UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Winter 2019**

Cecylia Bocovich
Yaser Baseri

ASSIGNMENT 3

Assignment due date: **Tuesday, April 2nd, 2019 4:00 pm**

Total Marks: 55

Written Response Questions TA: Jiayi Chen j524chen@uwaterloo.ca
(Office hours: Friday 14:00 pm–15:00 pm in DC 3333)
Programming Questions TA: Navid Esfahani nnasresf@uwaterloo.ca
(Office hours: Tuesday 1:00 pm–2:00 pm in DC 3333)

Please use Piazza for all communication. Ask a private question if necessary. The TAs' office hours are also posted to Piazza for reference.

Written Response Questions [25 marks]

1. [4 marks total] **Two-time Pad**

   In the One-time Pad (OTP) cryptosystem, the secret key must be truly random and should never be reused. Let C denote the ciphertext, P denote the plaintext, and K denote the key. OTP encryption is defined as $C = P \bigoplus K$, where $\bigoplus$ is the bitwise XOR operator and the key has the bit length l as both the ciphertext and the plaintext.

   The following subquestions will help you understand why two-time pad is insecure. Suppose that we have two ciphertexts C_1 and C_2 using the same key K (i.e., "Two-time Pad"), where their plaintexts P_1 and P_2 as well as the key are **unknown** to us.

   (a) [2 marks] Please describe how to obtain $P_1 \bigoplus P_2$ and provide a brief proof.

   (b) [2 marks] Suppose that both plaintexts are meaningful texts, containing ASCII characters from only 52 English letters (i.e., "A-Z a-z") and space. Please describe how to recover the plaintexts P_1 and P_2.

2. [8 marks total] **GnuPG**

   A GnuPG public key for j524chen@uwaterloo.ca is provided along with the assignment on the course website (j524chen.asc). Perform the following tasks. You can install GnuPG on your own computer, or use the version we have installed on the ugster machines.

(a) [2 marks] Generate a GnuPG key pair for yourself with 2048-bit RSA as the encryption algorithm. Use your real name and your uWaterloo email address for the key pair generation. Export this key using ASCII armor into a file called **key.asc**.

(b) [2 marks] Use this key to sign (not local-sign) the j524chen@uwaterloo.ca key. Its true fingerprint is: 2590 9C18 2FBE 3546 87E2 3F7D C9B4 AA02 8994 3E76. Export your signed version of the j524chen@uwaterloo.ca key into a file called **j524chen-signed.asc**; be sure to use ASCII armor.

(c) [2 marks] Create a message containing your userid and name. Sign it using the key you generated, and encrypt it to the j524chen@uwaterloo.ca key. You should do both the encryption and signature in a single operation. Make sure to use ASCII armor, and save the output in a file called **message.asc**.

(d) [2 marks] Briefly explain the importance of fingerprints in GnuPG. In particular, explain how users should check fingerprints and what type of attacks are possible if users do not follow this procedure properly.

3. [5 marks total] **Diffie-Hellman**

The Diffie-Hellman (DH) key exchange protocol is designed to establish a common secret between two parties when using an insecure channel. See http://mathworld.wolfram.com/Diffie-HellmanProtocol.html for details.

(a) [2 marks] Assume Alice and Bob agree to use modulus $p = 83$ and base $g = 35$. Then, Alice chooses secret parameter $a = 17$ and Bob chooses secret parameter $b = 48$. What are the public values that Alice gives to Bob and Bob gives to Alice? What is the resulting secret key that is generated as a result of the DH protocol?

(b) [3 marks] During the key exchange, Eve observes the values p, g, as well as the public parameters $A = g^a \pmod p$ and $B = g^b \pmod p$. Can Eve recover the secret parameters a and b? Please explain.

4. [8 marks total] **Inference attacks**

In the database of a hospital, the employee's information is maintained in a table called $\mathrm{Employee}$. The table has N records with the employee's id (the unique identifier for each employee), name, position (for simplicity, we only consider three types: doctor, nurse, or non-medical), and salary (annual income). We assume the composition of the employees in this table is 15 % doctors, 60% nurses, and 25% non-medical employees. Below is an excerpt (not the entire table):

ID	Name	Position	Salary
10001011	Patrick Harvey	Doctor	$212,000
10002087	Shannon Byrnes	Nurse	$79,000
10004005	Jordan Mason	Doctor	$133,000
10008197	Natasha Matthew	Nurse	$67,000
10012007	Olivia Burton	Doctor	$258,000
10012097	Nina Mccarthy	Nurse	$97,000
10015003	Amber Richardson	Non-medical	$112,000
10020023	Jason Barnes	Non-medical	$48,000

To deter employees from learning each other's salaries, the database is set up to suppress the Salary field in the output of queries. However, employees can execute queries in the following form:

$$\mathrm{SELECT\ SUM(Salary)\ FROM\ Employee\ [WHERE\ ...]}$$

where queries that match fewer than k or more than $N - k$ (but not all N) records are rejected. In this question, we will use $k = \frac{N}{8}$.

