

CENG 471- TERM PROJECT

STAGE 2



May 5, 2019

Hasan yeniada

220201024

**Table of Contents**

1. **Introduction3**
2. **Summary of the Report3**
3. **First Step - Public / Private Key Generation4**
4. **Second Step – Public Key Sharing6**
5. **Third Step - Encryption / Decryption Operations** **7**
6. **Performance Measurements** **9**
7. **Conclusion** **10**
8. **References** **10**
9. Introduction

In that second stage of term project, it is asked for us to implement the same scenario with first stage, but in that time RSA encryption scheme should be used. The implemented modulos in the first stage can be used in that stage withoud any change, so I just added the RSA package to our project.

In that time, the sender and receiver generate their own RSA public-private key pairs and share their public keys. Therefore, in the first step of the report, I will show how to generage RSA public-private key pairs in Java, and how the sender and receiver share their public keys between each other. In RSA encryption scheme, the sender and receiver use **different** keys in encryption and decryption operations, however in the first stage, they have used the **same** key in encryption and decryption and that same key is shared using Diffie-Helman Key Exchange scheme.

After generation of the public-private key pairs, using RSA encryption scheme, different length of documents are encrypted by sender and decrypted by receiver. I used the files used in the first stage also in that stage, while measuring the performance of the resulting code. I will also add these performance measurement results using some screenshots and tables to compare the RSA and AES better.

In RSA, there are no key distribution problem, because both the sender and receiver use different keys in encryption-decryption operations, however, the main problem in RSA is that the message length is limited to encrypt, and I will explain that detailly in the third step of the report. Because of that limitation, I have used again AES in that stage to solve that message length problem to help RSA scheme.

1. SUMMARY OF THE REPORT

In the first stage, I will show how to generate public-private key pairs in Java. While generating these keys, we need to use GCD, multiplicative inverse and relatively prime checking operations of the modulo Euclid’s Extended Algorithm, which I implemented in the first stage of the project. We also need Fermat Little Theorem while generating big prime numbers p and q.

In the second stage, I will show how to encrypt and decrypt a basic String and different length of documents using RSA encryption scheme, and in that part we also need Fast Exponentiation modulos which is implemented in the first stage. I have used the AES in RSA encryption scheme to solve the message length limitation problem.

At the end, I will show the performance measurements of the RSA and compare it with the AES using some tables and screenshots.

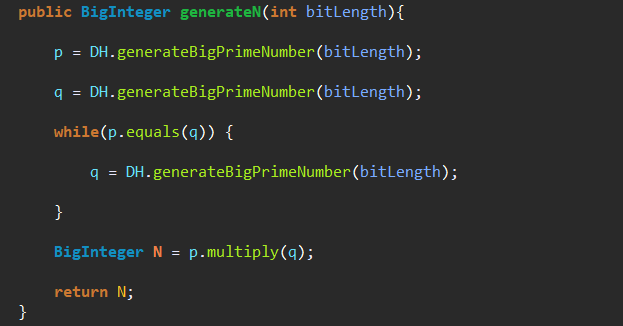
1. FIRST STEP – PUBLIC / PRIVATE KEY GENERATION

In the first step, the sender and receiver generates their own public and private keys. In RSA, the sender and receiver use different keys in encryption and decryption operations, and to generate these keys, the following steps are performed:

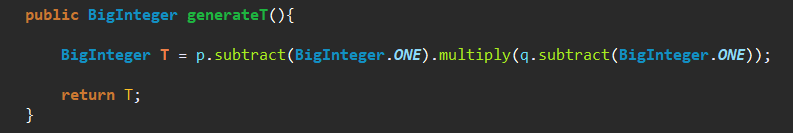
* The first step is to generate big prime numbers p and q as 1024 bits length. To generate these big prime numbers, I used the same method in the first stage, and I have explained that method detailly in the first stage.



