! **Try again once you are ready**

TO PASS 80% or higher

Try again

GRADE
40%

Week 3 - Problem Set

LATEST SUBMISSION GRADE

40%

1. **Suppose a MAC system (S, V) is used to protect files in a file system by appending a MAC tag to each file. The MAC signing algorithm S is applied to the file contents and nothing else. What tampering attacks are not prevented by this system?**

0 / 1 point

- ○ Changing the name of a file.
- ○ Changing the first byte of the file contents.
- ○ Appending data to a file.
- ◉ Replacing the contents of a file with the concatenation of two files on the file system.

> ! **Incorrect**
> The MAC tag will fail to verify if any file data is changed.

2. **Let (S, V) be a secure MAC defined over (K, M, T) where $M = \{0,1\}^n$ and $T = \{0,1\}^{128}$. That is, the key space is K, message space is $\{0,1\}^n$, and tag space is $\{0,1\}^{128}$.**

0 / 1 point

Which of the following is a secure MAC: (as usual, we use $\|$ to denote string concatenation)

- ☐ $S'(k, m) = S(k, m)$ and
  $V'(k, m, t) = [V(k,\ m,\ t) \text{ or } V(k,\ m \oplus 1^n,\ t)]$
  (i.e., $V'(k, m, t)$ outputs ``1" if t is a valid tag for either m or $m \oplus 1^n$)
- ☑ $S'(k, m) = S(k, m \oplus m)$ and
  $V'(k, m, t) = V(k,\ m \oplus m,\ t)$

> ! **This should not be selected**
> This construction is insecure because an adversary can request the tag for $m = 0^n$ and thereby obtain a tag for any message. This follows from the fact that $m \oplus m = 0$.

- ☑ $S'(k, m) = S(k, m \oplus 1^n)$ and
  $V'(k, m, t) = V(k, m \oplus 1^n, t)$

> ✓ **Correct**
> a forger for (S', V') gives a forger for (S, V).

- ☐ $S'(k,\ m) = [t \leftarrow S(k, m),\ \text{output } (t, t)\,)$ and
  $V'(k, m, (t_1, t_2)) = \begin{cases} V(k, m, t_1) & \text{if } t_1 = t_2 \\ \text{"0"} & \text{otherwise} \end{cases}$
  (i.e., $V'(k, m, (t_1, t_2))$ only outputs "1" if t_1 and t_2 are equal and valid)
- ☐ $S'(k, m) = \begin{cases} S(k, 1^n) & \text{if } m = 0^n \\ S(k, m) & \text{otherwise} \end{cases}$ and
  $V'(k, m) = \begin{cases} V(k, 1^n, t) & \text{if } m = 0^n \\ V(k, m, t) & \text{otherwise} \end{cases}$
- ☑ $S'((k_1, k_2),\ m) = (S(k_1, m), S(k_2, m))$ and
  $V'((k_1, k_2), m, (t_1, t_2)) = [V(k_1, m, t_1) \text{ and } V(k_2, m, t_2)]$
  (i.e., $V'((k_1, k_2), m, (t_1, t_2))$ outputs ``1" if both t_1 and t_2 are valid tags)

> ✓ **Correct**
> a forger for (S', V') gives a forger for (S, V).

3. **Recall that the ECBC-MAC uses a fixed IV (in the lecture we simply set the IV to 0). Suppose instead we chose a random IV for every message being signed and include the IV in the tag. In other words, $S(k, m) := (r,\ \text{ECBC}_r(k, m))$ where $\text{ECBC}_r(k, m)$ refers to the ECBC function using r as the IV. The verification algorithm V given key k, message m, and tag (r, t) outputs ``1" if $t = \text{ECBC}_r(k, m)$ and outputs ``0" otherwise. The resulting MAC system is insecure. An attacker can query for the tag of the 1-block message m and obtain the tag (r, t). He can then generate the following existential forgery: (we assume that the underlying block cipher operates on n-bit blocks)**

0 / 1 point

- ○ The tag $(r \oplus 1^n,\ t)$ is a valid tag for the 1-block message $m \oplus 1^n$.
- ○ The tag $(r,\ t \oplus r)$ is a valid tag for the 1-block message 0^n.
- ○ The tag $(r \oplus t,\ m)$ is a valid tag for the 1-block message 0^n.
- ◉ The tag $(m \oplus t,\ r)$ is a valid tag for the 1-block message 0^n.

> ! **Incorrect**
> The right half of the tag, r, is not likely to be the result of the CBC MAC.

