Homework 2: EncryptedIM

COSC435 - Homework 2

This homework is worth 100 points in total, and contains two parts (written questions, whose answers should be submitted as a PDF; and an encrypted IM program). As explained below, all parts are due on the same day/time, but should be submitted separately to **Autolab** (https://autolab.georgetown.edu).

Assigned Sept 12, 2019; last updated ~~Sept~~ ~~25~~ ~~30~~ October 7th, 2019; **due ~~October 8th~~ October 10th at 11:59pm**.

Part One: Written questions [25 points]

1. **[10 points]** A cryptosystem that offers ***perfect secrecy*** (https://en.wikipedia.org/wiki/Information-theoretic_security) prevents an eavesdropper who observes an encrypted transmission from learning anything about the plaintext, other than its size.

Show with a counterexample that the Shift Cipher doesn't provide perfect secrecy.

2. **[10 points]** Consider the following modification to one-time pad (OTP) encryption. Suppose that in an attempt to use a one-time pad more than once, Prof. Pedantic, the esteemed Ineptitude Professor of Computer Science and Quackery at Wikipedia University, conjures up the following scheme:

Let P be a one-time pad shared by Alice and Bob, and further suppose that Alice and Bob only want to communicate messages of $|P|$ bits, where $|P|$ is the number of bits in P. Under Prof. Pedantic's scheme, Alice can send any number of messages $M_1, M_2, \ldots, M_n$ via:

$$A \longrightarrow B : IV_i, M_i \oplus (P \oplus IV_i)$$

where IV_i is a fresh IV for message M_i.

Explain the flaw in Professor Pedantic's scheme. That is, why is the scheme vulnerable to ciphertext only attacks if Alice uses the one-time pad P more than once?

3. **[5 points]** Prof. Pedantic, the esteemed Ineptitude Professor of Computer Science and Quackery at Wikipedia University, is developing a new terminal program (and associated service) to log into the servers in his lab. Although he is aware of ssh, he refuses to use it. Instead, he decides to construct his own novel protocol. Like telnet and ssh, his remote console/terminal program should allow a remote user to type commands and execute them on a remote machine. Since Prof. Pedantic doesn't trust anyone —

particularly the students in his introduction to network security class — he decides that all communication should be encrypted.

Prof. Pedantic decides to use the AES encryption algorithm in ECB mode. Is this a good choice? Give two reasons why or why not.

Please submit your solutions to the above problems as a PDF file (no Microsoft Word!) to **Autolab (https://autolab.georgetown.edu)** .

Part Two: A Simple, Encrypted P2P Instant Messenger [75 points]

As promised, you will be extending your earlier unencrypted messaging application (or the one provided by the teaching staff) with encryption! We'll call this new program **EncryptedIM**.

Your program should encrypt messages using AES-256 in CBC mode, and use HMAC with SHA-256 for message authentication. IVs should be generated randomly.

Your program must use an encrypt-then-MAC scheme.

You should revise (and rename) your server and client programs to encryptedIMserver.py and encryptedIMclient.py, respectively.

The two programs should be invoked as:

```
python3 encryptedIMclient.py -p port -s servername -n nickname -c confidentialitykey -a authenti
citykey
```

```
python3 encryptedIMserver.py -p port
```

Note that the server should **not** be provided with either the confidentialitykey or the authenticitykey.

For the client, the argument to *-c* specifies the confidentiality key used for AES-256-CBC encryption, and *-a* option specifies the authenticity key used to compute the SHA-256-based HMAC.

To force the keys to be 256 bits long, you must use the SHA-256 hash function on the arguments passed to *-c* and *-a*.

As an example, you may run

python3 encryptedIMclient.py -s turing -n jane -c 435isawesome -a iluvhomework -p 9999

Note that as with the first homework, the server must be started before any client connects.

Also, unlike **Homework 1 (https://georgetown.instructure.com/courses/82306/pages/homework-1-basicim)** , we are adding in an explicit port option (via the -p flag). This avoids some of the "OS: Address already in use" errors that can occur during autograding.

