CS 1501 – Algorithm Implementation – Assignment#4[1]

Due: Monday November 23rd @ 11:59pm

Late submission: Wednesday November 25th @11:59pm with 10% penalty per late day

OVERVIEW

Purpose: The purpose of this programming assignment is to implement a primitive secure communications system utilizing the RSA cryptosystem and two primitive symmetric ciphers – a substitution cipher and an additive cipher.

INSTRUCTIONS

You can find on Canvas a primitive "Secure Chat Server" program, called SecureChatServer, which will allow multiple clients to connect to it and send messages to each other, such that the messages themselves are encrypted prior to being sent. The server and its clients work together as described below.

- The server can accept multiple clients, and each message sent by a client is received by the server and forwarded to all of the clients (including the sender).
- Each client **connects to the server in the following way**. The order of these statements is important. The server is already written, and it will be executing corresponding statements. To really see what is going on, look at how the server accepts client connections in the SecureChatServer.java source code.
- **Client** operation (you should implement these in SecureChatClient.java):
    - It opens a connection to the server via a Socket at the server's IP address and port. Use 8765 for the port. Have your client prompt the user for the server name. More than likely you will always be using "localhost" for the server name, since you will be running the server on your own machine.
    - It creates an ObjectOutputStream on the socket (for writing) and immediately calls the flush() method (this technicality prevents deadlock).
    - It creates on ObjectInputStream on the socket (be sure you create this AFTER creating the ObjectOutputStream).
    - It receives the server's public key, E, as a BigInteger object.
    - It receives the server's public mod value, N, as a BigInteger object.

[1] Assignment adapted from Dr. John Ramirez's CS 1501 class.

- It receives the server's preferred symmetric cipher (either "Sub" or "Add"), as a String object.
- Based on the value of the cipher preference, it creates either a Substitute object or an Add128 object, storing the resulting object in a SymCipher variable. See details below on the requirements for the Substitute and Add128 classes.
- It gets the key from its cipher object using the getKey() method, and then converts the result into a BigInteger object. To ensure that the BigInteger is positive (a requirement for RSA), use the BigInteger constructor that takes a sign-magnitude representation of a BigInteger – see the API for details.
- It RSA-encrypts the BigInteger version of the key using E and N, and sends the resulting BigInteger to the server (so the server can also determine the key – the server already knows which cipher will be used).
- It prompts the user for his/her name, then encrypts it using the cipher and sends it to the server. The encryption will be done using the encode() method of the SymCipher interface, and the resulting array of bytes will be sent to the server as a single object using the ObjectOutputStream.

– At this point the "handshaking" is complete and the client begins its regular execution. The client should have a nice user interface and should allow for deadlock-free reading of messages from the user and posting of messages received from the server. All messages typed in by the user should be sent to the server and should only be posted after they are received back from the server. You can find a primitive (unsecure) chat client that you may use as a starting point. See ImprovedChatClient.java for details. You may base your client on this if you like, or you may design it on your own.
– All messages typed in from the user must be encrypted using the encode() method of chosen cipher object, then sent to the server for distribution using the ObjectOutputStream. For details on encode(), see below.
– All messages received by the client should be read from the ObjectInputStream as byte[] objects. They should then be decrypted using the decode() method of the chosen cipher and posted to the client's window or panel. When the client quits (either through some functionality in the program or by simply closing the window) the message "CLIENT CLOSING" should be sent to the server. This message should be encrypted like all other messages, but should not have any prefix (ex: no client name). The server will use this special message as a sentinel to close the connection with the client.
– Note that the server provided will not compile or run without the SymCipher interface and the Add128 and Substitute classes. You must implement Add128 and Substitute before using the server.

SymCipher, Add128 and Substitute

Both the SecureChatServer and the SecureChatClient programs rely on the SymCipher interface and the Add128 and Substitute classes. I have provided the SymCipher interface for you. The Add128 and Substitute classes, which you must write, must implement SymCipher and your client should be able to handle both the Add128 and Substitute encryption schemes. See the SecureChatServer class for some insight on how to use SymCipher, Add128 and Substitute.