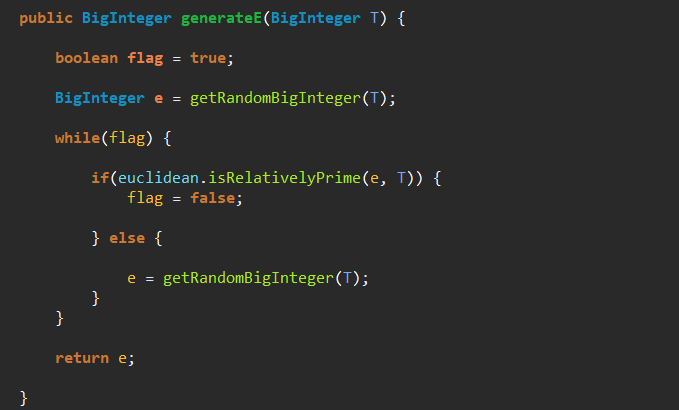
* The second step is to generate N using the p and q, and N = p \* q, here is the implementation of the generation of p, q and N:



* The third step is to generate T using again p and q, and here is the implementation of the generation of T:

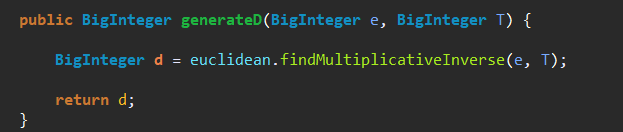


* The fourth step is to choose e, such as GCD(e, T) = 1, GCD(e, T) = 1 means that e and T are relatively prime, so in that part, I used the isRelativelyPrime method of ExtendedEuclidean, here is the implementation:



In that code, we choose a random BigInteger e, and until we find an e such as that e is relatively prime with the generated T. Then returns the chosen e.

* The fifth step is to find d such as d \* e = 1 (mod T), means that d is the multiplicative inverse of e in mod T, so I used the findMultiplicativeInverse method of the of ExtendedEuclidean class, here is the implementation:

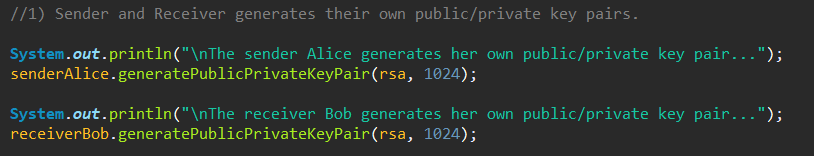


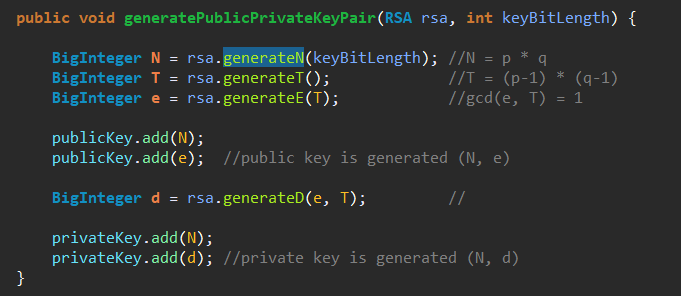
* At the end, we have generated public and private keys which are:

**-Public key = (N, e)**

**-Private key = (N, d)**

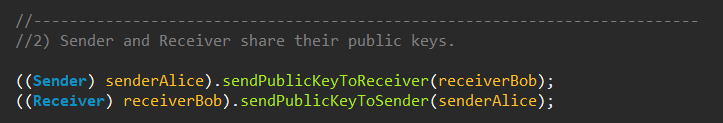
Here is the complete key generation implementation:

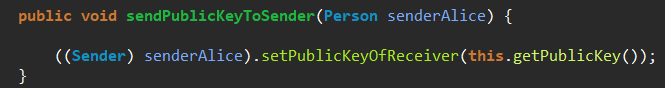


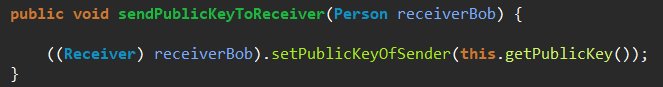


1. SECOND STEP – PUBLIC KEY SHARING

Both Alice and Bob generated their own public and private key pair as I shown, and next step is thay they share their public keys between each other:







Now Alice has the public key of Bob, and Bob has the public key of Alice.

1. THIRD STEP – ENCRYPTION / DECRYPTION OPERATIONS

Now, the sender Alice has the public key of the Bob, and she can send a message to Bob encrypting the message with the public key of Bob, and only the Bob who has the private key can decrypt the message. Here is the Encryption and Decryption functions:

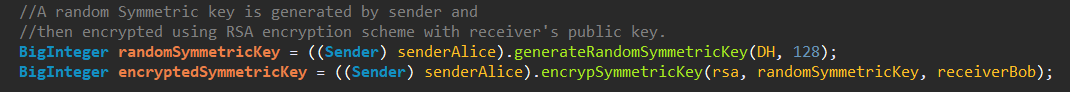
**Encryption Function** = Cipher Data = (Plain Data ^^ e) (mod N)

