CS114 - Homework 1, part 2[*]

Due 11:59pm on March 5th, 2023

Prof. Daniel Votipka

1 Written questions {40 points}

(a) {10 points} A cryptosystem that offers *perfect secrecy* prevents an eavesdropper who observes an encrypted transmission from learning anything about the plaintext, other than its size.

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

(b) {10 points} Consider the following modification to one-time pad (OTP) encryption. Rather than share a single one-time pad, Alice and Bob have shared knowledge of two pads, P_1 and P_2.

Given a plaintext M, Alice creates the ciphertext $C = M \oplus P_1 \oplus P_2$, where $\oplus$ denotes xor and $|M| = |P_1| = |P_2|$ (i.e., the size of the message and the two pads are all equal). To decrypt, Bob takes the ciphertext and xors it with P_1 and P_2; i.e., $D(C) = C \oplus P_1 \oplus P_2$.

Argue that if a one-time pad offers perfect secrecy, then the above scheme must also be perfectly secure.

(c) {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 because he doesn't like being hushed.[1] 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.

(d) {15 points} Prof. Pedantic designed a "secure" communication protocol for two parties (Alice and Bob) that have preshared secrets k_1 (the confidentiality key) and k_2 (the authenticity key).

[*]Last revised on January 16, 2024.

[1]Extra credit {0.0000001 points}: Explain that joke.

Prof. Pedantic doesn't believe in traditional MACs, so he constructs his protocol as follows: to send a message m, Alice (A) sends to Bob (B) the following:

$$\begin{aligned} A \to B : \langle \quad & r, \\ & \mathrm{iv}_1, \\ & \mathrm{iv}_2, \\ & \mathrm{S}(k_1, \mathrm{iv}_1) \oplus (m \| r) \\ & \mathrm{S}(k_2, \mathrm{iv}_2) \oplus (m \| r) \quad \rangle \end{aligned}$$

where $\|$ denotes concatenation, r is a nonce (to prevent replay attacks), iv_1 and iv_2 are fresh initialization vectors (IVs), and $\mathrm{S}(k, \mathrm{iv})$ denotes a cryptographically secure pseudorandom sequence based on key k and IV iv (i.e., a stream cipher).

The professor claims that the protocol achieves *confidentiality* and *authenticity*, **as defined as follows**:

- *confidentiality:* an eavesdropper that observes a run of the protocol cannot learn the message m unless it knows the confidentiality key k_1; and
- *authenticity:* if Bob receives $\langle r, \mathrm{iv}_1, \mathrm{iv}_2, \mathrm{S}(k_1, \mathrm{iv}_1) \oplus (m \| r), \mathrm{S}(k_2, \mathrm{iv}_2) \oplus (m \| r) \rangle$ and r is a fresh nonce and the decryption of $\mathrm{S}(k_1, \mathrm{iv}_1) \oplus (m \| r)$ equals the decryption of $\mathrm{S}(k_2, \mathrm{iv}_2) \oplus (m \| r)$ (using the corresponding IVs and keys), then message m must have been transmitted by a party that knows both the confidentiality and authenticity keys (i.e., k_1 and k_2).

The professor's intention is that Bob obtains m by decrypting $\mathrm{S}(k_1, \mathrm{iv}_1) \oplus (m \| r)$ using key k_1 and iv_1. Further, Bob performs an authenticity check by ensuring that the decrypted message matches the decryption of $\mathrm{S}(k_2, \mathrm{iv}_2) \oplus (m \| r)$ (via key k_2 and IV iv_2). He reasons that only a sender that knows *both* k_1 and k_2 can cause the decryptions to match.

Does Prof. Pedantic's scheme achieve confidentiality and/or authenticity, as defined above? Briefly argue why or why not, for both confidentiality and authenticity. Assume that k_1 and k_2 are random 128-bit keys that have been securely shared apriori between Alice and Bob, that $k_1 \neq k_2$, and that the two IVs are also fresh.

2 Eavesdropping on unencryptedim {15 points}

Show that the `unencryptedim.py` program from Part I of Homework 1 is susceptible to eavesdropping.

To do this, you will look at a packet capture I generated while running my version of the program and write down what the password was that I transmitted between the client and server. The pcap can be found at https://www.cs.tufts.edu/comp/114/2024S/hws/hw1p2.pcap

To open the captured pcap file you can use Wireshark, which will help you visualize the stream of packets sent between the server and client. Note that Wireshark is available (for free!) on Linux, Mac OSX, and Windows. Unless you already have it, you will need to install it. Submit the password string that I typed to receive points for this section of the homework.

3 A Simple, Encrypted P2P Instant Messenger {35 points}

As promised, you will be extending your earlier unencrypted messaging application (or the one provided by the teaching staff[2], provided on the course Box) with encryption! We'll call this new program `encryptedim.py`.

