INF-2301: Nettverk og sikkerhet

# Mandatory Assignment 2

Jesper Rødstrøm

October 3rd, 2018

## 1 Introduction

In this assignment we will implement a file-sharing solution. The assignment consists of two parts, one of them being the implementation itself and the other part being analysis of that implementation. The first part requires a client and a server implemented. These two will communicate securely by encrypting the transferred data in such a way that an attacker would have a tough time getting the information. The second part is where we will reflect on the implementation, finding possible information leaks and determining what can be done to further increase security.

### 1.1 Requirements

In the assignment we will do as follows:

#### PART A

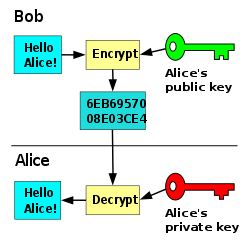
* Create a server which stores one or more files.
* The server should only require a local file path which it will serve and a TCP port number where it will listen for incoming connections.
* Create a client which can download the server files.
* The client should only require the hostname / IP address of the server it will connect to and the TCP port number where that server is listening.
* The file transferring must be encrypted using both symmetric and asymmetric encryption.

#### PART B

* Analyze and scrutinize the file-sharing system with regards to the CIA and AAA security principals, described in the course syllabus.
* The findings must clearly be documented in the report.

## 2 Technical Background

2.1 Asymmetrical encryption:

Asymmetrical encryption consists of two keys that is used for encrypting and decrypting data. One of the keys is a public key that can be distributed and shared freely with other systems. This key is typically used for encrypting data that will be sent back to the source that created this key. The other key is a private key which will not be shared with any other systems. When the source receives information that are encrypted using its public key, the private key is used to decrypt this data.

This encryption method is usually not used for big blocks of data because of the computational complexity. Instead it is used on smaller blocks. Since symmetrical encryption outclasses asymmetrical in terms of computational speed the most common way to use the encryption method today, is by sharing a symmetrical key between two systems and using this for file sharing instead. The reason why asymmetrical is slower originates from the algorithms used for encryption. Asymmetrical encryption is based on mathematical problems that do not have any efficient solution.

Figure 1 – Asymmetrical encryption-decryption

Another optional feature of asymmetrical encryption is signatures. This adds another element to the data sent. Creating a signature is done by combining the data with a private key. By adding this the key creator can validate the received data by checking if the signature is correct. If the data is modified in any way, it will not validate as a correct signature and the data can be handled accordingly.

### 2.2 Symmetrical encryption:

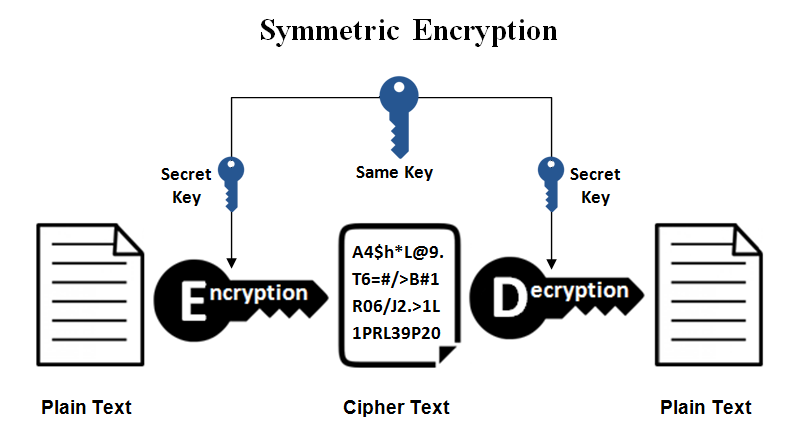
Symmetrical encryption consists of only one key. Therefore, if two parties want to exchange information they both need to have access to the same key. Because of this symmetrical encryption is inflexible on its own. However, if the key could be shared securely this downside can be erased. This is where the synergy between asymmetrical and symmetrical becomes apparent since asymmetrical can be used to safely share the symmetrical key.

Figure 2 – Symmetrical encryption-decryption

Due to the faster computational complexity, symmetrical encryption is preferred for exchanging large amount of the data. The reason why the computational speed is faster is because symmetrical encryption is based on simpler mathematical problems compared to its asymmetrical counterpart.   
  
There are mainly two types of encryption cyphers used for symmetrical encryption. Stream ciphers and block ciphers. Stream ciphers encrypt letters or digits individually throughout a message, while block ciphers encrypts a group of bits together as a single unit, as well as padding the text to make it a multiple of the block size used.

## 3 Design

This code was implemented using the Socketserver library for the server, and the socket library for the client. Socketserver is specialized for server use, while socket can be used both for servers and clients. AES is used for the symmetrical encryption, while RSA is used for the asymmetrical one. The overarching library used for both encryption methods is Pycrypto (CryptoDome). This library offers a wide range of methods and classes used for encryption/decryption. As for the type of cipher used for the symmetrical encryption, the block cipher was chosen.

An additional library named AST is also being used to convert a string into a tuple, this was required to get the symmetrical key back to its original state encrypted state after sending it through http.

## 4 Implementation

### 4.1 Server

The whole implementation consists of two main implementations and a module created for handling AES encryption/decryption.

For the server implementation there is one major class “TCP handler”, which has methods that handle the different instances of requests that will be received from the client. The main handle method which is overwritten from the super class contains all the other methods and ties the whole handling process together.

The main handle methods’ primary job is to receive the request from the client read it and then decide on which case-specific method it will use to handle that request. A method that handles post requests has the job of extracting the public key from the http request message and use this to encrypt and send the symmetric key to the client. Another method that handles get requests extracts the data from the file requested, encrypts it and send the contents to the client.

