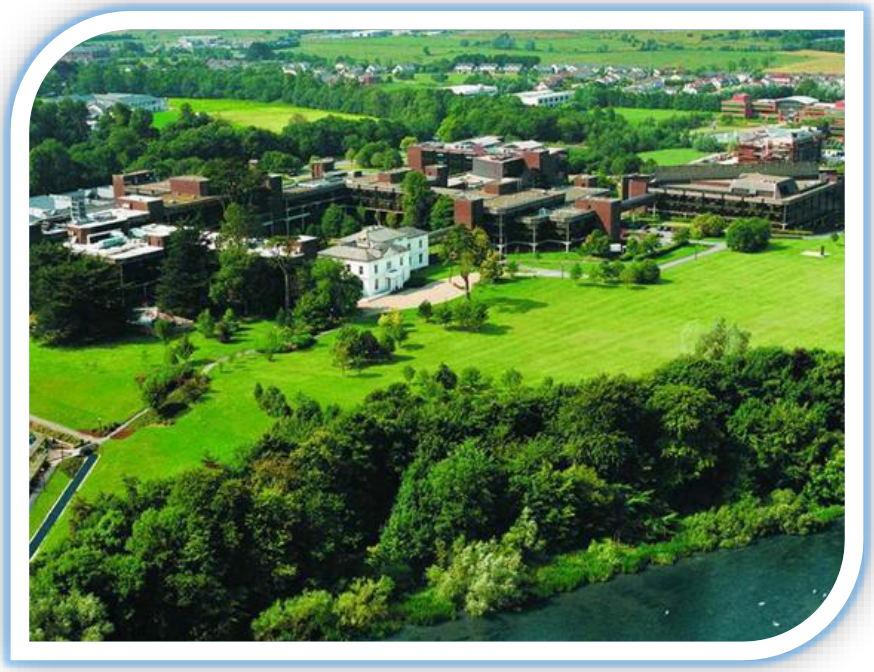
UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Secure and Insecure Chat Implementation

(Department of Electronic and Computer Engineering)



Project: Final Master Thesis Project

Student: Aleix Masdeu Medina

Profesor: Thomas Newe

Date: 15/12/2015

CONTENTS

1. INTRODUCTION

Internet is not a single network, but a worldwide collection of loosely connected networks that are accessible by using computers, smartphones or tablets, in a variety of ways, to anyone with a computer and a network connection. Thus, individuals and organizations can reach any point on the internet without regard to national or geographic boundaries or time of day.

However, along with the convenience and easy access to information come risks. Among them are the risks that valuable information will be lost, stolen, changed, or misused. If information is recorded electronically and is available on networked computers, it is more vulnerable than if the same information is printed on paper and locked in a file cabinet. Intruders do not need to enter an office or home, they may not even be in the same country. They can steal or tamper with information without touching a piece of paper or a photocopier. They can also create new electronic files, run their own programs, and hide evidence of their unauthorized activity.

As it can be seen, securities issues are of a high relevance because Internet is a network of networks in which the information sent can travel over a secure or insecure network. Nowadays, messaging or chat applications are used every day by millions of users to exchange text messages, pictures, videos, contacts, documents and so on. It is translated to a huge amount of data travelling by the network in a transparent way for the users involved.

For all this reasons, applying security to the information sent is really important to guarantee data protection. In our case, it has been desired to implement a chat application in which a Server and a Client can exchange, for instance, messages, pictures, documents, songs, and videos. In this application, both entities should have the possibility of sending the information over a secure or insecure channel. If the secure way is chosen some cryptographic protocols, such as symmetric and asymmetric cryptography methods, will be applied to guarantee the most important security aspects: confidentiality, security, reliability and no repudiation. In the insecure way, any cryptographic method is applied and the information will be sent in clear.

To be able to exchange messages and several types of documents, it is desired to implement an interface using JAVA language in which an option has to offer to the user the possibility of switching between the secure and insecure channel. Moreover, this interface should have a box in which to store the information sent and received and a copy of this information should be stored in a backup file for further treatment.

After that, it is desired to compare the packets sent and received over the local loop using Wireshark tool to verify how the information is sent when using the secure and insecure way.

2. BACKGROUND

A communication is encrypted only when transmitter and receiver are able to extract the message information. Therefore anyone outside communication will be able to see only gibberish and the message content will be completely hidden.

Although we can think of Alan Turing, Claude Shannon or the NSA as references in the field of cryptography (and, of course, they are), this art goes back much further in time. Message encryption has been practiced for over 4,000 years and, indeed, the origin of the word cryptography is found in the Greek krypto (hidden), and graphos (writing).

The Spartans were also using cryptography to protect their messages, specifically, a technique known as transposition cipher consisting of a parchment roll on a stick that served to sort letters and displayed the message. To decrypt, the receiver must have a scytale with the same diameter as the issuer had used (symmetric cryptography) because it was the only way to display the message in the same way.

Over time, cryptography has become a key part of armies worldwide. During the French Wars of Religion (which confronted the state with the Huguenots), decode enemy messages became a tactical objective and Antoine Rossignol became, in 1628, one of the most important cryptographers in France.

Cryptography was the key during World War II (1939-1945) because it changed the course of the war. Germany had mastered the North Atlantic with its submarine fleet, and communications were indecipherable because of the Enigma machine. Besides the traditional fronts and battles between the armed forces it had opened a new battlefield: break enemy communications. A task that the Allies commissioned a group of mathematicians, engineers and physicists who worked from Bletchley Park facilities and among them was Alan Turing.

Perhaps Alan Turing's work and the Allies is the best known work on cryptography during World War II. However it was not the only one. The encryption communications marked the conflict and a varied set of techniques used to prevent the enemy from intercepting communications. United States, for example, rescued a technique already used successfully during the First World War and, instead of resorting to complex encryption algorithms, opted for use as code the language of Native Americans.

After World War II, cryptography made a big leap by Claude Shannon, known as the father of the theory of communication. In 1948, Shannon, who worked at Bell Labs published "A Theory of Secrecy Systems Communications" a seminal paper in which coding techniques were modernized to transform them into advanced mathematical processes. While the frequency analysis was based on statistics, Shannon proved mathematically that fact and introduced the concept of "unicity distance" that marked the length of ciphered text is needed to be able to decrypt it.

But it wouldn't be until March 17, 1975 when would reach the first "public preview" (not dependent on the NSA) linked to the cryptography world. The International Business Machines

Company (IBM), developed the encryption algorithm Data Encryption Standard (DES), and two years later, would become a Federal Information Processing Standard (FIPS 46-3) and its use spread worldwide. In 2001, DES ceded his post to Advanced Encryption Standard (AES) which, after 5 years of review, became a standard.

The second major public developments also had its origin in the 70. All systems discussed before were symmetrical and it means that both sender and receiver must handle the same code. So there was a necessity of being synchronized. However, Whitfield Diffie and Martin Hellman laid the foundations of asymmetric cryptography (public key and private key) in the article "New Directions in Cryptography" published in 1976. Asymmetric cryptography today is fundamental to transactions conducted through the Internet.

3. SECURITY ISSUES

In this section, it is explained the technologies used in the project concerning to security issues. First, asymmetric cryptography is explained, in which a public key and private key are used to encrypt and decrypt messages and files.

After that, symmetric encryption is explained as it is used in the project in combination with asymmetric cryptography to generate a session key which is later used to encrypt the information before sending and decrypt the information after receiving.

Then, Diffie-Hellman is explained because it is used to send a pre master secret between both entities, Client and Server which is used to generate the session key.

In all this security methods, the man in the middle attack (MITM) is explained as the weaknesses of the analyzed method and then a solution is proposed to overcome this attack.

3.1 ASYMMETRIC CRIPTOGRAPHY [1]

Asymmetric cryptography or public-key cryptography **(Figure 1)** is a cryptography method in which a pair of keys is used to encrypt and decrypt a message so that it arrives securely.

Initially, a network user receives or generates a public and private key pair. Any other user who wants to send an encrypted message can get the intended recipient's public key from a public directory. They use this key to encrypt the message, and they send it to the recipient. When the recipient gets the message, they decrypt it with their private key, which no one else should have access to.

Two of the best-known uses of public-key cryptography are:

1. **Public-key encryption**, in which a message is encrypted with a recipient's public key. The message cannot be decrypted by anyone who does not possess the matching private key, who is thus presumed to be the owner of that key and the person associated with the public key. This is used in an attempt to ensure confidentiality.

2. **Digital signatures**, in which a message is signed with the sender's private key and can be verified by anyone who has access to the sender's public key. This verification proves that the sender had access to the private key, and therefore is likely to be the person associated with the public key. This also ensures that the message has not been tampered with, as any manipulation of the message will result in changes to the encoded message digest, which otherwise remains unchanged between the sender and receiver. With digital signatures it can be assured data integrity and no repudiation.

The central problem with the use of public-key cryptography is confidence that a particular public key is authentic, in that it is correct and belongs to the person or entity claimed, and has not been tampered with or replaced by a malicious third party. The usual approach to this

problem is to use a **public-key infrastructure (PKI)**, in which one or more third parties, known as certificate authorities, to certify ownership of key pairs.

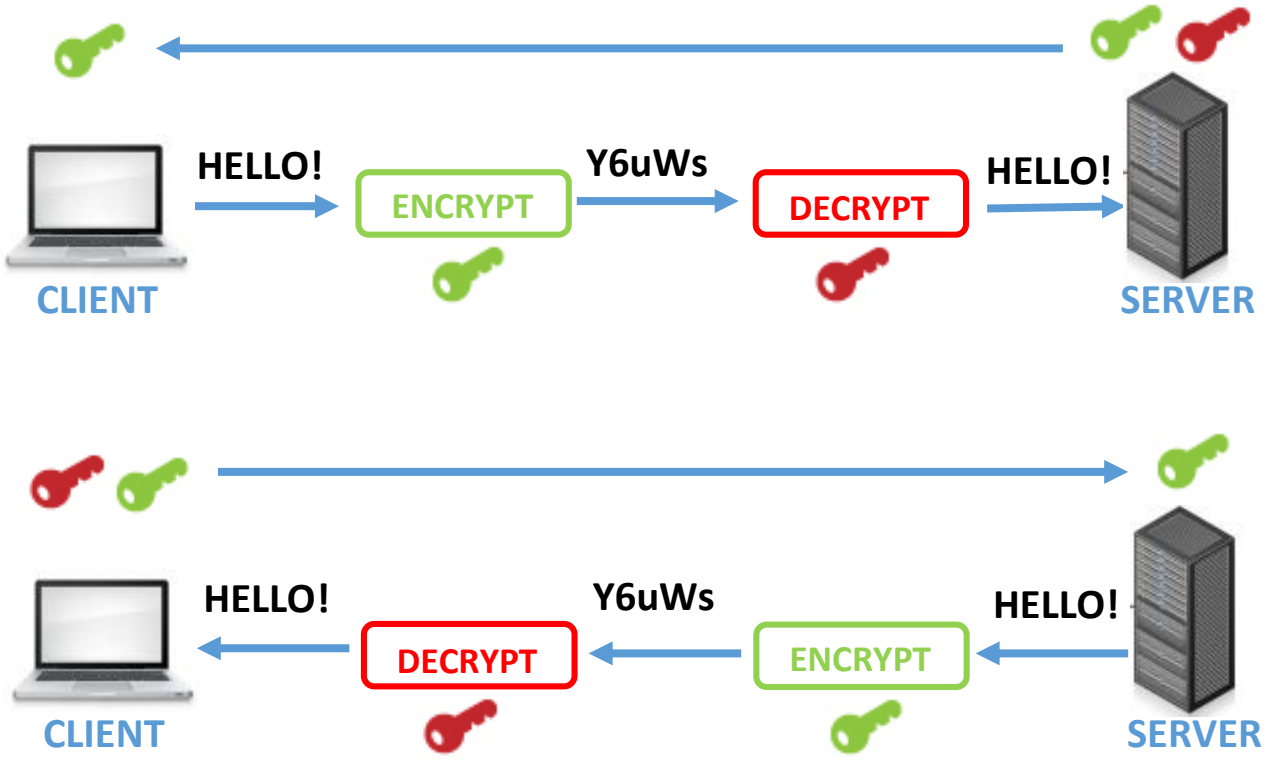


Figure 1 - Server and Client public key exchange

3.1.1 MAN IN THE MIDDLE ATTACK

Suppose Client wishes to communicate with Server. Meanwhile, Mallory wishes to intercept the conversation to eavesdrop and optionally deliver a false message to Server.

The man in the middle attack starts when Client asks Server for his public key and when Server sends his public key to Client and Mallory is able to intercept it. Then, Malory sends a forged message to Client hat claims to be from Server but instead includes Mallory's public key.

Client, believing this public key to be Server's public key, encrypt the message with Mallory's public key and sends the enciphered message back to Server. Mallory's intercepts again the message and decrypt the message with his private key as the message is encrypted with his public key. After that, Mallory's encrypts again the message with Server's public key and sends the message to the Server. When Server receive the message it believe it came from Client. The steps are as follows and can be seen in the next picture **(Figure 2)**:

1. Client sends a message to Server, which is intercepted by Mallory:

   Client: "Hi Server, it's Client. Give me your key." → **Mallory**

2. Mallory relays this message to Server and Server cannot tell it is not really from Client:

   Mallory: "Hi Server, it's Client. Give me your key." → **Server**

3. Server responds with his encryption key:

   Mallory ← [Server's public key] **Server**

4. Mallory replaces Server's key with her own, and relays this to Client, claiming that it is Server's key:

   Client ← [Mallory's key] **Mallory**

5. Client encrypts a message with what she believes to be Server's key, thinking that only Server can read it:

   Client: "Meet me at the bus stop!" [Encrypted with Mallory's key] → **Mallory**

6. However, because it was actually encrypted with Mallory's key, Mallory can decrypt it, read it, modify it (if desired), re-encrypt with Server's key, and forward it to Server:

   Mallory "Meet me at the shopping center!" [Encrypted with Server's key] → **Server**

7. Server thinks that this message is a secure communication from Client.

This attack can be done in one of the sides or in both sides. If Mallory wants Client to believe he is Server, he only has to repeat the process again.

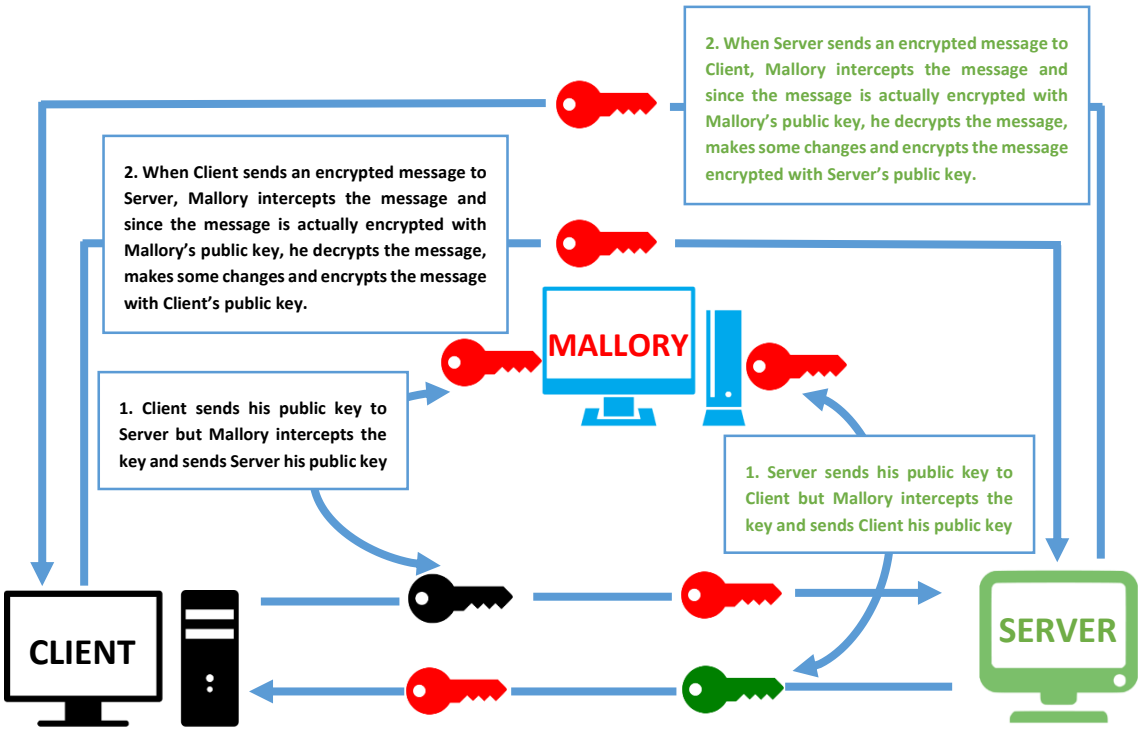


Figure 2 - Public key Criptography attack

This example shows the need for both Client and Server to have some way to ensure that they are truly using each other's public keys, rather than the public key of an attacker. Otherwise, such attacks are generally possible, in principle, against any message sent using public-key technology. Fortunately, there are a variety of techniques that help defend against MITM attacks.

All cryptographic systems that are secure against MITM attacks require an additional exchange or transmission of information over some kind of secure channel. Many key agreement methods have been developed, with different security requirements for the secure channel. Various defenses against MITM attacks use authentication techniques that include:

1. DNSSEC: Secure DNS extensions

2. Public key infrastructures: Transport Layer Security is an example of implementing PKI over TCP protocol. This is used to prevent man in the middle attack over a secured HTTP connection on internet. Client and Server exchange PKI certificates issued and verified by a common certificate authority.

- **2.1 PKI mutual authentication:** The main defense in a PKI scenario is mutual authentication. In this case applications from both Client and Server mutually validates their certificates issued by a common root certificate authority. Virtual Private Networks do mutual authentication before sending data over the created secure tunnel, however mutual authentication over internet for HTTP connections are seldom enforced.

3. Stronger **mutual authentication**, such as:

- **3.1 Secret keys** (which are usually high information entropy secrets, and thus more secure), or
- **3.2 Passwords** (which are usually low information entropy secrets, and thus less secure.

3.1.2 PUBLIC-KEY INFRASTRUCTURE (PKI)

Enveloped Public Key Encryption (EPKE) is the method of applying public-key cryptography and ensuring that an electronic communication is transmitted confidentially, has the contents of the communication protected against being modified (communication integrity) and can't be denied from having been sent (non-repudiation). This is often the method used when securing communication on an open networked environment such by making use of the Transport Layer Security (TLS) or Secure Sockets Layer (SSL) protocols.

EPKE **(Figure 3)** consists of a two-stage process that includes Public Key Encryption (PKE) and a digital signature. Both Public Key Encryption and digital signatures make up the foundation of Enveloped Public Key Encryption.

For EPKE to work effectively, it is required that:

1. Every participant in the communication has their own unique pair of keys. The first key that is required is a public key and the second key that is required is a private key.

2. Each person's own private and public keys must be mathematically related where the private key is used to decrypt a communication sent using a public key and vice versa. Some well-known asymmetric encryption algorithms are based on the RSA cryptosystem.

3. The private key must be kept absolutely private by the owner, though the public key can be published in a public directory such as with a certification authority.

To send a message using EPKE, the sender of the message must first sign the message using their own private key, which ensures **non-repudiation** of the message. The sender then encrypts their digitally signed message using the receiver's public key thus applying a digital envelope to the message. This step ensures confidentiality during the transmission of the message. The receiver of the message then uses their private key to decrypt the message thus removing the digital envelope and then uses the sender's public key to decrypt the sender's digital signature. At this point, if the message has been unaltered during transmission, the message will be clear to the receiver.

Due to the computationally complex nature of RSA-based asymmetric encryption algorithms, the time taken to encrypt a large documents or files to be transmitted can take an increased amount of time to complete. To speed up the process of transmission, instead of applying the sender's digital signature to the large documents or files, the sender can rather hash the documents or files using a cryptographic hash function and then digitally sign the generated hash value, therefore enforcing non-repudiation. Hashing is a much faster computation to complete as opposed to using an RSA-based digital signature algorithm alone. The sender would then sign the newly generated hash value and encrypt the original documents or files with the receiver's public key. The transmission would then take place securely and with confidentiality and non-repudiation still intact. The receiver would then verify the signature and decrypt the encrypted documents or files with their private key.

The problem of using public key cryptography is the key size needed to avoid attacks which reduces the protocol efficiency and for this reason public key cryptography is used with symmetric cryptography to exchange a session key used later for both encryption and decryption.