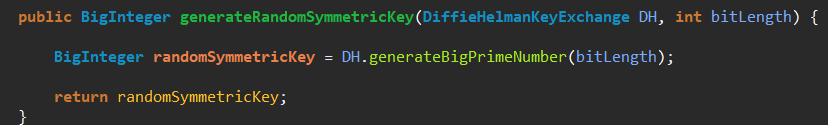
**Decryption Function** = Decrypted Data = (Cipher Data ^^ d) (mod N)

Note that, **(N, e)** is the **public key** of the **Bob**, and **Alice** encrypts **Plain Data** with that **public key** and sends that message to **Bob**, then **Bob** uses his **private key** **(N, d)** and decrypt **Cipher Data**.

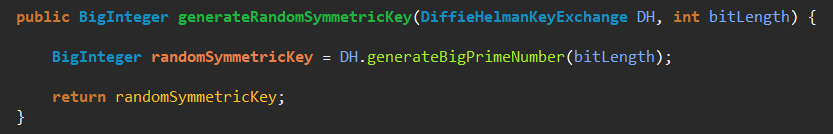
Here, the main problem in RSA is that Alice can send a message with limited length, because is the message length exceed the generated number N, then the message Alice sent cannot be the same with the message Bob get, so this situation limits the message length which is sent by RSA encryption scheme. To solve that issue, I did some researches on the internet and found a method named Hybrid Encryption, and in that encyrption scheme, there is no restriction on the message length, and here are the steps of that approach:

* Generate a random symmetric key:

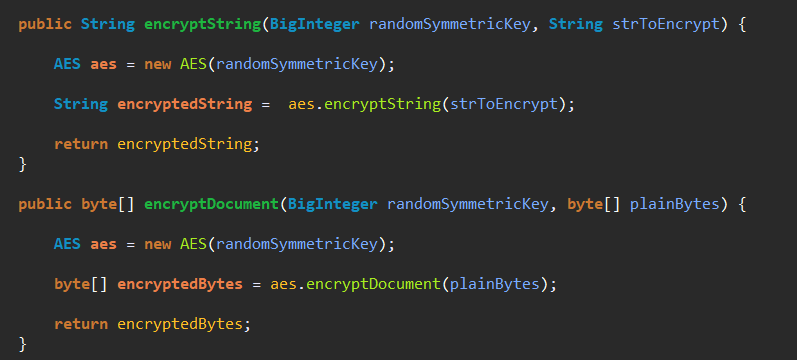




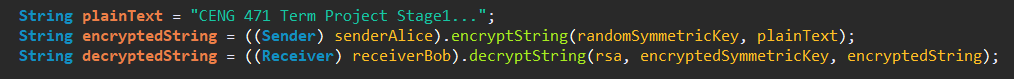
* Encrypt it using RSA encryption scheme:

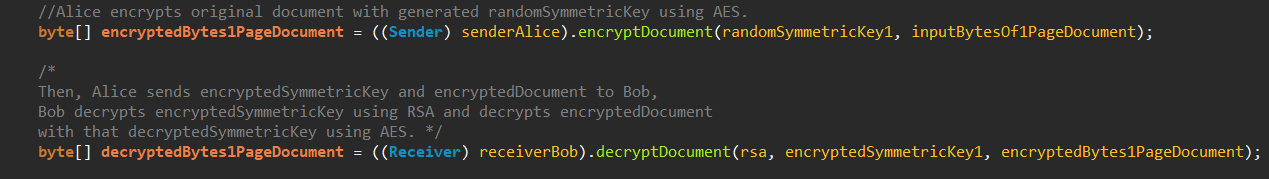


* Encrypt the Plain Data with the AES encryption scheme:



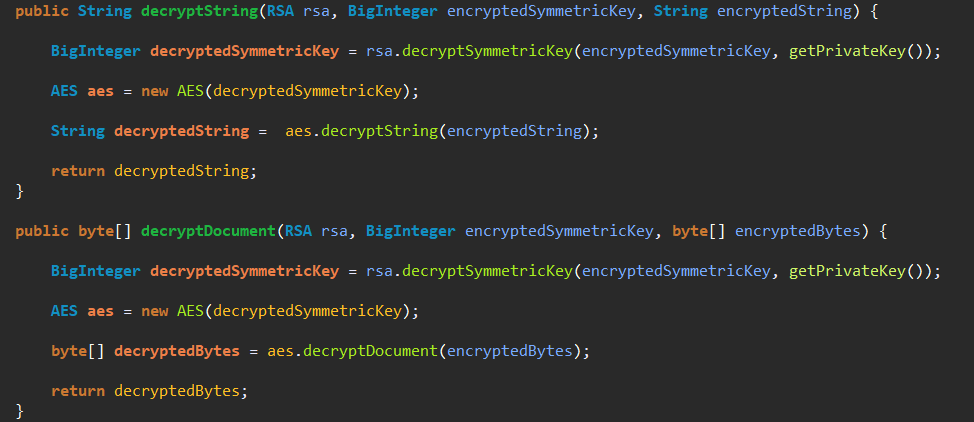
* Send both (EncryptedSymmetricKey by RSA) and (EncryptedPlainData with AES) to Bob:





