Amjad Alqahtani
Assignment 3 answers

Question 1 Answer:

We have given a secure MAC Π = (Gen, Mac, Vrfy). It works only on fixed message lengths with **n bits**. The goal is to define a MAC that works on arbitrary-length messages. Based on construction 4.7, we need to compute the size of the final tag t. We have to find the length in bits of the tag, as explained in 4.7, the tag is $t=\langle r,t_1,t_2,...,t_d \rangle$ computed for this 4096-bit message.

We have n = 256 bits fixed-length messages. We have Message: M = 4096 bits. To calculate block size, each message block m_i has length n/4 bits: 256/4 = 64 bits each. With M being 4096 bits and n = 256, we have 4096 / 256 = 16 blocks. So the message is split into blocks: block $m_1, m_2, \ldots, m_{64}$

Each tag t_i is 256 bits (output of a fixed-length MAC). So for each block m_i , we get:

$t_i \in \{0, 1\}^{256}$ There are 64 such tags.

$r \in \{0, 1\}^{n/4} = r \in \{0, 1\}^{64}$. So the message identifier r is **64 bits**

Therefore, t becomes:

Final tag size = 64+(64×256)=64+16384= 16448 bits

For a 4096-bit message, and n = 256 bits, Construction 4.7 divides the message into 64 blocks. Each block has a size n/4 = 64 bits. Each block gets its own MAC tag (t_1, t_2, ..., t_{64}), and a random value r is added. Each tag is 256 bits. Therefore, the total tag size is: 64 bits + 64 × 256 bits (tags) = 16448 bits.

Question 2 Answer:

n = 512 bits. This is the size of the final digest message. l = 64 bits means that the length of the original message. n' = 1024 bits where each input to the compression function is 1024 bits total. The maximum file size that can be hashed using the Merkle–Damgård transform (Construction 6.3) is: 2^{64} -1 bits = 2^{61} - 1 bytes. Output size of the hash: n=512 bits.

Question 3 Answer:

The size of the hash in bits is n = 512 bits. The hash size given a file size of 2^32 bits is 512 bits. Original message length is 2176 bits. Padding rules (Construction 6.3), we will append a single 1-bit, then enough zeros to make the length l mod n', and finally append the 64-bit representation of the length. Total padded message length: 2176 (original message) +1 (added)+831(zeros)+64(Then add 64-bit length encoding)= 3072 bits. Then, we break it into blocks: 3072/1024 = 3 blocks. So, the number of applications of h_s is 3.

Question 4 Answer:

To dissect the question, let me show you what we've been given:

Block cipher: Key size = 256 bits, Block size = 128 bits. Message length = 4096 bits.

We have also given two Usings: Figure 4.1 = Basic CBC-MAC (for fixed-length messages) Figure 4.2 = Secure CBC-MAC (for arbitrary-length messages).

To answer the first question: Message = 4096 bits, Block size = 128 bits, 4096/128 = 32 blocks. In Basic CBC-MAC, the message is divided into 128-bit blocks. The MAC is computed by XORing each message block with the previous ciphertext and encrypting. So, 32 block cipher evaluations.

Figure 4.2 adds an extra encryption of the message length to the beginning: still processes 32 message blocks: 32 evaluations +1 extra evaluation for the message length pre-processing. 33 block cipher evaluations.

If the key size changed from 256 bits to 128 bits:

The key size does not affect the number of evaluations. It only affects the security level (e.g., brute-force resistance). We still process the same number of blocks the same way. To sum up, no change in the number of evaluations. We still have 32 (basic) and 33 (secure).

Question 5 Answer:

With the same (email ID and password) at two different sites: A and B, that use the standard hash function f(h). The stored hashed password for the user should be **different** at each site.

The password hashing scheme uses the user's password as input to the hash function. This means that the **same password** will always produce the **same hash value**. This is a **without a salt**.

A random value called a salt is added to the user's password before hashing. The salt is unique for each user and site combination. So, the hashed password at **site A** will **be different** from the hashed password at **site B**. This is **with salt.**