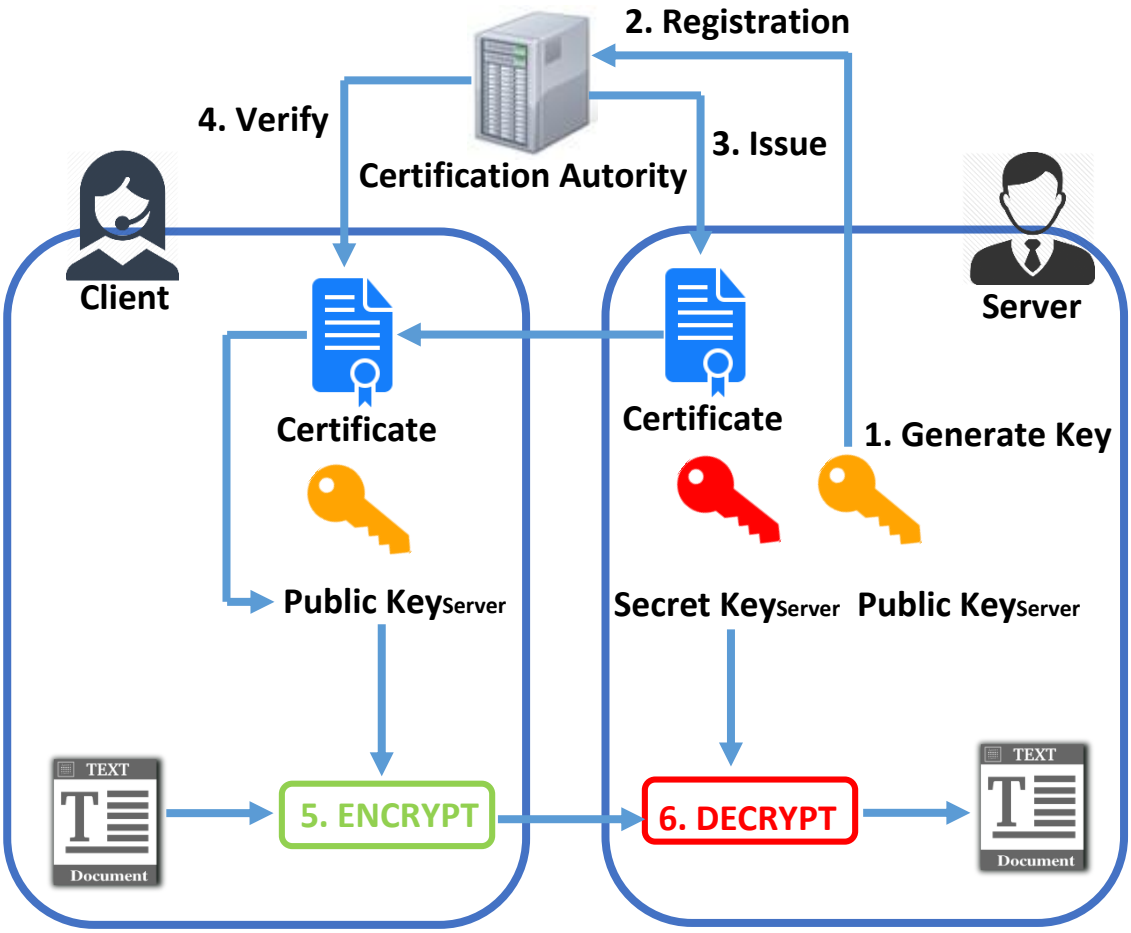


Figure 3 - Public key infrastucture

3.2 SYMMETRIC CRIPTOGRAPHY [2]

Symmetric cryptography **(Figure 4)**, also called secret key cryptography, is a cryptographic method where the same key is used to encrypt and decrypt messages. The two communicating parties have to agree in advance on the key used. Symmetric encryption is the oldest and best-known technique. A secret key, which can be a number, a word, or just a string of random letters, is applied to the text of a message to change the content in a particular way. This might be as simple as shifting each letter by a number of places in the alphabet. As long as both sender and recipient know the secret key, they can encrypt and decrypt all messages that use this key.

The problem with secret keys is exchanging them over the Internet or a large network while preventing them from falling into the wrong hands. Anyone who knows the secret key can easily decrypt the message.

One answer is asymmetric encryption, in which there are two related keys. A public key is made freely available to anyone who might want to send you a message. A second, private key is kept secret, so that only you know it.

Any message (text, binary files, or documents) that are encrypted by using the public key can only be decrypted by applying the same algorithm, but by using the matching private key. Any message that is encrypted by using the private key can only be decrypted by using the matching public key.

This means that you don't have to worry about passing public keys over the Internet (the keys are supposed to be public). A problem with asymmetric encryption, however, is that it is slower than symmetric encryption. It requires far more processing power to both encrypt and decrypt the content of the message.

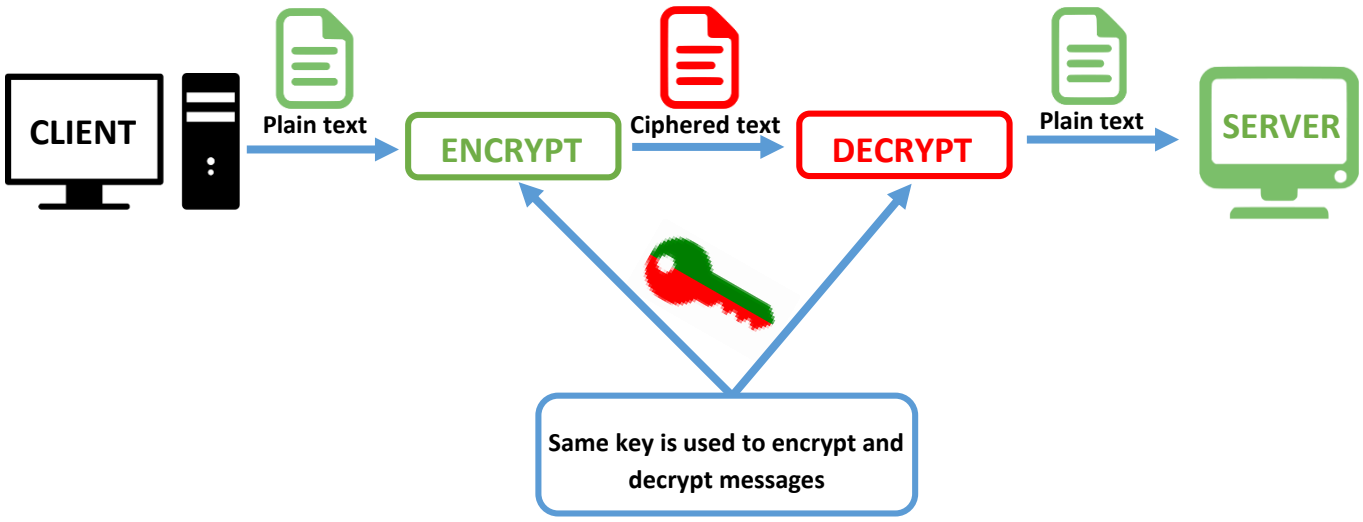


Figure 4 - Symmetric cryptography

3.3 DIFFIE-HELLMAN [3]

Diffie-Hellman key exchange is a specific method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols originally conceptualized by Ralph Merkle. D-H is one of the earliest practical examples of public key exchange implemented within the field of cryptography. Traditionally, secure encrypted communication between two parties required that they first exchange keys by some secure physical channel, such as paper key lists transported by a trusted courier. The Diffie-Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

Diffie-Hellman is used to secure a variety of Internet services. However, research published in October 2015 suggests that the parameters in use for many D-H Internet applications at that time are not strong enough to prevent compromise by very well-funded attackers, such as the security services of large governments.

Although Diffie-Hellman key agreement itself is a non-authenticated key-agreement protocol, it provides the basis for a variety of authenticated protocols, and is used to provide forward secrecy in Transport Layer Security's ephemeral modes (referred to as EDH or DHE depending on the cipher suite).

The method was followed shortly afterwards by RSA, an implementation of public-key cryptography using asymmetric algorithms.

3.3.1 IMPLEMENTATION

To implement Diffie-Hellman **(Figure 5)**, the two end users Client and Server, while communicating over a channel they know to be private, mutually agree on positive whole numbers p and q, such that:

- p is a prime number

- q is a generator of p.

The generator *q* is a number that, when raised to positive whole-number powers less than p, never produces the same result for any two such whole numbers. The value of p may be large (should be the same size as the public key used) but the value of q is usually small (1bit).

Once Client and Server have agreed on p and q in private, they choose positive whole-number personal keys a and b, both less than the prime-number modulus p. Neither user divulges their personal key to anyone.

In the next step, Client and Server compute public keys a* and b* based on their personal keys according to the formulas:

$$a^* = q^a \bmod p$$

and

$$b^* = q^b \bmod p$$

The two users can share their public keys a* and b* over a communications medium assumed to be insecure, such as the Internet or a corporate wide area network (WAN). From these public keys, a number *x* can be generated by either user on the basis of their own personal keys. Client computes x using the formula:

$$x = (b^*)^a \bmod p$$

Server computes x using the formula:

$$x = (a^*)^b \bmod p$$

The value of x turns out to be the same according to either of the above two formulas. However, the personal keys a and b, which are critical in the calculation of x, have not been transmitted over a public medium. Because it is a large and apparently random number, a potential hacker has almost no chance of correctly guessing x, even with the help of a powerful computer to conduct millions of trials. The two users can therefore, in theory, communicate privately over a public medium with an encryption method of their choice using the decryption key x.

The most serious limitation of Diffie-Hellman in its basic or "pure" form is the lack of authentication. Communications using Diffie-Hellman all by itself are vulnerable to man in the middle attacks. Ideally, Diffie-Hellman should be used in conjunction with a recognized authentication method such as digital signatures to verify the identities of the users over the public communications medium. Diffie-Hellman is well suited for use in data communication but is less often used for data stored or archived over long periods of time.

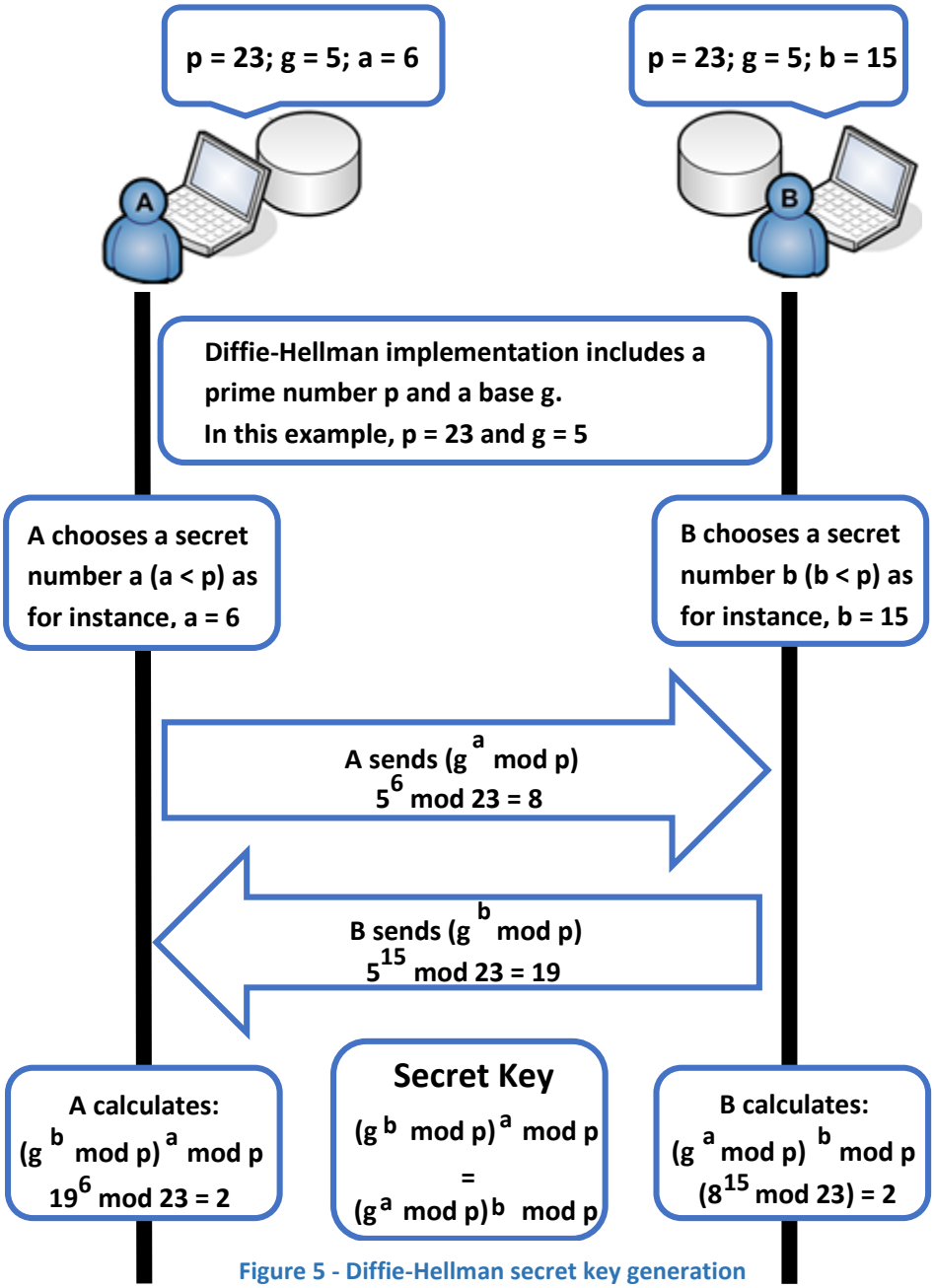


Figure 5 - Diffie-Hellman secret key generation

3.3.2 MAN IN THE MIDDLE ATTACK

The protocol is sensible to active man in the middle attacks **(Figure 6)**. If communication is intercepted by a third party, it can be passed by the sender to the receiver and vice versa face because it does not have any mechanism to validate the identity of the participants involved in the communication. Thus, the man in the middle could agree a key with each participant and relay data between them, listening to the conversation both ways. Once the attacker established

symmetrical communication must follow amid intercepting and modifying traffic to not realize. Note that for the attack to be operating the attacker must know the symmetric encryption method to be used. Concealment based on symmetric encryption algorithm does not meet the principles of Kerckhoffs (the effectiveness of the system should not depend on its design remains secret).

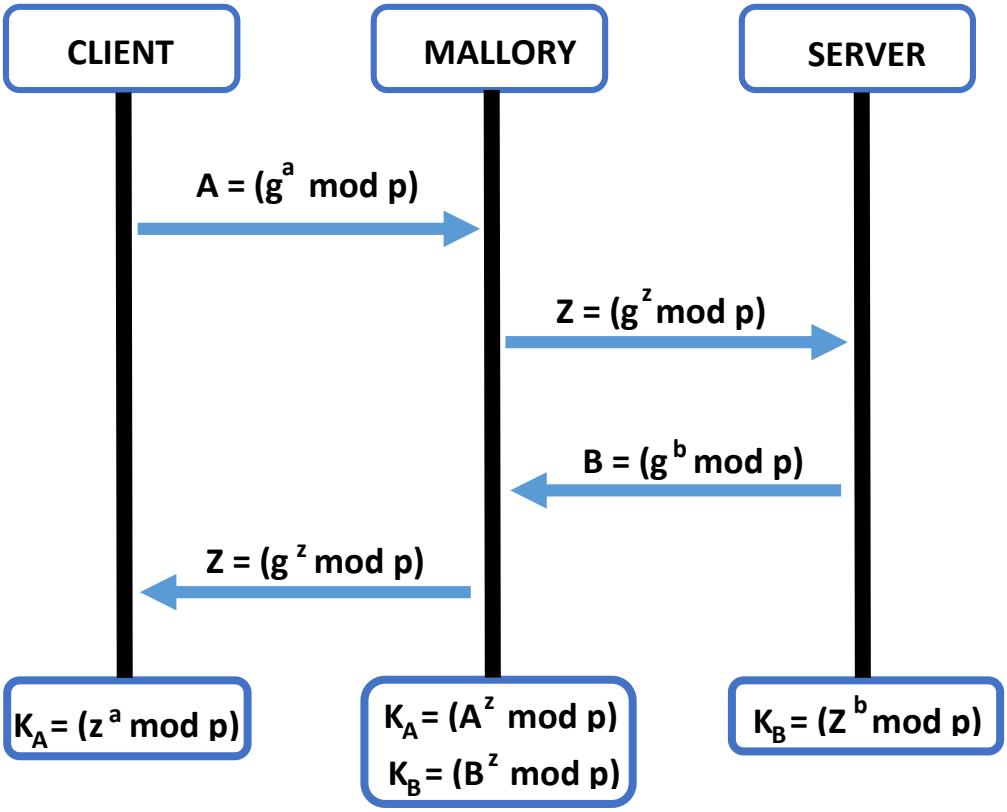


Figure 6 - Diffie-Hellman MITM

To prevent this type of attack is often used one or more of the following techniques:

1. Timing control

2. Pre-authentication of the parties: For example use underlying layer authentication protocol. We could first establish a TLS connection on that layer and apply the Diffie-Hellman.

3. Content Authentication: For example we could use MAC on the message content.

4. Encrypting public keys with a public key algorithm (asymmetric), avoiding the problem of Man-in-the-middle.

3.4 SSL [4]

SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol but rather two layers of protocols, as illustrated in **Figure 7**.

The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the hypertext transfer protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are defined as part of SSL: the Handshake Protocol **(Figure 8)**, the Change Cipher Spec Protocol and the Alert Protocol.

Two important SSL concepts are the SSL session and the SSL connection, which are defined as follows:

- **Connection:** A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Session:** An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

Between any pair of parties (applications such as HTTP on client and server), there may be multiple secure connections.

There are actually a number of states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states.

A session state is defined by the following parameters (definitions from the SSL specification):

- **Session identifier:** An arbitrary byte sequence chosen by the server to identify an active or presumable session state.
- **Peer certificate:** The X509 certificate of the peer. This element of the state may be null.
- **Compression method:** The algorithm used to compress data prior to encryption.
- **Cipher spec:** Specifies the bulk data encryption algorithm (such as null, DES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash size.
- **Master secret:** Forty eight byte secret shared between the Client and Server.
- **Is presumable:** A flag indicating whether the session can be used to initiate new connections.

A connection state is defined by the following parameters:

- **Server and Client random:** Byte sequences that are chosen by the Server and Client for each connection.
- **Server write MAC secret:** The secret key used in MAC operations on data sent by the Server.

- **Client write MAC secret:** The secret key used in MAC operations on data sent by the Client.
- **Server write key:** The conventional encryption key for data encrypted by the server and decrypted by the client.
- **Client write key:** The conventional encryption key for data encrypted by the client and decrypted by the server.
- **Initialization vectors:** When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphered text block from each record is preserved for use as the IV with the following record.
- **Sequence numbers:** Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero.

The SSL Record Protocol provides two services for SSL connections:

- **Confidentiality:** The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
- **Message Integrity:** The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

When the Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled and then delivered to higher-level users.

