Arika Khor
Computer Security CS-5173-001

Assignment 2

Question 1. What are the properties of hash functions?
Hash functions have four main properties:

- Performance: The input function H(m) is easy to compute, but obtaining the input is much harder. Essentially, the computation/function is much simpler going forwards than backwards.
- One-way property: Even if one knows the algorithm, it's computationally infeasible to calculate the input.
- Weak collision resistance: It is virtually impossible to find two colliding inputs such that they match each other (which would thus break the security
- Strong collision resistance: In the same manner of the possibility of the inputs colliding, this also applies to the outputs colliding, stating that it is computationally infeasible to find two inputs that hash to the same output function.

Ref: Lecture 9 Slides

Question 2. Explain meet-in-the-middle attacks against double-DES.
To explain the attack, we will use a case study of an example attack with the given information:

- Take two keys K_1 and K_2 of some total size being 112bit.
- For some P plaintext, C ciphertext, and I intermediate text, we then have some model like so:

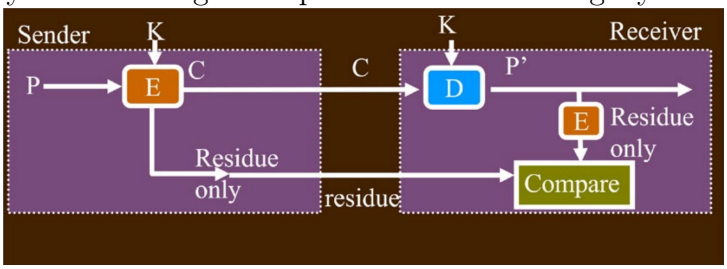
$$P \rightarrow E(K_1) \rightarrow I \rightarrow E(K_2) \rightarrow C$$

The attackers steps would then be as follows:

1) Encrypt P with a single DES to generate a table of all 2^{56} possible values of K_1. This would contain the intermediate I text values.
2) Using the same table, we can then decrypt C by again generating another 2^{56} values of K_2 which would also contain I.
3) This essentially makes it so even though the double DES uses 2^{112} keys, the strength is actually only 2^{57} as when we combine steps 1 and 2, we get $2^{56} + 2^{56} = 2^{57}$.
4) From here we can use a simple lookup between these lists to find a matching value. Once that value is found we've found the secret key and thus broke the security of the double DES.

While this attack would be quite costly, it is still very much possible given today's computing standards (combined with how advance our distributed systems have gotten), hence showing how a meet-in-the-middle attack could theoretically prove double DES to be insecure. Source: Lecture 7 Slides

Question 3. Explain why the following attempt cannot ensure integrity.



Given that an attacker could intercept and change C and residue, it's entirely possible for one to change the intermediate C text. It doesn't exactly matter if the attacker can decode the message as the question asks for integrity, though if they were to intercept the residue too, that would compromise the integrity. If we wanted to secure this we could ensure that there are no intermediate values left out in the open to prevent man-in-the-middle attacks. This can be done by using a HMAC or key based signature authentication (using something like a GPG key).
Source: Lecture 9 slides

Question 4. Alice designs a new double-DES scheme. The scheme will first DES-encrypt a message using K1 to get an intermediate ciphertext, then DES-decrypt the intermediate ciphertext using K2 to get the final ciphertext. Is there any vulnerability in Alice's design?
Yes, there is an issue with the new scheme (as it is still not perfectly secure). The decryption works in the same manner as encryption but reverses the order in which round keys are applied. Hence, we can still treat the final key strength somewhere around (or more likely an exact) 2^{57}, which as seen in question 2 does not hold. This similarity between encrypting and decrypting is a common feature of Feistel ciphers, and is not necessarily a bad thing given the quick time it takes to decrypt a key.
For some P plaintext, C ciphertext, I intermediate text, D decryption, E encryption we would have something along the lines of:

$$P \rightarrow E(K_1) \rightarrow I \rightarrow D(K_2) \rightarrow C$$

Source: Lecture 5/6 slides

Question 5. Suppose the sub-key generation function is to reverse all the bits of the key K (e.g., 0 1 1 1 $\rightarrow$ 1 1 1 0), and the scrambling function is f = M XOR K', where M is the second half of input bits and K' is the sub-key (i.e., the reverse of the original key K). Now given the original K = 0011, and input bits 1111 0000, compute the output of the Feistel Cipher.
We begin by formally defining all the input variables:

- input = 1111 0000
- M = 0000
- K = 0011
- K' = 1100
- f = M $\oplus$ K' = 1100
- Rounds: 2 (Following example from slides)

Then for the first round of the Feistel Cipher:

- Left half = 0000
- Right half = 1111 $\oplus$ 1100 = 0011
- Combined first round: $\boxed{0000 \quad 0011}$

Second round:

- Left half: 0011
- Right half: 0011 $\oplus$ 1100
- Combined second round: $\boxed{0011 \quad 1111}$

Source: Lecture 5 slides

Question 6. A and B want to ensure the integrity and authenticity of the messages between them, but they do NOT care about the confidentiality. Assume A and B share a key K. Answering two questions

1) How can they achieve their goal only with symmetric key cryptography?
   One modern day example of shared key cryptography is the usage of GPG (GNU Privacy Guard) keys. For example, A would both tag their message (such as a git commit) with B's public key to encrypt it. B would verify and decrypt the message with their private key using a command like gpg –verify. This ensures that the message sent was not tampered with in any way.
   In a high level view, GPG uses the same principles as shared key cryptography using a public encryption key and private decryption key. The public keys are usually shared through the keyservers (such as through https://keys.openpgp.org/) and can be signed by multiple users.
   Integrity and authenticity aren't covered given they aren't cared for.
2) How can they achieve their goal with hash function H?
   A and B cannot use a basic cryptographic hash given one could easily use a length extension attack. Hence, A and B should use a HMAC (Hashed Message Authentication Code), which combines a hash function and a key together. We take some function HMAC(Key,Map) = MAC. Then when the end user wants to decrypt the message they can simply use the same key and hash

to decrypt the function. If the message is equal it it will be equal to the original MAC, else it has been tampered with.

Question 7. Bob is assigned a task to design a way to allow encryption of files stored in the system: all files are stored in an encrypted form. If a block of a file is requested, the system should retrieve the block, decrypt it and return the plaintext to the host. Similarly, if a host writes a block to the storage system, the system should retrieve the right keying material, encrypt the block, and only save the ciphertext on disk. Consider the modes of operations discussed in class (i.e., ECB, CBC, CFB, CTR). Which one should be used in terms of read/write efficiency? Why? (You do NOT need to consider the key storage problem.)

Counter mode is likely the best choice out of the 4 presented for this for reading and writing efficiency. This is due to its random access nature , allowing for extremely fast lookup and write times similar to a hashmap. The parallel processing allows for faster performance on larger files, and is fairly secure with several modern day application using CTR. The potential risk with CTR is reusing counter values which can lead to a breach of security.

If we were more worried about perfect security then CBC (the block chaining method) would be better given how long it takes to verify each block, and linking each block making them all depend on each other. Of course, all of these techniques would be combined other ones, and in reality one would end up using a well known cryptographic library instead of these raw formats.

Source: https://crypto.stackexchange.com/a/33861 and Lecture Slides