

Implementation of Cryptographic Algorithms

Assigned: Nov. 13, 2020

Due: Nov. 20, 2020

Version: 0.1.1

In this assignment, you will gain better understanding of cryptography by implementing a simplified version of RSA encryption and then by using public keys generated by the software, PGP. In the first part of the assignment we will extend the client and server written for Project 1. Now the server process will send its public key to a client process. The client process will then use the public key to send a short session key to the server. The server will decrypt that key, which will eventually be used in a symmetric encryption implementation. The second part of the assignment involves you generating public and private key pairs to send messages to each other with confidentiality.

Part A: Simplified RSA Encryption

As we discussed in class, RSA encryption is computationally expensive and so it is typically used to distribute session keys for a symmetric encryption algorithm. In this assignment we will simulate the existence of two parties, a client and a server, that use public key encryption to exchange a session key that will be used to encrypt data using the simplified AES algorithm. The approach that we are using gives you an insight of how TLS/SSL works.

Initially the client will say hello to the server and indicate a list of symmetric and asymmetric algorithms that it can support. The server will respond with its own hello message, which includes one symmetric, and asymmetric algorithm that it can support from the list supplied by the client. This hello message will also include the server's public key as well as a 16-bit pseudorandom string. The client will then send a session key message that will comprise of the "103 SessionKey " string followed by a symmetric key that is encrypted with the server's public key, this is in turn followed by the nonce encrypted with the symmetric key algorithm. The server will verify that the received nonce matches what was sent. If there is no match, the server will send the "400 Error" message to the client and close the connection to the client. If there is a match, the server will send the "106 " message to the client as in Project 1. The server will follow that message with the "107 Polls Open" message. Operation continues as specified for Project 1, except that this time the votes, i.e., the integer for each candidate, will be encrypted using the simplified AES algorithm. The encrypted votes will be sent to the server. The server will decrypt the votes for each client and then return the winning candidate to the client.

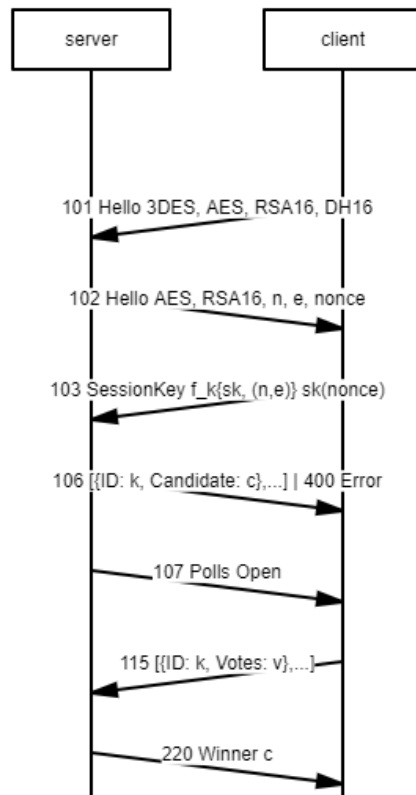


Figure 1: Messages exchanged for cryptography project

Server Implementation

Write a function that receives as its parameters primes p and q , calculates public and private RSA keys using these parameters. You should print n , $\varphi(n)$, d , and e to standard output. *You must use the Extended Euclidean Algorithm to compute d .* Write another function that decrypts data. That function will accept as input an integer parameter for the ciphertext and return the plaintext. This function *must* make use of the `modular_exponentiation` function, that is implemented in the `NumTheory` class.

When you start up your server you will accept the two prime integers and compute the public and private keys using the function defined above. After the client says “Hello” the server will respond to the client’s message with a “Hello” message that contains the server’s public key as well as a 16-bit pseudorandom string. The client will send a session key message that will comprise of the “103 SessionKey” string followed by a 16-bit symmetric key that is encrypted with the server’s public key, this, in turn, is followed by the nonce encrypted with the symmetric key algorithm. The server will decrypt the symmetric key, and then use the decrypted symmetric key to extract the nonce. Finally, the server will compare the decrypted message with the nonce that was sent. If there is a match, the server will send the “106...” message followed immediately with the “107 Polls open” message. If there is no match, then the server will send the “400 Error” message to the client and close the connection to the client. The rest of the operation is as specified for Project 1, except that this time the total votes at the client for each candidate will be encrypted using the simplified AES algorithm.

