The Hong Kong Polytechnic University

COMP 3334 – Computer Systems Security (Semester 2, 2024)

Assignment

This is an individual assignment. You may use the course material and Internet resources to answer the questions. However, you should not post the questions online and ask for help. Discussion among your peers is encouraged; however, you must produce answers by yourself and in your own words. Any suspicion of plagiarism will be thoroughly investigated. Copying answers from GenAI tools into your assignment is a form of plagiarism. This assignment is due on **Sunday, 17 March 2024, 23:59**.
Late submissions will be subjected to a 15% penalty per day, starting at 00:01.
Total: 100 points. Course weight: 10%.

Submission requirements

Submit on Blackboard a single ZIP file containing:

1. A PDF file named as `comp3334-`*`studentid`*`.pdf` for your written answers. Change "studentid" with your actual student ID. The file must also include your name and student ID on the first page.
2. The three Python files as requested in the exercises below: `exercise{1,3,4}.py`.

Double check your submission. Any incorrect submission format may result in a zero mark for this assignment.

Exercise 1: Misusing AES [30 pts]

A website authenticates its users by asking for a login/password, and sends them a cookie C, valid for one minute, to keep track of their authentication status. The cookie C is formed such as $C =$ Enc(`"user=`*username*`,tmstmp=`*timestamp*`"`), with *username* = `"anonymous"` for unauthenticated users, or the name of the user when authenticated; and *timestamp* is a Unix-formated timestamp[1] representing the time up to which the user is authenticated (current time plus one minute). Enc$(\cdot)$ designates the AES256 encryption in OFB-mode using iv as a random IV and k as a random key; both k and iv are unknown to us. The OFB mode of operation for encryption is described in Figure 1.

In this exercise, we consider cookies delivered on February 1st, 2024 at 00:00am UTC. At that time, an unauthenticated user coming to the website will receive a cookie:

$$C_U = \text{AES256-OFB}_{k,iv}(\texttt{"user=anonymous,tmstmp=1706745660"})$$

The value 1706745660 corresponds to 00:01am on February 1st. We denote by P_U the plaintext version of the cookie.

a) Give the plaintext cookie, denoted P_A, that corresponds to the authenticated `admin` user if he logged in at the same time. Compare the length of P_A and P_U. [3 pts]

b) Describe how to modify the cookie C_U into $C_A = \text{AES256-OFB}_{k,iv}(P_A)$ without knowing k nor iv. You may rethink about the value of P_A so that P_U and P_A have the same length. Note that the cookie may authenticate the user `admin` for as long as you want (but at least the original one minute duration). [7 pts]

c) Implement in Python the attack that would turn C_U into a valid C_A for at least the original duration. Prepare a single Python file named `exercise1.py` that contains a function `modifycookie()` that takes as argument a base64-encoded cookie and returns the base64-encoded modified cookie. Your attack should work for different timestamps. You will get full marks if, given an encrypted cookie

[1] https://www.epochconverter.com/

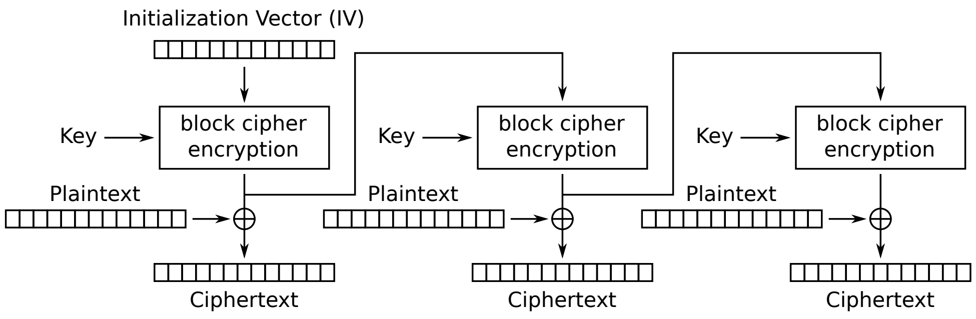

Figure 1: Output Feedback (OFB) mode of operation (during encryption)

issued at any later date than February 1st this year, you are able to turn this cookie into a valid admin cookie for at least the original duration. Make sure your code uses meaningful variable names, consistent indenting scheme, and comments. [20 pts]

Exercise 2: Lan Manager hash [20 pts]

Back in the days, up to Windows XP, Windows account passwords were hashed using Microsoft's LAN Manager (LM) hash function, which works as follows:

Step 1 The password is converted into upper case, null-padded to 14 characters (or truncated to 14 characters), and split into two 7-character halves.