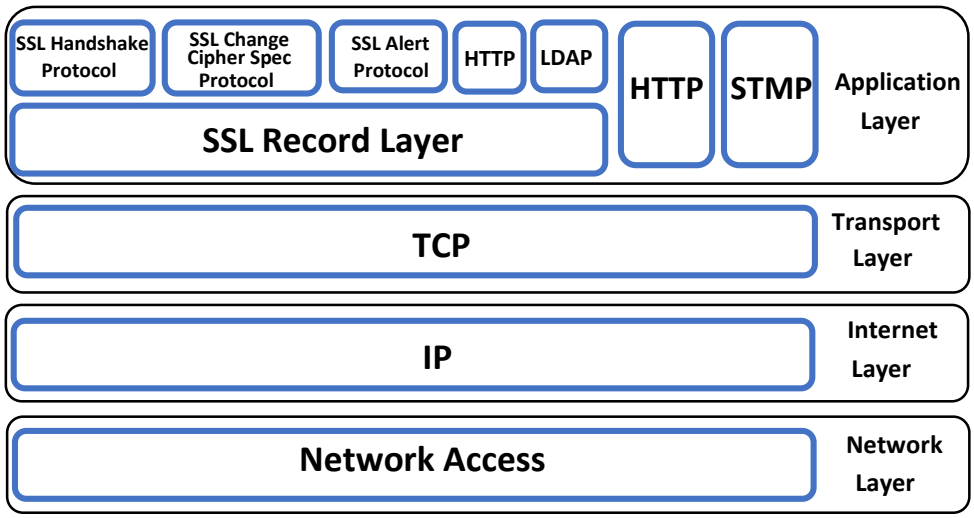


Figure 7 - SSL protocol stack

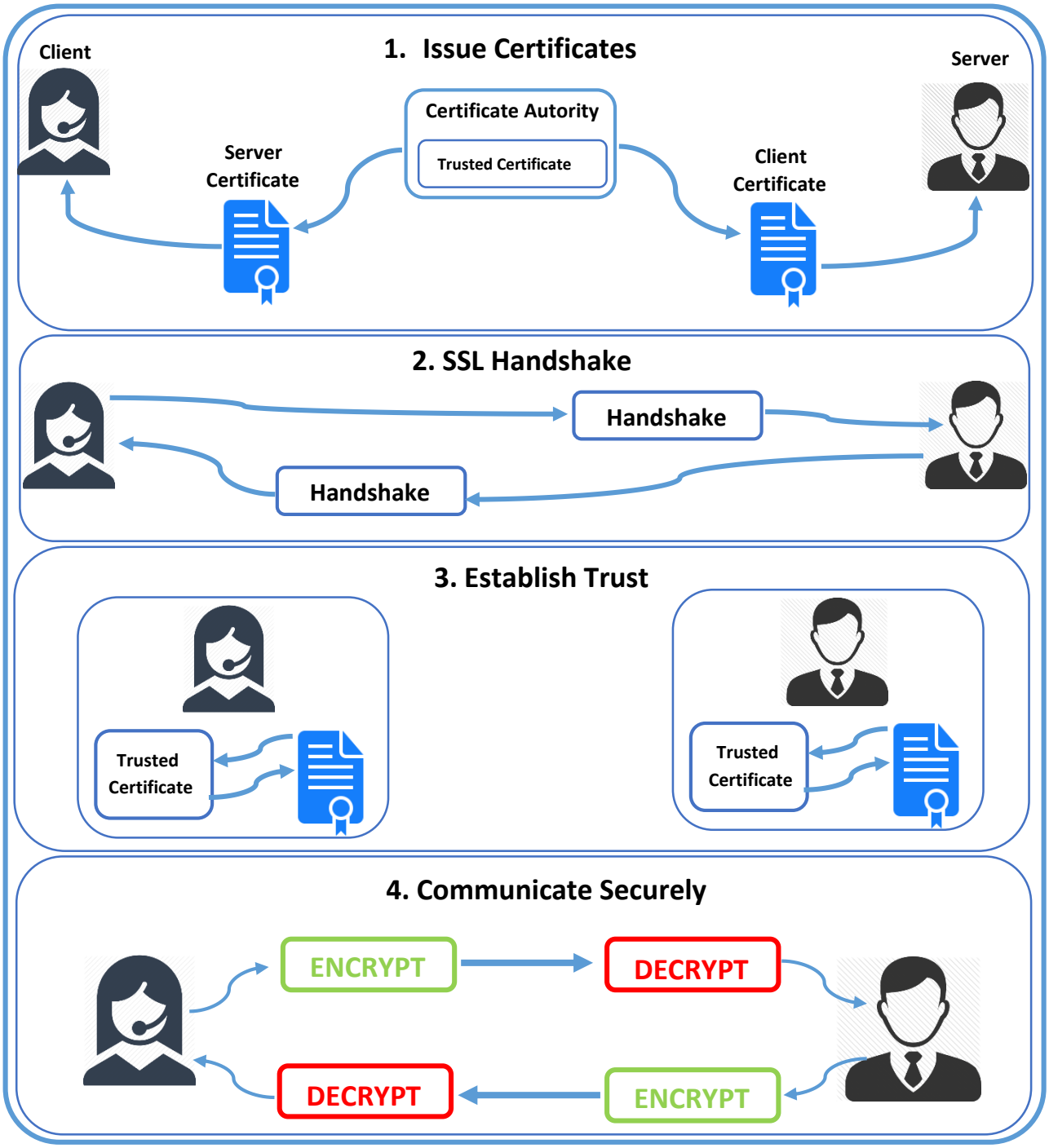


Figure 8 - SSL procedure

3.4.1 SOCKET [5]

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

Socket designates an abstract concept by which two programs (possibly located in different computers) can exchange any data stream, generally reliable and orderly manner.

The term socket is also used as the name of an application programming interface (API) for the Internet Protocol Suite TCP / IP, usually provided by the operating system.

Internet sockets are the mechanism for delivery of data packets from the network card to the appropriate threads or processes. A socket is defined by a pair local and remote IP addresses, a transport protocol and a pair of numbers of local **(Figure 9)** and remote **(Figure 10)** port.

For two programs can communicate with each other is necessary that certain requirements are met:

1. A program to be able to locate the other.

2. Both programs are able to interchange any sequence of bytes, that is, data relevant to its purpose.

For this, two resources that originate the concept of socket are necessary:

1. A couple of network protocol addresses (IP address, if the TCP / IP protocol is used), identifying the source computer and the remote:

2. To do the two resources that originate the concept of socket are necessary. A pair of port numbers that identify a computer within each program.

The sockets allow you to implement a client-server architecture. Communication should be initiated by a software program called "client". The second program waits for the other to initiate communication, for this reason is called "server" program.

A socket is a process or thread existing on the client machine and the server machine, used ultimately for the server and the client program to read and write information. This information will be transmitted by different network layers.

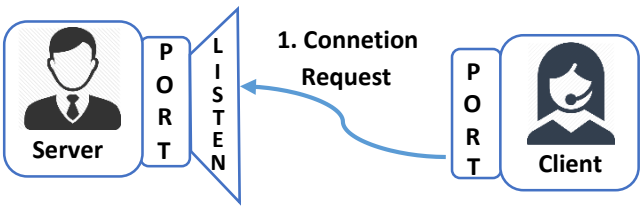


Figure 10 - Socket connection request procedure

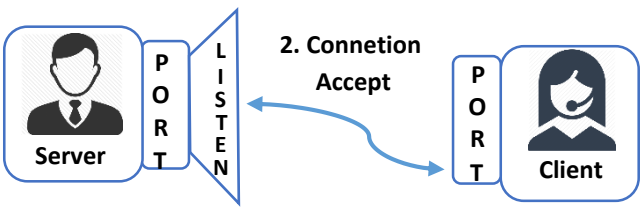


Figure 9 - Socket connection accept procedure

4. TECHNOLOGIES

In this chapter it is explained the related technologies involved in the project development. As it can be seen below, JAVA language has been used to develop project's code, Eclipse is the program used to develop the project, Keytool is used to generate a pair of certificates for both Client and Server and Wireshark is used to analyze the packets and the differences between both secure and insecure channels. Finally, Model View Controller technology is explained in detail because it has been used to structure the code, separating the different created classes to maintain the scalability and feasibility to changes.

4.1 JAVA [6]

Java is a programming language object oriented which was specifically designed to have as few implementation dependencies as possible. His intention is to enable application developers to write the program once and run it on any device.

The Java programming language was originally developed by James Gosling at Sun Microsystems (which was acquired by Oracle Corporation) and released in 1995 as a key component of the Java platform from Sun Microsystems. Its syntax derive largely from C and C ++ but has fewer low-level utilities than any of them. Java applications are typically compiled to bytecode (Java class) that can run on any Java (JVM) virtual machine regardless of the underlying computer architecture.

The company Sun developed the original reference implementation for Java compilers, virtual machines, and class libraries in 1991 and was first published in 1995.

The Java language was created with five main objectives:

1. The paradigm of object-oriented programming should be used.
2. It should allow execution of the same program on multiple operating systems.
3. It should include default support for networking.
4. It should be designed to execute code on remote systems securely.
5. It should be easy to use and take the best from other object-oriented languages like C ++.

The first feature, object oriented ("OO"), relates to a programming method and language design. Although there are many interpretations for OO, the first idea is to design the software so that various types of data used are attached to their operations. Thus, data and code (functions or methods) are combined into entities called objects. An object can be seen as a package containing the "behavior" (the code) and the "state" (data). Often, changing a data structure implies a change in the code that operates on them, or vice versa. This separation into coherent and independent objects provides a more stable environment for the design of a software system base. The goal is to make large projects are easier to manage and handle, improving their quality and, as a result, reducing the number of failed projects. Another of the great promises

of object-oriented programming is creating more generic entities (objects) that enable software reuse between projects, one of the fundamental premises of Software Engineering.

The second characteristic, platform independence, means that programs written in Java can also run on any hardware. This is the meaning of being able to write a program once and can run on any device, such as Java follows the axiom, "write once, run anywhere".

For this, the source code written in Java language is compiled to generate a code known as "bytecode" (specifically Java bytecode). This part is "halfway" between source code and machine code understood by the target device. The bytecode is then executed on the Java virtual machine (JVM), a program written in native code of the destination (which is the one that understands your hardware) which interprets and executes the code platform. Furthermore, additional libraries are provided to access the features of each device (such as graphics, execution by threads or threads, and the network interface) as one. It should be noted that, although no explicit compilation stage, the generated bytecode is interpreted or converted to native machine code instructions for the JIT (Just in Time).

The JRE (Java Runtime Environment, or Runtime Environment for Java) is required to run any application developed for Java platform software. The end user uses the JRE as part of the software or plug-ins (or connectors) packets in a Web browser. Sun also provides the Java 2 SDK or JDK (Java Development Kit) within which resides the JRE, and includes tools such as the Java compiler, to generate Javadoc documentation or the debugger. It is also available as a separate package, and can be considered necessary to run a Java application environment, whereas a developer must also have other facilities offered by the JDK.

4.2 ECLIPSE [7]

Eclipse is a computer program comprising a set of programming tools to develop open source platform so the project called “Enriched client applications” as opposed to “Client-light” browser-based applications. This platform has typically been used to develop integrated development environments (IDE) such as the Java IDE called Java Development Toolkit (JDT) and compiler (ECJ) delivered as part of Eclipse (which are also used to develop the same Eclipse). However, it can also be used for other types of client applications, such as BitTorrent and Azureus.

Eclipse is also a community of users, constantly expanding application areas covered. One example is the recently created Eclipse Modeling Project, covering almost all areas of Model Driven Engineering. It was originally developed by IBM as a successor to its family of tools for VisualAge. Eclipse is now developed by the Eclipse Foundation, as an independent nonprofit organization that promotes open source community and a set of complementary products, skills and services.

Eclipse was originally released under the Common Public License, but then was re-licensed under the Eclipse Public License. The Free Software Foundation has said that both licenses are free software licenses, but are incompatible with the GNU General Public License (GNU GPL).

The widgets are implemented by a tool called widget for Java Standard Widget Toolkit, unlike most Java applications, are using standard options as Abstract Window Toolkit (AWT) or Swing. The Eclipse UI also has an intermediate layer called JFace GUI that simplifies the construction of SWT based applications.

The integrated development environment (IDE) used modules (plug-in English) to provide full functionality in front of the rich client platform, unlike other monolithic environments where functionalities are all included, the user needs or not. This mechanism module is a light platform for software components. In addition to allowing Eclipse extended using other programming languages as are C / C ++ and Python, allowing Eclipse to work with languages like LaTeX text processing, network applications such as Telnet and management system database. The plugin architecture allows writing any desired extension in the environment, as would be configuration management. Java and CVS support in Eclipse SDK is provided. And not have to be used only with these languages as it supports other programming languages.

4.3 X.509 CERTIFICATES [8]

X.509 **(Figure 11)** was officially published in 1988 and started in conjunction with the X.500 standard and assumed a strict hierarchical system of certification authorities (CAs) to issue certificates.

In cryptography, X.509 is an ITU-T standard for public key infrastructure (PKI) that specifies, among other things, standard formats for public key certificates and algorithm validation certificate path. Its syntax is defined using language ASN.1 (Abstract Syntax Notation One), and the most common encoding formats are DER (Distinguish Encoding Rules) or PEM (Privacy Enhanced Mail). X.509 also includes standards for implementing certified revocation lists (CRLs), and neglected aspects of PKI systems.

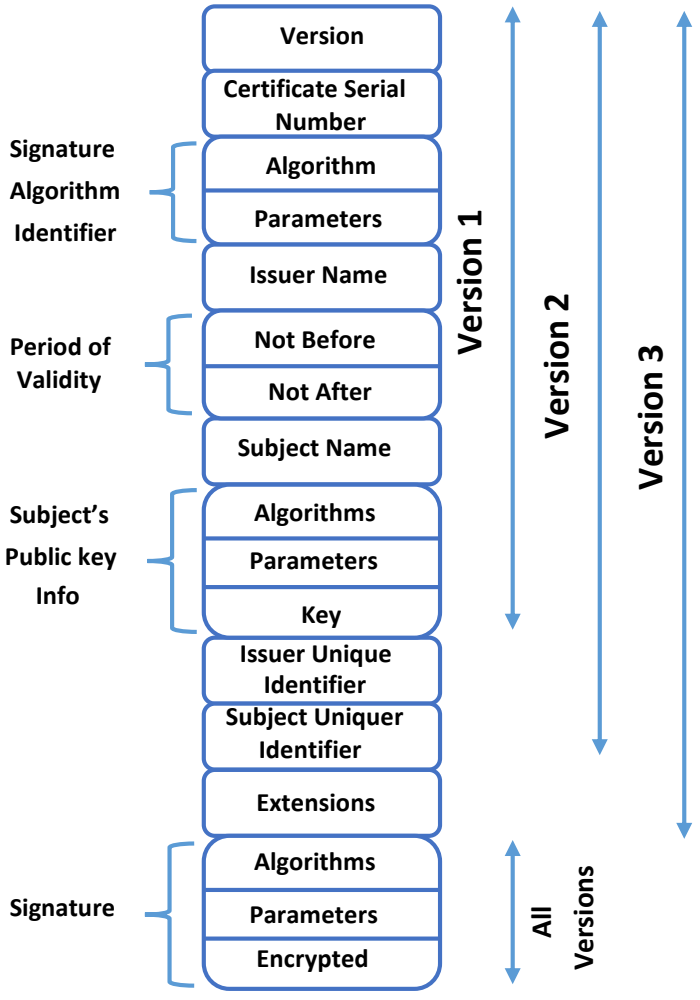


Figure 11 - X.509 Certificates structure

4.3.1 CERTIFICATION AUTHORITY

A certification authority (CA) is an entity that issues digital certificates to be used by third parties. It is an example of a trusted third party. CAs are characteristic in many schemes of public key infrastructure (PKI).

In the X.509 system, a certificate authority issues a certificate to associate a public key to a particular Distinguished Name in the X.500 tradition or an alternate name such as an email address.

X.509 is the centerpiece of the public key infrastructure and the data structure that links the public key to the data that identify the holder. Its syntax is defined using ASN.1 language (Abstract Syntax Notation One) and the most common encoding formats are DER (Distinguished Encoding Rules) or PEM (Privacy-enhanced Electronic Mail). Following the notation of ASN.1, a certificate contains several fields, grouped in three main groups:

1. The first field is the subject (subject), which contains data identifying the subject holder. These data are expressed in notation DN (Distinguished Name), where a DN in turn consists of various fields, the most common being the following; CN (Common Name), OU (Organizational Unit), O (Organization) and C (Country).

> Besides the name of the titular subject (subject), the certificate also contains data associated with the digital certificate itself as the version of the certificate, your ID (serial Number), the signing CA (issuer) and the validity period (validity).
>
> The X.509.V3 version also allows you to use optional fields (alternative names and permitted uses for the key location of the CRL and the CA).

2. The certificate contains the public key, which expressed in ASN.1, consists of two fields, first, showing the algorithm used to create the key (RSA), and secondly, the public key itself.

3. CA added the sequence of fields that identify the signature of the previous fields. This sequence contains three attributes, the signature algorithm used, the hash of the signature, and the digital signature itself.

4.4 KEYTOOL [9]

When two SSL socket try to establish a connection (a client socket and a server socket), the first thing they do is to present to each other and each one finds out if the other is trustable. If the information exchanged is correct then the connection is established. If one of both entities doesn't trust the other one then the connection isn't stablished.

To present themselves a certificate has to be created for each socket. This certificate is merely to generate a file with the keytool that is integrated with Java. Two certificates are created, one for the server and one for the client.

To know if each entity trust with the other one, each socket has a file store with the trusted certificates this entity trust with. Therefore, in the customer's warehouse the server certificate must be added and vice versa. These stores are just created files with java keytool.

Java Keytool is a key and certificate management utility. It allows users to manage their own public/private key pairs and certificates. Java Keytool stores the keys and certificates in what is called a keystore. It protects private keys with a password.

Each certificate in a Java keystore is associated with a unique alias. When creating a Java keystore a .jks file will be created. Initially will only contain the private key, and then a CSR is generated. After that, the certificate is imported to the keystore including any root certificates.

The three main functions used in this project are:

1. Generate a Java keystore with a certificate **(Figure 12).**

2. Export the certificate from the keystore in a CER file **(Figure 13).**

3. Import the certificate to the certificate store of the other entity **(Figure 14).**

Server's Certificate Generation

1. **keytool** -**genkey** -**keyalg** RSA -alias serverKay -**keystore** serverKey.jks -**storepass** serverpassword

   - Genkey: Indicates the generation of a certificate.

   - keyalg: Indicates that RSA encryption will be used.

   - Alias: As in the certificates store can be more than one certificate this alias will be to identify the certificate within the store.

   - keystore: Indicates the file that will work as the certificates store.

   - Storepass: A password is needed to log in the certificates store. If the keystore doesn't exist then it will be created.

Figure 12 - Server's certificate generation

Server's Certificate Exportation

2. keytool -**export** -**keystore** serverkey.jks -**alias** serverKay -**file** ServerPublicKey.cer

- Export: Indicates that the operation done will be an exportation.

- Alias: As in the certificates store can be more than one certificate this alias will be to identify the certificate within the store.

- keystore: Indicates the file that will work as the certificates store.

- File: Contains the file in which the certificate exported will be stored.

Figure 13 - Server's certificate exportation

Server's Certificate Importation

3. keytool -**import** -**alias** serverKay -**file** ServerPublicKey.cer -**keystore** clientTrustedCerts.jks -**keypass** clientpassword -**storepass** clientpassword

- Import: Indicates that the operation done will be an importation.

- Alias: As in the certificates store can be more than one certificate this alias will be to identify the certificate within the store.

- keystore: Indicates the file that will work as the certificates store.

- File: Contains the file in which the certificate exported will be stored.

Figure 14 - Server's certificate importation

CLIENT

1. keytool -**genkey** -**keyalg** RSA -alias clientKay -**keystore** clientKey.jks -**storepass** clientpassword

2. keytool -**export** -**keystore** clientkey.jks -**alias** clientkay -**file** ClientPublicKey.cer

3. keytool -**import** -**alias** clientKay -**file** ClientPublicKey.cer -**keystore** serverTrustedCerts.jks -**keypass** serverpassword -**storepass** serverpassword

Figure 15 - Client's certificate procedure

4.5 WIRESHARK [10]

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.

Wireshark is cross-platform, using the GTK+ widget toolkit in current releases, to implement its user interface, and using pcap (packet capture) that consists of an application programming interface (API) for capturing network traffic. It runs on Linux, OS X, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. Wireshark, and the other programs distributed with it are free software and released under the terms of the GNU General Public License.

Wireshark is a software that "understands" the structure (encapsulation) of different networking protocols. It can parse and display the fields, along with their meanings as specified by different networking protocols. Wireshark uses pcap format to capture packets, so it can only capture packets on the types of networks that pcap supports.

Wireshark lets the user put network interface controllers that support promiscuous mode into that mode, so they can see all traffic visible on that interface, not just traffic addressed to one of the interface's configured addresses and broadcast/multicast traffic. However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all traffic through the switch is necessarily sent to the port where the capture is done, so capturing in promiscuous mode is not necessarily sufficient to see all network traffic.

Unfortunately, on Windows, Wireshark is unable to capture packets or traffic sent from a host machine to that same host machine. This is due to the fact that such local traffic is not sent over a real network interface, but instead (in many cases) is sent over a loopback interface. Then to be able to capture this traffic a program called RAWCAP is used because also enables the user to save captured traffic as .pcap file and this means that Wireshark can be used to analyze the resulting file.

4.6 MODEL VIEW CONTROLLER (MVC) [11]

The Model-View-Controller (MVC) is a standard software architecture that separates data and business logic of an application from the user interface and the module responsible for managing events and communications. To do so, MVC proposes the construction of three distinct components that are the model, view and controller. This pattern software architecture is based on the ideas of code reuse and separation of concepts, features that are intended to facilitate the task of application development and subsequent maintenance.

Generically, the MVC components **(Figure 16)** could be defined as follows:

- Model: It is the representation of information with which the system operates, therefore manages all access to such information, both queries and updates, ad also implementing privileges access have been described in the specifications of the application (business logic). It sends the view that part of the information at all times is asked to be shown (typically to a user). Requests for access or manipulation of information reach the model through the controller.

- The View: It presents the model (information and business logic) in a suitable format to interact (usually the user interface) and therefore requires that model the information to be represented as output.

- Controller: It responds to events (typically user actions) and invokes requests to the model when a request for information is made (by such as editing a document or record in a database). You can also send commands to its associated view if a change is requested in the way the model is presented so you could say that the controller is the intermediary between the view and model.

Although you can find different implementations of MVC, the control flow that follows is usually the following:

1. The user interacts with the user interface in some way (for example, the user presses a button, link and so on).

2. The controller receives the notification of the action requested by the user. The controller manages the event that arrives, frequently through an event handler (handler) or callback.

3. The controller accesses the model, updating it, possibly modifying appropriately to the action requested by the user. The complex controllers are often structured using a command pattern that encapsulates the actions and simplifies his extension.

4. The controller delegates the objects of sight the task of displaying the user interface. The view gets its data model to generate the appropriate user interface where changes are reflected in the model. The model must not have direct knowledge about the view. However, you could use the Observer pattern to provide some indirection between the model and the view, allowing the model to notify interested parties of any change. A view object can register with the model and

wait to change, yet the model itself still does not know anything about the view. This use of the Observer pattern is not possible in the Web application since classes are disconnected model view and controller. In general the controller fails domain objects (model) in sight although it can give the order in sight to be updated.