(a) [4 marks] Suppose that you are going to infer the salary of an employee named Nicky Snider (not shown on the excerpt) by doing several queries, but you do not know any other details about them. Use a tracker attack to design a tracker and a set of *three* queries based on this tracker that will let you infer the salary. In your solution, you should give 1) your tracker, 2) the set of three queries, and 3) how to obtain the salary.

(b) [4 marks] The hospital also maintains patient information in the database. For research purposes, the hospital is about to release the health data of some elderly patients in a privacy-preserving way. The hospital declares that *Name*, *Age*, and *Assisted Living Status* are identifiers. *Assisted Living Status* has the two possible values: *None*, meaning the patient receives no living assistance, and *Assisted* meaning they receive some level of living assistance. The hospital declares that the *Assisted Living Status* field is vital to research studies and should therefore not be masked. Thus, the hospital adopts the following way to anonymize the patient data:

i. *Name* fields are fully masked by stars;

Name	Age	Age (after)	Assisted Living Status	Disease
*	65	60–69	assisted	respiratory disease
*	68	60–69	assisted	respiratory disease
*	71	70–79	none	diabetes
*	72	70–79	assisted	diabetes
*	75	70–79	none	heart disease
*	78	70–79	none	Alzheimer's
*	78	70–79	assisted	respiratory disease
*	82	80–89	assisted	heart disease
*	85	80–89	assisted	heart disease
*	86	80–89	assisted	Alzheimer's

ii. *Age* fields are generalized to the following age groups: 60–69, 70–79, and 80–89.

The processed table is shown above.

Is the anonymized table 3-anonymous? If yes, please explain and give l for l-diversity. Otherwise, please present your solution and give l for l-diversity [You are allowed to do some obfuscation on *Age* fields by modifying the value to another integer within the range of ± 5 (e.g., change 72 into 75)].

Programming Questions [30 marks]

Background

In this section we will study padding oracle attacks. Block ciphers operate on a fixed number of bits, such as 64 (DES) or 128 (AES). To encrypt longer messages, a mode of operation such as CBC can be used (see Figure 1). However, this still requires that the message length is a multiple of the block size. In order to accommodate arbitrary-length messages, modes of operation apply a padding scheme. The decryption routine will need to check that the message padding is valid. Unfortunately, information about the validity of the padding can easily be leaked to an attacker. Based on the feedback regarding padding validity, an attacker might infer plaintext properties. More precisely, a "padding oracle", which reveals no information apart from the validity of the padding of a given ciphertext, can allow the attacker to decrypt (and sometimes encrypt) messages without knowing the encryption key.

The padding oracle attack was originally described in a 2002 paper by Serge Vaudenay. This paper is available at https://www.iacr.org/archive/eurocrypt2002/23320530/cbc02_e02d.pdf and will serve as your reference for this attack (Reading the first three sections will be enough to tackle this assignment; however, reading the whole paper is recommended). We provide some hints that might be helpful in understanding the paper in Section "Remarks on Required Readings". Padding oracle attacks similar to Vaudenay's attack have continued to be relevant since their inception, with new attacks being discovered regularly. One recent example is the POODLE attack on SSL 3.0 and TLS discovered by Google security engineers in 2014 (not a required reading).

