11 Message Authentication Codes

In a chosen-ciphertext attack, the adversary can decrypt arbitrary ciphertexts of its choosing. It would be helpful if there was a way for the decryption algorithm to distinguish between ciphertexts that were honestly generated (by the library itself) and ciphertexts created by the adversary. For example, if the decryption algorithm was able to detect and reject (*i.e.*, raise an error) ciphertexts not created honestly, this could be a promising approach to achieve CCA security.

The problem of *determining the origin* of information is known as **authentication**, and it's a different problem then that of *hiding* information. For example, we may wish to know whether a ciphertext was created honestly by the library itself, while there is no need to *hide* the ciphertext.

In the libraries that define CCA security, honestly generated ciphertexts are generated by the libraries with knowledge of the secret key k, and again decrypted by the library with knowledge of the key. We would like a way to prove to the receiver (decryption procedure) that a piece of information (the ciphertext) was created by someone (the encryption procedure) who knows the secret key.

The tool for the job is called a **message authentication code**, or **MAC** for short. A MAC scheme consists of two algorithms:

- KeyGen: samples a secret key for the MAC
- MAC(k, m): given the secret key k and a plaintext $m \in \mathcal{M}$, output a MAC (sometimes called a "tag") t. Typically MAC is deterministic.

We will follow the (confusing) convention of terminology, and use the term "MAC" to sometimes refer to the scheme itself, and sometimes just the "tag" associated with the message (i.e., "the MAC of m").

Think of a MAC as a kind of signature that will be appended to a piece of data. Only someone with the secret key can generate (and verify) a valid MAC. Our intuitive security requirement is that: an adversary who sees valid MACs of many messages cannot produce a **forgery** — a valid MAC of a *different* message.

11.1 Security Definition

This is a totally new kind of security requirement. It has nothing to do with *hiding* information. Rather, it is about whether the adversary can actually generate a particular piece of data (*e.g.*, a MAC forgery).

To express such a requirement in a security definition, we need a slightly different approach to defining the libraries. Consider a much simpler statement: "no adversary should be able to guess a uniformly chosen value." We can formalize this idea with the following two libraries:

```
$\mathcal{L}_{\text{left}}$
$r \leftarrow \{0,1\}^\lambda$

GUESS($g$):
    return $g \overset{?}{=} r$
```

```
$\mathcal{L}_{\text{right}}$
GUESS($g$):
    return FALSE
```

The left library allows the calling program to attempt to guess a uniformly chosen "target" string. The right library doesn't even bother to verify the calling program's guess — in fact it doesn't even bother to sample a random target string!

Focus on the difference between these two libraries. Their GUESS subroutines give the same output on nearly all inputs. There is only one input r on which they disagree. If a calling program can manage to find that input r, then it can easily distinguish the libraries. Therefore, if the libraries are indistinguishable, it means that the adversary cannot find/generate one of these special inputs! That's the kind of property we want to express.

Indeed, in this case, an adversary who makes q queries to the GUESS subroutine achieves an advantage of at most $q/2^\lambda$. For polynomial-time adversaries, this is a negligible advantage (since q is a polynomial function of λ).

The MAC security definition. Let's follow the pattern from the simple example above. We want to say that no adversary can generate a MAC forgery. So our libraries will provide a mechanism to let the adversary *check* whether it has a forgery. One library will actually perform the check, and the other library will simply assume that a forgery can never happen. The two libraries are different only in how they behave when the adversary calls a subroutine on a *true forgery*. So by demanding that the two libraries be indistinguishable, we are actually demanding that it is difficult for the calling program to generate a forgery.