5. User interface waits for new user interactions, starting the cycle again.

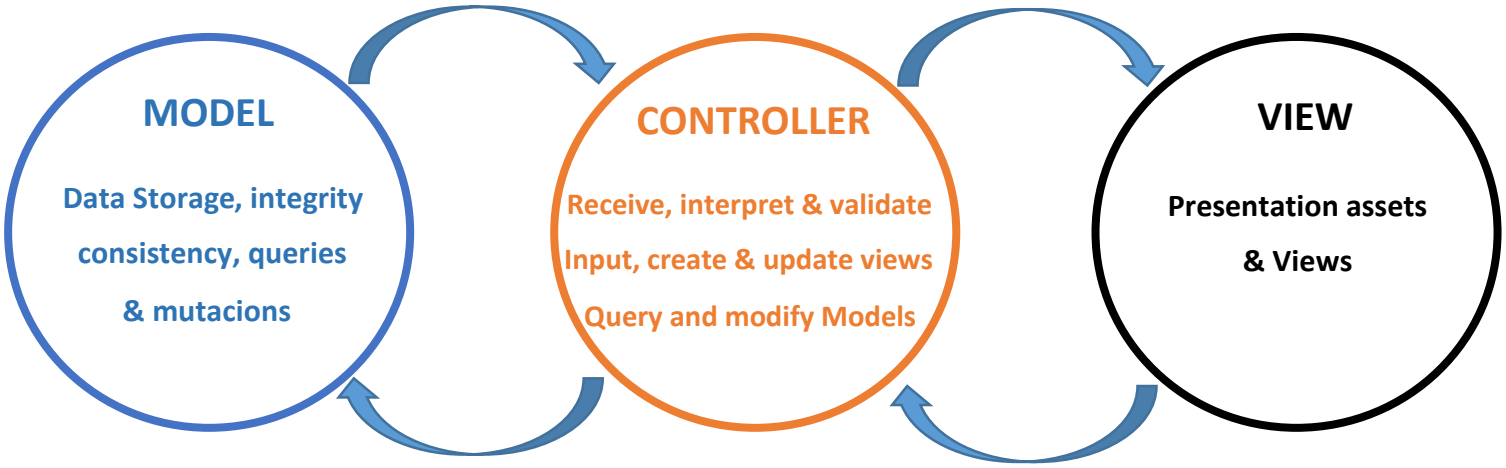


Figure 16 - Model View Controller

4.3 CLASS DIAGRAM

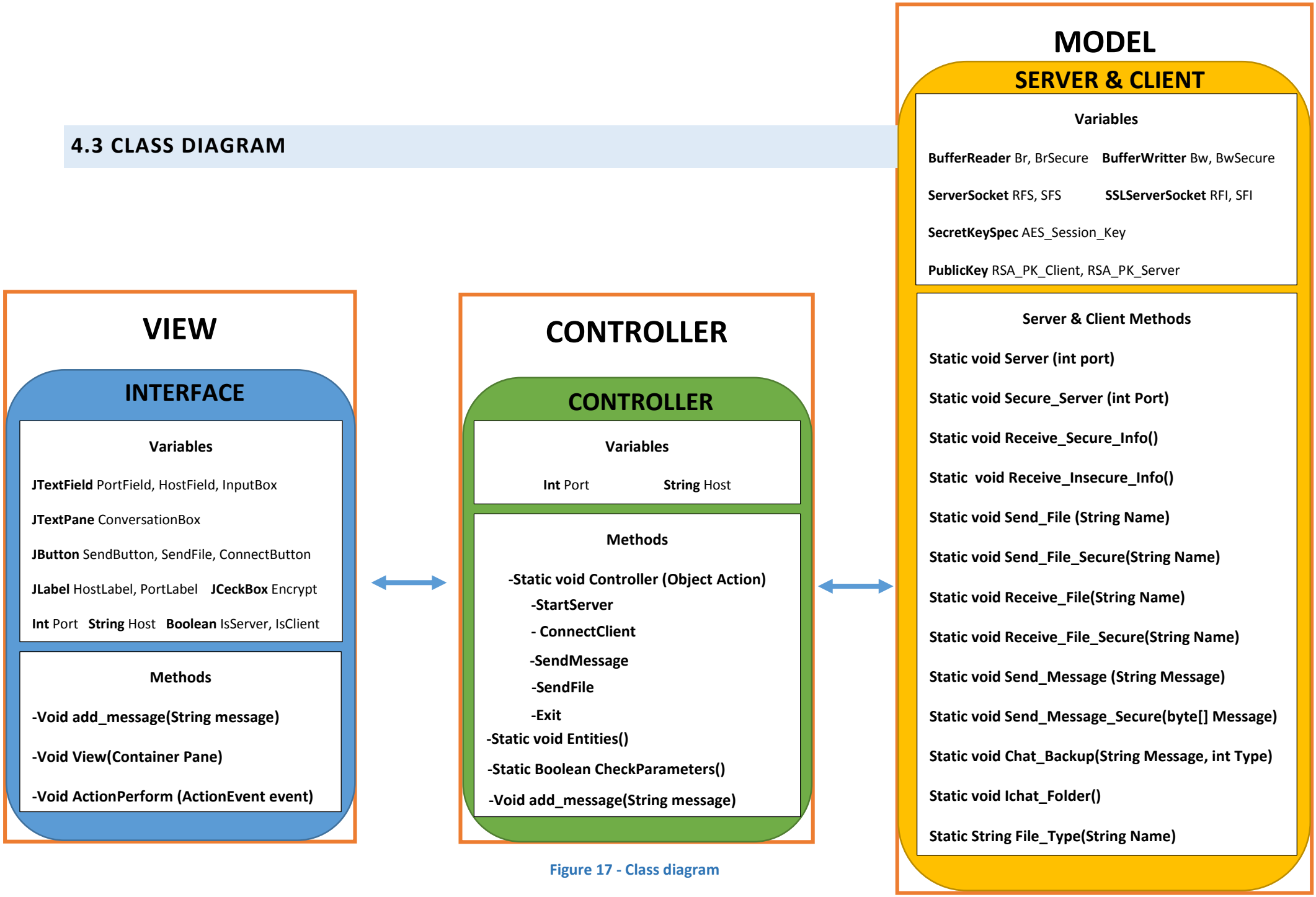


Figure 17 - Class diagram

4.7.1 INTERFACE

This function is the first one executed when the program is run and is in charge of building the graphical interface for both user and server. As the programming mode used is the Model View Controller (MVC) this class only contains those variables and methods involved in graphical issues and it is completely independent from the rest of the parts. Using this model, changes done in the graphical interface not will affect the rest of the other parts and will be transparent for the other entities.

VIEW

This class is in charge of initializing all the variables relatives to the graphical interface. When this method is called from the main class a JFrame is created and a JPane is added to this frame. After that, this method handles the position on the container of all the elements that build the interface. This elements are:

1. **JButtons:** All the buttons which the user can interact with and generate an event that will be analyzed by the Controller.
2. **JTextPane:** This element contains the conversation box between the server and the client.
3. **JTextFields:** The interface has a text field to write the port and the host and another one used as the input to send messages.
4. **JLabels:** This elements can't be modified and only indicate the text fields corresponding to the host and port.
5. **JCheckBox:** When this box is ticked both messages and files are sent over a secure channel.
6. **Image Icons**: It corresponds to the profile image of the chat and the rest of the buttons.

In the case of the buttons, a listener is needed to know when a button has been pressed to execute the corresponding event.

ACTIONPERFORMED

When a button is pressed then an event is generated. This function is in charge of analyzing this event and converting it into an action. Then, this action is send to the Controller for treatment. This method is used as a connection between the view and the controller.

ADD_MESSAGE

This function is used every time a writing operation is done. Initially, the idea was to use a JTextArea to write all the messages but the option of changing the color of the different lines was not let and then it was needed to use a JTextPane which let us to paint every specific character using the desired color. As the JTextPane has a limited size, after sending some messages a scroll bar is needed and another problem was found because the JTextPane doesn't have a scroll bar by itself. Then, the solution was to include a scroll bar independent from the JTextPane. Besides being used to write the messages in a different color depending on the information is generated or is received from an external entity, this method updates the view of the scroll bar if needed.

It has to be mentioned that every time a message or a file is received it is shown in the conversation box and before the message, a headboard is added. This headboard contains the date in which the message has been received or sent. All the message sent are painted with blue color whilst all the messages received are painted with gray color.

As this form part of the view, when a writing operation is required by the Server_Model or Client_Model, a function of the Controller is called which sends the message to the view using the Add_Message function. This avoids the Models of directly affecting the view and reinforces the idea of our programming model. Then if a change in the message appearance is desired only the class view has to be modified and this modification will be transparent for the rest of the classes.

4.7.2 CONTROLLER

This method is acting as a connection between the sight and the logic of the application. All the actions received for the view because of the user interaction are analyzed and sent to the model to be processed.

The actions received from the view can be:

1. Start the server: When this button is pressed and a true value is returned by the Check_Parameters function, then a SSL server socket and a server socket are generated in the desired port and the server will be waiting for a client to be connected using both a secure and an insecure channel.

2. Connect the client: When this button is pressed and a true value is returned by the Check_Parameters function, then a SSL socket and a socket is generated in the desired port which has to be the same as the port chosen by the server. Then the client is linked to the server and messages and files can be exchanged between them.

3. Send messages: As it will be explain in detail in the testing part, the server and the user can exchange messages that will travel encrypted or in clear from the sender to the receiver. A JCheckBox has been created and when it is clicked a secure channel will be used and the data will be encrypted using an asymmetric key as it is explained later in the Model part. Later, in the testing part, it is shown that the synchronism between the client and the server will be important when sending encrypted messages.

4. Send files: Both entities, server and client, can exchanged files over a secure or insecure channel and the option of saving the file or not is decided by the receiver itself. This option will be very useful to compare between both methods to see how many packets are sent depending on the choice done and the size of those packets. An error control is deployed before enabling the sender to send the file because files larger than 6 MB aren't allowed.

5. Shut down the application: When any of the two sides want to finish the connection, there is a button to do so. When this button is pressed a confirmation message is shown to ensure it has not been an error. Moreover, the interface has a button in the upper right side that has the same function.

It can be sent that in any case the view will be in direct contact with the model without using the controller as a connection between them. So, the changes made to the model part will be completely transparent to the view.

CHECK_PARAMETERS

This function is executed every time an event is generated by the user when the start or connect button are clicked. This function returns a boolean variable which takes a true value if the rules are accomplished or false in case of not.

The parameters controlled by this function are:

1. The host has to be composed of numbers and dots and can't be changed from the one shown by default. This is done because in the future this application could be run not only in the local host.

2. The port has to be a number compressed between 44000 and 65535 because numbers lower than 44000 may be reserved and the bound number is because a higher port is not available.

As it can be seen this function is only controlling the two parameters that are involved to create the server and the client. If these rules are fulfilled then the program can be run and depending on the role chosen a client or a server will be deployed. In case there is something wrong alert messages will be shown to the user and the possibility of trying again will be an available option.

ENTITIES

When a true value is returned by the Check_Parameters function one of this two entities can be chosen:

1. **Server:** Then a method from the Server_Model will be called by the Controller to prepare the SSL server socket and server socket using the host and port chosen by the user. In case any problem is detected when stablishing the sockets a report message will be shown to the user.

2. **Client:** As it happens with the server, a method from the Client_Model class will be called by the Controller**.** This time, the procedure is a little bit sophisticated because the port in which the connections will be focused on has to be the same chosen by the server because if not the tunnel between the server and the client can't be set up. If the port introduced is correct a SSL socket and a socket will be generated and the server will indicate with two messages than a secure and an insecure client have been connected.

After having the client connected to the server both entities are let to send messages and files using the secure or insecure way. Now, the Controller will be in touch with the client and the server when an event is generated by the user's activity.

As it will be explained later in more detail several folders are created to store the files and the information transmitted between them.

Having the choice of sending the information using a secure or an insecure channel will allow us to check and analyze by using Wireshark both ways.

SEND_MESSAGE

When a message is sent to any of the two entities involved, this function is not used because the message is written directly in the input box and then, adding the message can be done in the view part but when a message is received in the buffer's reader, this message is located in the Model part and then to be appended in the conversation box has to be sent to the Controller using the Send_Message function which receives a string as a parameter. After that, the only function this method has is to send the message to the view and the view is who adds the headboard and the color to this specific message.

Using this structure the MVC is followed and it facilitates the work if some updates have to be done. Moreover, it guarantees the totally transparence between the View and the Model which let us to have a more scalable project.

4.7.3 SERVER MODEL

This class implements all the methods of the server side. When the start button is pressed and there isn't any errors with the host the port and, then an event is received by the Controller and the server's methods can be called. As it has been mentioned before, this class only interacts with the Controller and never with the view as the controller is in between of them isolating both parts.

SERVER

This is the first method called by the Controller when the start button is pressed. This method receives an integer as the main parameter. Then the entity server is created and the variables referring to the server part are set up. After that, a server socket is created and then the class Secure_Server is called to create the SSL server socket. The secure channel is created in the port chosen and the insecure channel is created in the next available port. It has been implemented this way because the most important part is the secure channel and the insecure channel is used for comparing and checking which information can be sniffed using Wireshark tool.

Moreover, two threads are executed. It has been implemented this way because the server has to be able to receive both secure and insecure information at any time and the only way to do that is creating two independent threads. One thread is listening in the secure port to be able to receive secure messages and files and the other one is listening to the insecure port. Each of these threads are listening to its buffer reader where it is received the information sent from the server.

Finally, the IChat_Folder method is called and a group of folders are created in a hierarchical way to store and classify the information received.

ICHAT_SERVER

This function is in charge of creating a set of folders for storage. As it can be seen in the picture below **(Figure 18)**, a folder called IChat is created in the system root folder where the program is deployed. Inside this folder, the server's folder is created and then inside this folder other five folders are created: Pictures, Chats, Documents, Videos and Others. It can be guessed what these folders will contain but not which formats are allowed:

1. **Pictures:** In this folder are stored all the images with jpg, png or bmp extension.
2. **Documents:** In this folder are stored those document which extension is txt, pdf, doc and docx.
3. **Videos:** All the videos which extension is mp3, mp4 or wax are stored in this folder.
4. **Others:** The rest of the files which extension is not one of the mentioned previously are stored in this folder.
5. **Chats:** Every time the program is deployed, a txt file is created, the name of this file is the following: Year-Month-Day Hour Minute Second. Using this nomenclature, it can be ensured the independence of all the files. This archives are very important because are used as a backup of all the conversations done by the server and in this file it can be seen all the messages and files sent and received, the date in which they were sent or received and if they were sent or received using the secure or insecure channel. This backup has been implemented because in most of the messaging or chat applications it is present and really useful. A complete explanation of this files can be seen in the testing part.

It has to be said that there is a restriction concerning to the size of the files. The maximum size allowed is 6MB. It is a design decision but it has been thought that files larger than this size would slow down the program and for our purposes is not useful. When a file is sent, a confirmation message is append to the conversation box indicating the file has been sent correctly. Otherwise, when a file is stored, a message is appended to the conversation box and then it is shown the folder in which the file has been stored and the file's name. As it is detailed in the testing part, when an existing file is received, the user is alerted with a notification message and the opportunity of renaming or not storing the file is shown and is up to the user.

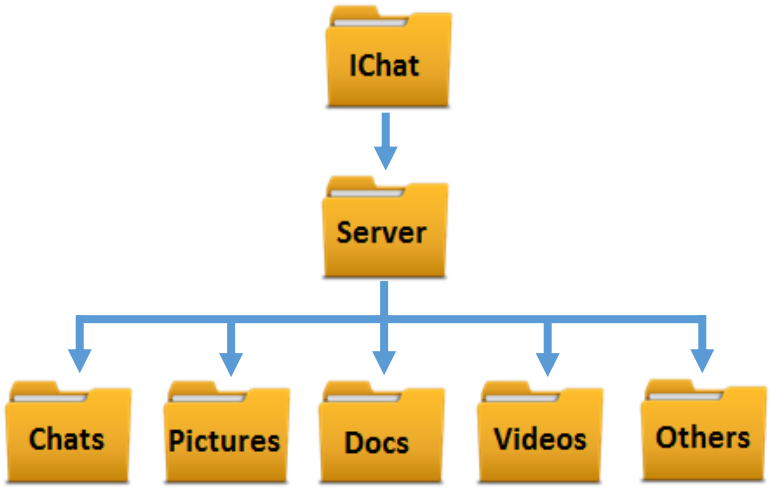


Figure 18 - Server folders structure

SECURE_SERVER

This method receives an integer as a parameter which is the port in which the SSL socket will be stablished. The first thing to do is to configure the SSL server socket. When two SSL sockets, a client and a server socket, want to stablish a connection the first think they have to do is to present itself to the other part and then each of them checks that the other side is a trustable entity. If the process is correct and both entities trust each other the connection is done. To present themselves they a certificate is needed and to believe the certificate of the other part is a trusted one they need to have this certificate stored in his a store of trustable certificates. To create the SSL socket, the factories that came by default with JAVA were used.

To configure the SSL server socket this following steps have to be followed:

1. The key store is initialized with a JKS extension and the store in which the server's certificate is located is loaded in this instance.
2. A key manager is initialized with the key store created before using the corresponding password chosen when the certificate's store was configured using Keytool program.
3. A trusted store is set up where the client's certificate was introduced using the Keytool program. This is an important step because when the client will try to connect to the server and will send its own certificate, if the server doesn't have this certificated added to its store as a trustable one, then the connection will be rejected. This step isn't necessary if the certificated had been signed by a certificate authority but as our certificates are auto generated then this step is fundamental to stablish the secure connection between both entities.
4. The SSL server socket factory is created and loaded with the key manager and the trusted manager. Both variables are arrays and contain the server's certificate and the client's certificate.
5. The SSL server socket is created using the SSL server socket factory configured previously and after that, the method accept connections is called to enable connections on that channel.

Finally, a buffer reader and a buffer writer are created and initialized using the SSL server socket previously created. The buffer reader is used to store the information received and the buffer writer is used to send information.

Now the server is properly configured to accept connections. When a connection is received there is the handshake that has the following steps:

1. Client cipher suite and other parameters
2. Server's certificate, cipher suite and other parameters
3. Client's certificate
4. Server key exchange
5. Change cipher suite
6. Client key exchange
7. Change cipher suite
8. Finished

The aim is to use a combination of asymmetric and symmetric key encryption. Finally, with the information exchanged during the handshake, both entities have to be able to create the same symmetric session key which will be used to encrypt and decrypt the information that will travel over the secure channel. For the asymmetric encryption RSA keys are used and for the symmetric encryption a 256 bits AES key is used. The aim of combining both methods is to prevent the MITM attacked. The handshake is explained in detail in the handshake chapter (page 35).

FILE_TYPE

This method is called every time a file is received. The main purpose of this function is to figure out the extension of the file received to decide in which of the created folder should be stored. To do so, this functions needs to receive the name of the file as a parameter. In the case of not being able to find out the extension or if this extension is not one of the desired extensions, the file is stored in the Others folder. This method has been created because it was hoped to create a chat application the most similar possible to the existing ones.

RECEIVE_SECURE_INFO

This function **(Figure 19)** is inside a thread because is needed to be able to receive messages and files at any moment over the secure and insecure channel. The procedure followed to know if a message has been received is to check if there is any data in the buffer reader. In case the buffer reader is filled, there are two possibilities:

1. **The encrypted check box is not clicked:** This is the worst case because it means that both entities aren't synchronized and then the information received will not be decrypted because the AES session key won't be used. So, the server, in its own conversation box, will see strange symbols and letters that won't make any sense. This method has been implemented to shown that the synchronism is a key issue in communications and it is useful to understand how the information travels over the secure channel and this information is what a third entity will be able to see if it was sniffing the traffic.