For the client implementation there is one major class Client which consists of three methods and a constructor. The primary method in client is also called handle and follows the same design architecture as the server, calling other methods that handle each request specifically.

The main handle method has the job of prompting the user to input the file they want to access, and then sequentially calling methods to create requests to send to the server. One of the methods creates post requests. This method sends a public key to the server and receives the response message with the symmetrical key, returning it to its original tuple state. The other method creates GET requests and receives the body containing the encrypted message, decrypting it and writing it to its own client file.

Lastly the module for AES encryption and decryption consists of three methods. One of the methods adds the padding to the data, another one creates the iv and encrypts data, while the last one decrypts the data returning a string containing the original message that was encrypted.

## 5 Discussion

This is where part B of the assignment will be answered, it will focus on potential threats, information leaks and how it can potentially be dealt with through redesign or adding new features.  
  
One of the potential threats that the implementation could be exposed for is a DDoS attack. DDoS stands for Distributed denial-of-service. Its purpose is to sabotage the server by bombarding it with many requests continuously. An attacker usually does this by using a large network of zombie computers, to continuously send requests to the server repeatedly until it would overload. This would result in the server not being able to provide the service it is intended for, preventing the client from getting information.  
  
To get an outlook on what information an attacker can get, the information that gets sent back and forth needs to be observed. There is a total of four chunks of data that is sent through the communication link. The first information sent is the public key, which can be used to encrypt data. The second piece is the encrypted symmetric key. The exposed symmetric key would be a valuable piece of information for an attacker since it would allow them to decrypt any messages sent from the server. The third piece is a simple get request and fourth being the encrypted message.  
  
This server would be severely exposed if the attacker managed to get a hold of the symmetric key since the key is not random. Someone would have to change it manually if there was an attack which is not preferable at all. To prevent this, the symmetric key could be set to be random which would result in the attacker having to acquire a new one for each session.   
  
For the attacker to get a hold of the symmetric key, they would have to find a way to get the private key and decrypt it. One of the methods they could use would be a brute-force attack. This is when the attacker tries to guess the key by having a computer try all the possible combinations until it reaches the correct one. This is a computationally heavy procedure and the time would vary on the key length and the attackers’ computer hardware. To reduce the risk of such an attack the best way is to increase the key size and increase the time needed exponentially.  
  
Another less computational way to get access to the private key exists. This way requires that the attacker knows what hardware the server is using. By knowing the exact amount of time that hardware needs to encrypt plaintext, they could simplify the search process for a potential key. This way of attacking is known as a side-channel attack since it revolves around the hardware of a computer system instead of the weaknesses in the algorithms themselves.  
  
Lastly there is something called a man-in-the-middle attack. This is when the information sent back and forth is intercepted by the attacker and changed. An attacker could then modify the data sent by the server making it send the attackers’ symmetric key instead, and then later malicious content to the client. They could also change the public key sent to the server with one of their own, which would enable the attacker to get access to the symmetric key.  
  
To improve the implementation and make it harder for an attacker digital signatures could be used. As described earlier in section 2.1 (asymmetrical encryption) there is a feature which would add a signature to the data sent. Using this the client can validate that the encrypted message they receive has not been changed one the way. Changing a single character in the message would result in the signature not being valid and could then be denied.   
  
Another way to improve the implementation is by using certificates. This change eliminates most risk when it comes to man-in-the-middle attacks since it allows the server to know who the key belongs to. When distributing the public key, a certificate is attached to it containing information about the creator of the key. This prevents the attacker from distributing his own key and allows the server to check the certificate to validate the data coming from the client.   
  
Now to conclude and tie all the concepts together for an improved implementation. Some steps should be taken first, like changing the symmetric key to be random and increasing the length of the asymmetric keys to reduce the risk of an attacker brute-forcing. Then the server should also create its own set of asymmetric keys to prevent the attacker from sending his own malicious data and validate the server responses. This is done by using the certificates. Then all public keys should have their own signatures to validate them.   
  
The whole communication process would be changed. First, the client would send his own public key to the server. The server can check the certificate to make sure it was received from the client. Then the server sends its own public key to the client. This key is also validated through a certificate. After this, the server uses its private key to encrypt the symmetrical key followed by encrypting it once again with the clients’ public key. The client receives the data and validates it by checking the signature on the public key. Then they decrypt it using their private key and use the public key they received earlier from the server to decrypt it once again and receive the data. By doing this the client ensures that the data they get is valid and not malicious. Now that both parts have the randomized symmetrical keys, the rest of the procedure would follow the old pattern. To prevent DDoS attacks the server would need stronger hardware and/or split the work load to more than one server. Backup servers could also be set up to engage if the server went offline.

## 6 Conclusion

The file-sharing solution has been completed successfully. With the help of libraries that simplifies sockets and reading of request data, the requests where divided down and handled appropriately. Part A revolved mainly around header handling and the assignment was solved by inheriting the BaseRequestHandler class. Part B revolved mainly around manipulation of JSON files and was solved by inheriting the StreamRequestHandler class.

From this assignment I have learned a lot about handling of strings through many different string methods like strip, split and manipulating string-lists. HTTP headers, how they work and how you structure a proper HTTP response by using them. Lastly, I have learned a lot about how you handle JSON in python through its functions and general syntax. Overall this information gives me a good base to continue building on in future assignments on encryption and security.

## 7 References

<https://code.tutsplus.com/tutorials/http-headers-for-dummies--net-8039>Downloaded 17.09.2018

<https://en.wikipedia.org/wiki/JSON>Downloaded 17.09.2018