* Bob decrypts the EncryptedSymmetricKey by RSA with his private key and decrypts EncryptedPlainData by AES with that DecryptedSymmetricKey and got the original message.

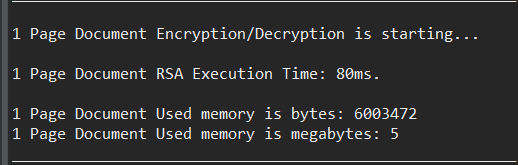
This is the Hybrid Encryption and I used the AES code of the first stage to implement that scheme.



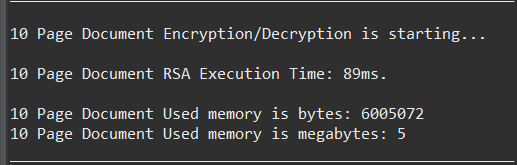
1. PERFORMANCE MEASUREMENTS

Until that, I have explained how to implement each step of stage1 one by one, and now it is time to discuss the performance of the resulting code. I will add the execution time and memory usage results of each different size documents.

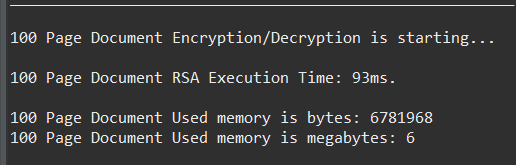
**1 Page Documment Execution Time and Memory Result:**



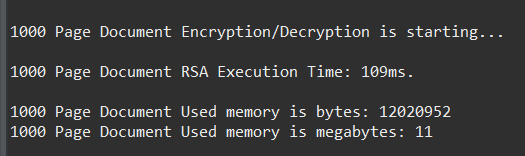
**10 Page Documment Execution Time and Memory Result:**



**100 Page Documment Execution Time and Memory Result:**



**1000 Page Documment Execution Time and Memory Result:**

****

**Let’s Summarize the Results in a table:**

|  |  |  |
| --- | --- | --- |
| RSA | Execution Time | Memory |
| 1 Page Document | 80 ms | 5 mb |
| 10 Page Document | 89 ms | 5 mb |
| 100 Page Document | 93 ms | 6 mb |
| 1000 Page Document | 109 ms | 11 mb |

1. CONCLUSION

When we compare the performance results of the AES in the first stage, and the performance results of the RSA with Hybrid approach, performance of the AES is better than the RSA, however security of the RSA is better than the AES because of there is no need to any key exchange scheme, because the sender and receiver use different keys in encryption and decryption. The execution times of the 1 page, 10 page, 100 page and 1000 page documents are very close to each other, because the most important factor that affects the execution time is public-private key generation part, and that is common for all of them, and the contents of the documents are encrypted and decrypted with AES and so that operation takes very less time than key generation part. As a result, RSA can be a secure and slow encryption scheme choice, with a message length limitation, however symmetric encryption schemes AES and DES are less secure and good performance encryption scheme choices.

1. REFERENCES

* <https://www.quaxio.com/exploring_three_weaknesses_in_rsa/>
* <https://crypto.stackexchange.com/questions/11904/why-does-plain-rsa-not-work-with-big-messages-mn>
* <https://hackernoon.com/how-does-rsa-work-f44918df914b>
* <https://stackoverflow.com/questions/51761721/large-data-not-encrypted-with-rsa-encryption>
* <https://security.stackexchange.com/questions/33434/rsa-maximum-bytes-to-encrypt-comparison-to-aes-in-terms-of-security>
* <https://stackoverflow.com/questions/10007147/getting-a-illegalblocksizeexception-data-must-not-be-longer-than-256-bytes-when/46828430#46828430>
* <https://www.sitepoint.com/encrypt-large-messages-asymmetric-keys-phpseclib/>