CS-E4340 Cryptography: Exercise Sheet 4

—Message Authentication Codes (MACs) & Pseudorandom Functions (PRFs)—

Submission Deadline: October 3, 11:30 via MyCourses

Each exercise can give up to two participation points, 2 for a mostly correct solution and 1 point for a good attempt. Overall, the exercise sheet gives at most 4 participation points.

Exercise Sheet 4 is intended to help...

(a) ...understand the *definition* of a message authentication code (MAC).
(b) ...understand the *relation* between PRFs and MACs.
(c) ...understand and practice how to define the security for a cryptographic primitive via security notions.

Exercise 1 shows that a UNF-CMA-secure MAC might leak the message it authenticates.
Exercise 2 explores the differences/similarities between one-way functions and UNF-CMA-secure MAC schemes.
Exercise 3 constructs variants of MAC schemes and the goal is to distinguish modifications which harm security from modifications which don't harm security.
Exercise 4 deepens understanding of the UNF-CMA game and provide some definitions in which no scheme can be secure. In this case, we say that these are variants of the MAC game which are *trivial to break* (trivial does not mean that the exercise is easy. Rather, it means that the constructions are not meaningful.).
Exercise 5 is an advanced counterexample for showing that not every UNF-CMA-secure MAC is a one-way function. (In fact, every UNF-CMA-secure MAC is a so-called *distributional* one-way function. Ask Miikka Tiainen if you are curious on the definition and its implications.)
Hint: *Lecture 4* and the beginning of *Lecture 5* cover message authentication codes, so you can have a look at both lecture videos to help with this exercise sheet.

Exercise 1 (MACs can leak the message)**.** Let m_1 be a UNF-CMA secure MAC scheme. Prove that also m_2 is a UNF-CMA secure MAC scheme, where

$$m_2.\mathsf{mac}(k, x) := m_1.\mathsf{mac}(k, x)||x$$

and

$m_2.\mathsf{ver}(k, x, t)$

assert $t \neq \bot$
$t' \leftarrow t_{1\ldots|t|-|x|}$
$x' \leftarrow t_{|t|-|x|+1\ldots|t|}$
if $x' \neq x$
  return 0
return $m_1.\mathsf{ver}(k, x, t')$

Hint: You need to provide a reduction in pseudo-code (main task) and show that the reduction works. In order to show that the reduction works as it should (explain in your solution what this means), you can either provide the main conceptual argument (in text) or an inlining proof (in pseudocode) as in the lecture.

Solution 1. We want to prove that for all PPT adversaries $\mathcal{A}$

$$\mathsf{Adv}^{\texttt{Gunf-cma}_{m_2}}_{m_2,\mathcal{A}}(\lambda) = |\Pr\left[1 = \mathcal{A} \rightarrow \texttt{Gunf-cma}^0_{m_2}\right] - \Pr\left[1 = \mathcal{A} \rightarrow \texttt{Gunf-cma}^1_{m_2}\right]|$$

is negligible when we know that

$$\mathsf{Adv}^{\texttt{Gunf-cma}_{m_1}}_{m_1,\mathcal{A}}(\lambda) = |\Pr\left[1 = \mathcal{A} \rightarrow \texttt{Gunf-cma}^0_{m_1}\right] - \Pr\left[1 = \mathcal{A} \rightarrow \texttt{Gunf-cma}^1_{m_1}\right]|$$

is negligible.
The conceptual argument is that the adversary should not benefit from the leaked message, since the adversary chooses the message himself, he knows it anyway and could append it to the MAC himself. Let's now make this argument rigorous.
Assume towards contradiction that there exists an efficient adversary $\mathcal{A}$ for which $\mathsf{Adv}^{\texttt{Gunf-cma}_{m_2}}_{m_2,\mathcal{A}}(\lambda)$ is non-negligible. Consider the following reduction *package* $\mathcal{R}$:

