NIS Assignment Write-up

Zachary Bresler	Justin Dorman	Chad Piha	Emil Kenguerli
BRSZAC002	DRMJUS001	PHXCHA001	KNGEMI001

1. Introduction

We were tasked to develop a secure communication system between two client applications, through the use of a server. These clients are initially expected to exchange and validate each other's certificates (which are issued by a Certification Authority trusted by both clients). The clients should then have the ability to communicate with each other by transmitting messages securely, by simulating the message confidentiality and authentication aspects of PGP (Pretty Good Privacy).

This report will describe the PGP implementation of our system in terms of authentication and confidentiality. It will then proceed to outline the choice of Crypto algorithms and how the keys will be managed. The communication connectivity model will be discussed, followed by our testing procedure and any assumptions made during development.

2. PGP Implementation

PGP provides a service to help implement authentication and confidentiality in a system where the transfer of data is required. In this assignment, we applied PGP authentication and confidentiality to a client-server application which deals with the sending and receiving of messages.

2.1 Authentication

The authentication aspect of the implementation verifies the authenticity of the message. The sender's plaintext message is captured and a hash of the message is computed (fingerprint). This hashing process enhances the performance of the encryption as it reduces the size of the message to a specific length. The private key of the sender is then used to sign the hashed message. The original message and the signed hash are then concatenated and finally, compressed.
To check for authenticity on the receiver's end, they will use the public key of the sender to decrypt the encrypted packet and get back the hash that the sender signed. At the same time, the original plaintext message is captured and hashed. The hash that the receiver has now computed is then compared with the hash obtained after the decryption. If these two hashes are identical then the message is authenticated.

2.2 Confidentiality

Confidentiality ensures that the data (message) being transferred is secure. The sender generates a secret/shared/session key that is used to encrypt the compressed packet from the authentication process. The public key of the receiver is then used to encrypt the session key, which results in increased security as only the receiver is able to decrypt the message. This is then concatenated with the encrypted data. On the receiver's side, they need to ensure the data received is confidential. The receiver uses their private key to decrypt the sender's encrypted session key. The session key will enable the receiver to decrypt the message contents. The decrypted contents are then decompressed to get back to the plaintext form.

2.3 Process

In PGP, the sender first authenticates the message and then ensures confidentiality. The receiver does the reverse of this process and first checks confidentiality and then the authenticity of the message.

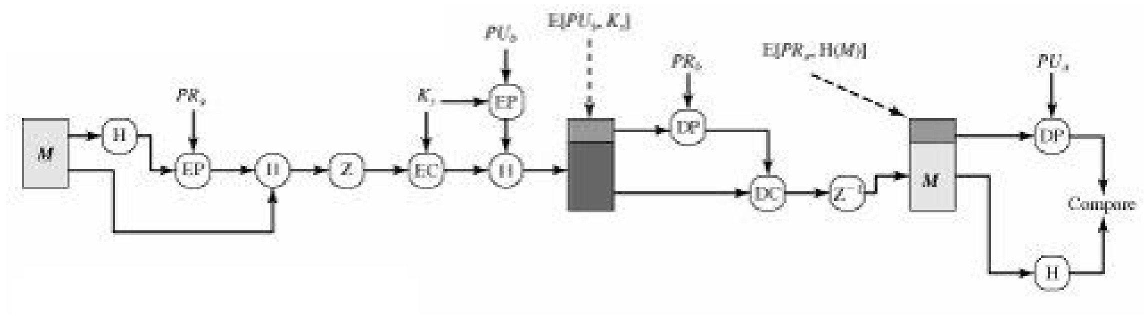

Figure 1: PGP Process including the authentication and confidentiality of messages sent

3. Choice of Crypto Algorithms

3.1 Key Generation

3.1.1 Public/Private Key Pair (asymmetric key generation)

To generate the public and private keys we used the RSA algorithm. The length of each key is 1024bits.

3.1.2 Secret/shared/session key (symmetric key generation)

To generate the secret/shared/session key we used the AES algorithm. The length of this key is 128bits.

3.2 Encryption

3.2.1 Digital Signature

The hash of the plaintext was encrypted using the client's private key and the RSA algorithm in Electronic Code Book (ECB) mode with PKCS1 padding.

3.2.2 Compressed payload

We encrypted the compressed payload using the secret/shared key and the AES algorithm in Cipher Block Chaining (CBC) mode with PKCS5 padding.

3.2.3 Secret/shared key

The RSA algorithm in ECB mode was used in conjunction with PKCS1 padding for the encryption of the secret/shared key.

4. Key Management

For the purpose of this PGP implementation, we are required to only have one communication session between a sender and a receiver. Their public and private keys are generated using asymmetric encryption generation that is discussed in section *3.1.1*, which is implemented using the KeyPairGenerator library in Java. These keys are generated upon the creation of each client. The private keys of the clients will be stored in a global variable of the respective client. The public keys will be stored in a text file where both clients can access it.
The public and private keys of the CA are generated as the server is started and will be stored in a text file for access by clients. The Session key is generated after the compression of the authenticated message (during the PGP process of sending a message). It is then encrypted and concatenated with the encrypted message.

For the purpose of this assignment, we have granted easy access to the CA's public and private keys as well as having a pre-created certificate for each client.

