COMS 4180 Network Security Spring 2015 Assignment 1

Due Tuesday, February 18th, 2014, 11:00pm Eastern time.

- This assignment is to be done individually.
- NO LATE SUBMISSIONS WILL BE ACCEPTED.
- The code for the programming problem must run in linux on the computer science department clic machines.
- Reminder: Each assignment is 20% of your final grade.

Submission Instructions

- Assignments will be submitted via the dropbox in courseworks.
- Submit a zip file containing (1) a tar file of your submission for the programming problems and (2) an ASCII text file or pdf file containing your answers to the written problems. The files for the programming part must only be the source code, makefile and readme file. Do not include any executables in your submission.
- The zip file name must be of the form UNI_#.<extension>, where "UNI" is your UNI and "#" is the number of the assignment, and the extension is zip or tgz.
- Please put your name at the top of each file submitted, including as a comment at the top of each file containing code.

Written Problems:

1. (10 points)

Explain how the round function of AES provides diffusion. (9 points)

When a 16-byte block of plaintext is encrypted with AES, how many rounds of AES are executed before every byte in the input (the plaintext) has influenced every byte in the 16-byte block as it is being processed? (1 point)

2. (10 points, 5 points for each part)

Suppose AES (or any block cipher) is used to encrypt data. Which mode of encryption (from lecture 1 slides/readings) is best suited for each of the following two scenarios and why? If more than one mode is suitable, pick one and explain why.

a. Streaming video between a server and client.

b. Disk encryption on a backup server storing system log files where the files are individually encrypted with one key used across all files. In addition, answer the following: Consider the first block of each encrypted file and the fact that the same key was used, what is the implication of this?

3. (10 points)

Use bc, an arbitrary precision calculator, to compute a shared key using Diffie-Hellman key exchange. (Pretend you are both Alice and Bob.) Use $p = 2^{61} - 1$ and g = 23489.

Alice and Bob's secret numbers are ***SA*** = 93573 and ***SB*** = 23903.

Use the script command to show each step.

bc is on the clic machines in /usr/bin/bc

Use "man bc" and "bc -h" for assistance.

4. (10 points) Lecture 3 is needed to do this problem.

Suppose Alice and Bob share a secret key K_{AB} and wish to authenticate each other. Alice always initiates contact with Bob. Consider the following protocol for mutual authentication between Alice and Bob.

Let rA and rb be nonces. E_K_{AB}{X} means X is encrypted with a symmetric key cipher using K_{AB} as the key.

• Alice sends $\{r_A\}$ to Bob

• Bob sends {E_K_{AB}(r_A, timestamp), r_B} to Alice
• Alice sends { E_K_{AB} (r_B, timestamp)} Bob
The timestamp is only accepted if it is within +/- 0.5 seconds of the receiving host's clock. When the three messages have been exchanged, can Alice be sure that she is talking to Bob? Can Bob be sure he is talking to Alice? Either justify why or show why not and how to fix the protocol.

5. (10 points) Lecture 3 is needed to do this problem.
Suppose Alice and Bob share a secret key K_{AB} and Bob wishes to authenticate Alice. Bob also is a stateless server so he will not remember what he sends to Alice. Consider the following protocol:
• Alice sends {"I'm Alice"} to Bob
• Bob sends {E_K_B (R), R} to Alice
• Alice sends {E_K_B (R), E_K_{AB} (R)} to Bob.
Where K_B is a key known only to Bob. When the three messages have been exchanged is Bob assured that is he talking to Alice? If yes, justify your answer. If no, describe how to fix the protocol.

6. (30 points)
Using the Sosemanuk paper from the software based stream ciphers selected by the estream competition http://www.ecrypt.eu.org/stream/ . **Use the link for the cipher under Profile 1 SW** from this site (don't use other links from the site that may contain prior versions of the algorithm). Explain how the cipher works, including each step of the algorithm. The explanation may include diagrams. The explanation should not be an identical copy of the algorithm description found in the article posted on the estream website. Your summary should indicate you understand the algorithm.

Programming Problem: (120 points)
This problem involves using crypto libraries/functions for encrypting, hashing and signing data. Use existing library routines as opposed to writing your own functions for any crypto algorithms needed. Do not download and install 3rd party crypto libraries. Your code must run on the clic machine without requiring that the TA install additional libraries. You may use any programming language on the clic machines. C/C++ or JAVA may be the easiest. If using C/C++, the openssl library contains the crypto functions. (Note: the openssl version on clic does support SHA-256 even if the help message does not display it in the list of algorithms.) If using JAVA, the JAVA crypto library has the required functions. There is no need to use a 3rd party JAVA library for the crypto.

Overview:
There will be two clients and a server with regular sockets (no TLS) between the clients and server. One client will encrypt and sign a file, then send an encrypted key, the encrypted file and the signature to the second client via the server.
The clients have RSA keys (using 2048 bit modulus) that will be used in this process. The server may be malicious and replace the file before sending it to the second client. The second client decrypts the file, checks the signature and indicates whether the signature verification failed or passed.
You are responsible for determining how to generate RSA keys and store the RSA keys so the clients can read the necessary keys.
Do no hard code the port numbers the clients and server use for the socket. The port numbers should be specified on the command line when starting the clients and server.