$\mathcal{R}$

$\mathsf{MAC}_{\mathcal{R}}(x)$

$t \leftarrow \mathsf{MAC}(x)$
return $t||x$

$\mathsf{VERIFY}_{\mathcal{R}}(x, t)$

assert $t \neq \bot$
$t' \leftarrow t_{1\ldots|t|-|x|}$
$x' \leftarrow t_{|t|-|x|+1\ldots|t|}$
if $x' \neq x$
  return 0
$d \leftarrow \mathsf{VERIFY}(x, t')$
return d

Now if we create a new adversary $\mathcal{B} := \mathcal{A} \to \mathcal{R}$ (i.e., replacing all $\mathcal{A}$'s calls to MAC by $\mathsf{MAC}_{\mathcal{R}}$ and $\mathcal{A}$'s calls to VERIFY by $\mathsf{VERIFY}_{\mathcal{R}}$), the resulting adversary $\mathcal{B}$ is efficient and it can distinguish $\texttt{Gunf-cma}^0_{m_1}$ and $\texttt{Gunf-cma}^1_{m_1}$. Let's prove the latter by code-equivalence. The real and ideal games $\texttt{Gunf-cma}^0_{m_2}$ and $\texttt{Gunf-cma}^1_{m_2}$ (which $\mathcal{A}$ can distinguish) are defined as:

$\underline{\underline{\texttt{Gunf-cma}^0_{m_2}}}$

$\underline{\mathsf{MAC}(x)}$

if $k = \bot$

  $k \leftarrow\!\!\$\ \{0,1\}^\lambda$

$t \leftarrow m_2.\mathsf{mac}(k, x)$

return t

$\underline{\mathsf{VERIFY}(x, t)}$

assert $t \neq \bot$

if $k = \bot$

  $k \leftarrow\!\!\$\ \{0,1\}^\lambda$

$d \leftarrow m_2.\mathsf{ver}(k, x, t)$

return d

$\underline{\underline{\texttt{Gunf-cma}^1_{m_2}}}$

$\underline{\mathsf{MAC}(x)}$

if $k = \bot$

  $k \leftarrow\!\!\$\ \{0,1\}^\lambda$

$t \leftarrow m_2.\mathsf{mac}(k, x)$

$\mathcal{L} \leftarrow \mathcal{L} \cup \{(x, t)\}$

return t

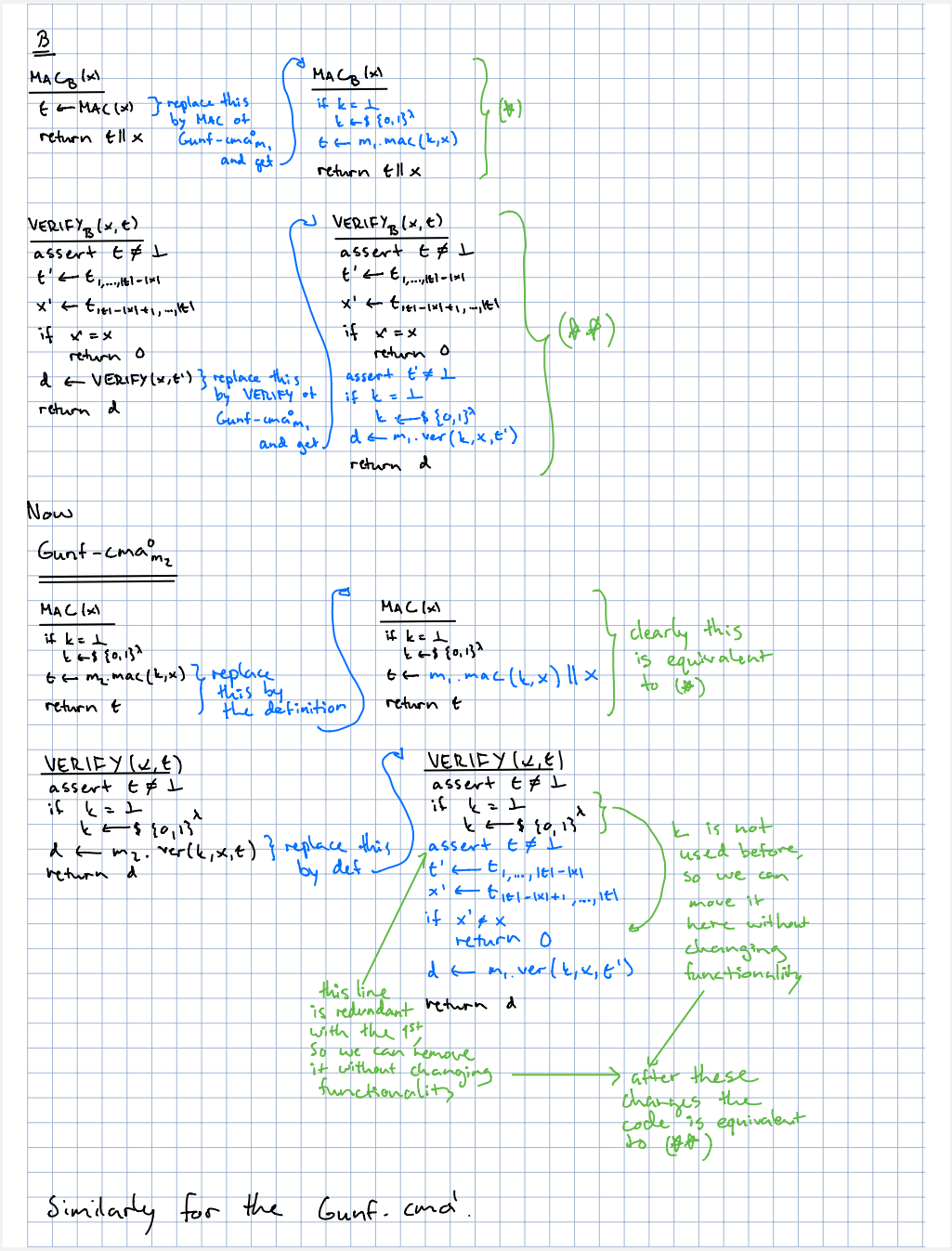
$\underline{\mathsf{VERIFY}(x, t)}$

if $(x, t) \in \mathcal{L}$:

  return 1

return 0

The rest of the proof can be found in the figure below.



While the proof for $\mathcal{R} \to \texttt{Gunf-cma}^1_{m_1} \overset{\text{code}}{\equiv} \texttt{Gunf-cma}^1_{m_2}$ is similar, we need to argue about the content of $\mathcal{L}$. Let us denote the list $\mathcal{L}$ in $\texttt{Gunf-cma}^1_{m_i}$ by $\mathcal{L}_i$ for $i \in \{0,1\}$. Now, $\mathcal{L}_1$ contains tags of the form (x,t), where $t \leftarrow\!\$\ m_1.\mathsf{mac}(k,x)$. In turn, $\mathcal{L}_2$ contains tags of the form $(x, t||x)$, where $t \leftarrow\!\$\ m_1.\mathsf{mac}(k,x)$. The input-output behaviour of the MAC oracles is not affected by this difference, because it only writes to the list, but never reads from the list. In turn, the VERIFY oracles behave the same, because the check $(x,t') \in \mathcal{L}_2$ is equivalent to splitting $t' = t'_{\text{left}}||t'_{\text{right}}$ and checking that $(x, t'_{\text{left}}) \in \mathcal{L}_1$ *and* $x = t'_{\text{right}}$.