5. Communication Connectivity Model

For the transmission of messages, we decided to use the chatroom application that we created for an assignment in 3rd year. The application allows clients to send each other messages through a server over TCP. We adapted this older application to accommodate the new requirements laid out in this assignment. We removed all the threading that allowed multiple clients from connecting and communicating at the same time. This was not necessary for the scope of this assignment, and it added additional complications. Hence, we changed it so that only one message can be sent and received at a time between two clients. Additionally, we altered the application so that there is a dedicated sender and receiver client. We did this because since we are only using one class for the clients (both the sender and receiver), it would complicate the asymmetric key generations (as it would be difficult to determine which key pairs belong to which client).

Our framework consists of the following classes: the Client class, Client.java which embeds PGPSend and PGPReceive, containing the relevant PGP methods. Furthermore, we have a Server class ChatServer.java, which embeds the classes ServerConnection.java and ChatGroup.java. The classes have to be run on the console in three separate terminals; two terminals for the two clients, and one terminal for the server. The Server class is required to be compiled prior to the client.

When the server class is run, it will wait and listen for clients trying to connect to it. At this point, the CA public and private key pair will be generated and written to text files that are accessible by the clients. The default server IP address is localhost, and the default port number is 60123. The clients and server will therefore only be able to connect to each other on the same device. Port number 60123 is a non-reserved port and should always be free for the server to use. When a client class is run (and the server is already running), the server will accept and set up the connection. The client will then be asked if they will be a sender or receiver for that specific session. There must be one sender client and one receiver client. The client will then be prompted to input a username which will identify them. Once the user inputs their username, their public and private key pair will be generated. The private key will be stored in the global variable of the client instance, and the public key will be written to the appropriate text file (sender or receiver client public key). The client's certificate will then be generated, by concatenating the client's public key with the client's username. The certificate will then be written to the respective text file (sender certificate or receiver certificate). If the client is a sender, the senders certificate will then be signed by hashing said certificate using SHA-256 and then signing it with the CA's private key. The signed certificate will then be written to a text file that can be accessed by the receiver. Since the clients are dedicated senders or receivers, the certificate will only be signed if the client is a sender.

The user will then be asked whether they want to create or join a chat room. A user can create a chat room by providing it with a name and password. Following this, another user will be able to join that chat room by inputting the credentials. We implemented this as an additional feature, to add an extra layer of security to the communication between the two clients. This adds another authentication step, as a private chat room gets created. Both clients are required to be present in the same chat room before communication can begin. If the client is a sender, they will be requested to enter a message that will be encrypted and sent to the other client in the chat room. Once the user enters the message and hits enter, the message will go through the entire PGP Authentication and Confidentiality process involved with sending a message. Ciphertext is sent as byte arrays. The ciphertext will be sent to the server on localhost address and port 60123, and thereafter be directed to the other client in the chat room. The receiver client then needs to hit enter to receive the message. Once the user hits enter, the sender client will be verified, prior to the ciphertext being unpacked. The public key of the CA will be used to decrypt the signature of the certificate to obtain the hash of the unsigned certificate. This will be compared to the hash that is computed on the unsigned certificate. This process is conducted to ensure that the sender is legitimate. If the authentication is successful, the PGP process of receiving a message will begin, with the signature being verified and the message being decrypted back to plain text. Else, communication will cease.

Output statements describing the PGP process will be displayed in the console of the respective clients, where appropriate. The encrypted and decrypted text at each stage will also be displayed, in the form of a string.

Upon receiving the byte[] packet, in order to read it, we sent the size of the final packet (along with the packet itself), and then created a buffer which completely reads in the packet before continuing.

The system is set up to have a continuous connection. Therefore, after the first message has been sent to the server and received by the other client following the PGP Authentication and Confidentiality process, both clients will be able to send and receive messages again (depending on their role of being a sender or receiver). If the user wants to end the connection for one of the clients, they should enter '/q' when prompted. This will result in the connection to the server being closed. Irrespective of how many client connections are closed, the server doesn't terminate and keeps its connection open for other clients to connect and create/join a chat room. If needed, the server has to be manually killed.

For the cryptography parts of the clients, we used java's java.security and javax.crypto libraries. For the compression, the Deflater and Inflater Java classes were used to compress and decompress the byte arrays containing the data.
We used Java's Socket and ServerSocket classes to handle the TCP connection. Furthermore, we used DataInputStream and DataOutputStream to read and write messages between the clients and the server.

6. Testing procedure

In the system, we ensured that each process of the PGP implementation is logged. A debug message is printed to the console of the respective client to clearly identify what is happening and exactly how the system is executing a start process, a complete process, and/or a fail process. Doing so will create transparency and clarity on exactly how and when the system starts, completes and/or fails a process. Encrypted and decrypted text, in String form, along with the packet size will be printed to the console to ensure the implementation on the data inputted is correct, and that packets do not lose data upon transmission.

7. Assumptions

In the implementation of the system we have assumed the following:

- The system will be run on a localhost and not over a public network
- The storage of keys need not be protected due to its execution on a localhost
- The system will only include two clients and a server
- One message is sent and received at a time as multithreading is out of scope
- Java's security libraries are implemented securely and are expected to work as documented
- The public and private key of the Certificate Authority (CA) will be directly accessible to both clients

The README included in the submission provides a description on how to execute the programs