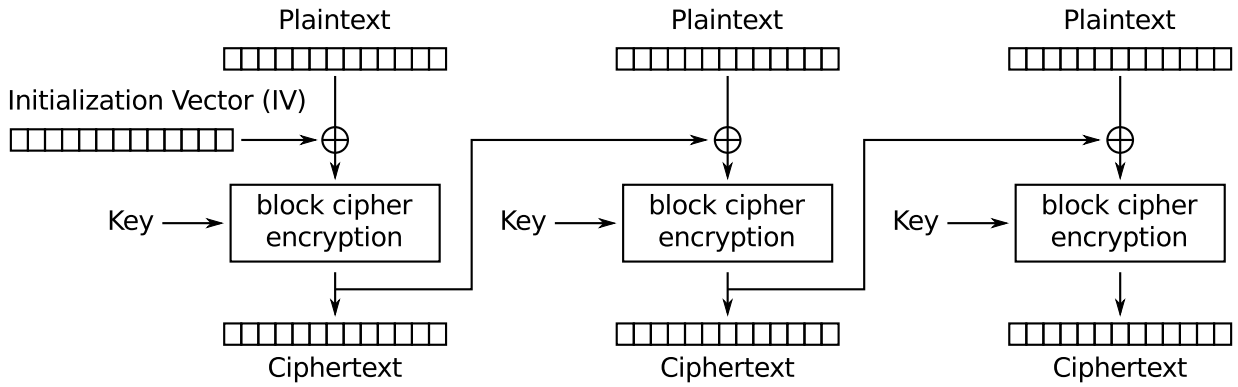

Figure 1: CBC Mode Encryption

Application Description

A simple web application is running at http://localhost:4555 on each ugster machine. To view this application in your web browser, you can use http://ugsterXX.student.cs.uwaterloo.ca:4555 (when off campus you need to use the university's VPN). However, please use http://localhost:4555 in your programs to ensure that they run quickly no matter which ugster machine we test them on.

When you visit the web application it will set a cookie named user in your browser. In other words, the HTTP response will contain a Set-Cookie header like the following:

Set-Cookie: user="7pEU3mxW4kuS1cGflOnc8DN6c4w8T/e8eMDUILO9sznE79EkYX9oEDSXev
8r5XcWqAqMuGZGh9qGF95w3w+iCw=="

The value of this cookie is encrypted with AES using a key known only to the web server. AES is used in CBC mode, and the randomly chosen IV (=nonce) is prepended to the ciphertext. The result is then Base64 encoded to form the cookie value. Base64 is a standard way of encoding binary data into printable ASCII characters. It is implemented in many programming languages and the base64 Unix utility, as specified in RFC 3548.

The web application uses *IP ESP* padding scheme. A padding of length n starts with the value 0x01, and is followed by a increasing sequence of numbers up to n. For example, 0x01 is a padding of length 1, 0x01 0x02 is a padding of length 2 , and 0x01 0x02 0x03 0x03 0x05 is a padding of length 5. All paddings start with a byte of value 0x01. There is always at least one byte of padding, i.e., messages whose length is a multiple of the block length receive an additional block of padding.

When you visit the web application again, your browser will send the cookie back to the server. This is done using a Cookie header of the same form as the Set-Cookie header above. The web server will attempt to decrypt any cookie sent to it using its encryption key. If the decryption is successful, the HTTP response will contain a 200 (OK) response code. However, if the decryption fails a 500 (Internal Server Error) response code will be returned. See the section below titled "HTTP with curl" for more information on working with HTTP. Instead of manually visiting the web application using your browser, you can also emulate browser behaviour in a programming language of your choice.

In this assignment, you analyze the scheme discussed in the above-mentioned paper, suggest a fix for the issue, then you modify the attack so that you can decrypt the cookies, and encrypt messages on behalf of the server, without having the secret key. Eventually, you implement one of the two other modes of operation.

Note that some of the questions require written answers, which should be provided as part of a3.pdf, together with the answers for the written part of the assignment.

Questions

1. [3 marks] How would you modify the attack described in the paper to account for the different padding scheme used by the web server? List the lines in Sections 3.1 and 3.2 that require changes and show how they should be changed.

2. [2 marks] What is the average case and worst case number of padding oracle calls required to perform the modified attack?

3. [2 marks] What cryptographic tool could be used to fix this vulnerability? Which of the properties Confidentiality, Integrity, Authenticity does the chosen cryptographic tool provide? How does it fix the vulnerability? Assume that the web server still has to return a different response for users with valid versus invalid cookies.

4. [10 marks] Write a program called decrypt that implements the padding oracle attack on the web application described above. Your program will be called with a single command line argument containing a Base64-encoded cookie value to be decrypted. These cookies will have at least two 16-byte blocks (the IV and at least one ciphertext block), but does not need to have the same number of blocks as responses returned by the server. Your program should generate the appropriate cookie values, send them to the web server using HTTP, and observe its response codes in order to decrypt the given cookie value. It should print the resulting plaintext to stdout, which will consist of only printable ASCII characters. You can print debug output to stderr if you like. Your program should run in a maximum of 10 minutes.