But in $\mathcal{R} \to \texttt{Gunf-cma}^1_{m_1}$, the reduction already checks that $x = t'_{\text{right}}$, so this additional check in the list is redundant and VERIFY of $\mathcal{R} \to \texttt{Gunf-cma}^1_{m_1}$ and $\texttt{Gunf-cma}^1_{m_2}$ also have the same input-output behaviour.
So $\mathcal{R} \to \texttt{Gunf-cma}^i_{m_1}$ has equivalent code to $\texttt{Gunf-cma}^i_{m_2}$ for $i = 0$ and 1. Hence, $\mathcal{A} \to \mathcal{R} \to \texttt{Gunf-cma}^i_{m_1}$ has equivalent code to $\mathcal{A} \to \texttt{Gunf-cma}^i_{m_2}$. Since $\mathcal{A}$ can distinguish $\texttt{Gunf-cma}^i_{m_2}$, it means that $\mathcal{A} \to \mathcal{R}$ (i.e. $\mathcal{B}$) can distinguish $\texttt{Gunf-cma}^i_{m_1}$. But this is a contradiction, since m_1 was secure, which concludes the proof.

Remark. The code of the reduction package $\mathcal{R}$ is very similar to the code of m_2. Conceptually, however, $\mathcal{R}$ is very different from m_2, because $\mathcal{R}$ does not know the key. That is, $\mathcal{R}$ is a package which transforms the adversary interface without knowing the key. The fact that the code of the reduction $\mathcal{R}$ is very similar to the code of the construction m_2 is very natural and will often happen in this course. Osama Abuzaid (who you also saw in one of the videos and who used to be a TA in this course before) used to call such a reduction a C.A.R., a construction analogous reduction.

Exercise 2 (OWFs & MACs)**.** Prove or disprove: For all one-way functions f, m is an UNF-CMA secure MAC scheme, where $m.\mathsf{mac}(k, x) = f(k||x)$ and

$$
\begin{array}{l}
\underline{m.\mathsf{ver}(k, x, t)} \\
t' \leftarrow f(k||x) \\
\textbf{if } t' \neq t \\
\quad \textbf{return } 0 \\
\textbf{else} \\
\quad \textbf{return } 1
\end{array}
$$

Hint: If you believe that this is an UNF-CMA secure MAC scheme, see hint for Exercise 1. If you believe that this is not necessarily an UNF-CMA secure MAC scheme, define a counterexample OWF f and provide an adversary in pseudocode and show that its advantage is non-negligible. See lecture notes for Lecture 4 for an example on how to write such adversary pseudocode.

Solution 2. Not true. Consider for example $f(x) := f_1(x_{1\ldots|x|/2})||0^{|x|/2}$ where f_1 is a length-preserving OWF. Such f is a OWF (this can be proven very similarly to examples in Exercise sheet 1). This OWF is a bad MAC because it doesn't use half of the input, but it doesn't use the message at all, just the key.
Now consider the following adversary:

$$
\begin{array}{l}
\underline{\mathcal{A}(1^\lambda)} \\
t \leftarrow \mathsf{MAC}(1^\lambda) \\
d \leftarrow \mathsf{VERIFY}(1^{\lambda-1}||0, t) \\
\textbf{return } d
\end{array}
$$

Now in the real and the ideal game $\texttt{Gunf-cma}^i_m$, in both cases t will get the value $f(k||1^\lambda)$. In the real case $\texttt{Gunf-cma}^0_m$, the value of d will always be 1, because

$$
\begin{aligned}
f(k||1^\lambda) &= f_1(k)||0^{\lambda/2} \\
&= f(k||1^{\lambda-1}||0)
\end{aligned}
$$

so the tag verifies.
However, in the ideal case $\texttt{Gunf-cma}^1_m$ the value of d will always be 0, because the message $1^{\lambda-1}||0$ is not in the table T.

Hence, the distinguishing advantage of $\mathcal{A}$ is $1 - 0 = 1$ which is non-negligible.

Remark. The above counterexample lets f use x without using k. One could also, e.g., choose a OWF f which leaks the key k and then use the key k to break the unforgeability.

Exercise 3 (Candidate MAC schemes)**.** Let f be a secure (λ, λ)-PRF. Consider the four MAC schemes m_1, m_2, m_3, m_4 given below.