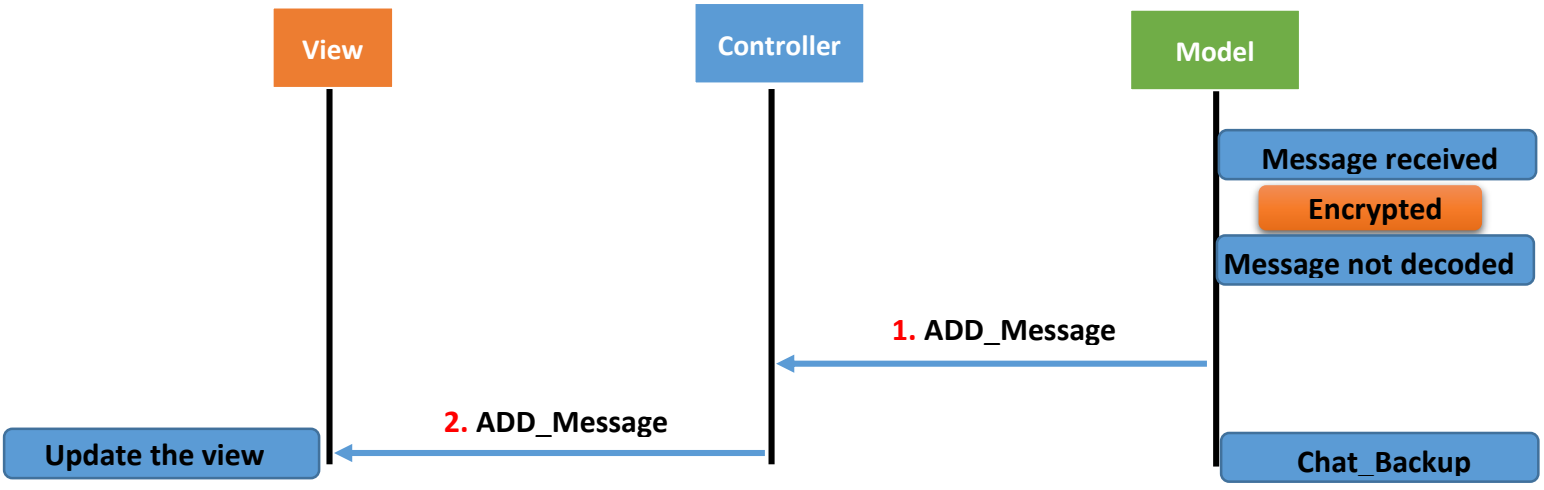


Figure 19 - Receive secure information diagram when not synchronized

2. The encrypted check box is clicked: This is the desired case **(Figure 20)** in which both sender and receiver are synchronized. Then the information encrypted by the Client with his own AES session key is decrypted easily by the Server which will used its AES session key. As it can be seen, the Client and the Server are using a symmetric encryption mode. Before sending the information, the message is encoded in Base 64 and for this reason, the Server has to decode the information following the same rule before using the AES session key. It has been done because if not there were times that if the message contained some accent or other special symbols the information was wrongly decoded.

As the session key used for both entities has a length of 256 bits, sometimes the information received can be larger than 256-bits and then a method is needed to decrypt the information because it has to be decrypted in chunks. Finally, all the chunks are joint to recover the original message and then the message is sent to the Controller for further treatment. The method that is called to decrypt the message in chunks is called Decrypt_Bytes_Array and as it suggested by the name, an array has to be sent as a parameter.

Otherwise, if a file is received, the server will know it because the Client will notify it with a message that will contain a concrete string. Then, the Server will called the method Receive_File_Secure and the Server will be waiting until the file transmission is finished.

After a file or a message is received, the generated backup file has to be updated by the Controller with the same information appended to the conversation box and adding a headboard to differentiate the messages or files received over the secure channel from those received over the insecure channel.

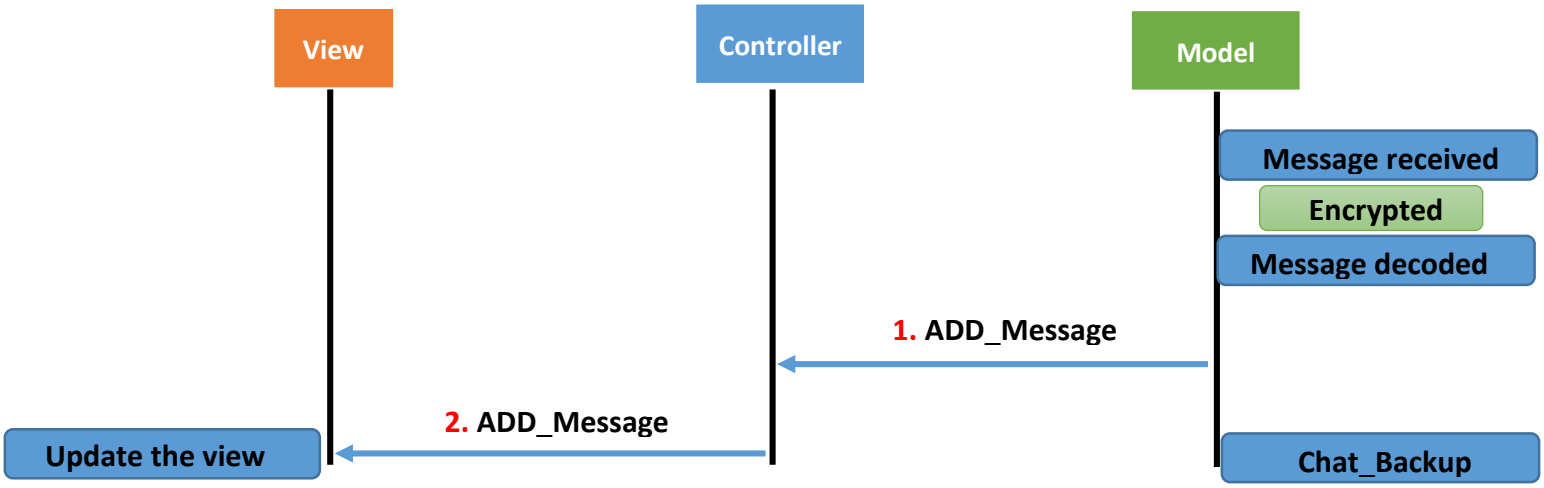


Figure 20 - Receive secure information diagram when synchronized

RECEIVE_INSECURE_INFO

As the previous method, this function **(Figure 21)** is inside a thread because is needed to be able to receive messages and files at any moment but in this case the channel used is the insecure one. The procedure followed to know if a message has been received is to check if there is any data in the buffer reader attached to the socket input stream.

In case the buffer reader is filled, the data is taken out and sent to the Controller which sends the message to the View for further treatment. After delivering the message to the Controller, the method Chat_Backup is called to update the file in which the conversation is stored and a headboard is added to know that this message was sent using the insecure channel and the time in which this information was sent. Otherwise, if a file is received, the method Receive_File is called which will call the method Chat_Backup to add that an insecure file was received.

Having this method is very useful because it can be sent the same message by both the secure and the insecure channel and then using the Wireshark tool both messages can be compared.

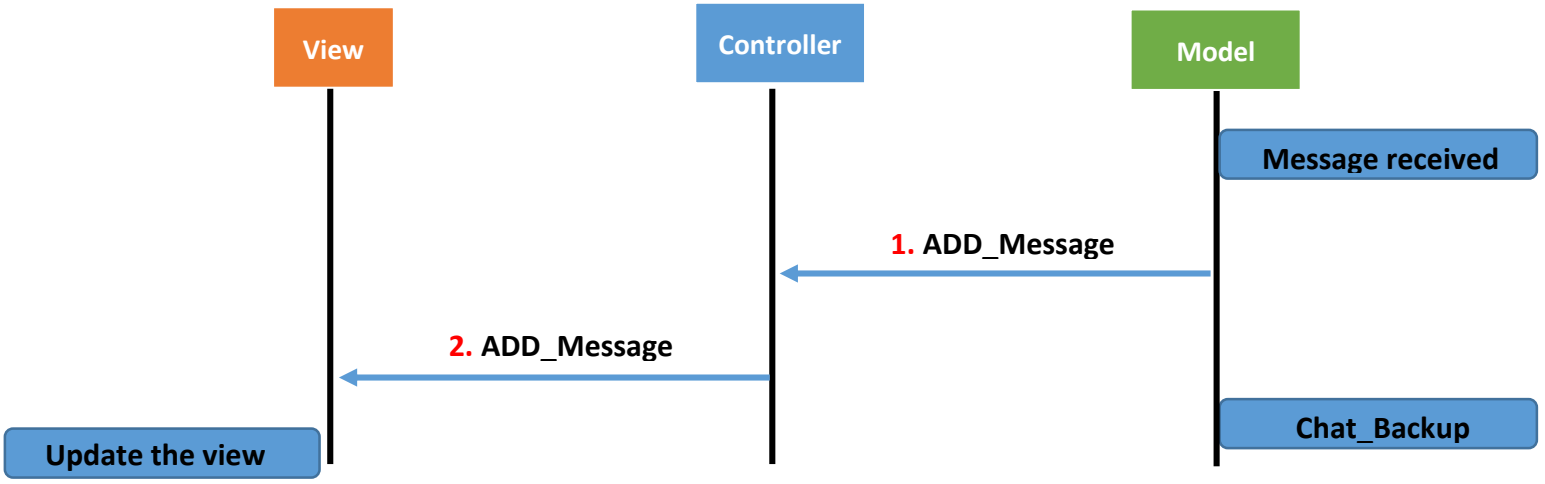


Figure 21 - Receive insecure information diagram

RECEIVE_FILE_SECURE

As the name suggests this method **(Figure 22)** is called every time a file is received by the SSL socket previously created.

When inside this function, a variable of the type Boolean is configured with a true value. This variable is used to create a loop that will not finish until this variable becomes false and it happens when all the bytes of the file are received. Then, the bytes are received in chunks, decrypted using the AES session key and stored in a byte array until the reading done returns a zero value which indicates there aren't bytes pending to be read. After that, the method File_Type is needed to find out the file extension which let the program to save the file in the corresponding folder. Once the extension is known, all the bytes are copied to a buffered output stream. This stream has the path in which the file has to be created. Finally, the file can be opened, read and modified if needed by the Server. The server knows the file location because

a message is added to the conversation box and the backup file showing a confirmation message and the folder in which the file has been saved.

If the sever had stored this file from previous conversations, an alert message is shown indicating the option of renaming or deleting the file.

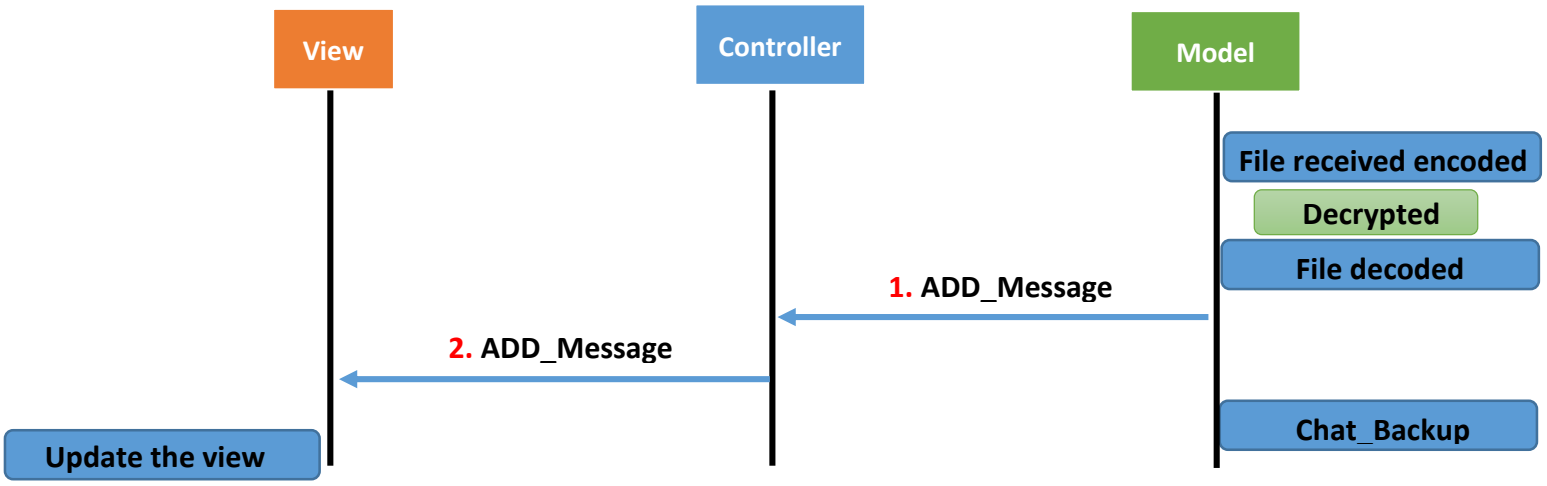


Figure 22- Receive file over SSL socket diagram

RECEIVE_FILE_INSECURE

This function **(Figure 23)** is called every time a file is received over the insecure channel what it means the information regarding the file has been travelling in clear over the channel and if a third malicious entity has been sniffing the traffic could have read the information.

When inside this function, a variable of the type Boolean is configured with a true value. This variable is used to create a loop that will not finish until this variable becomes false and it happens when all the bytes of the file are received. Then, the bytes are received in chunks and stored in a byte array until the lecture done returns a zero value which indicates there aren't bytes pending to be read. After that, the method File_Type is needed to find out the file extension which let the program to save this file in the corresponding folder. Once the extension is known, all the bytes are copied to a buffered output stream. This stream has the path in which the file has to be created. Now, the file can be opened, read and modified if needed by the Server. The server knows the file location because a message is added to the conversation box and to the backup file showing a confirmation message and the folder in which the file has been saved.

If the sever had stored this file from previous conversations, an alert message is shown indicating the option of renaming or deleting the file.

As it can be seen, the procedure is almost the same as when receiving a file over the secure channel but now as the bytes are not encrypted with the AES session key this step is skipped.

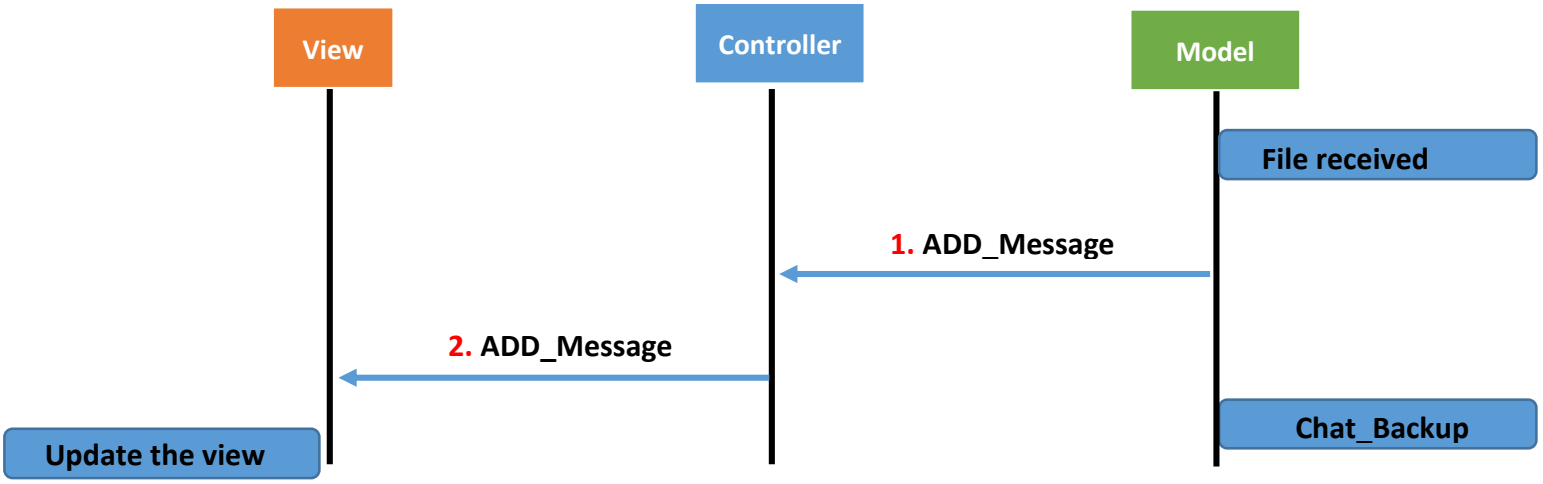


Figure 23 - Receive file over socket

SEND_FILE_SECURE

As the SSL server socket has been previously created, now all the steps concerting to the SSL handshake can be skipped.

When the encrypted check box is ticked and the Server needs to send an encrypted file to the Client, the button send file is clicked and then an additional interface is shown to the user. This interface is used by the user to navigate around the different folders and to choose the desired file. Then, a verification is done over the file to enssure the files don't overcome the size permitted and in case of trying to send a file too much longer an alert message is shown and then another file has to be chosen. When the size condition is fulfilled, this function **(Figure 24)** is called by the Controller and a string containing the path where the file is stored is sent.

After that, a byte array is created with the size of the file and a variable of the type file is set up with the path. Then, the file is read in chunks until the last byte and introduced to the output stream. Finally the file is delivered to the Client and the corresponding inputs and outputs streams are closed.

After sending the file, the Send_Message function is called and the Controller replies this information to the View for further treatment. As a final step, the Chat_Backup function is required to update the content.

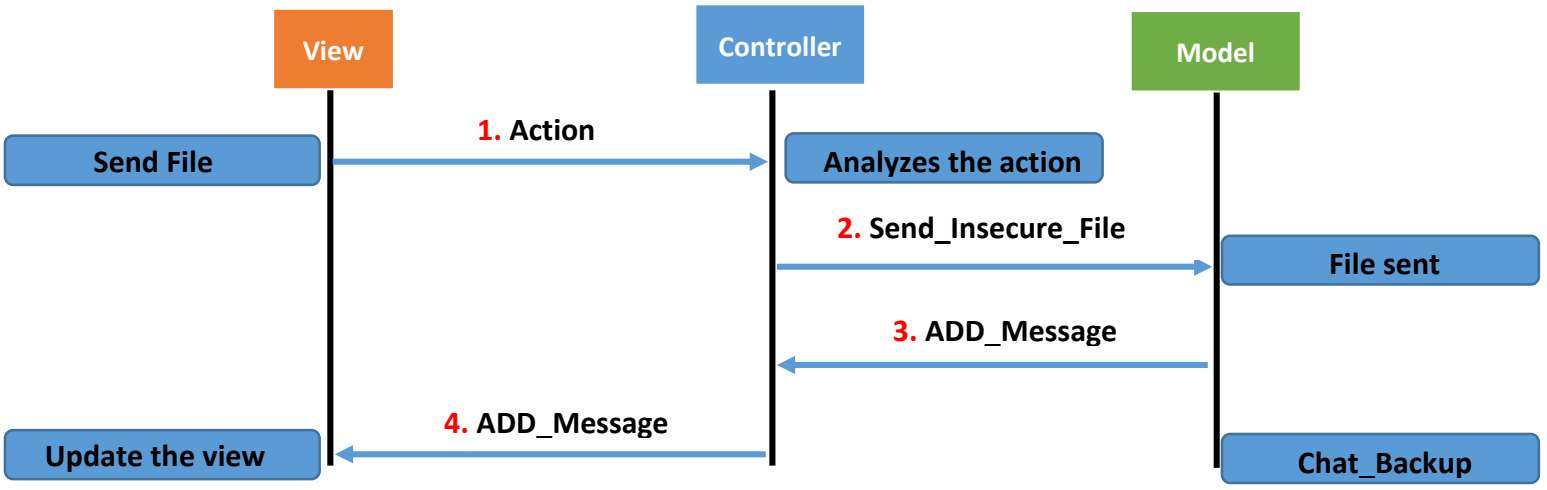


Figure 24 - Send file oevr SSL socket

SEND_FILE_INSECURE

The difference between this method and the previous one is that now **(Figure 25)** the insecure channel is used to send the file because the encrypted checkbox is not ticked. So, the information sent could have been modified by a third entity not directly involved in the conversation.

As in the secure case, after sending the file, the Send_Message function is called and the Controller replies this information to the View for further treatment. As a final step, the Chat_Backup function is required to update the content.

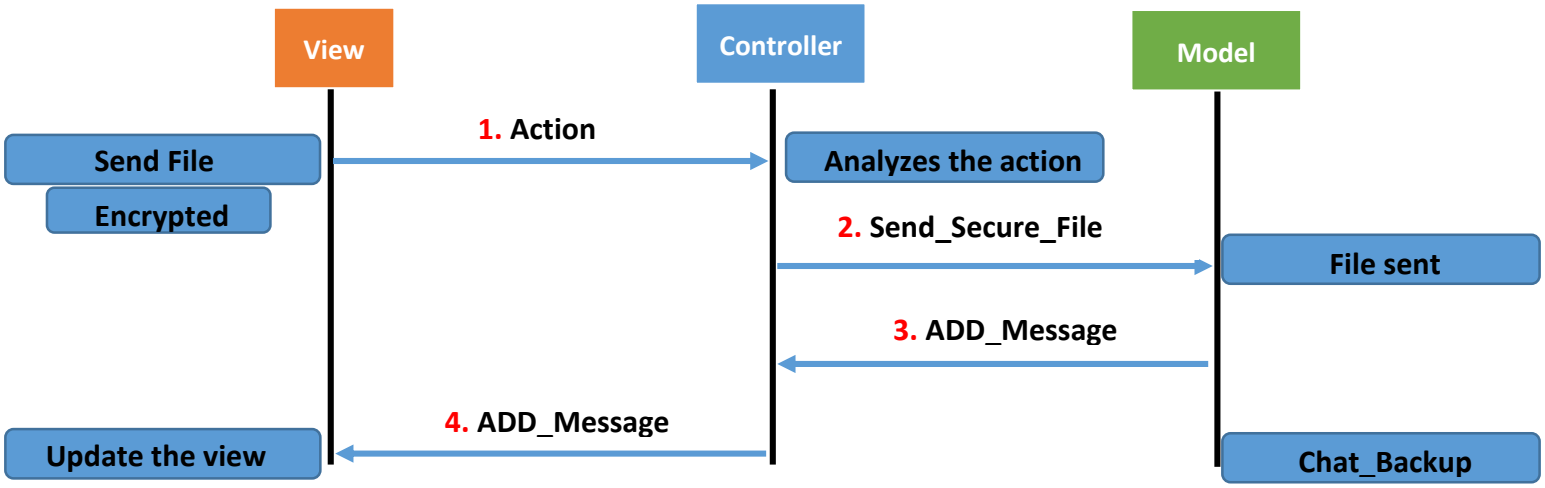


Figure 25 - Send file over socket

INSECURE_MESSAGE

When the send message button is clicked by the user, the input box is checked to ensure that isn't empty. If it is empty, an alerting message is displayed but if not then an action event is sent to the Controller and the function Insecure_Message is called. This method **(Figure 26)** receives a string as a parameter that is sent to the Client using the buffer writer.