- **SymCipher**: This interface contains 3 methods: getKey(), encode() and decode(). See more information in the file SymCipher.java.
- **Add128**: This class must implement SymCipher and meet the following specifications:
    - It will have two constructors, one without any parameters and one that takes a byte array. The parameterless constructor will create a random 128-byte additive key and store it in an array of bytes. The other constructor will use the byte array parameter as its key. The SecureChatClient will call the parameterless constructor and the SecureChatServer calls the version with a parameter.
    - To implement the encode() method, convert the String parameter to an array of bytes and simply add the corresponding byte of the key to each index in the array of bytes. If the message is shorter than the key, simply ignore the remaining bytes in the key. If the message is longer than the key, cycle through the key as many times as necessary. The encrypted array of bytes should be returned as a result of this method call.
    - To decrypt the array of bytes, simply subtract the corresponding byte of the key from each index of the array of bytes. If the message is shorter than the key, simply ignore the remaining bytes in the key. If the message is longer than the key, cycle through the key as many times as necessary. Convert the resulting byte array back to a String and return it.
    - Note that, due to wrapping and the fact that bytes are signed in Java, some or even many of your bytes will appear as negative numbers. This is normal.
- **Substitute**: This class must implement SymCipher and meet the following specifications:
    - It will have two constructors, one without any parameters and one that takes a byte array. The parameterless constructor will create a random 256-byte array which is a permutation of the 256 possible byte values and will serve as a map from bytes to their substitution values. For example, if location 65 of the key array has the value 92, it means that byte value 65 will map into byte value 92. Note that you will also need an inverse mapping array for this cipher, which can be easily derived from the substitution array (so you only need to send the original substitution array to the server). Be careful with this class since byte values can be negative, but array indices cannot be negative – this issue can be resolved with some thought. The other constructor will use the byte array parameter as its key. The SecureChatClient will call the parameterless constructor and the SecureChatServer calls the version with a parameter.
    - To implement the encode() method, convert the String parameter to an array of bytes, then iterate through all of the bytes, substituting the appropriate bytes from the key. Again, be careful with negative byte values.
    - To decode, simply reverse the substitution (using your decode byte array) and convert the resulting bytes back to a String.

PROGRAM OUTPUTS

To help with the grading and testing of your program, you must also do the following:

- Output the keys E and N received from the server to the console.
- Output the type of symmetric encryption (Add128 or Substitute) to the console once it is received from the server.

- Output the symmetric key that is generated by the client (and will be sent to the server) to the console.
- For each message that is encrypted, output the following to the console:
    - The original String message
    - The corresponding array of bytes
    - The encrypted array of bytes
- For each message that is decrypted, output the following to the console:
    - The array of bytes received
    - The decrypted array of bytes
    - The corresponding String

To help you understand how the server will work, a primitive (unsecure) server is provided for you. See ImprovedChatServer.java for details. You can run this server as is with the ImprovedChatClient class to see how interaction between client and server work and how the messages are displayed.

EXTRA CREDIT

If you design your client particularly well (i.e. make it a lot nicer than the one provided) you can receive extra credit.

SUBMISSION REQUIREMENTS

You must submit to **Gradescope** at least the following files:

1. Substitute.java, Add128.java, and SecureChatClient.java
2. Any other files that you have written
3. Assignment Information Sheet

The idea from your submission is that your TA (and the autograder if applicable) can compile and run your programs **from the command line** WITHOUT ANY additional files or changes, so be sure to test it thoroughly before submitting it. If the TA cannot compile or run your submitted code it will be graded as if the program does not work.

If you cannot get the programs working as given, clearly indicate any changes you made and clearly indicate why on your Assignment Information Sheet. You will lose some credit for not getting it to work properly, but getting the main programs to work with modifications is better than not getting them to work at all. A template for the Assignment Information Sheet can be found in the assignment's Canvas page. You do not have to use this template but your sheet should contain the same information.

Note: If you use an IDE such as NetBeans, Eclipse, or IntelliJ, to develop your programs, make sure they will compile and run on the command line before submitting – this may require some modifications to your program (such as removing some package information).

RUBRICS

Item	Points
Handshaking:	
E and N received and output:	10
Encryption type received and processed:	5
Random key generated and output:	10
Key encrypted and sent to server:	10
User name encrypted and sent to server:	5
Add128 Class:	
Constructors:	10
Message encryption process is correct:	10
Message decryption process is correct:	10
Substitute Class:	
Constructors:	10
Message encryption process is correct:	10
Message decryption process is correct:	10
Outgoing messages work (with trace):	15
Incoming messages work (with trace):	15
"CLIENT CLOSING" message:	10
Documentation and style	5
Assignment Information Sheet/Submission:	5