Definition 11.1 *Let Σ be a MAC scheme. We say that Σ is a* **secure MAC** *if $\mathcal{L}^{\Sigma}_{\text{mac-real}} \approx \mathcal{L}^{\Sigma}_{\text{mac-fake}}$, where:*

```
$\mathcal{L}^{\Sigma}_{\text{mac-real}}$
$k \leftarrow \Sigma.\text{KeyGen}$

GETMAC($m \in \Sigma.\mathcal{M}$):
    return $\Sigma.\text{MAC}(k,m)$

VER($m \in \Sigma.\mathcal{M}, t$):
    return $t \overset{?}{=} \Sigma.\text{MAC}(k,m)$
```

```
$\mathcal{L}^{\Sigma}_{\text{mac-fake}}$
$k \leftarrow \Sigma.\text{KeyGen}$
$\mathcal{T} = \emptyset$

GETMAC($m \in \Sigma.\mathcal{M}$):
    $t := \Sigma.\text{MAC}(k,m)$
    $\mathcal{T} := \mathcal{T} \cup \{(m,t)\}$
    return $t$

VER($m \in \Sigma.\mathcal{M}, t$):
    return $(m,t) \overset{?}{\in} \mathcal{T}$
```

Discussion:

- We do allow the adversary to request MACs of chosen messages, hence the GETMAC subroutine. This means that the adversary is always capable of obtaining valid

MACs. However, MACs that were generated by GETMAC don't count as forgeries — only a MAC of a *different* message would be a forgery.

For this reason, the $\mathcal{L}_{\text{mac-fake}}$ library keeps track of which MACs were generated by GETMAC, in the set $\mathcal{T}$. It is clear that these MACs should always be judged valid by VER. But for all other inputs to VER, $\mathcal{L}_{\text{mac-fake}}$ simply answers false.

- ► The adversary "wins" by successfully finding *any* forgery — a valid MAC of *any* "fresh" message. The definition doesn't care whether it's the MAC of any particular *meaningful* message.

11.2 A PRF is a MAC

The definition of a PRF says (more or less) that even if you've seen the output of the PRF on several chosen inputs, all other outputs look independently uniformly random. Furthermore, uniformly chosen values are hard to guess. Putting these two observations together, and we're close to to the MAC definition. Indeed, a PRF is a secure MAC.

Claim 11.2 *The scheme* $\text{MAC}(k,m) = F(k,m)$ *is a secure MAC, when* F *is a secure pseudorandom function.*

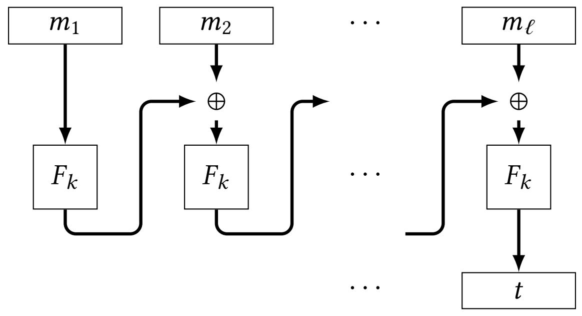
to-do *The main idea of the proof is straight-forward enough. But the presence of a* VER *subroutine in the MAC security definition makes the proof quite tedious. An adversary might make a query* $\text{VER}(m,t')$ *and then later ask for* $\text{GETMAC}(m)$ *for the same* m.

11.3 CBC-MAC

A PRF typically supports only messages of a fixed length, but we will soon see that it's useful to have a MAC that supports longer messages. A classical approach to extend the input size of a MAC involves the CBC block cipher mode applied to a PRF.

Construction 11.3 (CBC-MAC) *Let* F *be a PRF with* $in = out = \lambda$. *Then for every* ***fixed*** *parameter* ℓ, *CBC-MAC refers to the following MAC scheme:*

```
$\text{CBCMAC}^F(k, m_1 \cdots m_\ell)$:
  $t := 0^\lambda$:
  for $i = 1$ to $\ell$:
      $t := F(k, m_i \oplus t)$
  return $t$
```



CBC-MAC differs from CBC encryption mode in two important ways: First, there is no initialization vector. Indeed, CBC-MAC is deterministic (you can think of it as CBC encryption but with the initialization vector fixed to all zeroes). Second, CBC-MAC outputs only the last block.

