I collaborated with ********** INSERT COLLABORATORS HERE (INDICATING SPECIFIC PROBLEMS) **********.

Problem 5-1 (Block-by-block MAC Composition) 14 Points

Our method of choice so far for MAC-ing long messages was to use a universal hash function. In this problem you will examine several more straightforward (but less efficient and not necessarily “secure”) ways to MAC long messages. Assume we have a secure MAC \((G, T, V)\) that operates on \(k\)-bit messages. We want to design a MAC for much longer messages. For concreteness, assume we want to MAC messages of length up to \(k^3\) (in particular, the length could be variable). Given a corresponding block length \(l\), we write \(m = m_1 \ldots m_n\) (so that \(nl\) is the total length; for simplicity, we assume that all message lengths are multiples of \(l\)). For each of the following suggestions \(M_s\), determine if \(M_s\) is a secure MAC. In case the scheme is insecure, determine the smallest number of calls to the MAC-ing oracle that suffice for the forgery. Indicate also which insecure schemes for variable-length messages become secure with fixed-length (i.e., \(n\) is fixed) messages.

(a) (2 points) Let \(l = k\). Define \(M_s(m) = (T_s(m_1), \ldots, T_s(m_n))\).

Verification checks that all blocks are correctly MAC-ed.

Solution: **************** INSERT PROBLEM 1a SOLUTION HERE ***********

(b) (2 points) Let \(l \approx k - 3 \log k\). Define \(M_s(m) = (T_s(m_1, 1), \ldots, T_s(m_n, n))\).

Namely, the block number \(i\) (which takes at most \(3 \log k\) bits) is explicitly MAC-ed together with the message block. Verification checks that all blocks are correctly MAC-ed and block numbers match.

Solution: **************** INSERT PROBLEM 1b SOLUTION HERE ***********

(c) (3 points) Let \(l \approx k - 6 \log k\). Define \(M_s(m) = (T_s(m_1, 1, n), \ldots, T_s(m_n, n, n))\).

Namely, both the number of blocks and the current block number are explicitly MAC-ed together with the message block. Verification checks that all blocks are correctly MAC-ed, and that the overall number of blocks is \(n\).

Solution: **************** INSERT PROBLEM 1c SOLUTION HERE ***********
(d) (3 points) Let \( l \approx k/2 \). Define \( M_s(m) = (r, T_s(m_1, r + 1), \ldots, T_s(m_n, r + n)) \), where \( r \in \{0, 1\}^{k/2} \) is random and addition is modulo \( 2^{k/2} \).

Namely, \( r \) is a nonce serving as the “message id”. Verification checks that all blocks are correctly MAC-ed, and that nonces \( (r + i) \) go in increasing order with no “gaps”.

**Solution:** ********** INSERT PROBLEM 1d SOLUTION HERE **********

\[ \square \]

(e) (4 points) Let \( l \approx k/2 \). Define \( M_s(m) = (r, T_s(m_1, r + 1, n), \ldots, T_s(m_n, r + n, n)) \), where \( r \in \{0, 1\}^{k/2} \) is random and addition is modulo \( 2^{k/2} \).

Namely, \( r \) is a nonce serving as the “message id”, and both \( r + i \) and \( n \) are MAC-ed with the corresponding message block. Verification checks that all blocks are correctly MAC-ed, that nonces \( (r + i) \) go in increasing order with no “gaps”, and that the number of blocks is \( n \).

**Solution:** ********** INSERT PROBLEM 1e SOLUTION HERE **********

\[ \square \]

**Problem 5-2 (CBC-MAC) 6 (+6) Points**

Recall that the CBC-MAC was defined by \( \text{Tag}_s(m) = f_s(m_n \oplus f_s(m_{n-1} \oplus \ldots f_s(m_1) \ldots)) \), and was proven to be a secure MAC (in fact, even a PRF) under the following 3 assumptions:

- The number of message blocks \( n \) was fixed.
- The initialization vector \( IV \) was fixed (in particular, we fixed it to \( 0^k \) in the equation above).
- The function family \( \{f_s : \{0, 1\}^k \rightarrow \{0, 1\}^k\} \) was a PRF family.

You will show that all three of these assumption are necessary, and dropping any one of them makes CBC-MAC an insecure MAC (and, thus, not a PRF as well).

(a) (3 points) Argue that CBC-MAC is insecure for authenticating variable length messages. Concretely, argue that after knowing \( \text{Tag}_s(m_1) \) there exists a block \( m_2 \) for which one can forge \( \text{Tag}_s(m_1 || m_2) \). What is \( m_2 \)?

