Problem 1-1. Message authentication codes Recall that in class we mentioned the hash-then-MAC paradigm, which uses a MAC for messages of fixed length n (such as AES, in which case $n = 128$), and a hash function with range $\{0,1\}^n$, to MAC messages of arbitrary length, by first hashing the message to an element in $\{0,1\}^n$ and then applying the underlying MAC to the hash value.

In class we mentioned that for the resulting MAC to be secure the hash function must be collision resistant, and thus this paradigm cannot be used with AES since its domain consists of messages of length 128, and there does not exist a collision resistant hash function with 128-bit security that maps to tags of bitlength 128, due to the Birthday Paradox.

In this problem we consider using the hash-then-MAC paradigm with AES but with a hash function with a *secret* seed. In particular, consider the following hash function H: It takes as input a (secret) seed $(a_1, \ldots, a_t)$ and a message $(M_1, \ldots, M_t)$, where each a_i and M_i is a block of 128 bits. The hash function outputs

$$H((a_1, \ldots, a_t), (M_1, \ldots, M_t)) = \sum_{i=1}^{t} a_i M_i.$$

The arithmetic is done over a finite field of size 2^{128} (known as the Galois Field and denoted by $\mathsf{GF}[2^{128}]$), though how the arithmetic is done is not important, the only important thing is that the output is in $\{0,1\}^{128}$, and that in every (finite) field (and in particular in $\mathsf{GF}[2^{128}]$) every non-zero element has an multiplicative inverse.

(a) Define the MAC scheme obtained by applying the hash-then-MAC paradigm to the hash function $H : (\{0,1\}^{128})^t \times (\{0,1\}^{128})^t \to \{0,1\}^{128}$ and AES as the underlying MAC.

Solution:

$$\mathsf{MAC}'((K, (a_1, \ldots, a_t)), (M_1, \ldots, M_t)) = \mathsf{MAC}(K, H((a_1, \ldots, a_t), (M_1, \ldots, M_t)))$$

(b) 1. Recall the attack for the hash-then-MAC using AES and a hash function without a secret seed (assuming the message space is $(\{0,1\}^{128})^t$ for $t \geq 2$). *Hint:* This attack takes time roughly 2^{64}.

Solution: The attack is to generate random messages $M_1, \ldots, M_T \in (\{0,1\}^{128})^t$ until there exists $i \in [T-1]$ such that $H(M_T) = H(M_i)$. Note that H can be computed by the adversary since it does not have a secret seed! By the birthday paradox $T \approx 2^{64}$, and since the messages were chosen at random from a very large message space ($t \geq 2$) then with overwhelming probability $M_T \neq M_i$, in which case the adversary can query the signing oracle for a signature of M_i, and output this signature as a valid signature for M_T.

2. Explain why this attack fails if we use the hash function above (with a secret seed), assuming the attacker sees many fewer than 2^{64} tags. **Solution:** When the hash function has a secret seed the adversary does not know if $H(M_T) = H(M_i)$, and hence cannot execute the above attack.

3. Is this hash-then-MAC scheme (with H as defined above and AES) secure if the attacker gets to see more than 2^{64} tags for messages of his choice? *Hint:* Use the fact that AES is injective (i.e., if $\mathsf{AES}(K, M) = \mathsf{AES}(K, M')$ then $M = M'$), and the fact that H is a linear function, together with the fact that one can efficiently solve t linear equations with t variables (using Gaussian elimination).

   Solution: No! An adversary can query the signing oracle to sign many messages until he finds t collisions; namely, until he finds t message pairs $\{M_i, M_i'\}_{i\in[t]}$ such that $M_i \neq M_i'$ and

$$\mathsf{AES}(K, H((a_1, \ldots, a_t), M_i)) = \mathsf{AES}(K, H((a_1, \ldots, a_t), M_i')).$$

   This implies that $H((a_1, \ldots, a_t), M_i) = H((a_1, \ldots, a_t), M_i')$, which reveals one linear equation about the secret $a_1, \ldots, a_t$. Repeating this t times we get t linear equations about $a_1, \ldots, a_t$, and these linear equations are likely to be independent. As a result the adversary can find the secret seed $a, \ldots, a_t$ using Gaussian elimination. Once the adversary finds the seed he can break the scheme as above.
   Here's a cleverer break that only requires one collision due to Adam Zheng from Fall 2021's 6.S060.
   Suppose that the CMA adversary obtained 2^{64} tags, and it found a collision. As noted in the above solution, suppose we have found two messages $M_1 \neq M_2$ which cause a collision, i.e.,

$$AES(k, H(a, M_1)) = AES(k, H(a, M_2)) \implies H(a, M_1) = H(a, M_2)$$

   This implies $a * M_1 = a * M_2$ (where $*$ is the dot product), and thus $a * (M_1 - M_2) = 0, mod 2^{128}$.
   Now have the CMA adversary pick any message M_3 (not already submitted to the oracle) which has tag $AES(k, H(a, M_3))$. Then, $M_3 + (M_1 - M_2)$ also has that tag, since $H(a, M_3 + (M_1 - M_2)) = a * (M_3 + (M_1 - M_2))$ $= a * M_3 + a * (M_1 - M_2) = a * M_3 = H(a, M_3)$ by linearity of H. Since $M_1 \neq M_2$, $M_3 + (M_1 - M_2) \neq M_3$.

