Final Year Project Report

Full Unit – Final Report

Prime Numbers and Cryptosystems

Svetoslav Mihovski

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: prof. Zhaohui Luo

Department of Computer Science

Royal Holloway, University of London

March 28, 2018

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 21229

Student Name: Svetoslav Mihovski

Date of Submission: 28.03.2018

Signature: Svetoslav Mihovski

Contents

Abstract

Today information security is a subject with relatively high profile and importance. With the rise of the significance of information security comes and the important link with cryptography. As a result of that, cryptography has become widespread in our everyday world. To be able to achieve the required security cryptography is often related with prime numbers. In encryption it is common to use large semi-prime numbers, which are resulted by multiplication of two prime numbers. Cryptosystems in cryptography are associated with cryptographic algorithms that implement a particular security service. A cryptosystem uses three different algorithms for: key generation, encryption, and decryption.

One of the first public-key cryptosystems is RSA which is often used about secure data transmission. This particular cryptosystem uses public encryption key and a different private decryption key. The main idea behind RSA is based on that it is easy to multiply two very large prime numbers however, it is near impossible to reverse the operation– meaning to find the two prime numbers from a very large number. Because of RSA being relatively slow algorithm on its own, it finds more practicality as the keys distributor in symmetric key cryptography and performs at much higher speed. Symmetric key algorithms are algorithms that are using identical keys for encryption of the plain text and the decryption of the cipher text.

Focusing more on the project itself, the proof-of-concept software is representing simple encryption and decryption implementation of RSA and block ciphers. Further, the third proof-of-concept program required a graphical user interface which represents a local-host based chat between a Client and a Server. The combination of the three proof-of-concept programs, forms the prototype of the end system, the ‘Secure Chat Prototype’. Moreover, many changes and improvements were done over the prototype in order to progress towards achieving the final version of the software.

In the fast developing world that we live with the extreme rise of the technology in our everyday lives there is no better opportunity to get involved with information security. This particular project allows me to get a better understanding of cryptography and what motivates me the most is its practical aspect. My expectations are positive and I am looking forward in getting involved with the project because of the valuable experience and knowledge I will get at the end of my final year. I believe that this project will be beneficial and extremely helpful for my future career development in information security.

Project Specification

Prime Numbers and Cryptosystems

Aims: The project is associated with implementing basic techniques with prime numbers and then developing cryptographic functions using them.

Background: Modern cryptography allows us to perform different types of information exchange over insecure channels. One of these task is to agree on a secret key over a channel where messages can be overheard. This is achieved by Diffie-Hellman protocol. Other tasks include public key and digital signature schemes; RSA key exchange can be used for them. These protocols are of great importance for bank networks.

Most such algorithms are based upon number theory, namely, the intractability of certain problems involving prime numbers.

Early Deliverables:

1. Proof of concept program: An implementation of RSA using standard data types in Java or C++ encrypting and decrypting numbers.

2. Proof of concept program: A simple program generating keys.

3. Report: A description of the RSA encryption/decryption procedure.

4. Report: Examples worked out on paper and checked using the proof of concept program.

Final Deliverables:

1. The program must have a full object-oriented design, with a full implementation life cycle using modern software engineering principles.

2. The program will use integers of arbitrary size and encrypt and decrypt test messages and files.

3. An application with a GUI will perform a task such as a secure chat or a secure file transfer over the network.

4. A combination of RSA and a symmetric encryption scheme will be implemented, where RSA will distribute the keys for the symmetric scheme.

5. The report will contain an overview of modern cryptography.

6. The report will describe RSA and number theory issues such as primality checking and factorisation.

7. The report will describe the implementation issues (such as the choice of data structures, numerical methods etc.) necessary to apply the theory.

8. The report will describe the software engineering process involved in generating your software.

9. The report will contain the data on the running time of the program.

Suggested Extensions:

1. An advanced symmetric encryption scheme such as DES or AES will be implemented.

2. Different primality testing and factorisation methods will be implemented. Their performance will be compared and test results reported.

