1 Written questions {35 points}

(a) {10 points} Show with a counterexample that the Permutation Cipher doesn’t achieve 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 the above scheme is perfectly secure.

(c) {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). 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:

$$A \rightarrow B : \langle r, iv, RC4_{H(iv|k_1)}(m), H(m|k_2) \rangle$$

where $r$ is a nonce (to prevent replay attacks), $iv$ is a fresh initialization vector (IV), $RC4_k(m)$ denotes the encryption of message $m$ using RC4 (a stream cipher) with key $k$, 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. Pedantic embeds the IV into the effective encryption/decryption key using the hash function.)

The professor’s intention is that Bob obtains $m$ by decrypting $RC4_{H(iv|k_1)}(m)$ using key $k_1$ and $iv$. Further, Bob performs an authenticity check by ensuring that the hash of the decrypted message concatenated with $k_2$ is correct.

Describe a weakness of Prof. Pedantic’s scheme.

*Last revised on January 12, 2015.*
2 A Simple, Unencrypted P2P Instant Messenger {30 points}

To introduce you to network programming, you will build a simple unencrypted instant messenger. The program reads from standard input and sends input messages to another instance of the program running on a different machine; received messages are sent to standard output.

<table>
<thead>
<tr>
<th>ms2382-alice-HW1</th>
<th>ms2382-bob-HW1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Today was good</td>
<td>Today was good</td>
</tr>
<tr>
<td>Today was fun</td>
<td>Today was fun</td>
</tr>
<tr>
<td>Tomorrow is another one.</td>
<td>Tomorrow is another one.</td>
</tr>
<tr>
<td>From there to here</td>
<td>From here to there</td>
</tr>
<tr>
<td>From here to there</td>
<td>From here to there</td>
</tr>
<tr>
<td>Funny things are everywhere.</td>
<td>Funny things are everywhere.</td>
</tr>
</tbody>
</table>

Figure 1: An example conversation between two nodes, as seen by each of the two nodes. Red text is entered by the user at machine ms2382-alice-HW1 and blue text is entered by the user at ms2382-bob-HW1.

Figure 1 shows a conversation between instances of the IM client running on two machines, ms2382-alice-HW1 and ms2382-bob-HW1.

Your IM client should use TCP to send messages between hosts. Your program should run between machines netid-alice-HW1 and netid-bob-HW1, where netid is your Georgetown NetID. That is, you will each have two dedicated machines to use for this project. You should test your code on these two machines. Importantly, we will evaluate your code on these two machines.

**Important:** Your password on netid-alice-HW1 and netid-bob-HW1 is NOT your Georgetown NetID password. Your password for these hosts will be securely disseminated via Google Docs. You will receive instructions on how to log into netid-alice-HW1 and netid-bob-HW1 (including how to retrieve your password) via email and/or postings on Piazza.

**Program description.** Your program should read from standard input and send all input data to the other instance of your application (running on the other host), via TCP/IP over port 9999.

Here’s the tricky bit: Received messages should be immediately written to standard output. To do this, you will need to use the `select` call to block and wait for input either on standard input or the network socket. Descriptions of Python’s variant of `select` are available online.

(As an alternative to using `select`, you may use multiple threads, although this is far less efficient.)

Your program should be called UnencryptedIM.py.
Your program should have the following command-line options:

```
UnencryptedIM.py -s|-c hostname
```

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).

For example, you may run “UnencryptedIM.py -s” on `netid-alice-HW1`, and then start “UnencryptedIM.py -c netid-alice-HW1” on `netid-bob-HW1`. 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:

- You must write your program in Python.
- You may only use libraries already installed on `netid-alice-HW1` and `netid-bob-HW1`.
- 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.
- To aid in automated testing/grading, do not provide a prompt to the user, and only write received messages to standard out. (Text entered into standard input can also be shown; see Figure 1.) We will 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 than the `-s` or `-c hostname` options described above.
- It is OK if messages are only sent after the user presses [ENTER] after entering a line of text. However, incoming messages should be displayed immediately after they are received by the kernel.
- 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.
- Hint: To stress test your code, try using your program to copy a file between machines. You should be able to do this by redirecting standard input (at one end) and standard output (at the other).
3  Eavesdropping on Yourself {15 points}

Show that the UnencryptedIM.py program you wrote is susceptible to eavesdropping.