4. **Suppose Alice is broadcasting packets to 6 recipients $B_1, \ldots, B_6$. Privacy is not important but integrity is. In other words, each of $B_1, \ldots, B_6$ should be assured that the packets he is receiving were sent by Alice. Alice decides to use a MAC. Suppose Alice and $B_1, \ldots, B_6$ all share a secret key k. Alice computes a tag for every packet she sends using key k. Each user B_i verifies the tag when receiving the packet and drops the packet if the tag is invalid. Alice notices that this scheme is insecure because user B_1 can use the key k to send packets with a valid tag to users $B_2, \ldots, B_6$ and they will all be fooled into thinking that these packets are from Alice. Instead, Alice sets up a set of 4 secret keys $S = \{k_1, \ldots, k_4\}$. She gives each user B_i some subset $S_i \subseteq S$ of the keys. When Alice transmits a packet she appends 4 tags to it by computing the tag with each of her 4 keys. When user B_i receives a packet he accepts it as valid only if all tags corresponding to his keys in S_i are valid. For example, if user B_1 is given keys $\{k_1, k_2\}$ he will accept an incoming packet only if the first and second tags are valid. Note that B_1 cannot validate the 3rd and 4th tags because he does not have k_3 or k_4. How should Alice assign keys to the 6 users so that no single user can forge packets on behalf of Alice and fool some other user?**

1 / 1 point

- ☐ $S_1 = \{k_1, k_2\},\ S_2 = \{k_1, k_3, k_4\},\ S_3 = \{k_1, k_4\},\ S_4 = \{k_2, k_3\},\ S_5 = \{k_2, k_3, k_4\},\ S_6 = \{k_3, k_4\}$
- ☐ $S_1 = \{k_1\},\ S_2 = \{k_2, k_3\},\ S_3 = \{k_3, k_4\},\ S_4 = \{k_1, k_3\},\ S_5 = \{k_1, k_2\},\ S_6 = \{k_1, k_4\}$
- ☐ $S_1 = \{k_1, k_2\},\ S_2 = \{k_1, k_3\},\ S_3 = \{k_1, k_4\},\ S_4 = \{k_2, k_3, k_4\},\ S_5 = \{k_2, k_3\},\ S_6 = \{k_3, k_4\}$
- ☑ $S_1 = \{k_2, k_3\},\ S_2 = \{k_2, k_4\},\ S_3 = \{k_3, k_4\},\ S_4 = \{k_1, k_2\},\ S_5 = \{k_1, k_3\},\ S_6 = \{k_1, k_4\}$

> ✓ **Correct**
> Every user can only generate tags with the two keys he has. Since no set S_i is contained in another set S_j, no user i can fool a user j into accepting a message sent by i.

5. **Consider the encrypted CBC MAC built from AES. Suppose we compute the tag for a long message m comprising of n AES blocks. Let m' be the n-block message obtained from m by flipping the last bit of m (i.e. if the last bit of m is b then the last bit of m' is $b \oplus 1$). How many calls to AES would it take to compute the tag for m' from the tag for m and the MAC key? (in this question please ignore message padding and simply assume that the message length is always a multiple of the AES block size)**

1 / 1 point

- ○ 6
- ○ 5
- ◉ 4
- ○ n

> ✓ **Correct**
> You would decrypt the final CBC MAC encryption step done using k_2, the decrypt the last CBC MAC encryption step done using k_1, flip the last bit of the result, and re-apply the two encryptions.

6. **Let $H: M \to T$ be a collision resistant hash function. Which of the following is collision resistant: (as usual, we use $\|$ to denote string concatenation)**

0 / 1 point

- ☑ $H'(m) = H(m) \bigoplus H(m \oplus 1^{|m|})$
  (where $m \oplus 1^{|m|}$ is the complement of m)

> ! **This should not be selected**
> This construction is not collision resistant because $H(000) = H(111)$.

- ☑ $H'(m) = H(m \| m)$

> ✓ **Correct**
> a collision finder for H' gives a collision finder for H.

- ☐ $H'(m) = H(0)$
- ☐ $H'(m) = H(m)[0, \ldots, 31]$
  (i.e. output the first 32 bits of the hash)
- ☑ $H'(m) = H(m) \| H(m)$

> ✓ **Correct**
> a collision finder for H' gives a collision finder for H.