3. Attacks on RSA and/or the symmetric scheme will be implemented and tested.

Chapter 1: Introduction

The project requires both the combination of theoretical and practical experience in the aspect of information security. The leading task of the project is the development of a Java based application that would perform secure chat and file transfer over a network implementing modern cryptography techniques. This is to be achieved using requirements gathering, resource analysis, design patterns, proof-of-concept programs and testing.

1.1 Aims and Goals of the Project

The main distinguishable aims and goals of the project are grouped in two categories: project documentation and program development. Moreover, the progress of both divisions is driven by the deliverables described by the project specification and any additional identified deliverables.

The development focus in the first term was on creating a working prototype of the system that would be covering all of the early deliverables and most of the final deliverables from the project specification. In order to achieve this, the work process was broken down into the development of three proof-of-concept programs that would establish the fundamentals of the prototype. Moreover, these proof-of-concept programs stressed the knowledge and implementation of cryptographic algorithms in Java. The prototype itself represents a huge milestone of the project since it shows the core functionalities that the end system will perform. Furthermore, the completed version of the prototype allows end user testing to be proceed which would contribute in the identification of additional features and extensions of the program.

During the second term the goals of the project included both completing all of the deliverables from the specification and the identification of new additional features or improvements that can be implemented. Moreover, the aims and goals of the second term can be described as gradual improvement process of the prototype, which process would lead to the final form of the application. Further, the outlined changes stressed on interface changes, improvements in the encryption procedures, implementation of additional security features and a complete rework of the communication procedure.

1.2 Project Motivation

Having the opportunity to be involved in an information security related project was intriguing because of the large scope of skills it requires. From developer's point of view the project would allow me to expand on my current skills and gain beneficial knowledge in the area of cryptography. Furthermore, in terms of software engineering the project heavily requires on the valuable experience I have gained in working with Java over the past few years of the course, combined with the implementation of cryptographic algorithms and techniques. On the other hand, the project itself has an exceptional practical aspect focusing on procedures and systems used in the real world. Moreover, the knowledge from the project will be practically based and it would enhance my current skills with examples and situations from the real world- leading to an invaluable experience that could be related to career prospects and employability. Finally, the theoretical part of the project is also exceptionally convenient in terms of cryptography and how cryptographic encryption schemas and algorithms work, extending my current knowledge gained from last year information security related courses.

To conclude the combination of the listed above functional and analytical skills are the main source of motivation because they would not only be applied for academic use but those skills are also indispensable for a further career development in the area of information security.

1.3 Literature Review

The complexity of the project required both intense theoretical and implementation knowledge, the most important information of which is provided by the outlined literature in this section.

Initially, the project required a lot of abstract insight on overall information security in addition to the specific topics that the project focuses on. Most of this information was provided by the book "Everyday Cryptography: Fundamental Principles and Applications" (Keith M. Martin, 2017). This source of information was essential to gain enough background knowledge to be able to proceed with the more concrete topics of the project. The book contains a detailed description of most of the fundamental principles and applications of cryptography. In addition to that, there are numerous examples of the implementation of symmetric and asymmetric encryption schemes and their relevant algorithms. Another satisfying part about this literature, is the fact that it also covers practical examples of how cryptography can be applied. The design of the information provided by the book was suitable to allow me to begin my project research. The only downside of the book would be that it did not concentrate on the mathematics behind the cryptographic mechanisms provided.

Further theoretical research was provided by the research papers: "How RSA Works With Examples" (Doctorina, 2012), "The Math behind RSA" (Math Berkeley, 2002). The main focus of the listed sources of information is about the mathematics behind the implementation of different cryptosystems. The knowledge gained from these research papers was crucial in understanding the core of RSA and block cipher algorithms. Furthermore, the information was fundamental when implementing cryptographic algorithms in practice in the form of a proof-of-concept program that performs encryption and decryption of input data string.

The book "Introduction to Java Programming" (Svetlin Nakov, 2017) and the lecture slides "CS2800 - Software Engineering" (D. Cohen, 2012) were extremely supportive when it comes to the implementation of RSA algorithm in Java and the choice of the data structures. The sources provide a detailed description on how properly to achieve a full object-oriented design using modern principles. As well as that, they covered material regarding design patterns and their correct implementation in addition to how to write clean code and implementing proper development methodologies. To conclude, the listed sources of information were vital in the development of the code required for the project.