**Solution:** ********** INSERT PROBLEM 2a SOLUTION HERE **********

\[ \square \]

(b) (3 points) Another option is to consider a probabilistic variant of CBC-MAC where one chooses a random \( IV \) and outputs probabilistic MAC

\[ \text{Tag}_s(m; IV) = (IV, f_s(m_n \oplus f_s(m_{n-1} \oplus \ldots f_s(m_1 \oplus IV) \ldots)) \]

Argue that this construction is insecure (even for fixed number of blocks \( n \); e.g., \( n = 1 \)). Specifically, for \( n = 1 \) show that after seeing any valid tag \( (IV, f_s(m_1 \oplus IV)) \) of any message \( m_1 \) one can right away compute a valid tag for any other \( m'_1 \neq m_1 \). What is this valid tag?

**Solution:** ********** INSERT PROBLEM 2b SOLUTION HERE **********

\[ \square \]
(c*) **Extra Credit** (6 points) Finally, we can keep the number of blocks $n$ and the $IV$ fixed (say, to $0^n$, as earlier), but assume that $\{f_s\}$ is only *unpredictable* rather than pseudorandom. Namely, for any PPT $A$,

$$\Pr(f_s(m) = t \text{ and } m \text{ is new } | \ s \leftarrow \{0,1\}^k, (m,t) \leftarrow A^{f_s()} = \negl(k)$$

Argue the resulting CBC-MAC is not *necessarily* secure\(^1\) already for $n = 2$ (clearly, it is secure for $n = 1$). Concretely, assume $\{f_s' : \{0,1\}^k \rightarrow \{0,1\}^{k/2}\}$ is some PRF family, and define $f_s(x,y) = f_s'(x,y) \circ x$, where $x,y \in \{0,1\}^{k/2}$ and $\circ$ denotes concatenation. First argue (this is the simple part) that $\{f_s\}$ is an unpredictable family if $\{f_s'\}$ is a PRF. Second, assume the attacker asks an arbitrary 2-block CBC-MAC query, where $x,y,z,w \in \{0,1\}^{k/2}$. Show that after learning the answer $a \circ b$ (where $a,b \in \{0,1\}^{k/2}$), the attacker can forge a valid MAC $(a' \circ b')$ of some other 2-block message $(x' \circ y', z' \circ w')$. What is this message? I.e., write $x', y', z', w', a', b'$ as functions of $x, y, z, w, a, b$ and argue why they work.

**Solution:** **************** INSERT PROBLEM 2b SOLUTION HERE ****************

---

**Problem 5-3** (Authenticated Encryption using Nonce) 14 Points

Recall the notion of authenticated encryption, which we briefly remind for convenience. It is a triple of algorithms $(G, E, D)$. $G$ is the key generation algorithm, i.e. $G(1^k)$ produces the shared secret key (usually, a truly random string of some length). As usual, $c \leftarrow E_s(m)$ produces the ciphertext, while $D_s(c) \rightarrow \hat{m} \in M \cup \{ot\}$, where $M$ is the message space (say, $M = \{0,1\}^k$), and $\bot$ denotes “invalid”. For privacy, we want $(G, E, D)$ to be an IND-secure encryption scheme against CPA (chosen plaintext attack). However, now we also want $(G, E, D)$ to be a secure message authentication scheme (strongly) existentially unforgeable against chosen message attack. Notice, here a successful forgery constitutes producing $c$ s.t. $D_s(c) \neq \bot$, and $c$ was never returned by the “tagging” (i.e., “encryption”)\(^2\) oracles.

You are to examine the “R-Nonce” scheme as a candidate for authenticated encryption. Namely, let $\{f_s : \{0,1\}^{2k} \rightarrow \{0,1\}^{2k}\}$ be a family of efficient strong PRP's.\(^3\) To encrypt $m \in \{0,1\}^k$, choose a random “nonce” $r \in \{0,1\}^k$, and return $C = \langle f_s(m \circ r), r \rangle$, where $s$ is the shared key (and $\circ$ denotes concatenation). To decrypt $C = \langle c, t \rangle$, compute $(m \circ r) = f_s^{-1}(c)$ and output $m$ provided $r = t$. Otherwise, output $\bot$.

(a) (5 points) Show that R-Nonce scheme is a secure encryption. Why is it important to have a random $r$ for each invocation?

---

\(^1\)Of course, it *could* be secure for some unpredictable $f_s$, like if $f_s$ is actually a PRF, but not secure in general, under the sole assumption that $f_s$ is unpredictable.

\(^2\)Notice, in this scenario the “tagging” and “encryption” oracles are the same.

\(^3\)As you will see, the strongness is needed in part (b).

