CIS 4360- Homework 1, part 2[*]

Assigned September 11, 2019; Due 11:59pm on September 20, 2019

Prof. Kevin Butler

1 Written questions {35 points}

(a) {5 points} Prof. Bumblemore, the esteemed Ineptitude Professor of Computer Science and Quackery at Warthogs 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. 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. Bumblemore doesn't trust anyone — particularly the students in his computer security class — he decides that all communication should be encrypted.

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

(b) {15 points} Prof. Bumblemore 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).

Prof. Bumblemore 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:

$$\begin{aligned} A \rightarrow B : \langle \quad & r, \\ & \mathrm{iv}_1, \\ & \mathrm{iv}_2, \\ & \mathrm{RC4}_{H(\mathrm{iv}_1|k_1)}(r, m), \\ & \mathrm{RC4}_{H(\mathrm{iv}_2|k_2)}(r, m) \quad \rangle \end{aligned}$$

where r is a nonce (to prevent replay attacks), iv_1 and iv_2 are fresh initialization vectors (IVs), $\mathrm{RC4}_k(r, m)$ denotes the encryption of message m using RC4 (a stream cipher) with key k and nonce r, and $H(x|y)$ is the SHA-256 hash of x concatenated with y. (Note that RC4 does not natively accept an IV; hence, Prof. Bumblemore embeds the IV into the effective encryption/decryption key using the hash function.)

[*]Last revised on September 11, 2019.

The professor claims that the protocol achieves *confidentiality* and *authenticity*, **as defined in the following way**:

- *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 $\langle r, \mathrm{iv}_1, \mathrm{iv}_2, \mathrm{RC4}_{H(\mathrm{iv}_1|k_1)}(r, m), \mathrm{RC4}_{H(\mathrm{iv}_2|k_2)}(r, m)\rangle$ and r is a fresh nonce and the decryption of $\mathrm{RC4}_{H(\mathrm{iv}_1|k_1)}(r, m)$ equals the decryption of $\mathrm{RC4}_{H(\mathrm{iv}_2|k_2)}(r, m)$ (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 $\mathrm{RC4}_{H(\mathrm{iv}_1|k_1)}(r, m)$ using key k_1 and iv_1. Further, Bob performs an authenticity check by ensuring that the decrypted message matches the decryption of $\mathrm{RC4}_{H(\mathrm{iv}_2|k_2)}(r, m)$ (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. Bumblemore '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 *a priori* between Alice and Bob, that $k_1 \neq k_2$, and that the two IVs are also fresh.

2 Eavesdropping on Yourself {15 points}

Show that the UnencryptedIM program you wrote[1] for Part I of Homework 1 is susceptible to eavesdropping.

You can generate a trace of your own program if you developed the code on your local machine (or in a VM). Do this by using `tcpdump` to conduct a packet capture. If it's not already on your system, do `apt-get install tcpdump` or equivalent (depending on your flavor of Linux/Mac and corresponding package manager) to install it. Note that you will need to use root (admin) privileges to perform a packet capture, so when running `tcpdump`, you'll want to preface the command with `sudo` to run as root. You should also set the "snaplength" to 0 to capture packets in their entirety, and you'll want to save the capture to a file (see tcpdump's `-w` option).

Hint: The manual page for tcpdump is your friend. You can access it by typing `man tcpdump` on the Linux shell.

Note: If you don't have access to your own machine (or virtual machine) for running `tcpdump`, I will make a trace file available on Monday that you can analyze with Wireshark.

Then, open the captured pcap file with Wireshark, and take a screenshot that shows that an adversary can clearly see the plaintext messages as they traverse the network. Wireshark is available on the lab machines and is also freely available for Linux, MacOS, and Windows (you can use a package manager as described above, or download it from `wireshark.org`). Unless you already have it, you will need to install it. Submit your screenshot with this homework as evidence that an adversary can discern the plaintext IM messages.

Note that you do not need to write up anything for this question; just submit the screenshot, which should include the date and time.