Finally, the message is sent again to the Controller which replies the information to the View for further treatment and the method Chat_Backup is called to update the backup file.

As it can be seen in all the previous functions, the actions start in the view, are managed in the Controller, are executed in the Model and then the result is sent to the Controller and the Controller is in touch again with the View to update the conversation box.

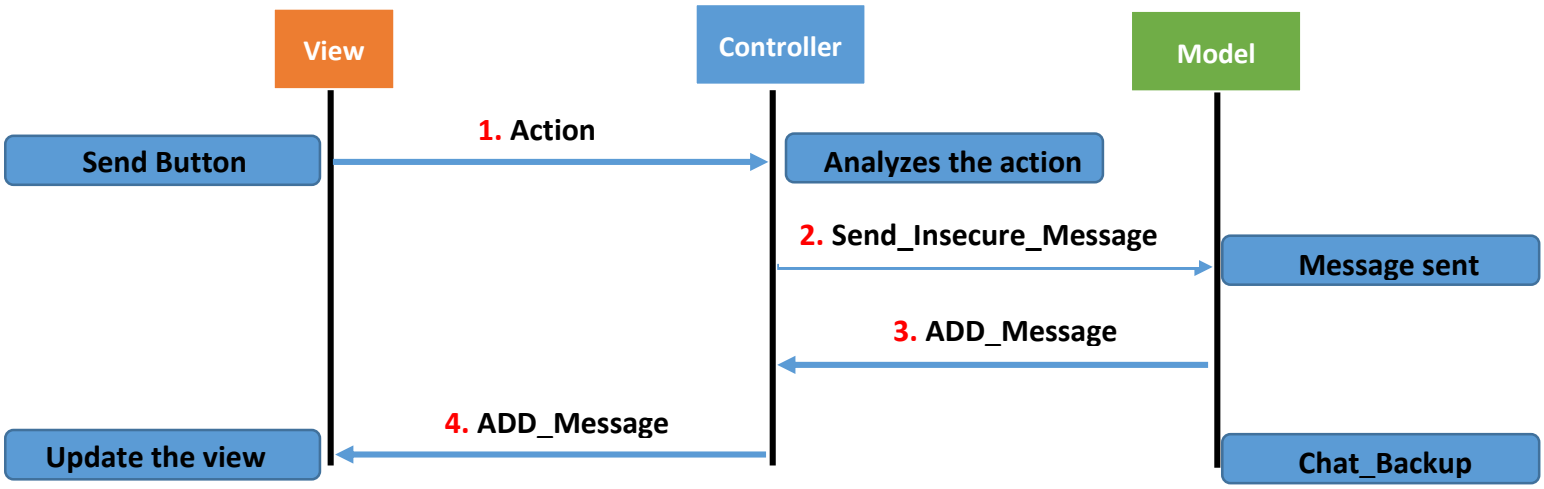


Figure 26 - Send message over an insecure channel

SECURE_MESSAGE

This method **(Figure 27)** is a little bit sophisticated than the previous one because the secure channel is used and some extra steps have to be done. This time, the AES session key has to be used to encrypt the message before sending over the SSL channel. When the message is encrypted is encoded in base 64 before transmitting and finally the message is sent using the buffer writer. After that, the message is delivered to the Controller which replies the information to the View for further treatment and the method Chat_Backup is called to update the backup file adding a header indicating this message has been sent in a secure way.

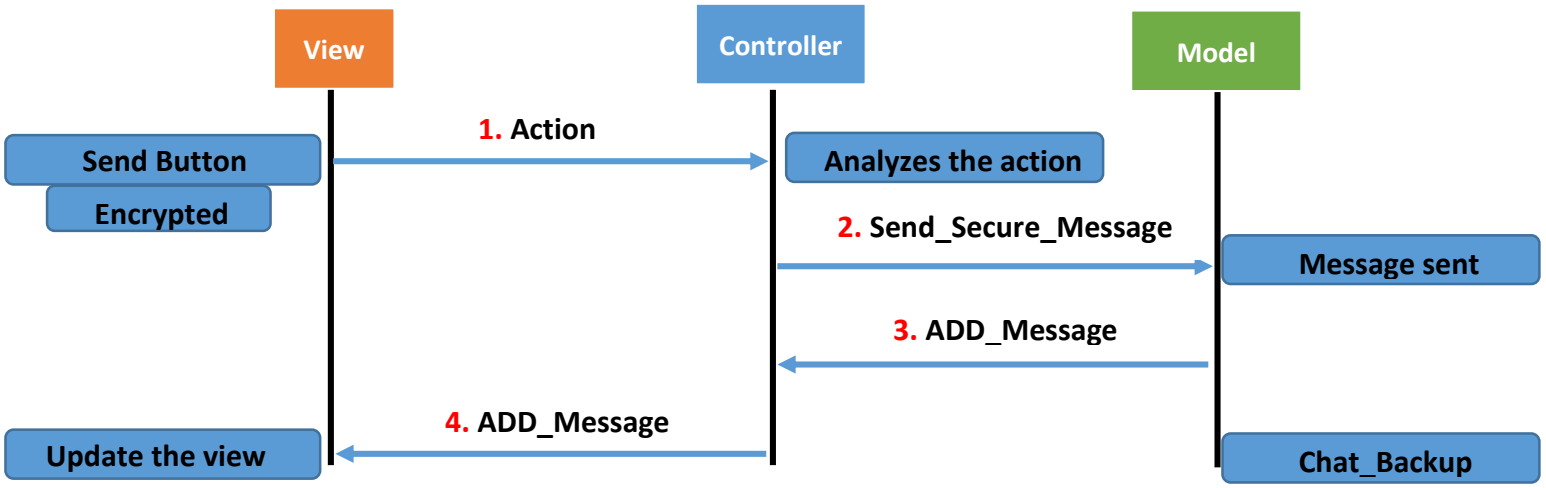


Figure 27 - Send message over a secure channel

4.7.4 CLIENT MODEL

This class implements the methods for the Client which will be called in some cases for the Controller and in other cases for some functions of the Client class. The main difference between both Models is that in the Server part a SSL server socket and a server socket are created whilst in the Client side a SSL socket and a socket are created. This is because the Client is who connects to the Server the first time and the Server is who is waiting for connections. A part from that both entities have the same classes because they are let to do the same things as it will be done in a messaging or chat application.

In the case the connection is lost or finished by one of the two sides, the other side is alerted with a message indicating that the connectivity has been lost.

ICHAT_CLIENT

This method is a bit different from the server's one because as this time an independent folder is created for the client. As in the Server method is done, the same folders **(Figure 28)** part are created under this folder called Client and then as it can be seen in the picture below the structure of folders is completely built and from so on both entities are able to send and receive messages and files in a secure and insecure way.

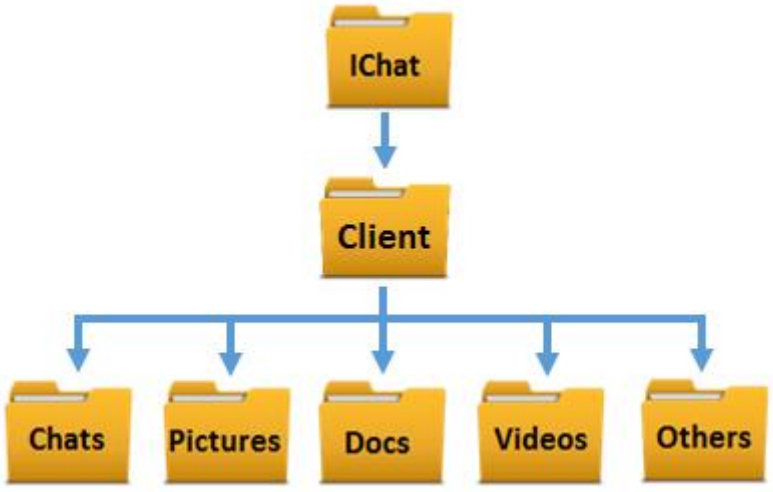


Figure 28 - Client's folders structure

When the Client has been correctly created and connected to the Server the resulting folders overview can be seen in the picture below **(Figure 29)**. As it can be seen both entities share the same root folder but they have independent folders to store the information received.

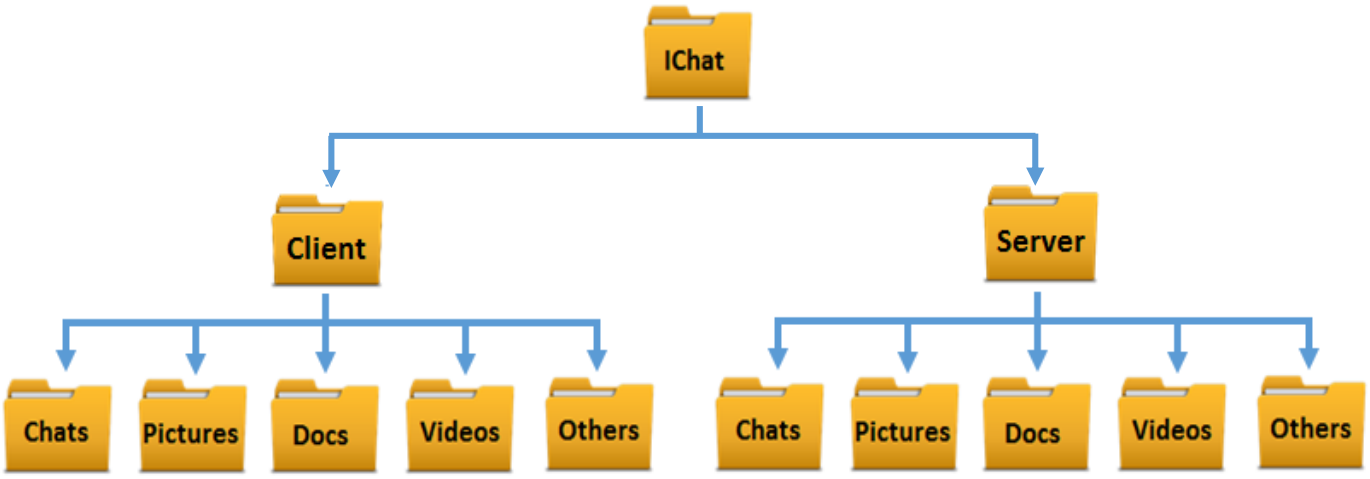


Figure 29 - Complete folder structure diagram

5. IMPLEMENTATION

In this chapter it can be seen all the procedures involved in the project development, as for instance, the graphical user interface implementation, the SSL handshake steps and a Wireshark analysis for both secure and insecure packets. Finally, a testing is done to verify and explain in detail all the functionalities developed in the project.

5.1 GRAPHICAL USER INTERFACE (GUI)

In computer science, a graphical user interface or GUI, is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces which require commands to be typed on the keyboard.

The actions in a GUI are usually performed through direct manipulation of the graphical elements. The elements used in our implementation are the following:

1. **JTextField:** is a lightweight component that allows the editing of a single line of text.

2. **JTextPane:** A text component that can be marked up with attributes that are represented graphically. This component models paragraphs are composed of runs of character level attributes. Each paragraph may have a logical style attached to it which contains the default attributes to use if not overridden by attributes set on the paragraph or character run. Components and images may be embedded in the flow of text.

3. **JButton:** Buttons can be configured and to some degree controlled, by Actions. Using an Action with a button has many benefits beyond directly configuring a button.

4. **JScrollPane:** Provides a scrollable view of a lightweight component. A JScrollPane manages a viewport, optional vertical and horizontal scroll bars and optional rows and columns heading viewports.

5. **JCheckBox:** This is an item which can be selected or deselected. The implementation of a check box displays its state to the user. By convention, any number of check boxes in a group can be selected.

6. **JLabel:** This component can display an area for a short text string, an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. However, a label can display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it. A JLabel object can display both text and images.

7. **ImageIcon:** An implementation of the Icon interface that paints Icons from Images. This images can be created from a URL, filename or byte array are preloaded using MediaTracker to monitor the loaded state of the image.

In the picture below **(Figure 30)**, it can be seen how these components have been combined all together and distributed to develop our interface.

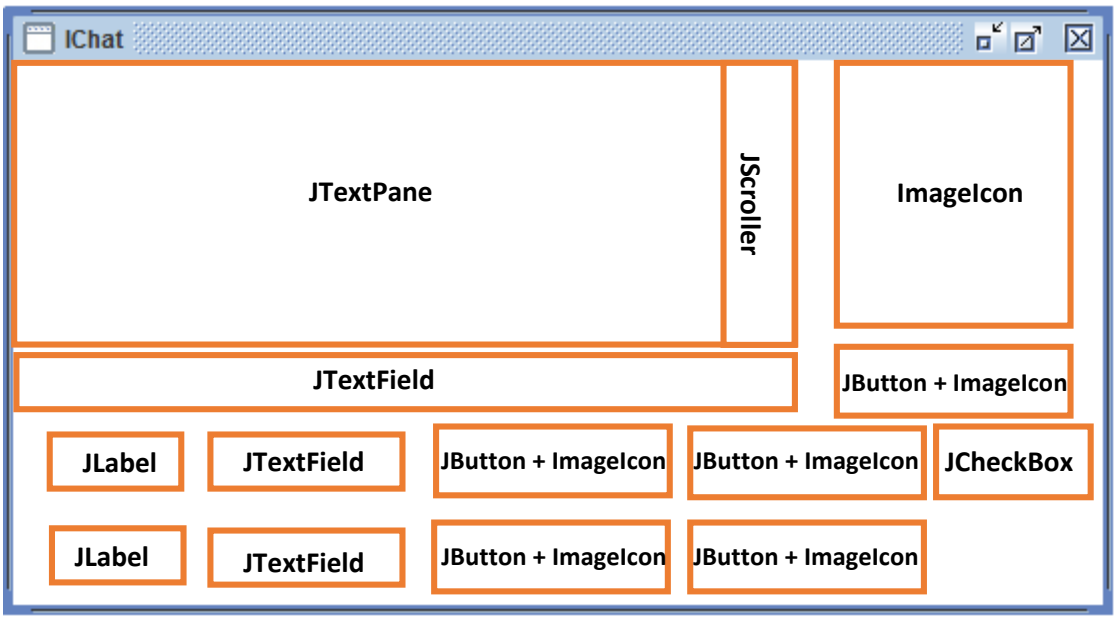


Figure 30 - Components of the interface

In the picture below **(Figure 31)**, it can be seen the interface physical appearance which has been built this way to be the easiest for the user to interact with.

Figure 31 - Interface

5.2 SSL HANDSHAKE

The steps involved in the SSL handshake **(Figure 32)** are as follows (note that the following steps assume the use of the cipher suites listed in Cipher Suites with RSA Key Exchange: Triple DES, RC4 and DES **(table [1])**):

1. The client sends the server the client's SSL version number, cipher settings, session-specific data, and other information that the server needs to communicate with the client using SSL.

   -SSL version: TLSv1.2.

   -Cipher settings: 32 cipher suites as it can be seen in the references **(table [1]).**

   -Signature algorithms: Sha512(ECDSA), SHA512(RSA), SHA384(RSA), SHA256(ECDSA), SHA256(RSA), SHA224(ECDSA), SHA224(RSA), SHA1(ECDSA), SHA1(RSA), SHA1(DSA) and MD5(RSA).

2. The server sends the client the server's SSL version number, cipher settings, session-specific data, and other information that the client needs to communicate with the server over SSL. The server also sends its own certificate, and if the client is requesting a server resource that requires client authentication, the server requests the client's certificate.

   -SSL version: TLSv1.2

   -Cipher settings: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

   -Signature algorithms: SHA512(RSA)

   -Handshake protocol: Server Hello, Certificate and Server Key Exchange

   -Prime number p: This number has a length of 2048 bits

   -Generator g: This number has a length of 1 bit

Server's Diffie-Hellman public value $A = g^X \bmod p$, where X is a private integer chosen by the server at random and never shared with the client.

When an ephemeral Diffie-Hellman cipher is used, the server and the client negotiate a pre-master key using the Diffie-Hellman algorithm. This algorithm requires that the server sends the client a prime number and a generator. Neither are confidential, and are sent in clear text. However, they must be signed, such that a MITM cannot hijack the handshake.

3. The client uses the information sent by the server to authenticate the server. If the server cannot be authenticated, the user is warned of the problem and informed that

an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client proceeds to step 4.

The step **4** can be deployed in two different ways:

4.1 Using all data generated in the handshake thus far, the client (with the cooperation of the server, depending on the cipher being used) creates the pre-master secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in step 2), and then sends the encrypted pre-master secret to the server.

4.2 If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case, the client sends both the signed data and the client's own certificate to the server along with the encrypted pre-master secret.

-Hanshake type: Client Key Exchange

-Public key length: 65 bits

In our case, the **4.2** has been the option selected as authentication for both entities has been the objective.

5. If the server has requested client authentication, the server attempts to authenticate the client. If the client cannot be authenticated, the session ends. If the client can be successfully authenticated, the server uses its private key to decrypt the pre-master secret, and then performs a series of steps (which the client also performs, starting from the same pre-master secret) to generate the master secret.

- **The ChangeCipherSpec** message is sent by both the client and the server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys. Reception of this message causes the receiver to instruct the record layer to immediately copy the read pending state into the read current state. Immediately after sending this message, the sender must instruct the record layer to make the write pending state the write active state. The ChangeCipherSpec message is sent during the handshake after the security parameters have been agreed upon, but before the verifying Finished message is sent.

6. Both client and server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity (that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection).This process is explained in detail in the Diffie-Hellman section.

7. The client sends a message to the server informing that future messages from the client will be encrypted with the session key. Then, the client sends a separate (encrypted) message indicating that the client portion of the handshake is finished.

8. The server sends a message to the client informing that future messages from the server will be encrypted with the session key. Then, the server sends a separate (encrypted) message indicating that the server portion of the handshake is finished.

9. The SSL handshake is now complete and the session begins. The client and the server use the session key to encrypt and decrypt the data they send to each other and to validate its integrity.

10. This is the normal operation condition of the secure channel. At any time, due to internal or external stimulus (either automation or user intervention), either side may renegotiate the connection, in which case, the process repeats itself.

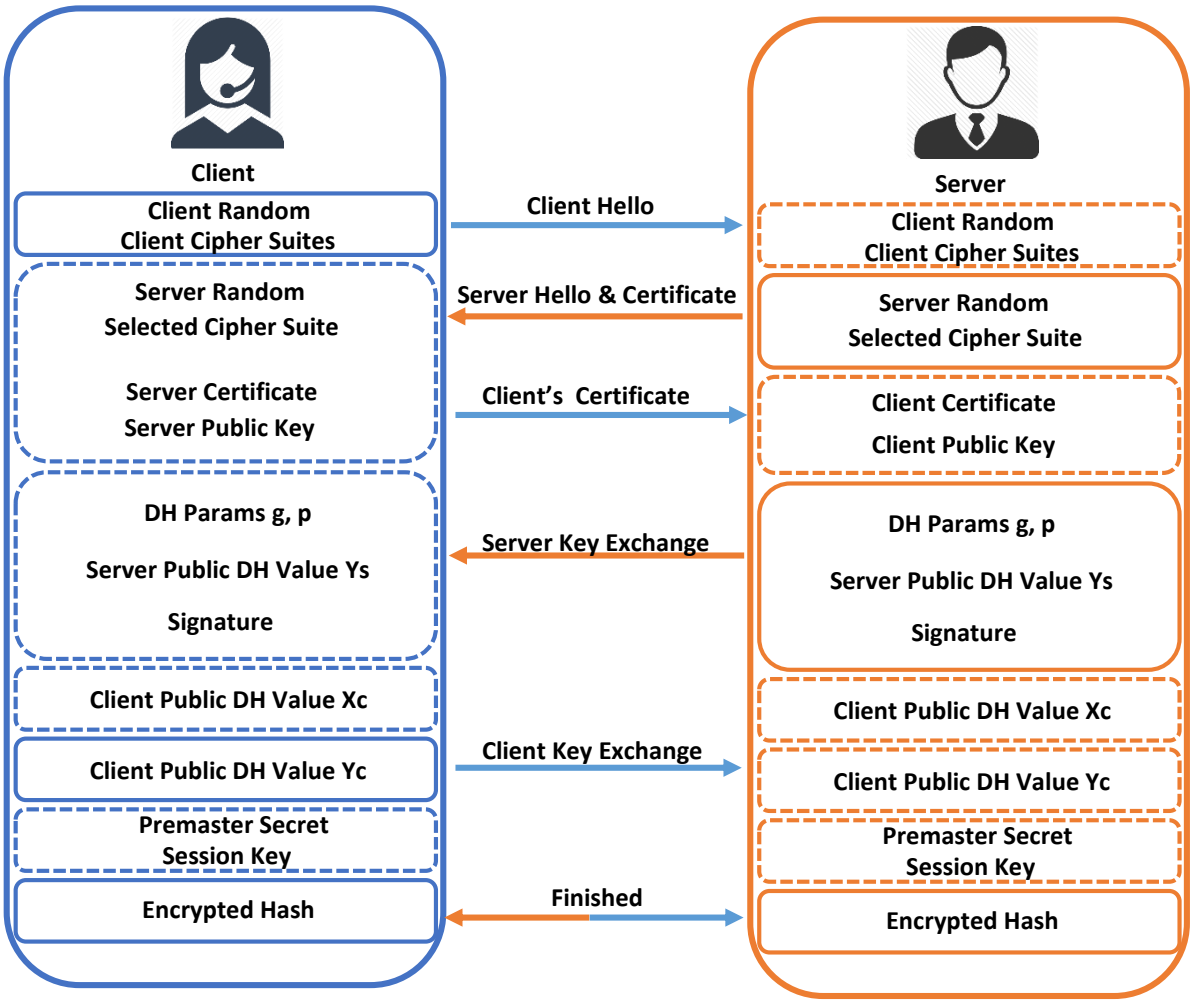


Figure 32 - SSL handshake

5.3 WIRESHARK ANALYSIS

As it has been commented before, in this project a chat has been implemented to be able to ensure both, secure and insecure communications. In this communications, each of both parts, Client and Server, can exchange messages and file, as for instance, videos, pictures and documents in different formats.