1. Which of these MAC-schemes are UNF-CMA and which are not? Justify your intuition.

2. Choose one of the MAC schemes m_i that you think is not UNF-CMA secure. Provide an adversary $\mathcal{A}$ (in pseudocode) that can distinguish between the real and ideal games $\texttt{Gunf-cma}_m^b$ (see Chapter 3 in the Crypto Companion). An intuitive explanation for why the adversary works suffices in this exercise.

Note: Below, the syntax $x[i]$ refers to the value assigned to x during the ith loop. Moreover, $y[i] \leftarrow f(k, x[i])$ is the output value of f during the ith loop.

$m_1.\mathsf{mac}(k, x)$

$j \leftarrow \lambda - (|x| \bmod \lambda)$

$x' \leftarrow x||0^j$, $h \leftarrow \frac{|x'|}{\lambda}$

for $i = 0$ **until** $h - 1$
  $\ell \leftarrow \lambda i + 1$
  $r \leftarrow \lambda i + \lambda$
  $x[i] \leftarrow x'_{\ell..r}$
  $y[i] \leftarrow f(k, x[i])$
$t \leftarrow (y[0], y[1], ..., y[h-1])$
return t

$m_2.\mathsf{mac}(k, x)$

$j \leftarrow \lambda - (|x| \bmod \lambda)$

$x' \leftarrow x||0^j$, $h \leftarrow \frac{|x'|}{\lambda}$

$y[-1] \leftarrow 0^\lambda$
for $i = 0$ **until** $h - 1$
  $\ell \leftarrow \lambda i + 1$
  $r \leftarrow \lambda i + \lambda$
  $x[i] \leftarrow x'_{\ell..r}$
  $y[i] \leftarrow f(k, x[i] \oplus y[i-1])$
$t \leftarrow (y[0], y[1], ..., y[h-1])$
return t

$m_3.\mathsf{mac}(k, x)$

$j \leftarrow \lambda - (|x| + 1) \bmod \lambda)$

$x' \leftarrow x||1||0^j$, $h \leftarrow \frac{|x'|}{\lambda}$

$y[-1] \leftarrow 0^\lambda$
for $i = 0$ **until** $h - 1$
  $\ell \leftarrow \lambda i + 1$
  $r \leftarrow \lambda i + \lambda$
  $x[i] \leftarrow x'_{\ell..r}$
  $y[i] \leftarrow f(k, x[i] \oplus y[i-1])$
$t \leftarrow (y[0], y[1], ..., y[h-1])$
return t

$m_4.\mathsf{mac}(k, x)$

$j \leftarrow \lambda - ((|x| + 1) \bmod \lambda)$

$x' \leftarrow x||1||0^j$, $h \leftarrow \frac{|x'|}{\lambda}$

$y[-1] \leftarrow 0^\lambda$
for $i = 0$ **until** $h - 1$
  $\ell \leftarrow \lambda i + 1$
  $r \leftarrow \lambda i + \lambda$
  $x[i] \leftarrow x'_{\ell..r}$
  $y[i] \leftarrow f(k, x[i] \oplus y[i-1])$
$t \leftarrow y[h-1]$
return t

$m_1.\mathsf{ver}(k, x, t)$

$j \leftarrow \lambda - (|x| \bmod \lambda)$

$x' \leftarrow x||0^j$, $h \leftarrow \frac{|x'|}{\lambda}$

$(y[0], y[1], ..., y[h-1]) \leftarrow t$
for $i = 0$ **until** $h - 1$
  $\ell \leftarrow \lambda i + 1$
  $r \leftarrow \lambda i + \lambda$
  $x[i] \leftarrow x'_{\ell..r}$
  if $y[i] \neq f(k, x[i])$