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

As promised, you will be extending your earlier unencrypted messaging application with encryption! We'll call this new program `EncryptedIM`.

Your program should encrypt messages using AES-128 in CBC mode, and use HMAC with SHA-256 for message authentication. IVs should be generated randomly.

Your program should have the following command-line options:

```
     EncryptedIM [-s <portnum> | -c hostname <portnum>] [-confkey K1]
[-authkey K2]
```

where the `-s` argument indicates that the program should wait for an incoming TCP/IP connec-

[1] **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.

tion on port 9999 (or optionally the port number specified by `portnum`); the `-c` argument (with its required `hostname` parameter) indicates that the program should connect to the machine `hostname` (over TCP/IP on port 9999, or the port number specified by `portnum`). `-confkey` specifies the confidentiality key (`K1`) used for encryption, and `-authkey` specifies the authenticity key (`K2`) used to compute the HMAC. You should use SHA-256 to hash keys `K1` and `K2` to ensure that they are of a constant size. You should take the first 128 bits of the two 256-bit hashes as your respective keys.

For example, you may run "`EncryptedIM -s 9001 -confkey FOOBAR -authkey CIS4360ISAWESOME`" on some machine, e.g., `lin113-01`, and then start "`EncryptedIM -c lin113-01 9001 -confkey FOOBAR -authkey CIS4360ISAWESOME`" on some other machine, e.g., `lin113-02`. Note that the "server" instance with the `-s` option must be started before the other instance.

Along with your code, you must submit a **brief** protocol document in plain ASCII (no MS Word please!) that describes the format of your messages. In particular, the document should describe how/where the IV is transmitted, and the locations of the ciphertext and HMAC in the messages.

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

- You may write your program in C or Python. For submissions done in C, we will ignore all submitted executables and will compile your code from the submitted source files.
- Your program should verify that the HMAC is correct. If it is not, it should exit with an error message. You should test that authentication is working properly by specifying *different* authentication keys on your client and server machines; this should produce your error message and cause the program to exit!
- Please use the libraries already installed on the lab machines. These will allow you to use openssl libraries and the Python Crypto libraries should also be installed. Please post to Canvas if you have difficulties with these.
- You may not collaborate on this homework. This project should be done individually. You may search the Internet for help, but you may not copy (either via copy-and-paste or manual typing) code from another source. You **may** use code from the textbook, or from the instructor or TAs.
- As with the first assignment, to aid in testing/grading, do not provide a prompt to the user, and only write received messages to standard out. We will (hopefully) 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.

Grading

This portion of HW1 is worth 70 points (35 parts for written/analysis questions; 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.

Description	**Deduction**
Only included executables (no source code; applies to C/C++)	35
Compilation / interpreter errors	20
Compiles, but IMs are neither successfully transmitted nor received	17
Communication only works in one direction	13
IMs are not encrypted	25
IMs are encrypted, but not successfully decrypted	12
Lack of HMACs	15
Lack of HMAC verification	10
Incorrect HMAC verification	7
Received messages only appear after user presses [ENTER] (indicates that *select* is used improperly)	10
General instability (e.g., occasional segfaults)	6
Run-time error (e.g., crash) on large input	5
Non-conformant command-line options (hinders automated testing)	5
No compilation instructions provided (applies to C/C++)	5

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

Submission Instructions

Submit your solution as a single tarball (tar.gz archive) through Canvas.

In the archive, include a single PDF or ASCII text document with your written answers to Question 1. **Writeups submitted in Word, PowerPoint, Corel, RTF, Pages, and other non-PDF or ASCII formats will be irritating and possibly unreadable.** Consider using LaTeX to format your homework solutions. (For a good primer on LaTeX, see the Not So Short Introduction to LATEX.)

Include in the archive the written responses, all source code, and the protocol description. If your program is written in C, please also provide compilation instructions.

Please post questions (especially requests for clarification) about this homework to Canvas.