1 Key Sharing, The Pedantic Way \{15 points\}

At a recent conference, Prof. Pedantic met a potential collaborator, Prof. Feckless. Over drinks, Prof. Pedantic and Feckless outlined a new super-secret research project that they would collaborate on throughout the year. Due to the nature of the work, both professors agreed that any future email between the two parties should be encrypted.

(a) \{7 points\} Suppose that during their encounter, Prof. Pedantic and Feckless securely exchanged a random, 16 bit key, $k_{16}$. Later, back at their respective institutions, they realize that 16 bits is too small. They decide to use the short key to communicate a longer secret, chosen by Prof. Pedantic, as follows:

Prof. Pedantic $\rightarrow$ Feckless : $E_{k_{16}}(k_{256}, MAC_{k_{16}}(k_{256}))$

They then communicate using the 256 bit key $k_{256}$ as follows:

Prof. Pedantic $\leftrightarrow$ Feckless : $E_{k_{256}}(M, MAC_{k_{256}}(M))$

What is the flaw in the two professors’ logic?

(b) \{8 points\} Suppose that the two professors each share a (separate) key with a trusted mutual friend, Dean Bureaucracy. With Dean B’s help, can they now securely exchange a key such that an external eavesdropper (i.e., anyone who is not the professors or the Dean) cannot learn it? If so, how? If not, why not? You can assume that Dean B is honest.
A Less Simple, Encrypted P2P Instant Messenger \{40 points\}

For this programming assignment, you will modify your (or our) encrypted P2P instant messenger from HW1, Part II.

Rather than input confidentiality and authenticity keys on the command-line, the IM clients will use the Diffie Hellman (DH) protocol to agree on a single ephemeral confidentiality key, which will be used for the duration of the exchange. Unlike HW1, Part II, we will not be using a MAC.

As before, your program should encrypt messages using AES-128 (or AES-256) in CBC mode. IVs should be generated randomly.

To obtain the correct size confidentiality key, take a hash of the key shared via DH and use the appropriate number of bits from that hash as your AES key.

Your program should have the following command-line options:

\[
\text{DHEncryptedIM } [-s|-c \text{ hostname}] 
\]

where the \text{-s} argument indicates that the program should wait for an incoming TCP/IP connection on port 9999; the \text{-c} argument (with its required \text{hostname} parameter) indicates that the program should connect to the machine \text{hostname} (over TCP/IP on port 9999). Note that no keys should be provided on the command-line; the confidentiality key should be negotiated through a DH exchange.

For example, you may run “DHEncryptedIM \text{-s}” on \text{netid-alice-HW1}, and then start “EncryptedIM \text{-c netid-alice-HW1}” on \text{netid-bob-HW1}. Note that the instance with the \text{-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 hardcode the DH parameters. Set \( g = 2 \) — that is, the generator should be two. Use the following 1024-bit prime:

```c
static unsigned char dh1024_p[] = {
    0xCC,0x81,0xEA,0x81,0x57,0x35,0x2A,0x9E,0x9A,0x31,0x8A,0xAC,
    0x4E,0x33,0xFF,0xBA,0x80,0xFC,0x8D,0xA3,0x37,0x3F,0xB4,0x48,
    0x95,0x10,0x9E,0x4C,0x3F,0xF6,0xCE,0xDC,0xC5,0x5C,0x02,0x22,
    0x8F,0xCC,0xBD,0x55,0x1A,0x50,0x4F,0xEB,0x43,0x46,0xD2,0xAE,
    0xF4,0x70,0x53,0x31,0x1C,0xEA,0xBA,0x95,0xF6,0xC5,0x40,0xB9,
    0x67,0xB9,0x40,0x9E,0x9F,0x05,0x02,0xE5,0x98,0xCF,0xC7,0x13,
    0x27,0xC5,0xA4,0x55,0x8E,0x80,0xBE,0x1E,0x0B,0x7D,
    0x23,0xFB,0xEA,0x05,0x4B,0x95,0x1C,0xA9,0x64,0xEA,0xEC,0xAE,
    0x7B,0xA8,0x42,0xBA,0xA,0x81,0x8C,0x45,0x3B,0xF1,0x9E,
    0xB9,0xC5,0xC8,0x6E,0x72,0x3E,0x69,0xA2,0x10,0xD4,0xB7,0x25,
    0x61,0xCA,0xB9,0x7B,0x3F,0x3B,0x60,0xB
};
```