- ☐ $H'(m) = H(m[0, \ldots, |m| - 2])$
  (i.e. hash m without its last bit)
- ☐ $H'(m) = H(m \| 0)$

7. **Suppose H_1 and H_2 are collision resistant hash functions mapping inputs in a set M to $\{0,1\}^{256}$. Our goal is to show that the function $H_2(H_1(m))$ is also collision resistant. We prove the contra-positive: suppose $H_2(H_1(\cdot))$ is not collision resistant, that is, we are given $x \neq y$ such that $H_2(H_1(x)) = H_2(H_1(y))$. We build a collision for either H_1 or for H_2. This will prove that if H_1 and H_2 are collision resistant then so is $H_2(H_1(\cdot))$. Which of the following must be true:**

1 / 1 point

- ○ Either x, y are a collision for H_2 or $H_1(x), H_1(y)$ are a collision for H_1.
- ○ Either $x, H_1(y)$ are a collision for H_2 or $H_2(x), y$ are a collision for H_1.
- ◉ Either x, y are a collision for H_1 or $H_1(x), H_1(y)$ are a collision for H_2.
- ○ Either $H_2(x), H_2(y)$ are a collision for H_1 or x, y are a collision for H_2.

> ✓ **Correct**
> If $H_2(H_1(x)) = H_2(H_1(y))$ then either $H_1(x) = H_1(y)$ and $x \neq y$, thereby giving us a collision on H_1. Or $H_1(x) \neq H_1(y)$ but $H_2(H_1(x)) = H_2(H_1(y))$ giving us a collision on H_2. Either way we obtain a collision on H_1 or H_2 as required.

8. **In this question you are asked to find a collision for the compression function: $f_1(x, y) = \text{AES}(y, x) \bigoplus y$ where $\text{AES}(x, y)$ is the AES-128 encryption of y under key x. Your goal is to find two distinct pairs (x_1, y_1) and (x_2, y_2) such that $f_1(x_1, y_1) = f_1(x_2, y_2)$. Which of the following methods finds the required (x_1, y_1) and (x_2, y_2)?**

0 / 1 point

- ○ Choose x_1, y_1, y_2 arbitrarily (with $y_1 \neq y_2$) and let $v := AES(y_1, x_1)$. Set $x_2 = AES^{-1}(y_2,\ v \oplus y_2)$
- ○ Choose x_1, y_1, y_2 arbitrarily (with $y_1 \neq y_2$) and let $v := AES(y_1, x_1)$. Set $x_2 = AES^{-1}(y_2,\ v \oplus y_1 \oplus y_2)$
- ○ Choose x_1, y_1, x_2 arbitrarily (with $x_1 \neq x_2$) and let $v := AES(y_1, x_1)$. Set $y_2 = AES^{-1}(x_2,\ v \oplus y_1 \oplus x_2)$
- ○ Choose x_1, y_1, y_2 arbitrarily (with $y_1 \neq y_2$) and let $v := AES(y_1, x_1)$. Set $x_2 = AES^{-1}(y_2,\ v \oplus y_1)$

> ! **Incorrect**
> You did not choose an option.

9. **Repeat the previous question, but now to find a collision for the compression function $f_2(x, y) = \text{AES}(x, x) \bigoplus y$. Which of the following methods finds the required (x_1, y_1) and (x_2, y_2)?**

0 / 1 point

- ◉ Choose x_1, x_2, y_1 arbitrarily (with $x_1 \neq x_2$) and set $y_2 = y_1 \oplus x_1 \oplus AES(x_2, x_2)$
- ○ Choose x_1, x_2, y_1 arbitrarily (with $x_1 \neq x_2$) and set $y_2 = AES(x_1, x_1) \oplus AES(x_2, x_2)$
- ○ Choose x_1, x_2, y_1 arbitrarily (with $x_1 \neq x_2$) and set $y_2 = y_1 \oplus AES(x_1, x_1)$
- ○ Choose x_1, x_2, y_1 arbitrarily (with $x_1 \neq x_2$) and set $y_2 = y_1 \oplus AES(x_1, x_1) \oplus AES(x_2, x_2)$

> ! **Incorrect**
> This does not work

10. **Let $H: M \to T$ be a random hash function where $|M| \gg |T|$ (i.e. the size of M is much larger than the size of T). In lecture we showed that finding a collision on H can be done with $O(|T|^{1/2})$ random samples of H. How many random samples would it take until we obtain a three way collision, namely distinct strings x, y, z in M such that $H(x) = H(y) = H(z)$?**

1 / 1 point

- ◉ $O(|T|^{2/3})$
- ○ $O(|T|^{1/2})$
- ○ $O(|T|)$
- ○ $O(|T|^{1/3})$

> ✓ **Correct**
> An informal argument for this is as follows: suppose we collect n random samples. The number of triples among the n samples is n choose 3 which is $O(n^3)$. For a particular triple x, y, z to be a 3-way collision we need $H(x) = H(y)$ and $H(x) = H(z)$. Since each one of these two events happens with probability $1/|T|$ (assuming H behaves like a random function) the probability that a particular triple is a 3-way collision is $O(1/|T|^2)$. Using the union bound, the probability that some triple is a 3-way collision is $O(n^3/|T|^2)$ and since we want this probability to be close to 1, the bound on n follows.