HW2 **CSL7480: Cryptography**

Due date: 15th Feb 2023. 11:59 pm.

Submission process: Submit three files in the Google classroom (NOT as a zipped archive. Submit 3 separate files): the source codes for the two questions and a pdf file explaining how did you develop the attacks in the two questions. Name your files as (i) rollno-code-1, (ii) rollno-code-2, and (iii) rollno-report.pdf (replace rollno with your entry number). Any programming language is acceptable for the two problems. Mention your roll number and name clearly at the top in all the files.

---

1. Suppose the length of a message m, which is to be encrypted with a block cipher E in CBC mode, is not a multiple of the block length of E. In this situation, we need to pad the message so that the padded message has a length which is multiple of the block length. For example, if E is AES and m is 28 bytes long then we need to pad the message with 4 bytes. However, the receiver should be able to know what was the original message, and hence needs to know the pad length. One way this was done in many internet standards earlier was to use the padding byte values to be b if the number of bytes being filled were b. In our example, the 4 bytes of the padding will be `0x04 0x04 0x04 0x04`. If the message length was 15 bytes then the padding byte will be `0x01`.

   In this exercise, you will write a program to decrypt ciphertexts encrypted with AES in CBC mode, *without knowing the secret key*, when the padding scheme mentioned earlier is being used. You will *guess* a plaintext, and verify if it satisfies the padding rule. If not then you will be able to reject the guess and try the next guess. You will have access to a *padding oracle*, which will decrypt a given ciphertext = (IV, c) and give you a true/false answer about the padding being correct or not. You can use this oracle as many times as needed, and with different IV values.

   The main idea is as follows: The real decryption function for (IV, c) is $m = D_k(c) \oplus IV$. But you will use a different initialization vector IV' and call the padding oracle on (IV', c). The oracle with compute $m' = D_k(c) \oplus IV'$, and return true or false depending on whether m' satisfies the padding rule. Repeating this process a number of times will enable you to know the last byte of m. Using the same strategy, you can keep on recovering the other bytes of the message one by one. It is easy to visualize the attack for one block of ciphertext (i.e. c being 16 bytes in our example). But the idea can be trivially extended to multi-block ciphertexts as well.

   Use any library which implements AES and the CBC mode, and implement the following:

   (a) Write functions for CBC encryption with AES, and CBC decryption with AES. The encryption function should implement the padding scheme described earlier. Generate a random AES key for these functions which should, of course, be the same in both these functions.

   (b) Write a function named "padding_oracle" (using the above functions) which decrypts a given (IV, c) pair, and produces a true/false answer as mentioned earlier. The padding oracle knows the secret key being used by the encryption/decryption functions.

   (c) Create a plaintext with the following format: "your roll number, your name, your father or mother's name, and your hometown". If the plaintext is already a multiple of 16 bytes, then add some more details in the text to ensure that it is NOT a multiple of 16.

(d) Encrypt the plaintext with the CBC encrypt (with appropriate padding) to produce the output (IV, c).

(e) Decrypt the ciphertext (IV, c) by calling the padding oracle function as many times as needed. Note that you are not supposed to know the secret key in this attack. The only operation you are allowed to perform are modify IV's and call padding oracle on different inputs to get a true/false answer.

2. This question explores the concept of differential cryptanalysis against an AES like cipher.

(a) We will use the excellent tutorial by Howard Hayes on differential attack for this problem. The tutorial is attached with the assignment.

(b) Look at the design of an SPN based block cipher on Page 4 of the tutorial. The description of the S-box and the permutation layer are given on page 3.

(c) Read sections 4.1, 4.2, 4.3 and 4.4 of the tutorial to understand how does the differential cryptanalysis work. The top level idea is to look at many pairs of inputs with a fixed difference between them and observe the output difference between them. That is, we fix a value δx before the attack is mounted, and pick two different messages x' and x'' to encrypt, where $x' \oplus x'' = \delta x$. Let the ciphertexts be y' and y'' for these messages, respectively. We observe the difference $y' \oplus y''$. Repeated trials (in the chosen plaintext attack setting) will allow us to generate message pairs satisfying some fixed input difference δx mapping to a fixed output difference δy.

(d) Understand what is the difference distribution table (DDT) for the S-boxes used in the design.