Problem 1-2. Weaknesses of CPA-secure cryptosystems

Let $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a secure pseudorandom function. In class, we saw the CPA-secure encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with keyspace $\mathcal{K}$, where

$$\mathsf{Enc}(k,m) := \begin{cases} r \xleftarrow{R} \{0,1\}^n \\ c \leftarrow F(k,r) \oplus m \\ \text{output } (r,c) \end{cases} \qquad \mathsf{Dec}(k,(r,c)) := \begin{cases} m \leftarrow F(k,r) \oplus c \\ \text{output } m \end{cases} .$$

We will use a secure MAC scheme $\mathsf{MAC}\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ over the same keyspace $\mathcal{K}$.

MIT uses encryption to protect communications between department offices and its central payroll system. In particular, the EECS department office and MIT's payroll office share a secret encryption key $k_{\mathsf{Enc}} \xleftarrow{R} \mathcal{K}$ and a secret MAC key $k_{\mathsf{MAC}} \xleftarrow{R} \mathcal{K}$.

On each pay date, a machine in the EECS department sends a sequence of messages to the payroll office. Each plaintext message is a 54-byte ASCII-encoded string with format:

$$m = \text{“}\texttt{date:NNNN-NN-NN, mit\_id:NNNNNNNNN, amount:NNNNNNNNN.NN}\text{”},$$

where each `N` represents an ASCII-encoded decimal digit (i.e., ASCII values `0x30`–`0x39`).

Unfortunately, the designers of the system make the wrong decision and use MAC-then-encrypt rather than encrypt-then-MAC. In particular, the EECS department encrypts each message m as:

$$\mathsf{ct} = \mathsf{Enc}(k_{\mathsf{Enc}}, m \,\|\, \mathsf{MAC}(k_{\mathsf{MAC}}, m)).$$

The machine at the payroll office operates on each ciphertext ct as follows:

- Compute $(m' \| t') \leftarrow \mathsf{Dec}(k_{\mathsf{Enc}}, \mathsf{ct})$.
- Check that the bytes of the string corresponding to the MIT ID number and the payment amount are valid ASCII decimal digits. If not, send a `FORMAT ERROR` message to the EECS machine.
- Check that the date corresponds to today's date. If not, send a `DATE ERROR` message to the EECS machine.
- Check that $t' = \mathsf{MAC}(k_{\mathsf{MAC}}, m')$. If not, send a `MAC ERROR` message to the EECS machine.
- Append the $(\texttt{ID}, \texttt{amount})$ pair to a log file for later processing and send an `OK` message to the EECS machine.

Explain how a network attacker that can intercept ciphertexts and interact with the payroll server can recover **all but one of the bits in each secret digit** in the plaintexts that the EECS department sends. Recovering these bits for a single plaintext using your attack should take no more than than 10,000 interactions with the payroll server.

Solution: The attacker intercepts a ciphertext ct and recovers bits from the plaintext one unknown decimal digit at a time. (The attacker knows the date, since it is just the current date.) If we view each decimal digit as a 4-bit number, the attacker will be able to recover all but the least significant bits of the number.

For a number $i \in \{0, \ldots, 9\}$, let $\texttt{ASCII}(i)$ be the ASCII encoding of i as a byte. For a byte b $\texttt{IS\_ASCII}(b)$ be the function that returns "1" if the byte is a valid ASCII-encoded decimal digit and that returns "0" otherwise. For a number $i \in \{0, \ldots, 9\}$, define the *fingerprint* of i to be the 16 bits:

$$\big(\texttt{IS\_ASCII}(\texttt{ASCII}(i) \oplus \texttt{0x00}),\ \ldots,\ \texttt{IS\_ASCII}(\texttt{ASCII}(i) \oplus \texttt{0x0F})\big).$$

You can convince yourself (e.g., with a short Python program) that the only numbers in $\{0, \ldots, 9\}$ that have the same fingerprint are ones that are the same in their most-significant 3 bits. So 4 and 5 have the same fingerprint, but 4 and 7 do not. So, knowing the fingerprint of a number in $\{0, \ldots, 9\}$ is enough to determine the number's most significant three bits.