To check the differences between both channels, Wireshark application has been used to compare:

- Initial procedures

- Security aspects

5.3.1 SOCKETS

When sockets are implemented over the TCP protocol, they have the following properties:

1. They are connection oriented.
2. The data transmission is guaranteed without errors and omissions.
3. It is guaranteed that all the octets are going to get their destination in the same order in which they have been sent.

As it has been verified **(Figure 33)** using Wireshark tool, when packets are sent over a TCP channel, there isn't an initial procedure between both, sender and receiver, to stablish a secure channel or to present themselves and for this reason the information is sent over an insecure channel in which the confidentiality, integrity and security is not guaranteed because in this case, the port is the only property that identifies them. Then, an external Mallory entity can eavesdrop the channel, take the packets and figure out the information contained. Moreover, it can use the man in the middle attack to supply the original sender and modify the packet information.

As it can be seen, the data packets are sent over TCP but no security measures are applied and the information can be easily intercepted by a Mallory entity. As Internet is not a trustable medium this is not an optimal way when personal or private information has to be sent.

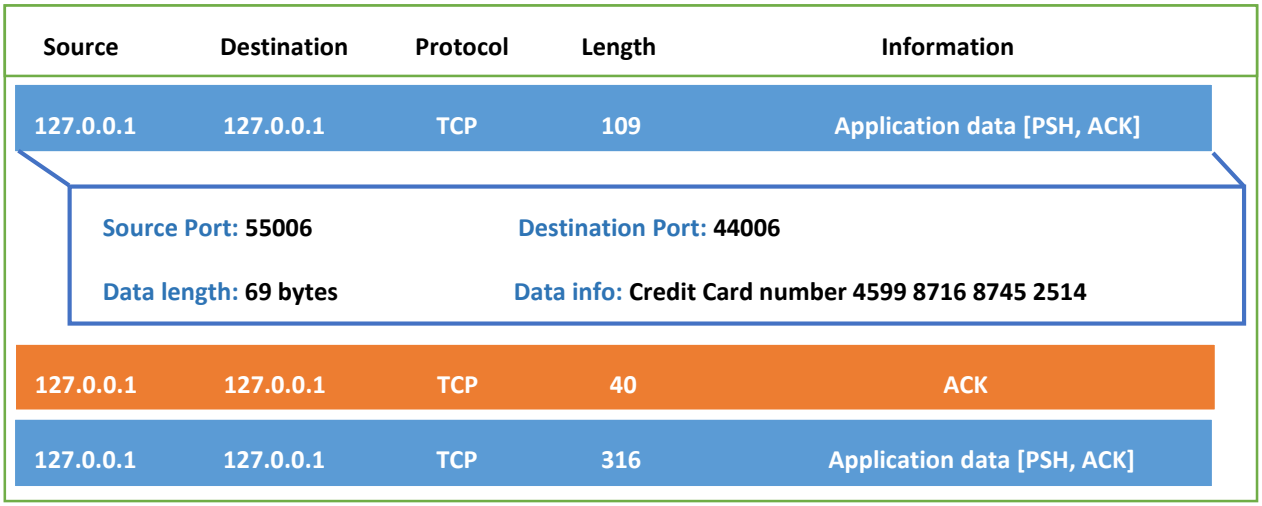


Figure 33 - Socket Wireshark capture

5.3.2 SSL SOCKETS

Such as sockets, SSL sockets are normal stream sockets, but they add a layer of security protections over the underlying network transport protocol, such as TCP. Those protections include:

1. **Integrity Protection:** SSL protects against modification of messages by an active wire tapper.
2. **Authentication:** In most modes, SSL provides peer authentication. Servers are usually authenticated, and clients may be authenticated as requested by servers.
3. **Confidentiality:** In most modes, SSL encrypts data being sent between client and server. This protects the confidentiality of data, so that passive wire tappers won't see sensitive data such as financial information or personal information of many kinds.
4. **No repudiation:** As in the handshake procedure both entities have exchanged their certificates and they trust each other, when information is received from the other side or vice versa, the receiver is sure whom this information comes from.

These kinds of protection are specified by a "cipher suite", which is a combination of cryptographic algorithms used by a given SSL connection. During the negotiation process, the two endpoints must agree on a cipher suite that is available in both environments. If there is no such a suite in common, no SSL connection can be established, and no data can be exchanged.

The cipher suite used is established by a negotiation process called "handshaking". The goal of this process, as it is explained in detail in the SSL section is to create or rejoin a "session", which may protect many connections over time.

Source	Destination	Protocol	Length	Information
127.0.0.1	127.0.0.1	TLSv1.2	294	Client Hello
127.0.0.1	127.0.0.1	TLSv1.2	1396	Server Hello, Certificate, Key Exchange
127.0.0.1	127.0.0.1	TLSv1.2	1396	Client Certificate, Client key Exchange
127.0.0.1	127.0.0.1	TLSv1.2	46	Change Cipher Spec
127.0.0.1	127.0.0.1	TLSv1.2	141	Encrypted Handshake Message
127.0.0.1	127.0.0.1	TLSv1.2	46	Change Cipher Spec
127.0.0.1	127.0.0.1	TLSv1.2	141	Encrypted Handshake Message

Figure 34 - SSL socket Wireshark capture

5.5 TESTING

In this section, it is explain in detail all the functionalities available in the project. Some of them are information messages shown depending on the user's behavior. Moreover, it is explained in depth how the messages are built into the chat box and their characteristics depending on if the information is sent or received. Finally, it is shown, with some examples, how the backup file works and its main features.

INVALID PORT NUMBER

When the start Server button is pushed or a connection to the Server is required by the Client pressing the connect button, then the host and the port fields are checked. In the case of the port number, it is verified if the number introduced is compressed between 40000 and 65535. The lower bound is because numbers lower can be used by other programs and the upper bound is because this is the maximum port allowed. If the number introduced is correct, the following step is to check the host and after that, the session can be begun.

Figure 35 - Invalid port number

INVALID HOST

As in the previous case explained, when the start Server button is pushed or a connection to the Server is required by the Client pressing the connect button, then the host and the port are checked. In the case of the host number, it is checked that the number introduced is 127.0.0.1 because the program is executed in the localhost machine. When both, port and host are correct parameters the Server can be started or the Client ca be connected to start a session.

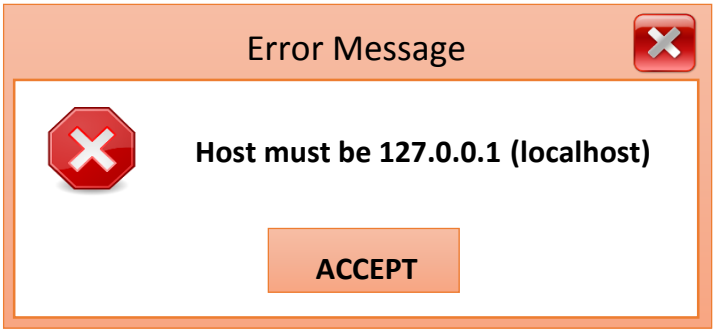


Figure 36 - Invalid host number

EMPTY MESSAGE

When the session has been started, both Client and Server can exchange messages and files but as it is obvious empty messages are not allowed because it doesn't make sense in a chat application and for this reason every time the send message button is pressed, the input of the corresponding text field is checked and if it is empty the following message is shown to the user.

Figure 37 - Empty message

START SERVER WITH CORRECT PARAMETERS

When the parameters introduced are correct then an information message is shown in the Server's chat to specify the listening port in which connections requests will be accepted. In this specific case **(Figure 38)**, the Server is waiting for Clients in the port number 44006.

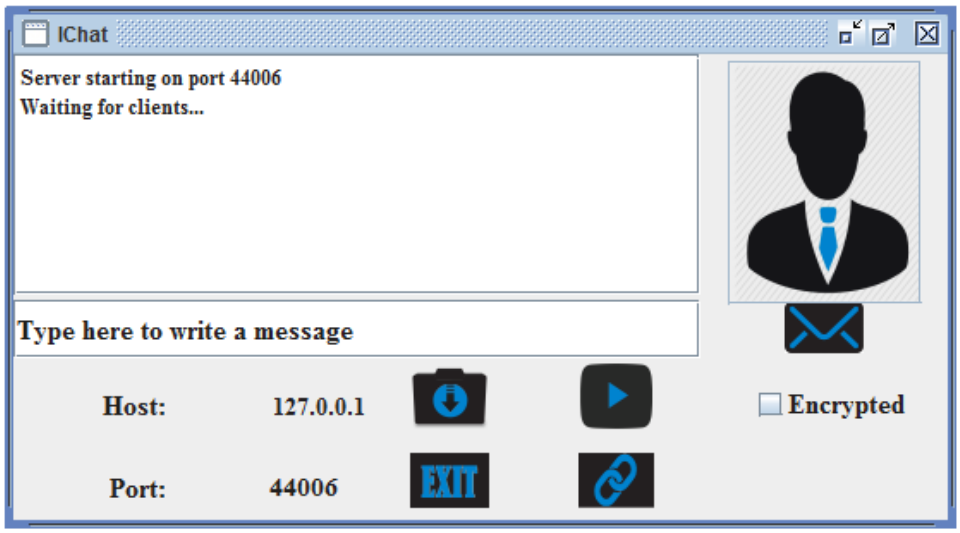


Figure 38 - Start Server with incorrect parameters

START CLIENT WITH CORRECT PARAMETERS

After the Server is running, the Client can be connected to establish a socket and a SSL socket between them. When the Client is well connected, a message with the connection has been stablished is showed as it is depicted in the picture below **(Figure 39)**.

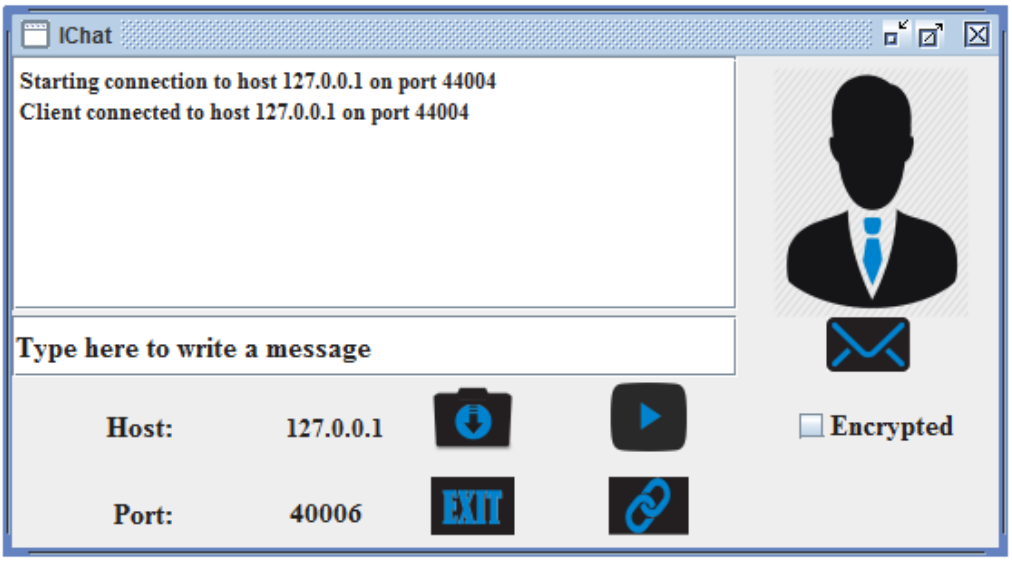


Figure 39 - Start Client with correct parameters

When a connection from a Client is received, a message is shown in the Server's chat indicating a secure and insecure Client has been connected. Then, both entities are able to exchange messages and files in both ways, secure and insecure channel using SSL sockets and sockets.

Figure 40 - Server message after the Client is connected

SEND INSECURE MESSAGE FROM SERVER TO CLIENT AND VICEVERSA

After the connection is established, as it can be seen in the picture below **(Figure 42)**, an insecure message has been sent from the Server to the Client and after that, the Client has answered the message **(Figure 41)**. In both chats, when a message is sent the color is blue and for the received messages the color is gray. This has been done this way because it is easier to the user to differentiate which messages has been sent and which of them have been received. Moreover, a header is added before every message showing the date in hour, minutes and seconds. When a message is added to the chat box the backup file is updated with a copy of the message and a header indicating if the message was sent or received in a secure or insecure way.

Figure 42 - Send insecure message from Server to Client

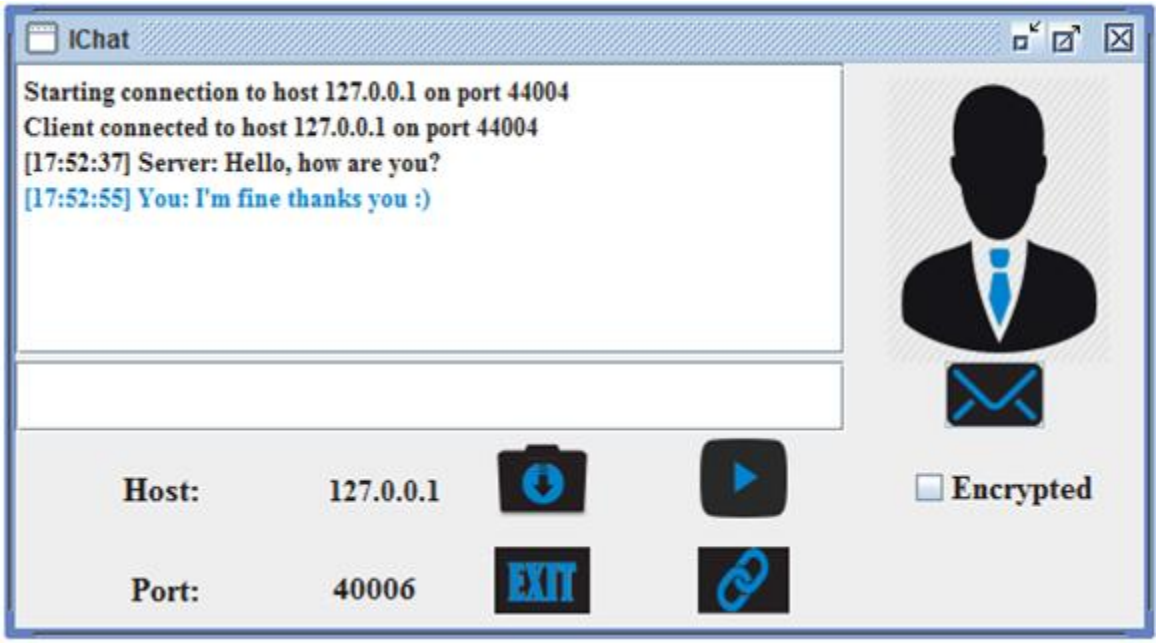


Figure 41 - Send insecure message from Client to Server

In this particular case, the Server wants to send to the Client a secure message and to do so the encrypted checkbox has to be clicked. If the Client has the encrypted checkbox clicked it means there is synchronism between both entities and the message will be decrypted in the Client side being able to recover the original information but if the Client isn't synchronized with the Server then the Client won't be able to decrypt the message because it won't use the session key and the original message won't be recovered.

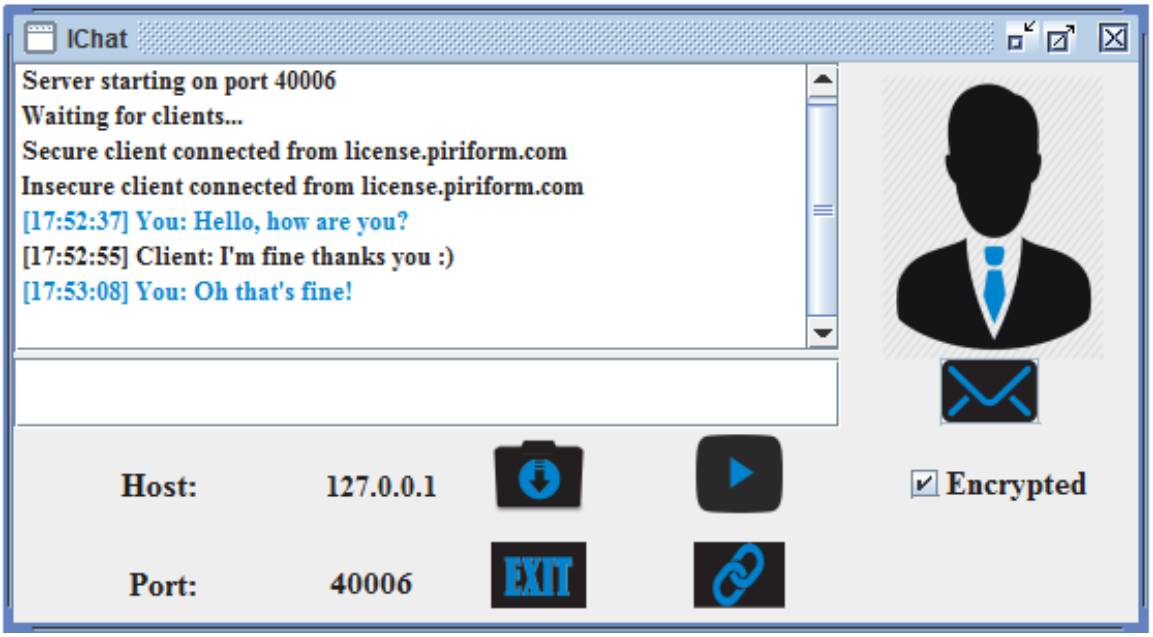


Figure 43 - Send a secure message to the Client without synchronization

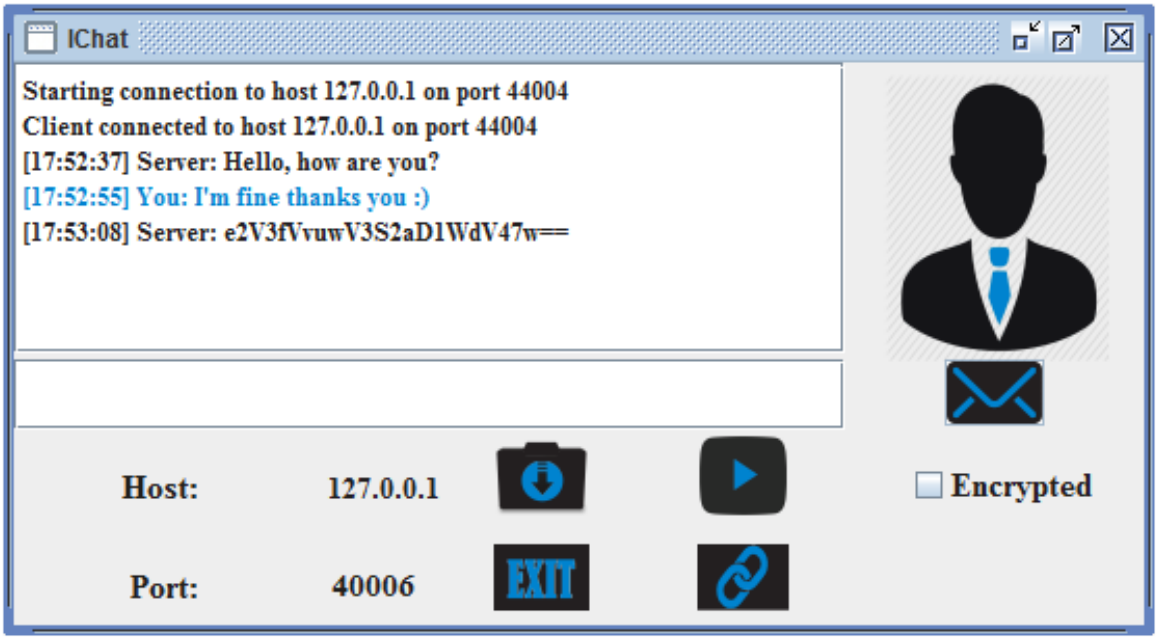


Figure 44 - Receive a message without synchronization

FILE SENT FROM THE CLIENT TO THE SERVER

When the Server or Client want to send a file they have to click to the send file button and then a file chooser is open **(Figure 45)**. When a file is chosen an information message is shown asking the user if it is sure of sending the file. Finally, if the yes button is clicked by the user, the file is sent and an information message is shown to the sender chat saying the file was correctly sent **(Figure 46)**.

