\text{-wins} \wedge \overline{\mathcal{A}\text{-ValidQuery}}] \qquad (2)$$

and

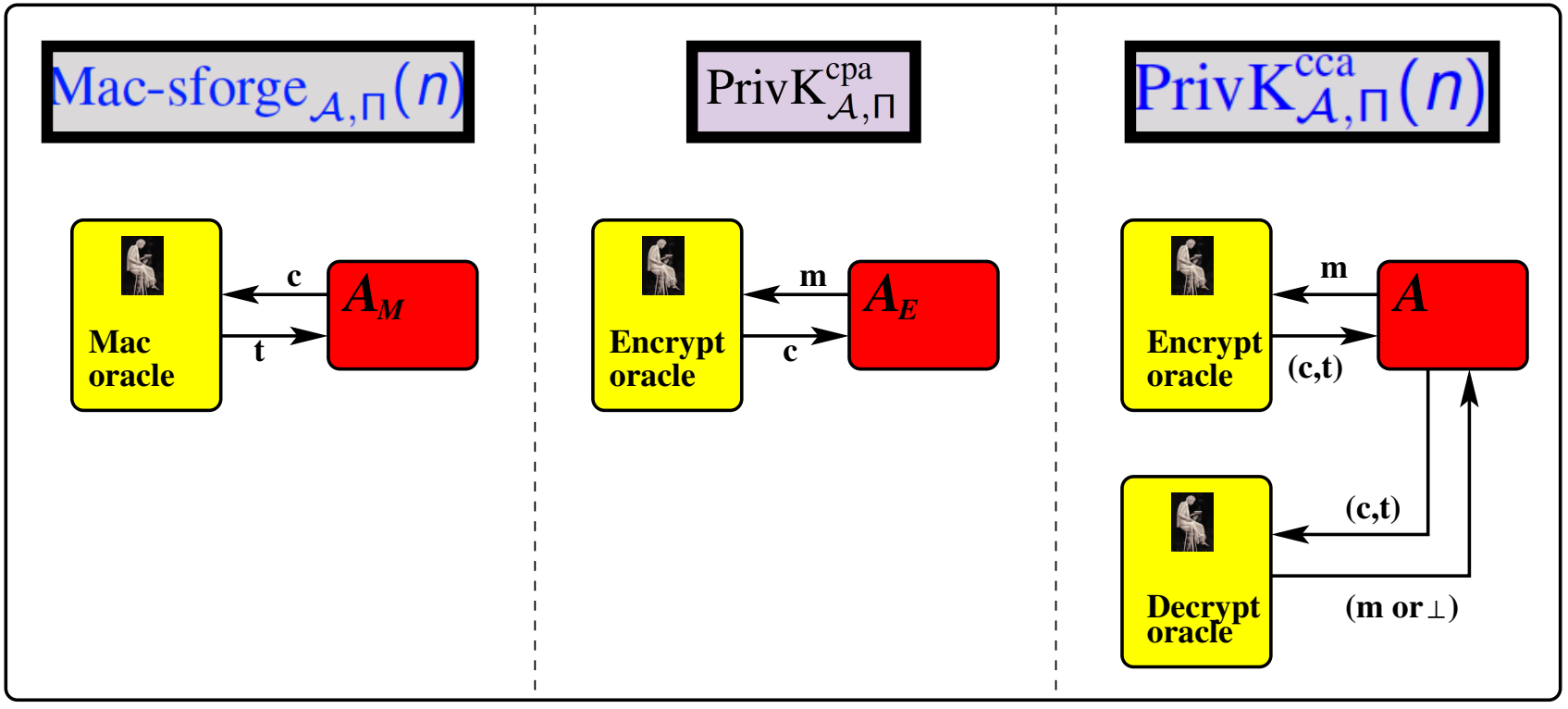
$$\Pr[\mathcal{A}_E\text{-wins}] \leq \tfrac{1}{2} + \text{negl}(n) \qquad (3)$$

Combining (1),(2),(3) we have

$$\Pr[\mathcal{A}\text{-wins}] \leq \frac{1}{2} + \text{negl}(n)$$

■

The three oracles involved...



$\mathcal{A}_M$ (Strong-Mac), $\mathcal{A}_E$ (CPA-secure) and $\mathcal{A}$ (CCA-secure)

Here the adversary $\mathcal{A}$ has access to two oracles: Encryption, and decryption. During the experiment, it may send queries to the two oracles, although, as before, the CCA experiment records the encryptions done, and only wins ($\mathcal{A}$-Wins) if it submits ciphertext not seen before.

Proof: CPA-secure $\wedge$ StrongMac $\longrightarrow$ CCA-secure

Task

We only outline the proof in class, and it may be helpful for you to read the proof in the book.

One may wonder if the rigorous treatment is a bit excessive. The treatment is necessary to catch certain subtleties. If you look at these questions it may be clearer:

1. Why do the keys k_M, k_E have to be generated independently?
2. Why does the Mac needed to be strongly secure?
3. Why do we need to consider the event $\mathcal{A}$-ValidQuery?

Symmetric encryption scheme

$$\mathrm{ENC}_k = (\mathrm{Gen}(1^n), \mathrm{Enc}_k(m), \mathrm{Dec}_k(c)) \qquad \textbf{(ch1)}$$

Properties	Defs	Constructions	Proofs
Perfect secrecy	2.3	(One time pad)	Thm 2.9
↕			Lemma 2.6
Perfect indistinguishability	2.5		
↓			
EAV-secure	3.8	3.17 (PRG)	Thm 3.18
↑			Proof given
EAV-Multi-encryption	3.19		
↑			
CPA-Multi-encryption	3.22		
↕			Thm 3.24
CPA-secure	3.23	3.30 (PRF)	Thm 3.31
↑			
CCA-secure	3.33		
+			
Unforgeable	4.16		
↕			Thm 4.19
Authenticated	4.17	4.18 (Enc-then-Auth)	

MAC scheme

$$\mathrm{MAC}_k = (\mathrm{Gen}(1^n), \mathrm{Mac}_k(m), \mathrm{Vrfy}_k(m,t)) \quad \textbf{(4.1)}$$

Properties	Defs	Constructions	Proofs
Unforgeable	4.2		
↑			Discussed
Strong	4.3	4.5 (Using PRF)	Thm 4.5
		4.7 (Variable length)	Thm 4.8
		4.11 (CBC-MAC)	Thm 4.12
		4.11(a) (Variable CBC-MAC)	