Client Implementation

You should write a function that takes in an integer parameter for the plaintext and returns its ciphertext when encrypted according to the RSA algorithm. This function must make use of the `modular_exponentiation` function, whose implementation has been given to you in the `NumTheory` class. You should also write a function that generates a random session key, which is at least 15 bits long and no more than 16 bits long. A lot of your processing will be done in the `start()` function. Please add the necessary logic to complete that function.

The client side of the execution will begin with the client saying “hello” to the server and listing a set of cryptographic algorithms that it can support. In our case, those algorithms are simplified triple DES, simplified AES, simplified RSA, and simplified Diffie-Hellman. The server will respond to the client’s hello message by selecting simplified AES and simplified RSA, followed by the server’s RSA public key, and a nonce. The client will then send a session key message that will comprise of the “103 SessionKey” string followed by a symmetric key that is encrypted with the server’s public key, this is in turn followed by the nonce encrypted with the symmetric key algorithm. The server will verify that the encrypted nonce matches what was transmitted. If there is no match, the server will send a “400 Error” message, otherwise, it will send down the list of candidates followed by the “107 Polls Open” message. From this point on, the implementation proceeds as in Project 1, except that the candidate votes will be encrypted using the simplified AES algorithm.

For debugging purposes, you should print out all the messages received from the server. Make sure to document clearly in your code any assumptions you make about the input and encryption algorithm.

Part B: Using PGP

The objective of this part is for you to become familiar with PGP and its use for secure communications. You may either use PGP on the lab computers under Linux, or you may download PGP for your personal computer. The information posted at <https://help.ubuntu.com/community/GnuPrivacyGuardHowto> is a helpful tutorial on using PGP on an Ubuntu machine. Please carry out the following tasks:

1. Generate a pair of keys. Export your public key into a .asc file.
2. Upload your public key to the MIT pgp keyserver, `pgp.mit.edu`.
3. Get at least two classmates to sign your public key, and sign at least two classmates’ public keys. Ensure they upload the signed key to the MIT keyserver. For each key you signed, explain the process you used to obtain, verify, and sign the key.
4. Use PGP to send a secure e-mail message to one of the persons with whom you exchanged public keys.

What to turn in:

- A copy of your public key, i.e., the .asc file.
- An explanation of the process used to obtain, verify, and sign the key.
- A screenshot of the encrypted email message including the header.

Rules

- **The project is designed to be solved independently.**
- **You may not share submitted code with anyone.** You may discuss the assignment requirements or your solutions away from a computer and without sharing code, but you should not discuss the detailed nature of your solution. If you develop any tests for this assignment, you may share your test code with anyone in the class. Please **do not put any code from this project** in a public repository.

What to turn in

1. A single zip|tar archive containing all the source files for your implementation, a PDF document with your tests and program documentation, and a copy of your public key. Your PDF document should also contain the screenshot for Part B. Your archive must unzip to a directory named with your student ID, and all of your files must be in that directory. The simplest way to achieve this is to download the starter code, unzip it, and rename the directory to have your student ID. When you are ready to submit, simply zip the directory with your student ID.
2. You will hand in the code for the client and server implementations along with screen shots of a terminal window, verifying that your programs actually carry out the computation that is specified.
3. For Part A, a separate (typed) document of a page or so, describing the overall program design, a verbal description of “how it works,” and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
4. A separate description of the tests you ran on your programs to convince yourself that they are indeed correct. Also describe any cases for which your programs are known not to work correctly.

Grading

- For Part A:
 - Program listing
 - Works correctly as specified in assignment sheet – 35 points
 - Contains relevant in-line documentation – 5 points
 - Design document
 - Description – 5 points
 - Thoroughness of test cases – 5 points
- For Part B:
 - Key generation - 5 points
 - Screen shots of encrypted email messages – 5 points.