$m_2.\mathsf{ver}(k, x, t)$

$j \leftarrow \lambda - (|x| \bmod \lambda)$

$x' \leftarrow x||0^j$, $h \leftarrow \frac{|x'|}{\lambda}$

$y[-1] \leftarrow 0^\lambda$
$(y[0], y[1], ..., y[h-1]) \leftarrow t$
for $i = 0$ **until** $h - 1$
  $\ell \leftarrow \lambda i + 1$
  $r \leftarrow \lambda i + \lambda$
  $x[i] \leftarrow x'_{\ell..r}$
  if $y[i] \neq f(k, x[i] \oplus y[i-1])$

$m_3.\mathsf{ver}(k, x, t)$

$z \leftarrow \lambda - ((|x| + 1) \bmod \lambda)$

$x' \leftarrow x||1||0^z$, $h \leftarrow \frac{|x'|}{\lambda}$

$y[-1] \leftarrow 0^\lambda$
$(y[0], ..., y[h-1]) \leftarrow t$
for $i = 0$ **until** $h - 1$
  $\ell \leftarrow \lambda i + 1$
  $r \leftarrow \lambda i + \lambda$
  $x[i] \leftarrow x'_{\ell..r}$
  if $y[i] \neq f(k, x[i] \oplus y[i-1])$

$m_4.\mathsf{ver}(k, x, t)$

$z \leftarrow \lambda - (|x| + 1) \bmod \lambda)$

$x' \leftarrow x||1||0^z$, $h \leftarrow \frac{|x'|}{\lambda}$

$y[-1] \leftarrow 0^\lambda$
for $i = 0$ **until** $h - 1$
  $\ell \leftarrow \lambda i + 1$
  $r \leftarrow \lambda i + \lambda$
  $x[i] \leftarrow x'_{\ell..r}$
  $y[i] \leftarrow f(k, x[i] \oplus y[i-1])$
  if $\lceil |x'| \rceil$

Solution 3. None of the four schemes are UNF-CMA secure. Example adversaries to attack each scheme are given below. In the following, we consider the case that $\lambda = 128$. That is, the PRF takes inputs of length 128.

- m_1 computes the MAC tag by applying f to each block of λ bits. Like ECB mode in block ciphers, this leaves a lot of attacks open. For example, for the case that $\lambda = 128$, the adversary can compute tag (t_1, t_2) for any 256-bit message (x_1, x_2), and then t_1 is a valid MAC tag for x_1. Now adversary can call the verification oracle with this pair: real game computes $f(k, x_1)$ and verifies the tag, but the ideal game rejects it, since x_1 has not been MAC'd previously.
- m_2 uses the tag of the previous block to affect the tag of the next block, so simple reordering is not possible. However, the simple padding with zeros causes problems: for example, two messages 0^{100} and 0^{101} are both padded to $x' = 0^{256}$, and therefore have the same MAC tag output. Knowing this, the adversary can only MAC one of them and gain a new valid (message, tag)-pair, and distinguish the games.
- In m_3, the padding problem is fixed. However, outputting the tags for each part of the chain leaves the scheme open to a truncating attack. For example, the adversary can compute the MAC tag (t_1, t_2) for message 1^{255} (which is padded to $x' = 1^{256}$). Then adversary knows that t_1 is a valid tag for 1^{127} (which is padded to $x' = 1^{256}$), and wins the game.
- m_4 seems to have fixed both padding problems and truncating attack vulnerabilities. However, more advanced attacks are available by XORing the computed tags into the messages. For example, the adversary can request MAC tag t for message $x = 1$ (which is padded to $x' = 11||0^{126}$). This tag is defined as $t = f(k, x')$. Now, knowing t, the adversary can form a new message $m = x'||(t \oplus x')$. This message has tag

$$f(k, (t \oplus x') \oplus f(k, x')) = f(k, f(k, x') \oplus x' \oplus f(k, x')) = f(k, x') = t$$

  and therefore the adversary has found a new valid (message,tag)-pair.