Claim 11.4 *If F is a secure PRF with in = out = λ, then for every fixed ℓ, CBC-MAC is a secure MAC for message space $\mathcal{M} = \{0,1\}^{\lambda\ell}$.*

to-do *The proof of this claim is slightly beyond the scope of these notes. Maybe I will eventually find a way to incorporate it.*

Note that we have restricted the message space to messages of exactly ℓ blocks. Unlike CBC encryption, CBC-MAC is **not** suitable for messages of variable lengths. If the adversary is allowed to request the CBC-MAC of messages of different lengths, then it is possible for the adversary to generate a forgery (see the homework).

11.4 Encrypt-Then-MAC

Our motivation for studying MACs is that they seem useful in constructing a CCA-secure encryption scheme. The idea is to combine a MAC with a CPA-secure encryption scheme. The decryption algorithm can verify the MAC and raise an error if the MAC is invalid. There are several natural ways to combine a MAC and encryption scheme, but *not all are secure!* (See the exercises.) The safest way is known as encrypt-then-MAC:

Construction 11.5 (Enc-then-MAC) *Let E denote an encryption scheme, and M denote a MAC scheme where $E.\mathcal{C} \subseteq M.\mathcal{M}$ (i.e., the MAC scheme is capable of generating MACs of ciphertexts in the E scheme). Then let EtM denote the **encrypt-then-MAC** construction given below:*

```
$\mathcal{K} = E.\mathcal{K} \times M.\mathcal{K}$
$\mathcal{M} = E.\mathcal{M}$
$\mathcal{C} = E.\mathcal{C} \times M.\mathcal{T}$

KeyGen:
  $k_e \leftarrow E.\text{KeyGen}$
  $k_m \leftarrow M.\text{KeyGen}$
  return $(k_e, k_m)$

Enc($(k_e, k_m), m$):
  $c \leftarrow E.\text{Enc}(k_e, m)$
  $t := M.\text{MAC}(k_m, c)$
  return $(c, t)$

Dec($(k_e, k_m), (c, t)$):
  if $t \neq M.\text{MAC}(k_m, c)$:
    return err
  return $E.\text{Dec}(k_e, c)$
```

Importantly, the scheme computes a MAC *of the CPA ciphertext*, and not of the plaintext! The result is a CCA-secure encryption scheme:

Claim 11.6 *If E has CPA security and M is a secure MAC, then EtM (Construction 11.5) has CCA security.*

Proof As usual, we prove the claim with a sequence of hybrid libraries:

```
$\mathcal{L}^{EtM}_{\text{cca-L}}$
$k_e \leftarrow E.\text{KeyGen}$
$k_m \leftarrow M.\text{KeyGen}$
$\mathcal{S} := \emptyset$

CHALLENGE$(m_L, m_R)$:
  if $|m_L| \neq |m_R|$
    return null
  $c \leftarrow E.\text{Enc}(k_e, m_L)$
  $t \leftarrow M.\text{MAC}(k_m, c)$
  $\mathcal{S} := \mathcal{S} \cup \{ (c,t) \}$
  return $(c,t)$

DEC$(c,t)$:
  if $(c,t) \in \mathcal{S}$ return null
  if $t \neq M.\text{MAC}(k_m, c)$:
    return err
  return $E.\text{Dec}(k_e, c)$
```

The starting point is $\mathcal{L}^{EtM}_{\text{cca-L}}$, shown here with the details of the encrypt-then-MAC construction highlighted. Our goal is to eventually swap m_L with m_R. But the CPA security of E should allow us to do just that, so what's the catch?