Next, observe that the attacker in our setting can compute the fingerprint of any of the unknown digits in the plaintext. To compute the fingerprint of the integer at the bth position in the plaintext, the attacker runs:

- For $\sigma \in \{\texttt{0x00}, \ldots, \texttt{0x0F}\}$:
  - Set ct′ $\leftarrow$ ct and then ct′$[b] \leftarrow$ ct$[b] \oplus \sigma$.
  - Send ct′ to the payroll server.
  - If the server replies `OK` or `MAC ERROR`, set $f_\sigma \leftarrow 1$, otherwise set $f_\sigma \leftarrow 0$.
- Return $(f_{\texttt{0x00}}, \ldots, f_{\texttt{0x0F}})$ as the fingerprint of the bth digit.

Finally, the attacker can put these ideas to recover the most-significant bits of each unknown message digit. For each unknown decimal digit in the plaintext, the attacker computes the fingerprint. The fingerprint then immediately gives the attacker the three most-significant bits of the unknown digit.

To compute the fingerprint, the attacker generates at most 16 ciphertexts per digit and there 20 unknown digits in the plaintext. So the attack requires at most 320 interactions with the payroll server. With a more careful fingerprinting technique, you can reduce the attack cost further.

Problem 1-3. Encryption and RSA

(a) Which of the following security goal(s) does encryption address :
(1) Confidentiality (2) Integrity (3) Sender authentication (4) Non-repudiation.

Solution: Encryption aims to provide confidentiality only. Non-repudiation is the concept of ensuring that a party in a dispute cannot repudiate, or refute the validity of a statement or contract.

(b) Suppose you obtain two ciphertexts C,C' encrypted using one-time pad, with key K and its bitwise complement $\bar{K} = K \oplus 1$ respectively. What can you infer about the corresponding plaintext messages?

Solution: You can obtain $m \oplus m$. Note that $K = K1$, therefore, $C \oplus C \oplus 1 = (m \oplus K) \oplus (m \oplus K \oplus 1) \oplus (1) = m \oplus m$.

(c) Alice and Bob are using public keys $(e_1, N_1), (e_2, N_2)$ respectively. Suppose you are informed that their RSA moduli $N1, N2$ are not relatively prime. How would you break the security of their subsequent communication? It is sufficient to show that you can get $\phi(N_1)$ and $\phi(N_2)$.
Hint: Euclid's algorithm allows us to compute GCD(x,y) efficiently.

Solution: Since N_1, N_2 share a common divisor (they are not relatively prime), let p be the shared prime divisor. Let $N_1 = p{\cdot}q$ and $N_2 = p{\cdot}r$. $p = GCD(N_1, N_2)$, which can be computed efficiently using Euclid's Algorithm. We do not expect the solution to outline the algorithm. $q = N_1/p$ and$r = N_2/p$ can be computed with the knowledge of p. $\phi(N_1) = (p-1)(q-1)$ and $\phi(N_2) = (p-1)(r-1)$ can be computed with the knowledge of p, q, r.

(d) Suppose that a system uses textbook RSA encryption. An attacker wants to decrypt a ciphertext c to obtain the corresponding confidential plaintext m. Assume that the victim system readily decrypts arbitrary ciphertexts that the attacker can choose, except for ciphertext c itself. Show that the attacker can obtain m from c even under this setting, i.e RSA is not CCA secure.

Solution: We know that $c = m^e \mod N$,i.e, the attacker aims to obtain m from c. To do this, the attacker uses the following scheme.

- Step 1. The attacker chooses a random number $'r'$ such that $gcd(r, N) = 1$. The attacker encrypts it as using the public key : $C_r = r^e \mod N$.

- Step 2.
  He computes $C = c \cdot C^r \mod N$, and asks the system to decrypt C
  Let the decryption be M.
  By definition of RSA encryption, we know that: $M' = (C')^d \mod N$.
  It follows that: $C' = ((m^e \mod N) \cdot (r^e \mod N)) \mod N. = (m \cdot r)^e \mod N$
  Therefore, $M' = C'^d \mod N, M' = (m \cdot r)^{ed} \mod N$
  This follows from Fermat's little theorem and the construction of e, d.
  $M = (m \cdot r) \mod N$
- Step 3. The attacker obtains M and recovers the confidential plaintext m by computing $M \cdot r^{-1} \mod N$.
  $M \cdot r^1 \mod N = m \cdot r \cdot r^{-1} \mod N = m \mod N$.
- Keep in mind that r has to be chosen such that its multiplicative inverse modulo N exists. This is true iff $gcd(r, N) = 1$.