Specifications:
• **client1**:
The first client (client1) accepts a 16 character password and a filename (full path if needed) as input parameters. Client1 will encrypt the file with AES in CBC mode. It also signs the plaintext of the file by hashing the plaintext file with SHA-256 then encrypting the hash with RSA using its private key. client1 sends the encrypted data and signature to the server. The file may be any format so make no assumptions

about the format. If necessary, you may assume the programs will not be tested with any file over 1 MB. client1 also encrypts the password used as the AES key with client2's public RSA key and sends it to client2 via the server. client1 disconnects from the server after sending the password, file and signature.

Inputs:
The following information should be input in the command line when executing client 1.

- Password: The 16 character password may contain any alphanumeric character and any character from , . / < > ? ; : ' " [ ] { } \ | ! @ # $ % ^ & * ( ) - _ = +
- filename: Clearly indicate in your README file if the path of the file provided as input must be the full path or relevant to the directory containing the executable. You may just require that the file be in the same directory as the executable.
- server IP address or name
- port number to use when contacting the server
- Necessary RSA key components: any inputs (filenames) to provide client1 the necessary information for the RSA keys . Key components should be read from files and not have to be typed by the user.

• **server:**
The server just passes the password received from client1 on to client2 and does nothing to it. The server, if in trusted mode, sends the file unmodified and the signature received from client1 to client2. The server, if in untrusted mode, replaces the encrypted file with a file called "serverdata" that is located in the same directory from which the server was started and sends it and the signature to client2. The signature is not modified.

The TAs will be using a file < 1MB for serverdata.

Inputs:
The following information should be input in the command line when executing the server.

- The port numbers on which the server will listen for connections from client1 and client2, respectively.
- mode: A single lowercase character of t or u. t means trusted mode, u means untrusted mode.

• **client2:**
client2 receives the encrypted password, encrypted file and signature from the server, decrypts the file and verifies the signature, writing the result ("Verification Passed" or "Verification Failed") to standard out (the terminal window). client2 writes the unencrypted file received from the server to disk in the same directory from which the client2 application was executed. It names the file "client2data" (no extension). client2 disconnects from the server after receiving the password, file and signature.
The file client2data will be compared to the original file when grading the program so it is recommended you diff the result with the original file when testing your programs. There should be no differences, including no differences for end-of-file.
You are responsible for determining the proper RSA key components client2 uses for decrypting the password and for verifying the signature.

Inputs:
The following information should be input in the command line when executing client2.

- server IP address or name
- port number to use when contacting the server
- Necessary RSA key components: any inputs (filenames) to provide client2 the necessary information for the RSA keys . Key components should be read from files and not have to be typed by the user.

Notes on the details of padding when using library functions:
1. When using AES from a crypto library, the function return value may indicate decryption failed and not produce a plaintext file. This occurs when the file it is attempting to decrypt has a length that is not an integer multiple of 16-bytes and/or does not have the proper padding CBC mode uses (the encryption function takes care of adding the padding). So if you test with an arbitrary "serverdata" file in untrusted mode, client2 will be able to write "Verification Failed" at this point. However, when testing, you should encrypt some file (different from what client 1 will encrypt) with AES in CBC mode using the same AES key used by client 1 and use the ciphertext as "serverdata" to verify client2 can correctly tell it was not the correct file based on the signature.
2. The standard for padding will result in 1 extra block being added when encrypting in CBC mode if the plaintext is an integer multiple of 16 (this extra block is padding that is needed so the recipient decrypting the data does not interpret the last block of the actual data as padding). This is done automatically by the library functions. If the plaintext was not an integer multiple of 16, the last partial block is padded and a full block is not added.

Error Handling:
Programs will be tested for handling of invalid/garbage/missing input. The programs must check the validity of the input parameters and exit nicely if anything is invalid, printing a message specifying the required input parameters before exiting. This includes but is not limited to missing parameters, improper values (length, type, value), out of order parameters, with the exception that invalid POSITVE NUMERIC values for RSA parameters may not be detectable until subroutines are called that uses these, at which point any detected error/exception should be handled by printing an informative message indicating the error and exiting nicely. Any runtime error must also be handled by printing an appropriate message and exiting nicely. (for example, segmentation faults in C/C++ will result in a grade of 0).
NOTE: Leaving one side of a socket open is NOT exiting nicely. For example, if the server side if a socket dies, the client's side should not print the default exception to the screen (such as occurs in JAVA when exceptions are not handled) or just hang. Your code must be commented. Uncommented and poorly commented code will lose points.

What to submit for the program:
- Do not submit any executables
- client1 source code titled client1.<ext> where <ext> depends on the language you used
- client2 source code titled client2.<ext>
- server source code titled server.<ext>
- The RSA files (RSA keys) with which you tested your programs
- any helper functions that are in separate files, including any program you wrote to generate RSA keys if you did not use an existing tool/openssl command line
- A Makefile - mandatory for C, C++, optional for JAVA. If you use JAVA and no makefile, your code will be compiled by typing "javac *.java" If you include a Makefile, your code will be compiled by typing "make"
- README file: (1) the steps you used to generate the RSA keys (2) How to run your programs. If you have files other than client1, client2 and server, describe any helper functions that are in separate files - include a list of such files with one or two lines stating the purpose of each.

If you are using JAVA, you will need to use client1, client2, and server as the class names for the two client and the server programs (these are in lower case). If you are using C/C++, the executables produced by the Makefile should be named client1, client2 and server.