---

PS5-3
(b) (5 points) Show that R-Nonce scheme is a secure message authentication scheme, implying that it is a secure authenticated encryption.

Solution: ****************** INSERT PROBLEM 3b SOLUTION HERE ******************

(c) (4 points) Assume we want to “optimize” the R-Nonce scheme and do not send the value \( r \) with the ciphertext. Namely, we simply let \( E_s(m) = f_s(m \circ r) \). Argue that the resulting scheme is no longer a secure authenticated encryption. Also, say which part fails: privacy, authenticity or both?

Solution: ****************** INSERT PROBLEM 3c SOLUTION HERE ******************

Problem 5-4 (Security of Universal-Hash-then-MAC) 15 Points

Recall that a PRF family \( \{f_s\} \) with \( k \)-bit inputs is also a MAC for \( k \)-bit messages. In order to MAC messages of length \( n > k \) we have shown that one can first hash the message using a universal hash function \( h_t : \{0,1\}^n \rightarrow \{0,1\}^k \) before applying \( f_s \): i.e., \( \text{Tag}_{s,t}(m) = f_s(h_t(m)) \). In this exercise we will show that this composition does not necessarily work if we use a general MAC \( \text{Tag}_s \) in place of the PRF \( f_s \).

(a) (4 points) Recall, one can view universal hash function \( h_t \) as a function where collisions are “hard to guess” without seeing the key \( t \) of \( h_t \). In this part, you will show that this property may no longer hold when the adversary is given oracle access to the function \( h_t \). Namely, if the adversary can see the function \( h_t(\cdot) \) evaluated on points of its choice, then he might be able to find “fresh collisions” \( (x, x') \), so that \( h_t(x) = h_t(x') \), \( x \neq x' \), and at least one of \( x \) or \( x' \) was not queried to the oracle.

To demonstrate this unfortunate attack, recall the Inner Product construction of universal hash function given in Lecture 11: let \( t = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n \) be the key and \( m = (m_1, \ldots, m_n) \in \mathbb{Z}_p^n \) be a message. The function is

\[
h_t(m) = \sum_{i=1}^n a_i \cdot m_i \mod p
\]

Show that an adversary \( A \) who is given oracle access to \( h_t(\cdot) \) can find a “fresh collision” (see above) with probability 1.

(Hint: There is a simple attack using \( n \) queries to \( h_t(\cdot) \). You can get an extra-credit if you show an attack using only 2 queries.)
Solution: ***************** INSERT PROBLEM 4a SOLUTION HERE ************

(b) (5 points) In this part you are to show that composing a MAC with a universal hash function is not always secure. In particular, let \((\text{Gen}', \text{Tag}', \text{Ver}')\) be a secure MAC on \(k\)-bit messages, and let \(\mathcal{H} = \{h_t(\cdot) : t \in \mathcal{K}\}\) be a family of universal hash functions that have the “bad property” you demonstrated in part (a) (i.e., an adversary can efficiently find a “fresh collision” if it is given oracle access to the function).

Construct a counterexample MAC \((\text{Gen}, \text{Tag}, \text{Ver})\) such that (1) it is secure on \(k\)-bit messages, but (2) it is no longer secure on \(n\)-bit messages when composed with a random hash function from the “bad” universal hash family \(\mathcal{H}\). Be sure to specify \((\text{Gen}, \text{Tag}, \text{Ver})\) from \((\text{Gen}', \text{Tag}', \text{Ver}')\) and describe an explicit attack for part (2).

(Hint: What happens if the algorithm \(\text{Tag}_a(m)\) outputs \(m\) in the clear?)

Solution: ***************** INSERT PROBLEM 4b SOLUTION HERE ************

(c) (6 points) Let us define a stronger notion of security for universal hash functions that we call “Oracle Universality”. Intuitively, this property says that an adversary who is given oracle access to the function \(h_t(\cdot)\) can find a collision \((x, x')\) only with negligible probability. More formally, a family of functions \(\mathcal{H} = \{h_t(\cdot) : t \in \mathcal{K}\}\) is “oracle universal” if:

\[
\Pr[\text{h}_t(x) = h_t(x') \land x \neq x' \mid t \leftarrow \mathcal{K}, (x, x') \leftarrow A^{h_t(\cdot)}(1^k)] \leq \text{negl}(k)
\]

Prove that composing a MAC with an oracle universal hash function \(h_t\) is secure.