$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
$x \leftarrow 0^{256}$	$x \leftarrow 0^{64}$	$x \leftarrow 1^{255}$	$x \leftarrow 1$
$t \leftarrow \mathsf{MAC}(x)$	$t \leftarrow \mathsf{MAC}(x)$	$t \leftarrow \mathsf{MAC}(x)$	$t \leftarrow \mathsf{MAC}(x)$
$t' \leftarrow t_{1..128}$	$x' \leftarrow 0^{128}$	$t' \leftarrow t_{1..128}$	$x' \leftarrow 11\|\|0^{126}\|\|(t \oplus (11\|\|0^{126})$
$x' \leftarrow 0^{128}$	$b \leftarrow \mathsf{VERIFY}(x', t)$	$x' \leftarrow 1^{127}$	$b \leftarrow \mathsf{VERIFY}[m](x', t)$
$b \leftarrow \mathsf{VERIFY}(x', t')$	**if** $b = 1$	$b \leftarrow \mathsf{VERIFY}(x', t')$	**if** $b = 1$
if $b = 1$	**return** 0	**if** $b = 1$	**return** 0
return 0	**else return** 1	**return** 0	**else return** 1
else return 1		**else return** 1	

All adversaries above do a constant number of oracle calls and are therefore efficient. Each of the adversaries provides a distinguishing probability of 1. In the real game, the new (message,tag)-pairs created are always valid (as was argued above), while in the ideal game, the pairs are always rejected since they haven't been added into the list of $\mathcal{L}$. Note that there is no chance the adversary fails here. This is because the ideal game behaves in a deterministic way, putting the produced tags into a list and then later checking if they are in the list.

Remark. There are many other options for adversaries. For example, for m_2, you could also consider the adversary $t \leftarrow \mathsf{MAC}(0^\lambda)$, **return** $\mathsf{VERIFY}(0^\lambda||t, (t, t))$. These schemes are completely broken and there are many ways to attack them.

Exercise 4. (Security Models, Weak Unforgeability) In this exercise, we aim to understand what would be a good model for a weak unforgeability definition that captures that only the message m is authenticated whereas the tag might be malleable, i.e., the adversary might be able to come up with a second, valid tag for a message m given a first valid tag for message m. We capture weak unforgeability as computational indistinguishability between a real game $\texttt{Gwunf-cma}^0_m$ and an ideal game $\texttt{Gwunf-cma}^1_m$. We define the real game for weak unforgeability under chosen message attacks (wUNF-CMA) as the real game for UNF-CMA security, i.e., $\texttt{Gwunf-cma}^0_m := \texttt{Gunf-cma}^0_m$, where m is a message authentication scheme. Below, we give three candidates for the ideal game $\texttt{Gwunf-cma}^1_m$. You need to choose one candidate such that weak unforgeability (integrity on the message only) is best captured. Justify your choice.

Package Parameters

λ : security parameter
m : MAC scheme

Package State

k : k
$\mathcal{L}$: list

MAC(x)

if $k = \bot$
  $k \leftarrow\!\$\ \{0,1\}^\lambda$
$t \leftarrow m.\mathsf{mac}(k, x)$
$\mathcal{L} \leftarrow \mathcal{L} \cup \{x\}$
return t

VERIFY(x, t)

assert $x \neq \bot$
if $(x) \in \mathcal{L}$ **and if** $m.\mathsf{ver}(k, x, t) = 1$
  return 1
else return 0

Package Parameters

λ : security parameter
m : MAC scheme

Package State

k : k
$\mathcal{L}$: list

MAC(x)

if $k = \bot$
  $k \leftarrow\!\$\ \{0,1\}^\lambda$
$t \leftarrow m.\mathsf{mac}(k, x)$
$\mathcal{L} \leftarrow \mathcal{L} \cup \{x\}$
return t

VERIFY(x, t)

assert $x \neq \bot$
if $(x) \in \mathcal{L}$
  return 1
else return 0

Package Parameters

λ : security parameter
m : MAC scheme

Package State

k : k
$\mathcal{L}$: list

MAC(x)

if $k = \bot$
  $k \leftarrow\!\$\ \{0,1\}^\lambda$
$t \leftarrow m.\mathsf{mac}(k, x)$
$\mathcal{L} \leftarrow \mathcal{L} \cup \{(x, t)\}$
return t