Another important source considering the practical application of the project is the provided by Oracle documentation regarding the 'BigInteger' class (Docs Oracle, 2007). The information provided was sufficient regarding how to use the class and the methods it provides. Furthermore, the knowledge gained was extensively used in the development of the proof-of-concept programs of the project.

Further, the book "The Laws of Cryptography with Java Code' (Neal R. Wagner, 2003) allowed me to understand with practical examples of how different security protocols, algorithms and procedures are implemented. Further, the source was extremely valuable when implementing the Advanced Encryption Standard algorithm in Java because of the combination of code examples and good description of how the code is realized. In addition to that, the book also supported the implementation of authentication and data integrity mechanisms in the program, related to the session key exchange (signing procedures).

The literature provided in this section showcases only a narrow part of the whole bibliography. However, the sources provided are considered essential in order to achieve the aims and goals of the project, moreover the program wouldn't be manageable without the support of those outlined books and online sources.

1.4 Milestones of the Project

1.4.1 Reports

The section formulates the sequence of the reports completed during the whole development process of the project. Moreover, the section represents the documentation and its relevance to the end system. The purpose of the reports is to establish an overview and work sequence of how the final product would be achieved.

- **Overview of Modern Cryptography:** The report provides an analysis of the structural principles of modern cryptography. Moreover, it establishes the required basic knowledge to be able to get involved with the project.

- **The RSA Cryptosystem:** The report stresses the topic of the public encryption scheme RSA, it considers its work process, functionality and implementation.

- **Proof-of-Concept Development:** The report is a documentation of the initially developed programs and their related user stories, development process and justification.

- **End System Development:** The report is a documentation of the whole development process that concerned the finalization of the application.

- **Software Engineering Process:** The report supports the techniques, methodologies and tools used in the development process of the program.

1.4.2 Programs

The section lists the programs completed during the aggregate development process of the project. Furthermore, the completed work indicates the progress made towards achieving the final product.

- **RSA Encryption Program:** The program is an implementation of the RSA algorithm in Java. It encrypts and decrypts given input data string under the rules of public key cryptography.

- **Feistel Block Cipher Encryption Program:** The program represents the combination of asymmetric and symmetric key cryptography, allowing the user to encrypt and decrypt data efficiently.

- **Chat Communication Program:** The program implements a simple chat using an interface. Moreover, the program allows the communication between two users, it is based on the use of Java sockets and works in a peer-to-peer mode.

- **Secure Chat Prototype Program:** The program serves as an early prototype of the end system, performing live encryption and decryption of messages and files sent over in a chat environment, represented by a graphical user interface.

- **Chat Client Program:** This is the first instance of the end product. It is the client part of the final application and it contains classes and methods regarding the encryption, authentication and communication procedures.

- **Chat Server Program:** This is the second instance of the end product. It is the server part of the final application and it represents the classes and methods that establish the communication with the client.

Chapter 2: Overview of Modern Cryptography

Cryptography or the form of secret writing has been used for thousands of years with a simple purpose- to hide secret information. However, in the present cryptography finds a much larger scope of use with the development of computer communications. Modern cryptography is based around designing cryptography algorithms around certain computational hardness that would be in practicality unbreakable by an adversary. It brings together different areas such as probability theory, computational-complexity theory and numbers theory and combines them under the study of cryptography.

More specifically cryptography today is the science of using mathematical techniques for all fields of information security. It primary concerns four goals:

1. **Confidentiality:** The process of having the information being in an unintelligible form for anyone but the receiver.

2. **Integrity:** The process of detecting whether or not the information between the sender and receiver has been altered in storage or in transit.

3. **Non-repudiation:** The process of identifying that the source of the information was the sender himself.

4. **Authentication:** The process of conformation between sender and receiver's identity and origin.

To conclude cryptography deals with more than just protecting the data from alteration or theft but also from modification, authentication and origin. To achieve these goals there are four main cryptographic schemas: symmetric-key cryptography, asymmetric-key cryptography, hash functions and digital signatures.

2.1 Numbers Theory

The section represents the essential theory required to understand the relation between numbers theory and cryptography, focusing on prime numbers, their discovery, implementation and relevance to the encryption algorithms.