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.**

Your program should have the following command-line options:

```
    python encryptedim.py [--s|--c hostname] [--confkey K1] [--authkey K2]
```

where the `--s` argument indicates that the program should wait for an incoming TCP/IP connection on port 9999; the `--c` argument (with its required `hostname` parameter) indicates that the program should connect to the machine `hostname` (over TCP/IP on port 9999). `--confkey` specifies the confidentiality key (`K1`) used for encryption, and `--authkey` specifies the authenticity key (`K2`) used to compute the HMAC.

To force the keys to be 256 bits long, you may use the SHA-256 hash function on the `K1` and `K2` parameters.

In order for your program to communicate correctly with the autograder, we need to standardize the sending protocol so it works with the autograder. You should begin each communication should have the following form:

$iv + E_{k1}(len(m)) + \text{HMAC}_{k2}(iv + E_{k1}(len(m))) + E_{k1}(m) + \text{HMAC}_{k2}(E_{k1}(m))$

[2]**Important note:** For the entirety of this homework, you may use the TAs'/instructor's solution to homework 1, part 1 rather than your own, if you prefer.

This means you'll begin by sending the IV in the clear, followed by the length of the message (encrypted-the-MACd), then finally the message (encrypted-then-MACd). You need to send the length of the message first so that you can handle an arbitrary-length message.

As an example, you may run "`python3 encryptedIM.py --s --confkey 'FOOBAR' --authkey 'CS114ISAWESOME'`" in one terminal window, and then start "`python3 encryptedim.py --c 127.0.0.1 --confkey 'FOOBAR' --authkey 'CS114ISAWESOME'`" in another terminal window. Note that the instance with the `--s` option must be started before the other instance.

Additional requirements and hints. Please make sure that your program conforms to the following:

- Your program should verify that the HMAC is correct. If it is not, it should print the error message "`ERROR: HMAC verification failed`" to stdout and exit. You should test that authentication is working properly by specifying *different* authentication keys for your client and server instance: this should produce your error message and cause the program to exit!
- You may use only the pycryptodome library for encryption and authentication. **DO NO CREATE YOUR OWN AES OR SHA-256 IMPLEMENTATIONS.**
- You may not collaborate on this homework. This project should be done individually. You may use, with proper citations as code comments, code and documentation from `https://pycryptodome.readthedocs.io/en/latest/`.
- As with the first assignment, to aid in automated testing/grading, do not provide a prompt to the user, and only write received messages to standard out. We may be using automated testing tools to evaluate your solutions, and printing additional messages or characters makes such automation far more difficult.
- Your program should not take in any additional command-line options other those described above. The `--confkey` and `--authkey` 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. To generate EOF from the terminal, press CTRL-D.
- You can test whether your program is actually encrypting data by using *tcpdump* to capture traffic on your loopback address using the command **tcpdump -i** <**loopback interface**> **-w** <**output file**>. This will listen on the loopback address and create a pcap file at the give file address that you can open and view with Wireshark. To determine what your loopback interface is, use the command **tcpdump -D**.

Grading

This portion of HW1 is worth 90 points (40 points for question 1; 15 points for question 2; and 35 points for the programming assignment). A non-comprehensive list of deductions for the programming portion of this assignment is provided in Table 1.

We will award partial credit when possible and appropriate. To maximize opportunities for partial credit, please rigorously comment your code. If we cannot understand what you intended, we cannot award partial credit.

Tests	**Scoring**
IMs are successfully transmitted	+5
IMs are successfully received	+5
Client and receiver can successfully communicate back and forth	+3
Passes complex input tests	+2
IMs are encrypted	+5
IMs are correctly encrypted	+10
HMAC is included	+5
HMAC is validated correctly	+10
Deductions	
Compilation / interpreter errors	-30
Received messages only appear after user presses [ENTER] (indicates that *select* is used improperly)	-10
Non-conformant command-line options (hinders automated testing)	-5
Includes unnecessary prompts (hinders automated testing)	-3

Table 1: Grading rubric. Note that this grading rubric is not intended to be comprehensive.

Submission Instructions

This submission will be made to in two files to Gradescope.

The first file will be a single PDF or ASCII text document with your written answers to Question 1, as well as the password identified through the eavesdropping exercise (Question 2). **Writeups submitted in Word, PowerPoint, Corel, RTF, Pages, and other non-PDF or ASCII formats will not be accepted.** Consider using LaTeX to format your homework solutions. (For a good primer on LaTeX, see the Not So Short Introduction to LATEX.) This archive should also include the protocol description from the programming assignment.

The second file will include the programming component of the assignment. Your submission **must** be called `encryptedim.py`.

Upload both files—pdf/text written answers and `encryptedim.py` to Gradescope.–before 11:59pm on March 5th. Please post questions (especially requests for clarification) about this homework to

Piazza.