Do this by using **tcpdump** to conduct a packet capture on **netid-alice-HW1**. You’ll need to use root (admin) privileges to perform a packet capture, so 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.

Then, on your own machine, 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. 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 your screenshot with this homework as evidence that an adversary can discern the plaintext IM messages.

(You do not need to write up anything for this question; just submit the screenshot.)

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

You will be extending your earlier unencrypted messaging application encryption! We’ll call this new program **EncryptedIM.py**.

Your program should encrypt messages using AES-128 in CBC mode, and use HMAC with SHA-1 for message authentication. IVs should be generated randomly.

Your program should have the following command-line options:

```
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.

You can use SHA1 to hash keys K1 and K2 to ensure that they are of a constant size.

For example, you may run “**EncryptedIM.py -s -confkey FOOBAR -authkey COSC235ISAWESOME**” on **netid-alice-HW1**, and then start “**EncryptedIM.py -c netid-alice-HW1 -confkey FOOBAR -authkey COSC235ISAWESOME**” on **netid-bob-HW1**. Note that the 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:

- 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 netid-alice-HW1 and netid-bob-HW1: this should produce your error message and cause the program to exit!

- Your program should not take in any additional command-line options other than those described above. The -confkey and -authkey arguments are mandatory; they are not optional.

5 Instant Messenger, now with Mutual Authentication! {35 points}

For the last part of the assignment, you’ll be adding mutual authentication using RSA signatures. The details of this part of the assignment will be provided shortly.

**Grading**

HW1 is worth 150 points. A non-comprehensive list of deductions 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.

**Submission Instructions**

Submit your solution as a single tarball (tar.gz archive) using Blackboard. To upload your assignment, navigate to the COSC535 course, click the “Assignments” link on the left hand side, and select “hw1”. Look for the “Attach File” section and upload your submission. Be sure to hit the “Submit” button when done. Upload your assignment before 11:59 pm on February 11th.

Include in the archive all source code, using the naming conventions described in this assignment.

Please post questions (especially requests for clarification) about this homework to Piazza.
<table>
<thead>
<tr>
<th>Description</th>
<th>Deduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation / interpreter errors</td>
<td>20</td>
</tr>
<tr>
<td>Compiles, but IMs are neither successfully transmitted nor received</td>
<td>17</td>
</tr>
<tr>
<td>Compiles, but IMs are either only transmitted or only received</td>
<td>14</td>
</tr>
<tr>
<td>Received messages only appear after user presses [ENTER] (indicates that select is used improperly)</td>
<td>10</td>
</tr>
<tr>
<td>General instability (e.g., occasional segfaults)</td>
<td>10</td>
</tr>
<tr>
<td>Run-time error (e.g., crash) on large input</td>
<td>7</td>
</tr>
<tr>
<td>Non-conformant command-line options (hinders automated testing)</td>
<td>5</td>
</tr>
<tr>
<td>Includes unnecessary prompts (hinders automated testing)</td>
<td>3</td>
</tr>
<tr>
<td>Communication only works in one direction</td>
<td>13</td>
</tr>
<tr>
<td>IMs are not encrypted</td>
<td>25</td>
</tr>
<tr>
<td>IMs are encrypted, but not successfully decrypted</td>
<td>12</td>
</tr>
<tr>
<td>Lack of HMACs</td>
<td>15</td>
</tr>
<tr>
<td>Lack of HMAC verification</td>
<td>10</td>
</tr>
<tr>
<td>Incorrect HMAC verification</td>
<td>7</td>
</tr>
</tbody>
</table>

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