As with **Homework 1, (https://georgetown.instructure.com/courses/82306/pages/homework-1-basicim)** you should use Google Protocol Buffers to serialize the messages sent between clients. Your .proto definition should include, for example, the IV being sent. Note that you should randomly generate IVs for each message sent.

Additional requirements, notes, and hints:

- This part of the assignment will be autograded. You will submit this, and all other parts of this assignment, via **Autolab (https://autolab.georgetown.edu/)** .
- **You should use Python version 3.7** (which is installed in Google Cloud). Don't use Python 2.7x -- it will be deprecated **very soon (https://pythonclock.org/)** .
- Your program should verify that the HMAC is correct. If it is not, it should not print the received message and instead print an error message (e.g., "received message that could not be authenticated!"). You should test that authentication is working properly by specifying different authentication keys on two client instances. This should produce your error message.
- If you are using Python, we recommend the built-in **hashlib (https://docs.python.org/3/library/hashlib.html)** for SHA-256 hashes.
- You may use any crypto library you like. For Python, we recommend **PyCrypto (https://pypi.org/project/pycrypto/)** .
- You may not collaborate on this homework. This project should be done individually.
- Your program should not take in any additional command-line options other those described above. The *-p*, *-c* and *-a* arguments are mandatory; they are not optional.
- Your program can terminate either when the user presses CTRL-C, or when end-of-file (EOF) is received, or when a client types "exit" (followed by the ENTER key). To generate EOF from the terminal, press CTRL-D.

Program naming conventions and other fun requirements

Your program will consist of multiple source files. **You must include with your submission the .proto Google Protocol Buffers definition.** However, you should also include the "compiled" Python classes **that were produced from the Google Protocol Buffers compiler (https://developers.google.com/protocol-buffers/docs/pythontutorial#compiling-your-protocol-buffers)** .

Because your program will be grader by an autograder, you need to be extra careful that you name your programs correctly. (Your supporting Python files can be named whatever you like.) Specifically,

- the server should be named encryptedIMserver.py

- the client should be named encryptedIMclient.py

These are case sensitive!

Computing resources

You may develop your code wherever you see fit (e.g., on **Google Cloud (https://georgetown.instructure.com/courses/82306/pages/creating-your-servers)** , your personal laptop, a computer in a lab, etc.). However, your code must operate correctly in the autograder, which itself is based on an Debian-based Linux distribution (as is, non-coincidentally, your Google Cloud instances). It is strongly recommended that you develop and test (especially the latter) your code on a virtual machine (VM) that you can create on Google Cloud.

Grading

EncryptedIM is worth 75 points. A **non-comprehensive** list of deductions is provided in the rubric below.

Description	**Deduction**
Compilation / interpreter errors	50
Compiles, but IMs are neither successfully transmitted nor received	50
Compiles, but IMs are either only transmitted or only received	50
Received messages only appear after user presses [ENTER] (indicates that *select* is used improperly)	15
General instability	10
Run-time error (e.g., crash) on large input	10
Any other run-time error (e.g., crash)	5
IMs are not encrypted	25
IMs are encrypted, but not successfully decrypted	12
	15

Lack of HMACs	
Lack of HMAC verification	10
Failure to abide by Encrypt-then-MAC	5
Incorrect HMAC verification	7

Note that this grading rubric is not intended to be comprehensive.

Submission Instructions

You will use the **Autolab** **(https://autolab.georgetown.edu)** system for submission.

IMPORTANT NOTE: There are three separate "assignments" listed in Autolab for this homework:

- hw2-questions: upload a PDF containing the answers to Part One of this assignment. This portion of the assignment will be manually graded.
- hw2-ungraded: upload a .zip file containing your source code for EncryptedIM; files must be in the top-level directory of the zip file. You have unlimited submissions for this ungraded Autolab assignment.
- hw2-graded: upload a .zip file containing your source code for EncryptedIM; files must be in the top-level directory of the zip file. You have 3 submissions to hw2-graded; we will grade the last submission.

Late assignments will be penalized according to the **Grading Policy** **(https://georgetown.instructure.com/courses/82306/pages/policy-grading)** .

Please post questions (especially requests for clarification) about this homework to **Piazza** **(https://piazza.com/georgetown/fall2018/cosc435/home)** .