As it has been explained before, both entities are able to send a file over a secure or insecure channel but is desired to send files over the secure channel because there can be personal information.

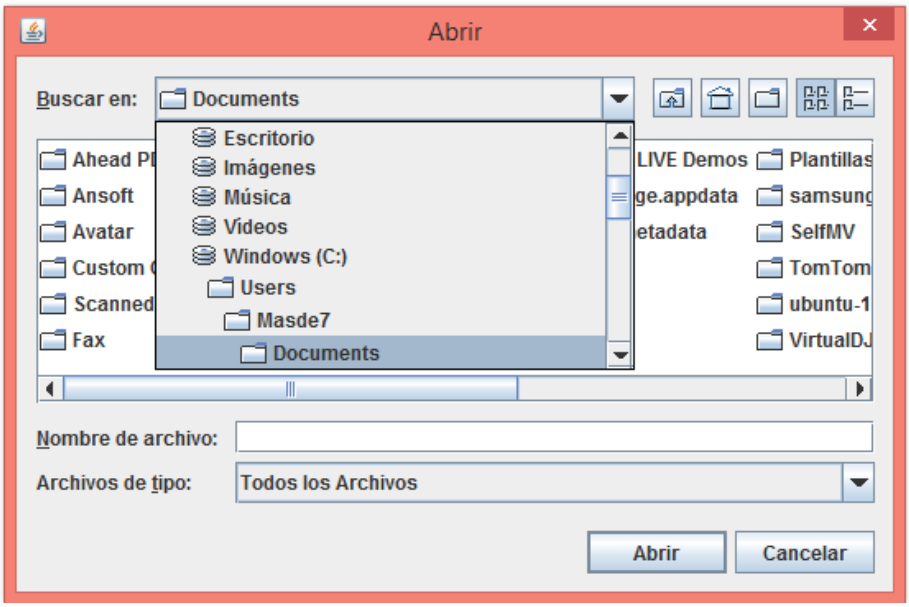


Figure 45 - File chooser

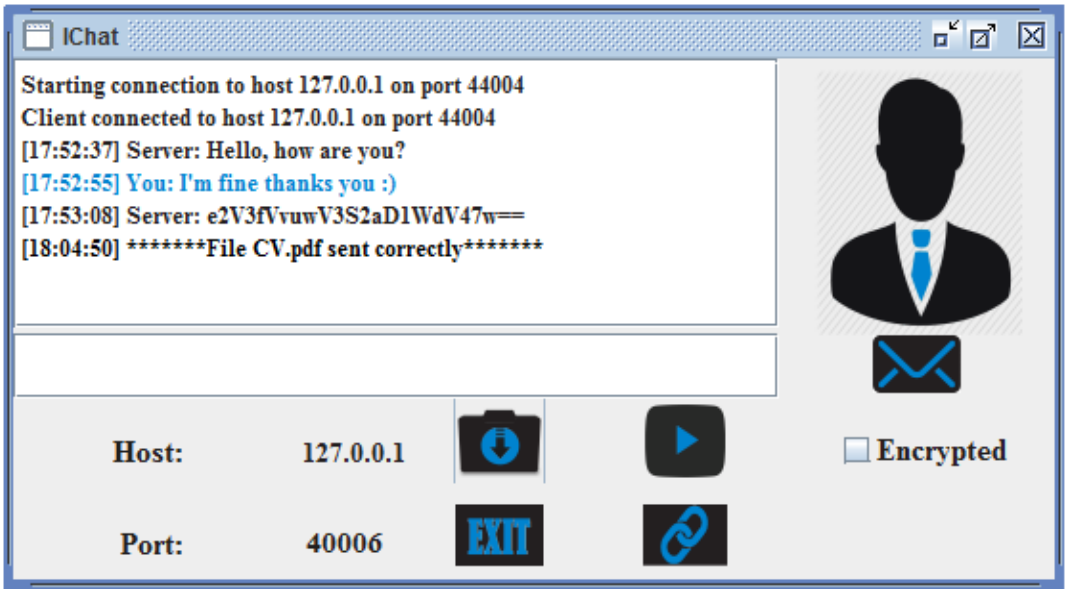


Figure 46 - File correctly sent

When the file is received by the user, a header is added to the chat showing the file was completely received and it can be seen the path in which the file is stored.

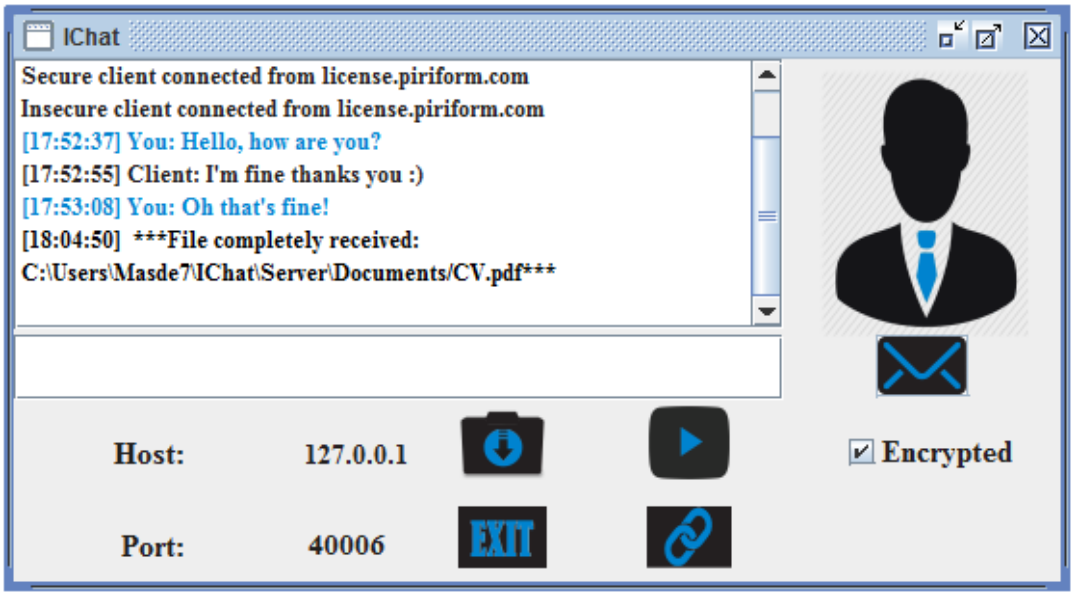


Figure 47 - File correctly received

MESSAGE CONFIRMATION BEFORE SENDING THE FILE

This is the confirmation message shown every time a file has to be sent by either Client or Server. If the yes button is clicked by the user, the file is sent and if the no button is clicked the user has the possibility of choosing another file. As it has been mentioned before, the maximum size allowed is 6Mb.

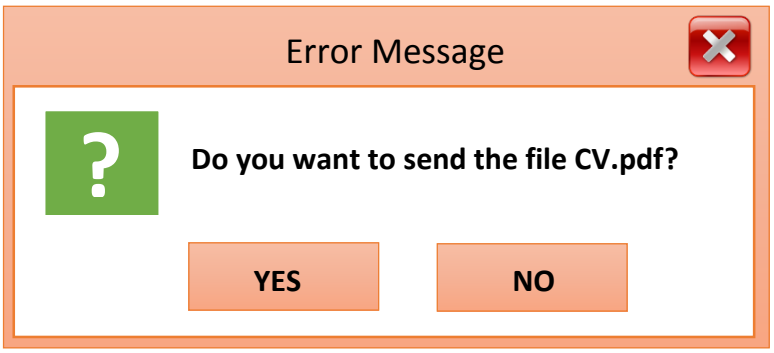


Figure 48 - Send file confirmation message

FILE EXCEEDS THE MAXIMUM SIZE ALLOWED

If the user tries to send a file larger than the maximum size allowed then an information message is shown indicating the file can't be larger than 6Mb and the user has the possibility of choosing another file. It has been thought that for the purposes of this project sizes larger than 6Mb would slow down the application.

Figure 49 - File excedes máximum size allowed

CLIENT IS DISCONNECTED

When the session is ended by the Client, a message is shown to the Server indicating the Client has left and then a new Client can be connected to the Server.

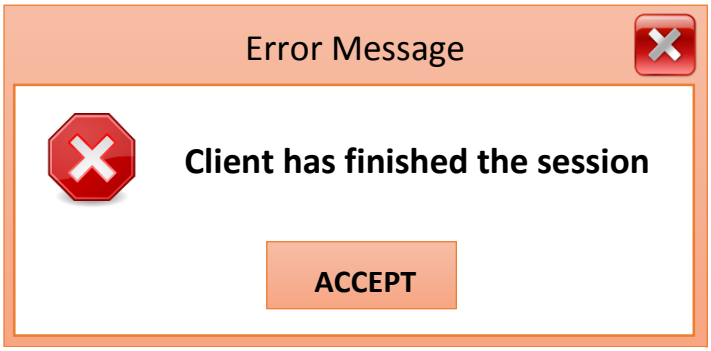


Figure 50 - Client ends the session

SERVER IS DISCONNECTED

When the Server is not available a message is shown to the user indicating the Server has left and it doesn't make sense to have the session running as messages and files can't be sent.

Figure 51 - Server ends the session

EXIT BUTTONCONFIRMATION MESSAGE

To leave the session, both Client and Server can use the exit button provided in the interface and then a message indicating if they are sure is shown. If the yes button is clicked the sockets are closed and the interface is cleared out form the screen. Then a message will indicate to the other entity that the communication isn't possible.

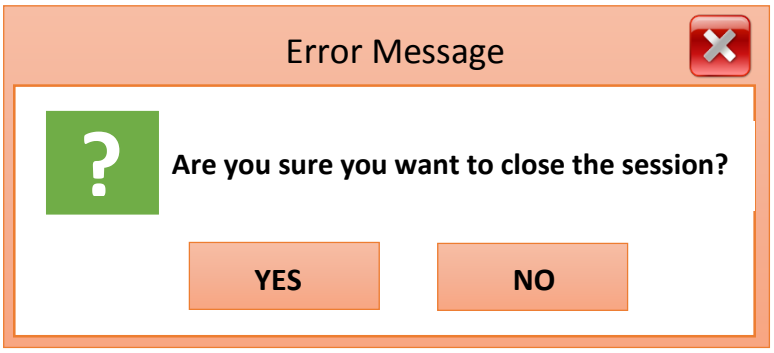


Figure 52 - Exit message confirmation

SERVER BACKUP FILE

When the Server is started, a backup file is created into the Server Chats folder with a unique name formed by the date in the following format: Year/Month/day/Hour/Minute/Seconds. Then, every time a message or a file is sent or received the chat box is updated with the corresponding information. After that, a copy of this information is stored in the backup file adding a header showing if the message or file was sent or received in a secure or insecure way.

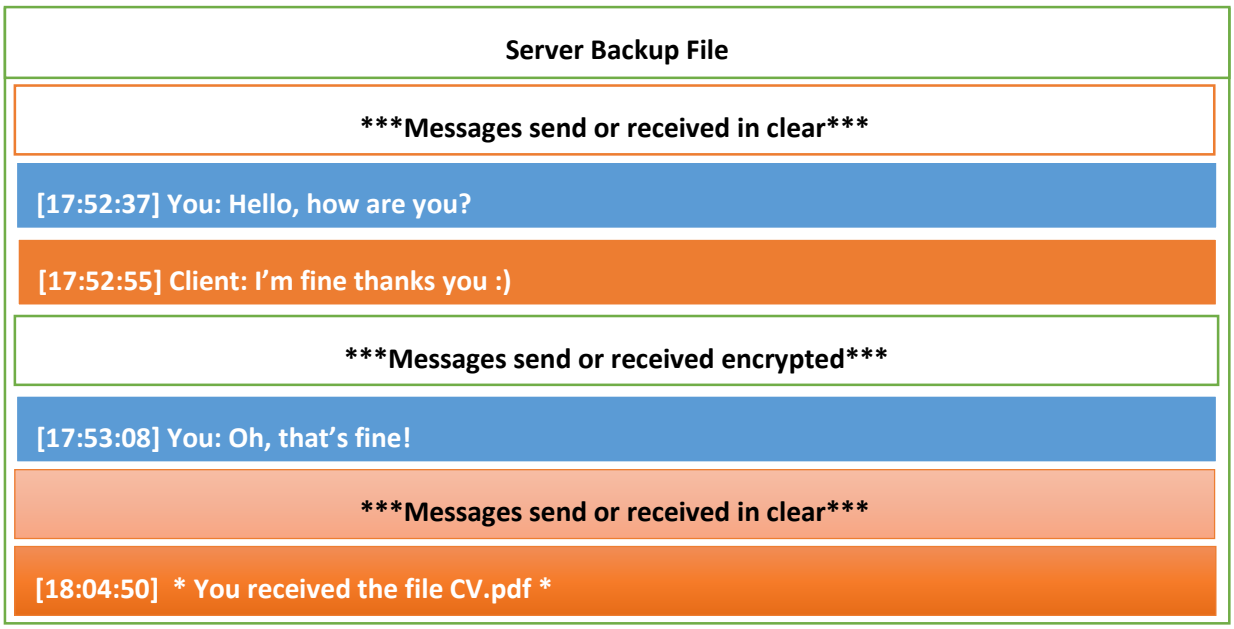


Figure 53 - Server backup file

CLIENT BACKUP FILE

When the Client is connected to the Server, a backup file is created into the Client Chats folder with a unique name formed by the date in the following format: Year/Month/day/Hour/Minute/Seconds. Then, every time a message or a file is sent or received the chat box is updated with the corresponding information. After that, a copy of this information is stored in the backup file adding a header showing if the message or file was sent or received in a secure or insecure way.

Figure 54 - Client backup file

6. CONCLUSIONS

During my degree in telecommunications specified in the telematics field and my master in telecommunications engineering I have been studying some subjects referring to security topics but I haven't been in direct touch because we did only the theory part but not the practice part. This was the key aspect I make this choice.

The idea of this project was to be able to create two interfaces using JAVA code, one for each Client and Server, in which messages and files could be sent using a secure or insecure channel using normal sockets or SSL sockets in which a pair of certificates would be needed to identify both sides. When a file or a message would be received, it was desired to store the information to be able to check this information when needed. To do so, it was implemented a folder's structure in which the information was stored depending on the content and the messages sent and received were stored in backup files.

In the first case, no cryptographic methods would be applied while in the second case, a mixed of some cryptographic methods such as symmetric, asymmetric and Diffie-Hellman would be combined.

Mixing the best properties of all this methods it was got our initial aim of having a chat in which two entities, a Server and a Client, were able to agree a session key to encrypt and decrypt messages and to send them over a SSL channel guaranteeing integrity, confidentially and no repudiation of the information transmitted. This three properties are really important because Internet is a network of networks and the information sent isn't always travelling over a trusted way and for this reason is crucial to protect the data before sending. To be able to create a SSL socket it was used the Keytool program to generate a pair of certificates.

During the creation of the secure channel it has been seen all the procedures involved in the SSL handshake in which both Client and Server exchange different information to be able to generate the same session key using the Diffie-Hellman method and to avoid sending the session key over Internet.

Moreover, it was desired to have an insecure channel in which no cryptographic methods were applied. This channel was used to check the differences respecting to the SSL channel and it has been seen that confidentiality, integrity and no repudiation can't be guaranteed when no cryptographic methods are applied to the information before sending over the channel. To do so, Wireshark tool has been used with RAWCAP tool to be able to analyze the packets transmitted over the local loop.

After comparing both methods, it has been verified that man in the middle attacks are really feasible over the insecure channel and it can affect to the transmitted information modifying the content or changing the remittent. Over the secure channel as several cryptographic techniques have been used the data is really difficult to decrypt by a Mallory entity and the receiver can be sure that confidentiality, integrity, security and no repudiation is maintained.

Finally, all the goals explained in the introduction were accomplished and all the theory learnt during the studies was physically applied.

7. REFERENCES

In this last chapter it can be seen the resources used in the project's development and some extra information as for instance tables about the encryption possibilities offered by JAVA and an overview of all the figures used in the project.

7.1 LINKS

[1] Asymmetric cryptography:

https://en.wikibooks.org/wiki/Cryptography/Asymmetric_Ciphers

http://hitachi-id.com/concepts/asymmetric_encryption.html

https://www.cs.utexas.edu/users/byoung/cs361/lecture44.pdf

http://computer.howstuffworks.com/encryption3.htm

https://www.comodo.com/resources/small-business/digital-certificates2.php

[2] Symmetric cryptography:

https://www.princeton.edu/~rblee/ELE572Papers/CSurveys_SymmAsymEncrypt-simmons.pdf

https://support.microsoft.com/en-us/kb/246071

http://www.webopedia.com/TERM/S/symmetric_key_cryptography.html

http://www.cs.cornell.edu/courses/cs5430/2010sp/TL03.symmetric.html

[3] Diffie-Hellman

http://mathworld.wolfram.com/Diffie-HellmanProtocol.html

https://www.ietf.org/rfc/rfc2631.txt

https://www.secpoint.com/what-is-diffie-hellman-encryption.html

[4] SSL

https://www.digicert.com/ssl.htm

https://www.instantssl.com/ssl.html

http://searchsecurity.techtarget.com/definition/Secure-Sockets-Layer-SSL

[5] Sockets

https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

http://www.tutorialspoint.com/java/java_networking.htm

[6] JAVA

https://www.codecademy.com/learn/learn-java

https://www.oracle.com/java/index.html

[7] ECLIPSE

https://eclipse.org/home/index.php

[8] X.509 Certificates

http://www.cypherpunks.to/~peter/T2a_X509_Certs.pdf

http://docs.oracle.com/javase/7/docs/api/java/security/cert/X509Certificate.html

[9] Keytool

http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html

https://www.sslshopper.com/article-most-common-java-keytool-keystore-commands.html

http://alvinalexander.com/java/java-using-keytool-list-query

[10] Wireshark

https://www.wireshark.org/

[11] Model View Controller (MVC)

https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

https://msdn.microsoft.com/en-us/library/ff649643.aspx

http://programmers.stackexchange.com/questions/127624/what-is-mvc-really

7.2 TABLES

[1] Cipher suites possibilities during the handshake negotiation:

CIPHER SUITE	KEY EXCHANGE	CIPHER	MAC
TLS_NULL_WITH_NULL_NULL	Null	Null	Null
TLS_RSA_WITH_NULL_MD5	RSA	Null	MD5
TLS_RSA_WITH_NULL_SHA	RSA	Null	SHA
TLS_RSA_WITH_NULL_SHA256	RSA	Null	SHA256
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	AES_128_CBC	SHA
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	AES_256_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA	AES_128_CBC	SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA	AES_256_CBC	SHA256
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH_DSS	AES_128_CBC	SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH_RSA	AES_128_CBC	SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE_DSS	AES_128_CBC	SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE_RSA	AES_128_CBC	SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA	DH_anon	AES_128_CBC	SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DH_DSS	AES_256_CBC	SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA	DH_RSA	AES_256_CBC	SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE_DSS	AES_256_CBC	SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	DHE_RSA	AES_256_CBC	SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA	DH_anon	AES_256_CBC	SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA256	DH_DSS	AES_128_CBC	SHA256
TLS_DH_RSA_WITH_AES_128_CBC_SHA256	DH_RSA	AES_128_CBC	SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256	DHE_DSS	AES_128_CBC	SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	DHE_RSA	AES_128_CBC	SHA256
TLS_DH_anon_WITH_AES_128_CBC_SHA256	DH_anon	AES_128_CBC	SHA256
TLS_DH_DSS_WITH_AES_256_CBC_SHA256	DH_DSS	AES_256_CBC	SHA256
TLS_DH_RSA_WITH_AES_256_CBC_SHA256	DH_RSA	AES_256_CBC	SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	DHE_DSS	AES_256_CBC	SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	DHE_RSA	AES_256_CBC	SHA256
TLS_DH_anon_WITH_AES_256_CBC_SHA256	DH_anon	AES_256_CBC	SHA256

7.3 TABLE OF FIGURES