Solution: ***************** INSERT PROBLEM 4c SOLUTION HERE ************

Problem 5-5 (“Trust but Verify” –Ronald Reagan) 14 Points

Recall the standard notion of security for MAC \(\text{MAC} = (\text{Gen}, \text{Tag}, \text{Ver})\), which is unforgeability against chosen message attacks (UF-CMA): for any PPT adversary \(A\),

\[
\Pr[\text{Ver}_s(m, t) = \text{accept} \mid s \leftarrow \text{Gen}(1^k), (m, t) \leftarrow A^{\text{Tag}_s(\cdot), \text{Ver}_s(\cdot)}(1^k)] \leq \text{negl}(k)
\]

where \(m\) has never been queried by \(A\) to the oracle \(\text{Tag}_s(\cdot)\).

We introduce the notion of strong UF-CMA (or sUF-CMA), which is the same as the above, except that we require the pair \((m, t)\) to be fresh; i.e., either \(m\) is “new”, or \(m\) is “old” but \(t\) must not have been obtained by querying the oracle \(\text{Tag}_a(m)\).

In this exercise we consider a variant of the unforgeability security notion, in which the adversary is not given access to the verification oracle \(\text{Ver}_s(\cdot, \cdot)\): for any PPT \(B\),

\[
\Pr[\text{Ver}_s(m, t) = \text{accept} \mid s \leftarrow \text{Gen}(1^k), (m, t) \leftarrow B^{\text{Tag}_s(\cdot)}(1^k)] \leq \text{negl}(k)
\]

Depending on whether we want regular or strong unforgeability, we call the resulting weaker notions UF-CMA-noV and sUF-CMA-noV, respectively (where “noV” stands for “no Verification”).

PS5-5
(a) (6 points) In this part, you are to show that for any MAC that is strongly unforgeable the verification oracle is not needed. Formally, sUF-CMA-noV security implies sUF-CMA (and, thus, UF-CMA) security. More precisely, you have to prove that if some PPT attacker \( A \) breaks sUF-CMA security (possibly using up to \( q \) verification queries), then some PPT attacker \( B \) breaks sUF-CMA-noV security (with no verification queries) with probability \( \epsilon' \geq \epsilon/q \). Be sure to precisely define \( B \) in terms of \( A \).

**Solution:** ***************** INSERT PROBLEM 5a SOLUTION HERE *****************

(b) (2 points) Show that any deterministic UF-CMA-noV MAC can be easily turned into a strongly unforgeable sUF-CMA-noV MAC (which, using part (a), must already be sUF-CMA secure). Hence, for deterministic MACs there is almost no difference between having verification queries or not.

(Hint: You only need to define a good verification algorithm.)

**Solution:** ***************** INSERT PROBLEM 5b SOLUTION HERE *****************

(c) (6 points) In this part we will show that, in general, the notion of UF-CMA-noV security does not imply UF-CMA security, even for deterministic MACs (where, of course, one does not first apply the simple transformation from part (b)).

You will show this by constructing a counterexample of a MAC, where it is hard to forge tags of new messages without the verification oracle,\(^4\) but where forging tags of any message becomes very easy with (cleverly chosen) verification queries.

Starting with any UF-CMA-secure MAC' = (Gen', Tag', Ver') whose secret key \( s = s_1 \ldots s_k \) has length \( k \), we construct the counter-example MAC = (Gen, Tag, Ver) as follows. The key generation is the same Gen = Gen', and the tagging is defined as Tag\(_s\)(\( m \)) = (Tag\(_{s'}\)(\( m \)), 0) (i.e., append a number zero to the old tag).

Complete the description of the “special” verification algorithm Ver below (by filling the dots) in such a way that the resulting MAC is:

- UF-CMA-noV-secure (hard to forge tags of new messages without the verification oracle).
- NOT UF-CMA-secure in a very strong sense: an adversary, who gets access to the verification oracle, can easily forge the tag of any message after a fixed number of verification queries.

In the description below, we use 0 for “reject” and 1 for “accept”. Also remember that you can only return 0/”reject” or 1/”accept” for each tag \( t \) (e.g., cannot return a \( k \)-bit string, for example).

\(^4\)Of course, by part (a) we know that your counter-example cannot be strongly unforgeable, so new valid tags of old messages are easy to forge.
\textbf{Ver}_s(m, t) :
  
  Parse \( t \) as \( (t', i) \), where \( i \geq 0 \).
  
  If \( \text{Ver}_s'(m, t') = 0 \) ("reject") \textbf{Then Return} . . .
  
  Else \textbf{If} \( i \notin \{1, \ldots, k\} \) \textbf{Then Return} . . .
  
  Else \textbf{Return} . . .

(Hint: What sensitive 1-bit information can you return in the last line that would eventually help to completely break the scheme?)

\textbf{Solution:} ****************** INSERT PROBLEM 5c SOLUTION HERE ***************

PS5-7