To apply the CPA-security of E, we must to factor out the relevant call to E.Enc in terms of the CPA library $\mathcal{L}^{E}_{\text{cpa-L}}$. This means that k_e becomes private to the $\mathcal{L}_{\text{cpa-L}}$ library. But k_e is also used in the last line of the library as $E.\text{Dec}(k_e, c)$. The *CPA* security library for E provides no way to carry out such E.Dec statements!

```
$k_e \leftarrow E.\text{KeyGen}$
$\mathcal{S} := \emptyset$

CHALLENGE$(m_L, m_R)$:
  if $|m_L| \neq |m_R|$
    return null
  $c \leftarrow E.\text{Enc}(k_e, m_L)$
  $t :=$ GETMAC$(c)$
  $\mathcal{S} := \mathcal{S} \cup \{(c,t)\}$
  return $(c,t)$

DEC$(c,t)$:
  if $(c,t) \in \mathcal{S}$
    return null
  if not VER$(c,t)$ :
    return err
  return $E.\text{Dec}(k_e, c)$
```

$\diamond$

```
$\mathcal{L}^{M}_{\text{mac-real}}$
$k_m \leftarrow M.\text{KeyGen}$

GETMAC$(c)$:
  return $M.\text{MAC}(k_m, c)$

VER$(c,t)$:
  return $t \overset{?}{=} M.\text{MAC}(k_m, c)$
```

The operations of the MAC scheme have been factored out in terms of $\mathcal{L}^{M}_{\text{mac-real}}$. Notably, in the DEC subroutine the condition "$t \neq M.\text{MAC}(k_M, c)$" has been replaced with "not VER(c,t)."

```
$k_e \leftarrow E.\text{KeyGen}$
$\mathcal{S} := \emptyset$

CHALLENGE$(m_L, m_R)$:
  if $|m_L| \neq |m_R|$
    return null
  $c \leftarrow E.\text{Enc}(k_e, m_L)$
  $t :=$ GETMAC$(c)$
  $\mathcal{S} := \mathcal{S} \cup \{(c,t)\}$
  return $(c,t)$

DEC$(c,t)$:
  if $(c,t) \in \mathcal{S}$
    return null
  if not VER$(c,t)$:
    return err
  return $E.\text{Dec}(k_e, c)$
```

$\diamond$

```
$\mathcal{L}^M_{\text{mac-fake}}$
$k_m \leftarrow M.\text{KeyGen}$
$\mathcal{T} = \emptyset$

GETMAC$(c)$:
  $t := M.\text{MAC}(k_m, c)$
  $\mathcal{T} := \mathcal{T} \cup \{(c,t)\}$
  return $t$

VER$(c,t)$:
  return $(c,t) \overset{?}{\in} \mathcal{T}$
```

We have applied the security of the MAC scheme, and replaced $\mathcal{L}_{\text{mac-real}}$ with $\mathcal{L}_{\text{mac-fake}}$.

```
$k_e \leftarrow E.\text{KeyGen}$
$k_m \leftarrow M.\text{KeyGen}$
$\mathcal{T} := \emptyset$
$\mathcal{S} := \emptyset$

CHALLENGE$(m_L, m_R)$:
  if $|m_L| \neq |m_R|$
    return null
  $c \leftarrow E.\text{Enc}(k_e, m_L)$
  $t := M.\text{MAC}(k_m, c)$
  $\mathcal{T} := \mathcal{T} \cup \{(c,t)\}$
  $\mathcal{S} := \mathcal{S} \cup \{(c,t)\}$
  return $(c,t)$

DEC$(c,t)$:
  if $(c,t) \in \mathcal{S}$
    return null
  if $(c,t) \notin \mathcal{T}$:
    return err
  return $E.\text{Dec}(k_e, c)$
```

We have inlined the $\mathcal{L}_{\text{mac-fake}}$ library. This library keeps track of a set $\mathcal{S}$ of values for the purpose of the CCA interface, but also a set $\mathcal{T}$ of values for the purposes of the MAC. However, it is clear from the code of this library that $\mathcal{S}$ and $\mathcal{T}$ always have the same contents.