   You may use any programming or scripting language available on the ugster machines. If your preferred language is not available, we may be able to accommodate requests. Most common programming languages have built-in libraries for making HTTP requests to web servers. Alternatively, you can use the curl program as described in the section below titled "HTTP with curl". You will submit a single file named src.tar. For evaluation, this file will be extracted and we will attempt to run an executable at the top level with ./decrypt ¡cookie¿. This executable could be your program itself, or it could be a script that compiles and then runs your program. For the example cookie shown above, the invocation would be:

   ./decrypt 7pEU3mxW4kuS1cGflOnc8DN6c4w8T/e8eMDUILO9sznE79EkYX9oEDSXev
   8r5XcWqAqMuGZGh9qGF95w3w+iCw==

5. [5 marks] Write a program called encrypt that encrypts arbitrary cookie values such that they will be correctly decrypted by the web application. Your program does not receive the web application's encryption key as input. It will be called with a single command line argument containing a printable ASCII plaintext to be encrypted. Your program should generate the appropriate cookie values, send them to the web server using HTTP, and observe its response codes in order to encrypt the given value. It should print the resulting Base64-encoded ciphertext to stdout. You can print debug output to stderr if you like. Your program should run in a maximum of 10 minutes.

Submission is similar to the previous question. Your program source files should be included in the same src.tar. For this question, we will attempt to run an executable at the top level with ./encrypt ¡plaintext¿.

Hint: In CBC mode there are three stages a block goes through in decryption. It starts as a ciphertext, the ciphertext is decrypted by the block cipher to an intermediary value, and then the intermediary value is XORed with the previous ciphertext block (or the IV for the first block) to form the final plaintext value. If we know the intermediary value for a given ciphertext block, we can manipulate the IV to gain complete control over what that block will be decrypted to. If we want to produce a plaintext x, then the IV should be x XORed with the intermediary value. This idea can easily be extended to multi-block messages. Start from the last block and move backwards. Pick any value to be the ciphertext for the last block and decrypt it (using your method from the previous question) to find the corresponding intermediary value. Then calculate the IV required to produce the desired plaintext. This IV will be the ciphertext for the second-last block, and so on.

For the next questions, if the script is called without any parameters, it should return the encryption algorithm that is used in that script with details including key length.

6. [5 marks] Discuss the susceptibility of EBC mode to padding oracle attack, and whether or not it can be used in this webserver.

For the programming questions, make sure your scripts do not generate any extra output (e.g. debugging output, compiling output, etc.)

Remarks on Required Readings

Here are a couple of hints that should help in understanding the paper:

- The author frequently uses the term "word" where we would usually say "byte"
- The author denotes the block cipher encryption (see Fig. 1) by C, i.e., $C(x)$ computes the encryption of the block x. The corresponding decryption function is consequently called C^{-1}.
- In Section 3.1. of the paper, 1 is the most likely correct padding of a random string because only the last byte has to be guessed correctly. In contrast, to have correct paddings of the form 22, 333, 4444, and so on, multiple bytes have to be correct.
- Step 5 of the algorithm in Section 3.1 is one way to check if a certain word is part of the padding or not. If changing the word r_n (by XOR-ing 1) affects the validity of the padding (i.e., turns it from valid into invalid), the n-th word is part of the padding, otherwise it is not.

HTTP with curl

HTTP (Hypertext Transfer Protocol) is a simple protocol for transferring text over a network. If you've used the Internet, you've used HTTP. In this protocol, a client makes a request to a server and the server replies with a response. A request generally asks for a resource located at a particular URL, and the response provides that resource. These resources are often HTML web pages, but here we'll be using mostly plain textual content. Let's see an example of a request and a response:

```
"$ curl -v http://localhost:4555
¿ GET / HTTP/1.1
¿ Host: localhost:4555
¿ User-Agent: curl/7.58.0
¿ Accept: */*
¿
¡ HTTP/1.1 200 OK
¡ Content-Length: 37
¡ Set-Cookie: user="7pEU3mxW4kuS1cGflOnc8DN6c4w8T/e8eMDUILO9sznE79EkYX9oEDSXev
8r5XcWqAqMuGZGh9qGF95w3w+iCw=="
¡ Server: TornadoServer/5.1.1
¡ Etag: "4b44c375286c4a668502645e4da51dd29441e4e9"
¡ Date: Tue, 05 Mar 2019 21:59:14 GMT
¡ Content-Type: text/html; charset=UTF-8
¡
Hello! I'll remember when I saw you.
```

