SOLUTIONS FOR ASSIGNMENT - 2

HUMESH REDDY VENKATAPURAM CS 513: COMPUTER SECURITY

1)

a) Because stream ciphers work in a set and predictable way with a limited set of possible outputs, they're bound to end up repeating the sequence of bits they produce for encryption. As the cipher goes through its cycle, moving from one state to the next in a specific order, it will eventually run out of new states and start over from a state it's been in before. This means the sequence of bits, or the keystream, will start repeating.

b) When the same sequence of bits (keystream) is used more than once in encryption, it becomes a major security issue. This is because of a simple math trick with XOR operations. If two messages are encrypted with the same keystream, an attacker can easily mix the encrypted messages and end up with a mix of the original messages. This gives them clues about the content of those messages. Stream ciphers are supposed to work like a one-time pad, meaning each keystream should be unique and used only once. If a keystream gets reused, it can lead to all messages encrypted with it being exposed.

2)

a)

$$C=P \oplus K$$

$$K=P \oplus C$$

In a stream cipher, we use a keystream (generated from a key) to encrypt plaintext. The XOR operation combines the plaintext with the keystream to produce the ciphertext. Trudy (the adversary) can determine the keystream by XORing the known plaintext P with the ciphertext C. This property is due to the commutative nature of XOR. Essentially, Trudy can figure out the keystream used for encryption.

b)

Trudy gets hold of the keystream used earlier let’s call it S. Trudy uses this keystream S to encrypt a new plaintext P’ and creates a fresh ciphertext C’. When Bob receives C’ and tries to decrypt it using the same key, he gets back P’. Bob won’t realize that the original plaintext has

been changed because he can't verify whether the text came from Alice or Trudy. So, Trudy effectively alters the message without Bob knowing.

$$P' \oplus S = C'$$

$$C' \oplus S = P'$$

3)

a) With the total 8 possible scenarios, the X register advances about 3/4 = 75% on avg of the time in 6 iterations.

b) With the total 8 possible scenarios, the Y register advances about 3/4 = 75% on avg of the time in 6 iterations.

c) Same like X and Y registers, the Z register progresses around 3/4 = 75% on avg of the time in 6 iterations.

d) With the total 8 possible scenarios, the all register steps 2 times about 1/4 = 25% on avg of the time.

e) In the 6 iterations, exact two registers advance around 3/4 = 75% of the time on average.

f) This situation is implausible because the majority value relies on having at least two bits should be same.

g) Here, if no registers step forward, the A5/1 stream cipher will not function.

4)

a) A Feistel Cipher is like a secret dance for encrypting messages. Imagine we have a block of text, let's call it P. We split it into two halves: L0 and R0. Now, for each round like a dance move, we do the following: Process one half (Ri-1) through a function (F) and combine it with the other half (Li-1) using XOR. Then, we swap the halves: Li becomes Ri-1, and Ri becomes the result of our dance move. We repeat this process for several rounds until we get our final encrypted message (ciphertext).

b) Yes, DES is a feistel cipher

c) No, AES is not a feistel cipher

d) TEA is similar to a Feistel cipher. It splits the plaintext into two halves. It processes both halves simultaneously left and right using addition and subtraction (instead of the usual XOR). But here's the twist: when decrypting, TEA doesn't follow the same steps as encryption (unlike a typical Feistel cipher).

5) We have a two-step process:

First, we encrypt the plaintext P using a key K1 to get an intermediate value M.

$$M = E(P, K1); C = D(M, K2)$$

Then, we decrypt M using another key K2 to get the ciphertext C. Now, imagine an attacker who knows both the plaintext P and the ciphertext C. Their goal is to find out the keys K1 and K2. The interesting part is that we can infer: M (the intermediate value) can also be obtained by encrypting C with K2.

$$M = E(C, K2)$$

Here's how the attack works: The attacker guesses all possible K1 values and computes M' for each guess. They store these M' values along with the corresponding K1'. Next, they guess K2 values and compute M' using C and K2'.

$$E(P, K1') = E(C, K2')$$

If any M' matches the stored ones, they've found the correct K1 and K2.

6)

a) The correct approach for adapting the meet-in-the-middle attack on double DES when only known plaintext is available:

For each of the 2^56 guesses for Key1, create a table mapping the known plaintext (P1) to a middle value (M1) for each guess like M1G1,M2G2,.....

Similarly, for each of the 2^56 guesses for Key2, create a table mapping the ciphertext (C1) back to a middle value (M'1) for each guess like M'1G1, M'1G2,.....

Now look for collisions matches between the middle values from both tables. A unique collision indicates the correct combination of Key1 and Key2. If there is more than one collision or no collision at all, the process may need to be repeated with another set of known plaintext and ciphertext pairs (P2, C2).

