Message Authentication Codes

1. Possible reasons are:
   a) The message was modified (error or changed by an adversary).
   b) The MAC tag is faulty (error or changed by an adversary).
   c) A computational error occurred during MAC generation or verification.
   d) The wrong key was used; the keys for MAC generation and verification do not match.

2. Both can be used to protect the integrity of messages against random events and errors. However, an adversary might change a message *and* the hash. In the presence of active adversaries, the authentic hash value of the original message is required to verify the integrity. In contrast, authenticity of a MAC is not a precondition, since an adversary should not be able to forge a valid MAC without the secret key.

3. A tag t is valid for the message m if $m \oplus t = k$. Suppose that k is a fresh uniform random key of length n. Then the probability that an adversary forges a valid MAC is only $\frac{1}{2^n}$ and hence negligible. Thus this MAC is secure, similar to a One-Time-Pad. However, this MAC is impractical, since the MAC and the key have the same length as the message. Furthermore, each key can only be used for one message.

4. An adversary chooses any message m and asks the oracle for the corresponding tag t. Then the adversary defines m' which differs from m only in the first bit. The tag t is also valid for m'. This message was not queried before and the adversary wins the MAC forgery experiment.

5. (See reference [KL15]). Let $l' = \lfloor \frac{l}{4} \rfloor$. Pad the message with zeros and split it into blocks $m_1, \ldots, m_N$ of length l'. For the i-th block, we compute the MAC of $r \| L \| i \| m_i$, padded by zeros if $4l' < l$. r is a uniform identifier that prevents the mixing of different messages. Including the block index number i and the encoded message length L protects against re-ordering and truncation attacks. The length of r, L and i is l'. Then we define the MAC of m as follows:

$$t_i = \mathrm{MAC}_k(r \| L \| i \| m_i) \text{ for } i = 1, \ldots, N \text{ and } \mathrm{MAC}_k(m) = (r \| t_1 \| \ldots \| t_N).$$

6. Suppose the tag is defined by appending the encoded message length and computing the CBC MAC. Let l be the block length and let L be the encoding of l. Then the tags of 0^l and 1^l are $t_0 = E_k(L \oplus E_k(0^l))$ and $t_1 = E_k(L \oplus E_k(1^l))$, respectively. Define messages $m_0 = 0^l \,\|\, L \| \, t_0$ and $m_1 = 1^l \,\|\, L \| \, t_1$ of length $3l$. One checks that the tag of both m_0 and m_1 is $E_k(L' \oplus E_k(0^l))$, where L' is the encoding of $3l$. An adversary asks for the tags of 1^l and m_0 and can give a valid tag of m_1.

7. This tag would always be valid: let $c_1, \ldots, c_{N-1}, c_N$ be the ciphertext blocks and assume the last block c_N is also used as CBC MAC. Decryption in CBC mode gives the plaintext blocks $m_1, \ldots, m_N$. The last block is $m_N = E_k^{-1}(c_N) \oplus c_{N-1}$. In order to verify the tag, one would encrypt the plaintext and always obtain c_N, even if the ciphertext has been tampered with:

$$E_k(m_N \oplus c_{N-1}) = E_k(E_k^{-1}(c_N) \oplus c_{N-1} \oplus c_{N-1}) = c_N$$

8. The construction is susceptible to a length-extension attack (see Exercise 5 of the preceding chapter): an adversary can compute $H(k\|m\|m')$ without knowing k. HMAC and NMAC are not affected by this attack, since an adversary only controls the output of the inner hash function, but not the outer hash function. The outer hash function takes a derived message of fixed length as input.

9.
```
sage: F.<t>=GF(2)[]; f = t^128 + t^7 + t^2 + t + 1
sage: f.is_irreducible
True
```

10. (a) $b \cdot (0 \dots 0) = (0 \dots 0)$, $b \cdot (10 \dots 0) = b$.
    (b) We have $b \cdot e_1 = b$ (see (a)). If $2 \leq i \leq 128$, then $b \cdot e_i$ can be computed by an $(i-1)$-fold multiplication of b by $(010 \dots 0)$. Each multiplication is given by $b \gg 1$, i.e., by right-shifting b; if the rightmost bit of b is 1, then the constant $R = 111000010 \dots 0$ has to be XORed.
    (c) Let $c = \lambda_1 e_1 \oplus \dots \oplus \lambda_{128} e_{128}$, where $\lambda_i \in \{0, 1\}$. Then
    $$b \cdot c = \lambda_1 (b \cdot e_1) \oplus \dots \oplus \lambda_{128} (b \cdot e_{128}),$$
    where the computation of $b \cdot e_i$ for $i = 1, \dots, 128$ is explained in (b).

11. Choose a uniform random IV of length 96 and set $ctr = IV\|0^{31}\|1$. Then $m = m_1$, $c_1 = E_k(ctr+1) \oplus m_1$, $c = IV\|c_1$, $H = E_k(0^{128})$, $A = 0^{128}$, $X_1 = 0^{128}$, $X_2 = c_1 \cdot H$, $X_3 = (X_2 \oplus (0^{64}\|0^{56}10000000)) \cdot H$. The tag is $t = X_3 \oplus E_k(ctr)$ and the authenticated ciphertext is (c, t).

12. An adversary can easily attack a block cipher in counter mode in a chosen ciphertext attack: let c be a challenge ciphertext in a chosen ciphertext experiment. The adversary changes one ciphertext bit and sends the modified ciphertext c' to the oracle, who returns the corresponding plaintext m'. The adversary flips the same bit as before and in this way obtains the plaintext m.

    This attack is not possible when the ciphertext is combined with a secure MAC (encrypt-then-authenticate approach), since the adversary cannot produce a valid tag of c'. Although the ciphertexts c and c' (and the corresponding plaintexts) differ only in one bit, their tags are not related. An adversary knows the tag of c (as part of the challenge ciphertext), but is unable to produce a valid tag of c' without the secret key. Any chosen ciphertext is most likely to be invalid, so that the oracle only returns the error symbol $\perp$ and not the corresponding plaintext. In other words, appending a secure MAC prevents an adversary from leveraging the decryption oracle.