(e) The propagation of the differences $\delta x \to \delta y$ (termed as the differential characteristic) and the conforming message pairs (x', x'') (i.e. the message pairs which satisfy the desired input and output differences) can be used to extract certain bits of the secret key. The main idea is to look for the propagation of the input differences to the outputs just before the last key XOR operation. Clearly, the XOR of a fixed value with two quantities does not change the XOR difference of the two quantities. That is, if $\alpha \oplus \beta = t$ then $(\alpha \oplus u) \oplus (\beta \oplus u) = t$ for any u. This means that the output difference is the same as the output difference just before the last key XOR operation.

(f) Now, the SPN design does not have the permutation layer in the last round. The next operation (from the decryption side) is the S-box layer. If the S-box used is $S(\cdot)$ then we can say that the operation we are looking at is $S(v \oplus k) = w$ for some u, w and secret key bits k. We already know w (this is the output difference at the end of the cipher). We can make a guess for the key bits involved and verify if the guess is correct using the DDT. This allows us to recover some key bits.

Now, we are in the position to state the problem.

The problem:

(a) Construct a DDT for the given 3 bit to 3 bit S-box:

Input:	000	001	010	011	100	101	110	111
Output:	110	101	001	000	011	010	111	100

(b) Consider the simple substitution permutation network shown in Figure 1 on the next page. Assume that the S-box defined above is used. Find the encryption of the plaintext "011010", using the key $(K1, K2, K3, K4) = (010101, 001011, 111000, 111110)$. For convenience, also show the intermediate results (i.e., the rows A, B, D, E, F , G, H, and J from the Figure.

(c) Consider two plaintexts P and P' such that $P \oplus P' = 000001$. What are the possible values of δH (input to the last round of the S-boxes).

(d) You are given the following Chosen plaintexts and ciphertexts encrypted using this cipher.

Plaintext	Ciphertext
000110	110110
000111	110010
001000	001101
001001	000011
001100	111001
001101	100000
011000	011101
011001	011111
011010	101001
011011	101000
100110	111110
100111	100100

Determine the last 3 bits of subkey K4.

(e) Can you determine more keybits?

Hints and suggestions

Q. 1: This attack was used to break SSL 3.0 encrypted traffic in practice. The initial attack was announced by a team of security researchers from Google in October 2014. However, the attack was originally described by Serge Vaudenay in Eurocrypt 2002, a full 12 years before this was used in practice! A variant of this attack was announced in December 2014 against TLS 1.0 as well. CVE-2014-3566 and CVE-2014-8730 were issued for these attacks in 2014.

Q. 2: All the details are already given in the brief writeup. In case of any confusion, please consult the tutorial by Hayes.

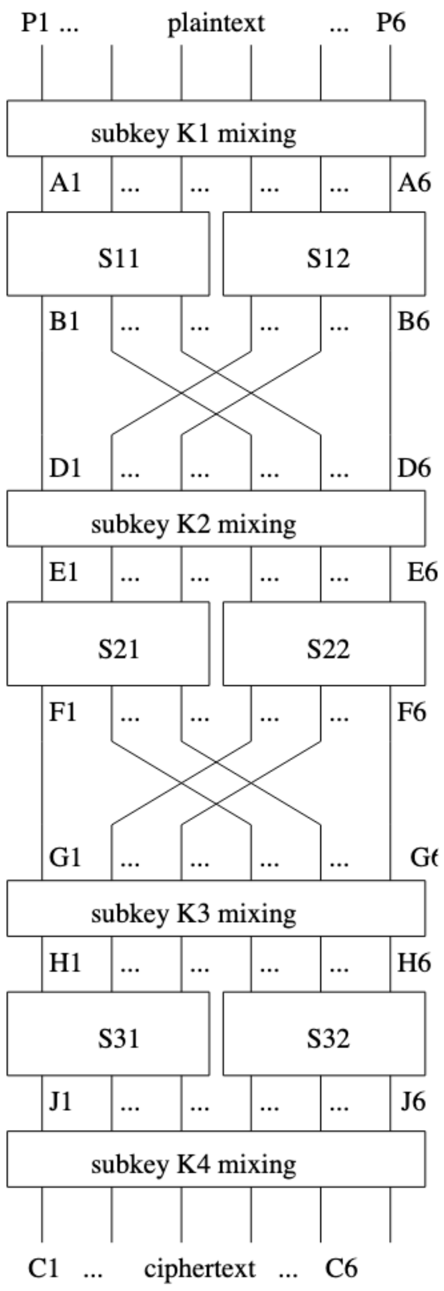

Figure 1: The SPN cipher