Step 2 Each half is separately converted into a DES key. This key is used to encrypt the ASCII string "`KGS!@#$%`", producing an 8-byte value.

Step 3 The two 8-byte values are concatenated, resulting in a 16-byte hash.

1. Suppose the attacker obtains a file with N hashed passwords. How much work would he need to do, at most, to crack these passwords by brute-force search? Show your calculations. Express the result in scientific notation ($m \times 10^n, m \in \mathbb{R} < 10, n \in \mathbb{N}$), and round it to two decimals. Assume that users could type any of the 95 printable characters found on a US keyboard i.e., letters, numbers, symbols, and punctuation marks, which are represented by codes 32 to 126 in the ASCII table. Passwords could be any length. [15 pts]
2. Knowing that a single modern NVIDIA GeForce RTX 4090 GPU can perform 151.1 GH/s for LM,[2] how long would it take an attacker with one such GPU to crack these N hashes? Assume that the overhead of matching a 64-bit string in a list of $2N$ 64-bit strings is negligible. [5 pts]

Exercise 3: PKCS#7 padding [20 pts]

The PKCS#7 padding scheme is commonly used to pad irregularly-sized plaintext messages to a specific block length before encryption, and is described in RFC5652. The algorithm simply consists in appending the required number of bytes up to nearest multiple of the block length. The value of each appended byte is equal to the number of bytes added. The maximum length of the pad is therefore 255. If the plaintext size is already a multiple of the block length, padding is still required. In this case, an entire block of padding bytes will be used. To remove the padding after decryption, the last byte of the decrypted ciphertext is read, which indicates how many bytes (of the same value) should be stripped from the end. For this exercise, **do NOT use any library/package** in your code.

1. Write a function `pkcs(plaintext, length)` in Python that takes a plaintext (`plaintext`) and a desired block length (`length`) as input and applies the PKCS#7 padding scheme. The function should return the padded input. An exception should be thrown if the block length is greater than the maximum pad length, using: `raise Exception("Invalid block size")`

   For instance, for the message `"YELLOW SUBMARINE"` and a block size of 20, the output should be `"YELLOW SUBMARINE\x04\x04\x04\x04"`. [5 pts]

[2]See Hashcat benchmark here: https://gist.github.com/Chick3nman/32e662a5bb63bc4f51b847bb422222fd

2. Write a function `validate_pkcs(plaintext, length)` in Python that verifies the validity of the padding, and returns an unpadded string. The function should throw exceptions using `raise Exception("Invalid padding")` when: 1) the length of the plaintext indicates that no padding has been used; 2) the value for the pad length is incompatible with the block length; 3) the value of the padding bytes is incorrect. [15 pts]

Prepare a single Python file named `exercise3.py` that contains both functions.

Make sure to use the exact exception messages given above throughout this exercise; otherwise, your function will not be evaluated properly for assessment.

Exercise 4: Near collisions [30 pts]

Referring to the exercise given in Lecture 4 (slide 23), find a "near collision" on SHA-256 by hashing values that *must contain your student ID*. A near collision is defined as a pair of inputs which hash to values that share the same first n bits. We define $n = 34$ for this exercise.

One method of finding such collisions is to hash an increasing counter (concatenated to your student ID) and keep n-bit prefixes in a dictionary. If you encounter a prefix you have already seen, you found a near collision.

1. Provide a function `find_near_collisions(studentid)` that takes as input your student ID as a string in the format 12345678 (without letter) and outputs a tuple of binary strings (val1, val2) that are near collisions, and `studentid` is a substring of each value. Your program should terminate within one minute on a reasonably modern laptop and with a correct output. [20 pts]

2. Provide a function `get_values()` that returns a tuple of binary strings (val1, val2) that you have previously computed and that satisfy the above criteria. Simply hardcode those values and return them. [10 pts]

   Example of correct outputs for student ID=12345678: (b'12345678288576', b'12345678335737').

Prepare a single Python file named `exercise4.py` that contains both functions.

Questions?

If you need a clarification about an exercise requirements, you can contact the following TA:

Bowen CUI: bowen.cui@connect.polyu.hk

TAs will not tell you whether your approach is correct or not, whether you got the right answer, etc.