CIS 4360- Homework 1, part 3[*]

Assigned October 1st, 2019; Due 11:59pm on October 14th, 2019

Prof. Kevin Butler

1 RSA {30 points}

(a) {5 points} Prof. Bumblemore, the esteemed Ineptitude Professor of Computer Science and Quackery at Warthogs University, generates an RSA keypair, consisting of a public key $\langle e, n\rangle$ and a private key $\langle d, n\rangle$. He saves the large (2048-bit) prime factors p and q used to compute n (i.e., $n = pq$) as well as his public key $\langle e, n\rangle$. Ever forgetful, he forgets to save his private key $\langle d, n\rangle$. D'oh! Will Prof. Bumblemore be able to recover and re-generate his private key? That is, can he learn d given the information that he recorded? Why or why not?

(b) {10 points} Prof. Bumblemore collaborates with his wife, Frau Professor Doktor Bumblemore, to create a variant of RSA, which he calls RASBB (the BB is for "Bumble and Bumble"). In RASBB, the private key exponent d is computed as $d = e^{-1} \bmod n$ (the modular inverse of the public key exponent e, modulo n), as opposed to $d = e^{-1} \bmod \Phi(n)$. The encryption and decryption functions also differ between RSA and RASBB, but knowing what these functions are isn't important for this question.

What **major** security flaw does RASBB's key generation algorithm introduce?[1]

(c) {5 points} Prof. Bumblemore gives you (securely) his RSA public key:

$$K^{+} = \langle e = 13, n = 77\rangle$$

What is the corresponding ciphertext for the plaintext message $M = 2$, encrypted with Prof. Bumblemore's public key? Show your work.

(d) {5 points} Given his public key $\langle e = 13, n = 77\rangle$, what is Prof. Bumblemore's private key?

(e) {5 points} Why were you able to answer the previous question?!? That is, is RSA broken? If it isn't broken, then why were you able to derive his private key from his public key?

[*]Last revised on October 1, 2019.

[1]Note that the notation $a = b^{-1} \bmod q$ is equivalent to $ab \bmod q = 1$. Both reflect the fact that a and b are modular inverses under modulo q.

2 Key Sharing $\{$15 points$\}$

At a recent conference after a falling out with Frau Bumblemore, Prof. Bumblemore met a potential new collaborator, Prof. Feckless. Over drinks, Prof. Bumblemore 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. Bumblemore 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. Bumblemore, as follows:

$$\text{Prof. Bumblemore} \rightarrow \text{Feckless} : E_{k_{16}}(k_{256}, \text{MAC}_{k_{16}}(k_{256}))$$

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

$$\text{Prof. Bumblemore} \leftrightarrow \text{Feckless} : E_{k_{256}}(M, \text{MAC}_{k_{256}}(M))$$

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.

3 *PayMe!* $\{$10 points$\}$

Hoping to become the next dotcom millionaire, Prof. Bumblemore decides to create an online money payment service similar to PayPal. His service, *PayMe!*, allows users to transfer money to other users of the system.

To ensure that no fraudulent activity takes places, the *PayMe!* service stores the public key of each user. (You should assume that the sharing of the public key is secure; that is, the server has each user's correct public key.)

If Alice ("A") wishes to give X dollars to Bob ("B"), she sends the following message to the *PayMe!* service ("S"):

$$A \rightarrow S : A, B, X, n, \{X|n\}_{A^-}$$

where n is a nonce, A^- is Alice's private key, and $\{M\}_{K^-}$ denotes[2] a digital signature over M computed using the private key K^-.

[2]Yikes! Another notation for signatures! There are several ways to denote the public key signature operation, including these three which appear frequently in the literature: $E_{K^-}(M) = \{M\}_{K^-} = \sigma_{K^-}(M)$. All three notations are equivalent.

(a) {5 points} What is a nonce, and why does Prof. Bumblemore include one in his protocol? Does it prevent any type of attack?

(b) {5 points} Explain how an active adversary can exploit a weakness in Prof. Bumblemore's protocol to steal money from an honest user, Alice.

(c) {5 points} Modify the protocol to fix this problem.

4 Encrypted P2P Instant Messenger with Key Agreement {40 points}

For this programming assignment, you will modify your (or our) encrypted P2P instant messenger from HW1, Part II.[3]

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:

```
DHEncryptedIM [-s <portnum> | -c hostname <portnum>]
```

where the `-s` argument indicates that the program should wait for an incoming TCP/IP connection 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`). 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 -s 9001`" on some machine, e.g., lin113-01, and then start "`DHEncryptedIM -c lin113-01 9001`" on some other machine, e.g., lin113-02. Note that the "server" instance with the `-s` option must be started before the other instance.

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

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:

  ```
  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,
  0x27,0xC5,0xA4,0x55,0xE2,0xE8,0x07,0xBE,0xDE,0x1E,0x0B,0x7D,
  0x23,0xFB,0xEA,0x05,0x4B,0x95,0x1C,0xA9,0x64,0xEA,0xEC,0xAE,
  0x7B,0xA8,0x42,0xBA,0x1F,0xC6,0x81,0x8C,0x45,0x3B,0xF1,0x9E,
  0xB9,0xC5,0xC8,0x6E,0x72,0x3E,0x69,0xA2,0x10,0xD4,0xB7,0x25,
  0x61,0xCA,0xB9,0x7B,0x3F,0xB3,0x06,0x0B,
  };
  ```

  Without the C notation, this number in hex is

  ```
  0x00cc81ea8157352a9e9a318aac4e33
    ffba80fc8da3373fb44895109e4c3f
    f6cedcc55c02228fccbd551a504feb
    4346d2aef47053311ceaba95f6c540
    b967b9409e9f0502e598cfc71327c5
    a455e2e807bede1e0b7d23fbea054b
    951ca964eaecae7ba842ba1fc6818c
    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.

- 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 (and PyCryptodome)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/). You'll want to use libgmp's exponentiation functions. Python natively supports large numbers.

- You may write your program in C, C++, or Python. Please see the teaching staff if you would like to use another programming language. For submissions done in C/C++, we will ignore all submitted executables and will compile your code from the submitted source files.
- You may only use libraries already installed on the lab computers. Please post requests for additional libraries to Canvas.
- 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 testing/grading, do not provide a prompt to the user, and only write received messages to standard out.
- Your program should not take in any additional command-line options other those described above.
- Your program should terminate under the following two conditions: when the user presses CTRL-C, and when end-of-file (EOF) is received. To generate EOF from the terminal, press CTRL-D.

Grading

This assignment is worth 100 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.

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

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 Canvas.

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.)

Include in the archive the written responses, all source code, and any descriptions required to understand the code (e.g., protocol descriptions if necessary). If your program is written in a C/C++, please also provide compilation instructions. Indicate the version of python you are using if using it.

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