Without the C notation, this number in hex is

```
0x00cc81ea8157352a9e9a318aac4e33
ffba80fc8da373fb44895109e4c3f
f6cedcc55c02228fccbd551a504feb
4346d2aeef705331ceaba95f6c540
b967b9409e9f0502e98cf71327c5
a455e2e807bede1e0b7d23fbea054b
951ca964eaecea7ba842ba1fc6818c
453bf19eb9c5c86e723e69a210d4b7
2561cab97b3fb3060b
```

- Normal, unoptimized exponentiation (i.e., computing \( x^y \) by multiplying \( x \) by itself \( y \) times) is too slow when dealing with large numbers. You’ll want to use an optimized exponentiation technique, which you can read about at [https://en.wikipedia.org/wiki/Exponentiation_by_squaring](https://en.wikipedia.org/wiki/Exponentiation_by_squaring).

- You must use a good, cryptographic source of randomness for the DH secrets (lowercase “a” and “b” in the class notes). Do not use Python’s `random.random` or C’s `rand()` functions. OpenSSL and PyCrypto have secure random number generators. Use them.

- If you are using C or C++, you will need to use a special library to handle large numbers: libgmp ([https://gmplib.org/](https://gmplib.org/)). You’ll want to use libgmp’s exponentiation functions. Python natively supports large numbers.
You may write your program in C, C++, Python, Ruby, Java, or Perl. Please see the teaching staff if you would like to use another programming language. For submissions done in C/C++/Java, we will ignore all submitted executables (or byte code) and will compile your code from the submitted source files.

You may only use libraries already installed on netid-alice-HW1 and netid-bob-HW1. Please post requests for additional libraries to Piazza.

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. For this homework only, as an exception to this rule, you may copy code for efficient exponentiation; however, you must properly attribute that code (e.g., by citing a URL or some other source) as comments in your source code. You may also use code from the textbook, or from the instructor or TAs.

As with the first two assignments, to aid in automated testing/grading, do not provide a prompt to the user, and only write received messages to standard out. 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 those described above.

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.

3 Man-in-the-Middle {15 points}

Briefly describe how an adversary, Eve, who is positioned between Alice and Bob, can perform an active attack against the above DH-enabled encrypted IM program to learn the contents of all messages exchanged between the two honest parties.
Grading

This portion of HW1 is worth 70 points. 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.

<table>
<thead>
<tr>
<th>Description</th>
<th>Deduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only included executables (no source code; applies to C/C++ and Java)</td>
<td>40</td>
</tr>
<tr>
<td>Compilation / interpreter errors</td>
<td>25</td>
</tr>
<tr>
<td>Non-functioning DH</td>
<td>17</td>
</tr>
<tr>
<td>Use of unsecure randomness (e.g., rand() or random() class functions)</td>
<td>5</td>
</tr>
<tr>
<td>Crashes on invalid command-line arguments</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Grading rubric. Note that this grading rubric is not intended to be comprehensive. Deductions listed in parts I and II of HW1 are not included here (for brevity); however, they still apply.

Submission Instructions

Submit your solution as a single tarball (tar.gz archive) using Blackboard. To upload your assignment, navigate to the COSC235 course, click the “Assignments” link on the left hand side, and select “hw1-part3”. Look for the “Attach File” section and upload your submission. Be sure to hit the “Submit” button when done. Upload your assignment before 11:59pm on September 27th.

In the archive, include a single PDF or ASCII text document with your written answers. 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.)

Don’t forget to answer Question 3!

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

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