Note that some extra debug output from curl has been omitted. The lines starting with "¿" are the client talking to the server, and lines starting with "¡" show communication in the other direction. The HTTP request starts with a method; we'll only be using GET. It then specifies the requested path and the protocol version. The following lines contain headers which specify additional information. These are present in both requests and responses. The response starts by confirming the protocol version. It then presents the status code, which is essential for us as it is the source of our padding oracle. The main response header of interest to us is the Set-Cookie header. We can use the base64 utility to decode it, although it will likely result in unprintable characters, so we'll display the binary data as a hex dump with xxd:

```
"$ echo "7pEU3mxW4kuS1cGflOnc8DN6c4w8T/e8eMDUILO9sznE79EkYX9oEDSXev8r5XcWqAqMu
GZGh9qGF95w3w+iCw=="— base64 -d — xxd
00000000: ee91 14de 6c56 e24b 92d5 c19f 94e9 dcf0  ....lV.K........
00000010: 337a 738c 3c4f f7bc 78c0 d420 b3bd b339  3zs.¡O..x.. ...9
00000020: c4ef d124 617f 6810 3497 7aff 2be5 7716  ..."$a.h.4.z.+.w.
00000030: a80a 8cb8 6646 87da 8617 de70 df0f a20b  ....fF.....p....
```

You can see that this cookie contains four 16-byte blocks. The first is the IV, and the remaining three are ciphertext blocks. To send the cookie back to the server, we can do the following:

```
“$ curl -v -b user=7pEU3mxW4kuS1cGflOnc8DN6c4w8T/e8eMDUILO9sznE79EkYX9oEDSXe
v8r5XcWqAqMuGZGh9qGF95w3w+iCw== http://localhost:4555
¿ GET / HTTP/1.1
¿ Host: localhost:4555
¿ User-Agent: curl/7.58.0
¿ Accept: */*
¿ Cookie: user=7pEU3mxW4kuS1cGflOnc8DN6c4w8T/e8eMDUILO9sznE79EkYX9oEDSXev8r5
XcWqAqMuGZGh9qGF95w3w+iCw==
¿
¡ HTTP/1.1 200 OK
¡ Date: Tue, 05 Mar 2019 22:02:05 GMT
¡ Content-Length: 47
¡ Etag: ”1f1f7c2cd946ee2d654c2ea63973520d37d5d65b”
¡ Content-Type: text/html; charset=UTF-8
¡ Server: TornadoServer/5.1.1
¡
Hello! I first saw you at: 2019-03-05 16:59:14
```

It would be best to use an HTTP library for your programming language when writing your solutions, but you can call the `curl` program directly as a last resort. `curl` is also very useful for manually debugging web applications.

Ugster availability

Some of the aforementioned programming languages and libraries will be made available on the Ugsters in case you do not have access to a personal development computer. We might be able to install additional languages if required.

What to hand in

All assignment submission takes place on the student.cs machines (not ugster or the virtual environments), using the submit facility. In particular, log in to the Linux student environment (linux.student.cs.uwaterloo.ca), go to the directory that contains your solution, and submit using the following command: submit cs458 3 . (dot included). CS 658 students should also use this command and ignore the warning message. Hand in the following files:

a3.pdf: A PDF file containing your answers for all written questions and programming questions when requested. It must contain, at the top of the first page, your name, UW userid, and student number. **-3 marks if it doesn't!** Be sure to "embed all fonts" into your PDF file. Some students' files were unreadable in the past; if we can't read it, we can't mark it.

key.asc, j524chen-signed.asc, message.asc: For the second question of the written part of the assignment.

src.tar: Your source files for your attacks, in your supported language of choice, inside a tarball. To create the tarball, cd to the directory containing your code, and run the command

tar cvf src.tar .

(including the .). If you are using an interpreted language, your source should include executable scripts named p1_attack and p2_attack that runs your code using the proper shebang. For compiled languages, include a Makefile in your source with a default target that builds executables named p1_attack and p2_attack.

prg.tar: A tar archive directly containing your decrypt, encrypt program source files. (Please notice that these programs need to be executables, and without any extensions, e.g., decrypt.sh **is wrong** and should be sent as decrypt.