CS 1501
Algorithm Implementation
Programming Project 4

Online: Friday, June 30, 2017

Due: All source files (including the server) and data files (keys.txt) and an Assignment Information Sheet zipped into a single .zip file and submitted via the submission site by 11:59PM on **Friday, July 14, 2017**

Late Due Date: All files submitted by Monday, July 17, 2017

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

I have written 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** (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).
  - 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
  - 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. I have written 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. If you design your client particularly well (i.e. make it a lot nicer than mine) you can receive **extra credit.**
- 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 / 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 will 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 / 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.

To help with the grading / 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:
    1. The original String message
    2. The corresponding array of bytes
    3. The encrypted array of bytes
- For each message that is decrypted, output the following to the console:
    1. The array of bytes received
    2. The decrypted array of bytes
    3. The corresponding String
- To help you understand how the server will work, I have also provided the code for a primitive (unsecure) server. 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.

W Students: Rewrite your paper from Assignment 1 such that it is markedly better than your original submission. Papers can almost always be improved, so even if your initial paper was pretty good you should still put a lot of effort into the rewrite so that you can make it even better. Use Jarrett's comments as a guide but do not hesitate to modify portions of your paper that had no comments. Reconsider both the form and the content of your paper and try to improve both. This will likely cause your revision to be longer than your original paper.