VERIFY(x, t)

assert $x \neq \bot$
if $(x, t) \in \mathcal{L}$
  return 1
else return 0

Solution 4. The model on the left is the most accurate one for capturing weak unforgeability. Consider the MAC oracle. After generating a tag t, the message x is stored in the list $\mathcal{L}$. This means that $\mathcal{L}$ consists only of the messages that were actually used within the $m.\mathsf{mac}$ algorithm for generating a tag t. Consider now the VERIFY oracle. VERIFY returns 1 as long as (1) the message x is in the list $\mathcal{L}$ and (2) the $m.\mathsf{ver}$ returns a 1. (2) checks for the correctness of the verify algorithm.

The model in the middle is similar to the model on the left. The difference is on the VERIFY oracle, which only checks if the message x is on the set $\mathcal{L}$, but does not check the correctness of the $m.\mathsf{ver}$ algorithm. This model does not capture the desired indistinguishable security property, since an adversary could use any tag known to be not correct to pass the ideal VERIFY. Consider the following adversary $\mathcal{A}$:

$\mathcal{A}$

$x \leftarrow\!\!\$\ \{0,1\}^\lambda$
$t' \leftarrow\!\!\$\ \{0,1\}^\lambda$
if $1 \leftarrow \mathsf{VERIFY}(x, t')$
  return 0
else
  $t \leftarrow \mathsf{MAC}(x)$
  if $1 \leftarrow \mathsf{VERIFY}(x, t')$
    return 1
  else return 0

$\mathcal{A}$ always distinguishes the middle model, so the middle option doesn't model any security at all!

The model in the right captures a stronger unforgeability property and is the definition that was discussed in the lecture. Namely, it captures that the message x is authenticated but also that the tag t is authenticated. As we can see in the MAC oracle, the list $\mathcal{L}$ consists of message and tag pairs. The VERIFY algorithm checks thus if the given message-tag pair is in the list.

Exercise 5. (Advanced counterexamples) Assume the existence of secure $(*, \lambda)$-PRFs. Prove that there exists an UNF-CMA-secure MAC scheme m such that the function

$$f_m : z \mapsto x||m.\mathsf{mac}(k, x) \text{ where } k = z_{1..\ell} \text{ and } x = z_{\ell+1..|x|} \text{ for } \ell = \left\lceil \frac{|z|}{2} \right\rceil$$

is not a one-way function. Notation: k is the first half of the input of f_m, and x is the second half of the input of f_m. When the input is of odd length, then x has one bit more than k.

Hint: Consult the counterexample theorems in the Crypto Companion.

Solution 5. (Sketch of solution)
The following 2 theorems are useful:

Theorem 1: When modifying an UNF-CMA-secure MAC scheme on a negligible fraction of the keys, then the resulting MAC-scheme is also UNF-CMA-secure.

Theorem 2: When modifying an UNF-CMA-secure MAC scheme by running it on the λ/c security parameter version, then it still remains UNF-CMA-secure, if c is a constant.

Solution to the exercise: Let's start with a MAC scheme that is built from a $(*, \lambda)$-PRF and thus has tags of length λ. First apply Theorem 2 so that we can ignore $\frac{c-1}{c}$ bits of the key of the MAC scheme and have space to play around. Choose $c = 4$ and divide the key into 4 equally long parts (or round...), part 1, part 2, part 3 and part 4:
The first part of the key is the MAC-key for the original MAC-scheme. That is okay by Theorem 2. When the second part of the key is all-zeroes, then our MAC-scheme will behave weirdly. This is okay by Theorem 1. The weird behaviour is that if the message is equal to the third part, then the scheme outputs the first $\frac{\lambda}{c}$ bits of part 4 of the key

as MAC. This yields an easy pre-image for any single message-tag pair (if the message is of length $\frac{\lambda}{c}$), although the MAC is secure.