This approach accurately captures the necessary adjustments when moving from chosen plaintext to known plaintext scenarios in a double DES attack. The main difference lies in the need to recreate the lookup tables from scratch for each new pair of plaintext and ciphertext, maintaining the computational work factor around 2^55 which is comparable to an exhaustive search for a single 56-bit DES key.

b) Creating the lookup table is a task you do just once, so if you use this table for multiple attacks on double DES, the effort it took to make the table becomes less significant with each

additional attack. If we don't count the initial effort to make the table, then the main work involves decrypting with different keys until a match is found in the table, which typically takes as much effort as trying to crack a single DES encryption by trying every possible key.

However, the initial creation of the lookup table is a big task that you can't spread out over multiple attacks. This means the overall effort required is roughly equivalent to trying two 56-bit keys (totaling 112 bits), which is a huge amount of work. If you're only working with known plaintexts (the encrypted messages you already have) instead of choosing your own plaintexts to encrypt, then the amount of work needed roughly doubles.

7)

a) IV (Initialization Vector) is like a special ingredient in encryption. Its job is to make sure that even if you encrypt the same thing twice, you get different results. Usually, IV should be random and unpredictable for better security. But in some cases, like stream ciphers, it’s okay if IV follows a sequence—as long as the encryption still relies on a strong key.

b) Using sequential initialization vectors (IVs) in encryption is generally not recommended because it can weaken security. Here’s why: Attackers might be able to predict the next sequential IVs used in future encryption. This predictability allows them to guess the encrypted data and potentially compromise the encryption.

8) No, it is not secure. Swapping the order of parameters in the counter mode encryption formula (using Ci = Pi XOR E(K, IV + i)) weakens the security guarantees. The original formula (Ci = Pi XOR E(IV + i, K)) ensures uniqueness and unpredictability of the keystream, which is essential for security.

9)

a) The corresponding decryption for the above rule:

P0 = IV $\oplus$ D(C0, K),

P1 = C0 $\oplus$ D(C1, K),

P2 = C1 $\oplus$ D(C2, K)

b) The rule for encryption and decryption is strictly sequential. In other words, the encryption of one piece of plaintext (Pi) depends on the encryption of the previous piece (Pi−1). This process cannot be done in parallel. It is less efficient. If an Trudy intercepts some of the ciphertext during transmission and replaces it with garbage values (for example, C1’ instead of C1), it can

lead to errors. For instance, P0 (the first plaintext block) will be correctly computed, but P1 (the second block) will be based on the incorrect C1'. This error can propagate, affecting subsequent blocks like P2.

This vulnerability is especially concerning in environments with high error rates, such as wireless communication. A single-bit error can disrupt entire blocks of data.

10)

a) If you use the same starting point for every message you encrypt, then sending the same message twice will end up with the same scrambled message each time, similar to what happens with a simpler encryption method called ECB. This goes against one of the key benefits of using CBC mode, which aims to make sure that doesn't happen.

b) When you use the same Initialization Vector (IV) for every message transmission, it leads to the generation of an identical keystream each time. Let's consider two separate messages: C1 and C2. C1 is the result of XORing the first plaintext block (P1) with a secret key (k). C2 is the result of XORing the second plaintext block (P2) with the same secret key (k). Now, if we compute the XOR of C1 and C2, we get P1 XOR P2. This reveals some information about the plaintexts. If Trudy intercepts any plaintext, it compromises the security of all future communications.

c) In simpler terms, when using CBC mode for encryption, only the first section of the encrypted message heavily depends on the initial setup value. But with CTR mode, the initial setup value directly affects every section of the encrypted message, without being influenced by the sections that came before it.

11) Given,

The plaintext block P - 8 bit = 10110101

The subkeys k1 through k4 means: k1,k2,k3,k4 = 1011, 0100, 0101, 1010

n = 4

Here in the fiestel cipher, plaintext splits into 2 halves right and left.

$R_i = L_{i-1} \oplus F(R_{i-1}, K)$ and $L_i = R_{i-1}$ where i = 1,2,3,4,5.....n rounds.

1st round:

n = 1

L1 = R0 = 0101

So, R1 = L0 $\oplus$ F(R0,K1) = 1011 $\oplus$ 0101 $\oplus$ 1011

=> R1 = 0101

2nd round:

n = 2

L2 = R01 = 0101

So, R2 = L1 $\oplus$ F(R1,K2) = 0101 $\oplus$ 0101 $\oplus$ 0100

=> R1 = 0100

3rd round:

n = 3

L3 = R2 = 0100

So, R3 = L2 $\oplus$ F(R2,K3) = 0101 $\oplus$ 0100 $\oplus$ 0101

=> R1 = 0100

4th round:

n = 4

L4 = R3 = 0100

So, R4 = L3 $\oplus$ F(R3,K4) = 0100 $\oplus$ 0100 $\oplus$ 1010

=> R1 = 1010

So, the ciphertext = 01001010