Therefore, the two conditions "$(c,t) \in \mathcal{S}$" and "$(c,t) \notin \mathcal{T}$" in the DEC subroutine are *exhaustive!* The final line of DEC is *unreachable.* This hybrid highlights the intuitive idea that an adversary can either query DEC with a ciphertext generated by CHALLENGE (the $(c,t) \in \mathcal{S}$ case) — in which case the response is null — or with a different ciphertext — in which case the response will be err since the MAC will not verify.

```
$k_e \leftarrow E.\text{KeyGen}$
$k_m \leftarrow M.\text{KeyGen}$
$\mathcal{S} := \emptyset$

CHALLENGE$(m_L, m_R)$:
  if $|m_L| \neq |m_R|$
    return null
  $c \leftarrow E.\text{Enc}(k_e, m_L)$
  $t := M.\text{MAC}(k_m, c)$
  $\mathcal{S} := \mathcal{S} \cup \{(c,t)\}$
  return $(c,t)$

DEC$(c,t)$:
  if $(c,t) \in \mathcal{S}$
    return null
  if $(c,t) \notin \mathcal{S}$:
    return err
  // unreachable
```

The unreachable statement has been removed and the redundant variables $\mathcal{S}$ and $\mathcal{T}$ have been unified. Note that this hybrid library never uses $E.\text{Dec}$, making it possible to express its use of the E encryption scheme in terms of $\mathcal{L}_{\text{cpa-L}}$.

```
$k_m \leftarrow M.\text{KeyGen}$
$\mathcal{S} := \emptyset$

CHALLENGE$(m_L, m_R)$:
  if $|m_L| \neq |m_R|$
    return null
  $c :=$ CHALLENGE$'(m_L, m_R)$
  $t := M.\text{MAC}(k_m, c)$
  $\mathcal{S} := \mathcal{S} \cup \{(c,t)\}$
  return $(c,t)$

DEC$(c,t)$:
  if $(c,t) \in \mathcal{S}$
    return null
  if $(c,t) \notin \mathcal{S}$:
    return err
```

$\diamond$

```
$\mathcal{L}^{E}_{\text{cpa-L}}$

$k_e \leftarrow E.\text{KeyGen}$

CHALLENGE$'(m_L, m_R)$:
  $c := E.\text{Enc}(k_e, m_L)$
  return $c$
```

The statements involving the encryption scheme E have been factored out in terms of $\mathcal{L}_{\text{cpa-L}}$.

We have now reached the half-way point of the proof. The proof proceeds by replacing $\mathcal{L}_{\text{cpa-L}}$ with $\mathcal{L}_{\text{cpa-R}}$, applying the same modifications as before (but in reverse order), to finally arrive at $\mathcal{L}_{\text{cca-R}}$. The repetitive details have been omitted, but we mention that when listing the same steps in reverse, the changes appear very bizarre indeed. For instance, we add an unreachable statement to the DEC subroutine; we create a redundant variable $\mathcal{T}$ whose contents are the same as $\mathcal{S}$; we mysteriously change one instance of $\mathcal{S}$ (the condition of the second if-statement in DEC) to refer to the other variable $\mathcal{T}$. Of course, all of this is so that we can factor out the statements referring to the MAC scheme (along with $\mathcal{T}$) in terms of $\mathcal{L}_{\text{mac-fake}}$ and finally replace $\mathcal{L}_{\text{mac-fake}}$ with $\mathcal{L}_{\text{mac-real}}$. ■

Exercises

11.1. Consider the following MAC scheme, where F is a secure PRF with $in = \lambda$:

```
KeyGen:
  $k \leftarrow \{0,1\}^\lambda$
  return $k$

MAC($k, m_1 \cdots m_\ell$): // each $m_i$ is $\lambda$ bits
  $t := 0^\lambda$
  for $i = 1$ to $\ell$:
    $t := t \oplus F(k, m_i)$
  return $t$
```

Show that the scheme is **not** a secure MAC. Describe a distinguisher and compute its advantage.

11.2. Suppose we expand the message space of CBC-MAC to $\mathcal{M} = (\{0,1\}^\lambda)^*$. In other words, the adversary can request a MAC on any message whose length is an exact multiple of the block length λ. Show that the result is **not** a secure MAC. Construct a distinguisher and compute its advantage.

