SCC 363 --- Crypto Assessment

This coursework is on designing, implementing, and testing components of a secure cloud file system, and performing cryptanalysis on simple substitution ciphers.

Submission deadline: Monday, 18 February 2019 at 1600 hours

Submission format: Please submit one .zip file named <studentid>-crypto.zip. If your student id is 3311224, submit 3311224-crypto.zip containing **dh.jar** and **cracker.jar**. You should also contain all of your source code (.java files). Please note the filenames should be lowercase only. Please adhere to the spec strictly! We absolutely cannot edit your programs to make them run even if the logic is correct.

Marking: Assignments will be marked by viva in the lab sessions during week 18. A schedule will be provided in due course.

Part 1: Cryptanalysis (50%)

A substitution cipher encrypts the message by replacing each letter of the plaintext with another letter (or possibly even a random symbol). The key of a substitution cipher is a mapping table that maps 26 letters to the corresponding cipher letters.

1. There is a block cipher that uses the substitution cipher in the ECB-mode, where the length of a block is one letter. Assume you have intercepted a chunk of ciphertext and the corresponding plaintext, denoted as c1 and p1. Try to recover the key of the substitution cipher.
2. Assume you got another ciphertext c2 that is encrypted under the same key as c1, please use the extracted key to decipher c2.
3. There is another block cipher that uses the substitution cipher in the CBC-mode, where the length of a block is one letter and, instead of XOR, we're using addition modulo 26. The IV is a single letter, which is provided with the ciphertext. Assume you have intercepted a chunk of ciphertext and the corresponding plaintext, denoted as c3 and p3. Try to recover the key of the substitution cipher. For this exercise, you will need to come up with a method for computing the key from the plaintext and ciphertext.
4. Assume you got another ciphertext c4 that is encrypted under the same key as c3 (in CBC-mode), please use the extracted key to decipher c4.

You will need to write a single program to perform these operations, which should be exported to a runnable jar file called “cracker.jar”. All ciphertexts and plaintexts will be provided as separate .txt files, which you should read as input. Your jar file should be run accepting the cipher mode as a string (either “ECB” or “CBC”), c1 filename, p1 filename, c2 filename, output filename, c1 IV number and c2 IV number. Your program will then be run as: java -jar cracker.jar <mode> <c1> <p1> <c2> <output> <IV1> <IV2>. If running in ECB mode, the IVs are not required.

Your program should print the computed key to the terminal, in the form m1-c1, m2-c2, with one per line. For example:
A->B
B->C
…

You should then write the decrypted contents of c2 to a txt file with the specified parameter <output> as the filename. We expect you to carry out this exercise in Java, and the above instructions are specific to Java, However, if you are more comfortable using another language then please contact module staff to discuss.

A separate document will be provided with an overview of the CBC encryption scheme in use.
We will provide you with a number of examples to test with. A different set will be used within the viva.
You must not use any external libraries for this task.

Marking criteria:
20% - Correct ECB key
10% - Correct ECB decryption
40% - Correct CBC key
10% - Correct CBC decryption
10% - Correct handling of parameters
10% - Code Commenting

Part 2: File retrieval with Perfect Forward Secrecy (50%)

Write a program called DiffieHellman.jar to carry out the following actions:

1. Implement the client side (Alice side) Diffie-Hellman (DH) key exchange protocol. Your program must connect with a cloud server (Bob) and compute a shared DH key as follows:
    a. Send $A=g^a$ mod p to the server in the format: **DHA**<A>****
    b. Receive $B=g^b$ mod p from the server: **DHB**<B>****
    c. Compute the DH shared key.
2. Using the shared DH key to compute a 128-bit session key.
. Generate a random string R of 4 letters (a-z), for example aabb..

a. Convert the key into a string of **12** characters. For eg. if the length of the key output by DH is too short, pad the string with 0s at the start e.g. “23”->”000000000023”.
b. Concatenate strings for R and DH key and obtain a 16-character string. For eg. “aabb000000000023”.
c. Generate the session key by hashing the concatenated string in c) using the MD5 algorithm, and take the first 16 bytes.
d. Send R to the server in the following format: **NONCE**<R>****
3. Request an encrypted file from the server.
a. *Server request format: **REQ****
b. *Server Response format:* **ENC**<encryptedcontents>****
The encrypted contents will be a BASE64 encoded string which should be BASE64 decoded before attempting decryption. The cipher is set up with the parameter"AES/CBC/PKCS5PADDING". The IV is the 16-character string generated in Step 2a. Decrypt the file using the AES-CBC-128 algorithm using the session key. The files should decrypt to text. The file will always start with “FILE:”.
5. Compute a MD5 hash of the file content (excluding “FILE:”).
6. Encrypt (AES-CBC-128) the hash with the session key (point 2) and send it back to the server as a BASE64 encoded string: **VERIFY**<encryptedhash>****
7. The server will reply with “**VERIFIED****” if the hash is correct.

Prepare a jar file, **dh.jar**, which should be runnable from the command line. Your jar file should be run accepting the server ip, server port, modulus (p) and base (g) and secret s as follows: dh.jar <ip> <port> <p> <g> <secret number>

You will be provided with the server implementation for testing, and your work will be tested with a server under our control with new input values. We expect you to carry out this exercise in Java, and the following instructions are specific to Java, However, if you are more comfortable using another language then please contact module staff to discuss.

You must not use any external libraries for performing the key exchange (it should be implemented by you), however you should use the built-in crypto libraries for carrying out encryption, decryption, and hashing.

DH key exchange uses large (>32bit) numbers as input. Use BigIntegers in Java in to handle these numbers while implement the algebra.

Marking criteria:
20% - Correct number handling (including the use of BigInteger)
20% - Correct secret key generated (correct Diffie Hellman implementation)
10% - Correct session key generation
10% - Correct decryption of data
10% - Hash decrypted data
10% - Encrypt hash and send to server
10% - Code Commenting