Hint: Request a MAC on two single-block messages, then use the result to forge the MAC of a two-block message.

★ 11.3. CBC-MAC is similar to CBC encryption, except that it outputs the last block only. For this problem, define a new variant of CBC encryption which outputs the XOR of all of the blocks.

```
NEWMAC$^F$($k, m_1 \cdots m_\ell$):
  $t := 0^\lambda$:
  for $i = 1$ to $\ell$:
    $t := t \oplus F(k, m_i \oplus t)$
  return $t$
```

1. Is NEWMAC a secure MAC for message space $\mathcal{M} = (\{0,1\}^n)^*$?
2. Is NEWMAC a secure MAC for message space $\mathcal{M} = \{0,1\}^{n\ell}$ (with fixed ℓ)?

11.4. Let E be a CPA-secure encryption scheme and M be a secure MAC. Show that the following encryption scheme (called encrypt & MAC) is **not** CCA-secure:

```
E&M.KeyGen:
  $k_e \leftarrow E$.KeyGen
  $k_m \leftarrow M$.KeyGen
  return $(k_e, k_m)$

E&M.Enc($(k_e, k_m), m$):
  $c \leftarrow E$.Enc($k_e, m$)
  $t := M$.MAC($k_m, m$)
  return $(c, t)$

E&M.Dec($(k_e, k_m), (c, t)$):
  $m := E$.Dec($k_e, c$)
  if $t \neq M$.MAC($k_m, m$):
    return err
  return $m$
```

Describe a distinguisher and compute its advantage.

11.5. Let E be a CPA-secure encryption scheme and M be a secure MAC. Show that the following encryption scheme Σ (which I call encrypt-and-encrypted-MAC) is **not** CCA-secure:

```
$\Sigma$.KeyGen:
    $k_e \leftarrow E$.KeyGen
    $k_m \leftarrow M$.KeyGen
    return $(k_e, k_m)$

$\Sigma$.Enc$((k_e, k_m), m)$:
    $c \leftarrow E$.Enc$(k_e, m)$
    $t := M$.MAC$(k_m, m)$
    $c' \leftarrow E$.Enc$(k_e, t)$
    return $(c, c')$

$\Sigma$.Dec$((k_e, k_m), (c, c'))$:
    $m := E$.Dec$(k_e, c)$
    $t := E$.Dec$(k_e, c')$
    if $t \neq M$.MAC$(k_m, m)$:
        return err
    return $m$
```

Describe a distinguisher and compute its advantage.

★ 11.6. In Construction 8.4, we encrypt one plaintext block into two ciphertext blocks. Imagine applying the Encrypt-then-MAC paradigm to this encryption scheme, but (erroneously) computing a MAC of *only* the second ciphertext block.

In other words, let F be a PRP with block length $blen = \lambda$, and let M be a MAC scheme for message space $\{0,1\}^\lambda$. Define the following encryption scheme:

```
KeyGen:
    $k_e \leftarrow \{0,1\}^\lambda$
    $k_m \leftarrow M$.KeyGen
    return $(k_e, k_m)$

Enc$((k_e, k_m), m)$:
    $r \leftarrow \{0,1\}^\lambda$
    $x := F(k_e, r) \oplus m$
    $t := M$.MAC$(k_m, x)$
    return $(r, x, t)$

Dec$((k_e, k_m), (r, x, t))$:
    if $t \neq M$.MAC$(k_m, x)$:
        return err
    else return $F(k_e, r) \oplus x$
```

Show that the scheme does **not** have CCA security. Describe a successful attack and compute its advantage.

Hint: Obtain valid encryptions (r, x, t), (r', x', t'), and (r'', x'', t'') of unknown plaintexts m, m', m''. Consider what information about the plaintexts is leaked via:

$$\text{Dec}((k_e, k_m), (r', x, t)) \oplus \text{Dec}((k_e, k_m), (r'', x, t)) \oplus x' \oplus x''.$$