2.1.1 Modular Arithmetic

Modular arithmetic is "a system of arithmetic for integers, which considers the reminder" (Brilliant, 2018). Moreover, modular arithmetic specifies with numbers extending until a certain given fixed quantity is attained (the quantity is also called the modulus) to leave a reminder. In practice a number '$a \pmod N$' is the same as computing the reminder of 'a' divided by 'N'. Furthermore, considering two numbers 'a' and 'b' that are in the same equivalence class (those numbers are congruent) modulo 'N' if after division those numbers have an equal reminder. Therefore it can be stated that '$a \equiv b \pmod N$'.

2.1.2 Prime Numbers

A prime number is a whole number which is greater than '1' and can be divided with '1' or itself without a reminder. On the other hand, coprime numbers are such integers 'a' and 'b' where the only positive factor that divides them both is '1' (H. G. Hardy; M. E. Wright, 2008 and H. Kenneth

Rosen, 1992). It is important to state that for every prime number ' p ' there exists a prime number ' q ' which is greater than ' p ', therefore there is no such thing as the largest prime, however prime numbers get much harder to be found with the increase of their size. Moreover, large prime numbers are extremely desirable and searched for, mainly because of their functionalities and their practical use in computers and especially cryptography. Simple examples for prime numbers are the numbers '2, 3, 5, 7'.

Primal factors are such prime numbers that can divide into an integer without a reminder (Hans Riesel, 1994). Moreover the prime factors of the number '10' are the numbers '2' and '5', because their multiplication would result the number '10' and in addition to that because both '2' and '5' are prime numbers.

Prime factorization is the process of finding the original number from the multiplication of which prime numbers. Moreover, prime factorization is the same as integer factorization with one crucial difference, every factor that results the original number must be a prime number. An example of prime factorization can be given with the number '147', the factors of the number are '3; 7; 7', therefore the multiplication of those three numbers equals '147', however '3' and '7' are both prime numbers so it can be concluded that '3; 7; 7' is the primal factorization of '147'.

In mathematics factorization is thought to be computationally challenging process. Moreover, the computational difficulty of factorization increases straightforward with the increase of the size of the number therefore, factorization is considered to be an inefficient process in terms of running time when working with large numbers.

Primality tests are specific algorithms which are used to determine if a number is prime or not. Moreover, the need of primality testing arises when dealing with large prime number, because prime factorization is extremely inefficient in terms of determining whether a number is prime or not. However, it is important to state that there is one main difference between primality testing and prime factorization and that is that primality testing algorithms output only a statement saying if the number is prime or not, while prime factorization outputs the prime factors of that number. Therefore, primality testing is rather fast, moreover "its running time is polynomial in the size of the data input" (Manindra Agrawal; Neeraj Kayal; Nitin Saxena, 2004).

One of the simplest methods to implement primality testing is trial division. "Given an input number ' n ', check whether any prime integer ' m ' from '2' to ' $\sqrt{n}$ ' evenly divides ' n ' (the division leaves no remainder). If ' n ' is divisible by any ' m ' then ' n ' is composite, otherwise it is prime" (Manindra Agrawal; Neeraj Kayal; Nitin Saxena, 2004).

2.1.3 Euler's and Fermat's Theorems

Firstly, Euler's theorem or also known as Euler's totient theorem proves that if there are two coprime positive integers ' n ' and ' a ', then following modular arithmetic is correct " ' $a^{f(n)} \equiv 1 (\mathrm{mod}\, N)$ ', in which ' $f(n)$ ' is the Euler's totient function, which function counts the integers between 1 and 'n' that are coprime to ' n '" (Kenneth Ireland; Michael Rosen, 1990 and Carl Gauss; Arthur A. Clarke, 1986). The theorem itself is a generalization of Fermat's little theorem.

On the other hand, Fermat's little theorem states "if ' p ' is a prime number, then for any integer 'a', the number ' a^p ' – ' a ' would be an integer that is multiple of ' p '" (Adrian A. Albert, 2015). This statement can be expressed in terms of modular arithmetic as: "' $a^p \equiv a (\mathrm{mod}\, p)$ '" (Adrian A. Albert, 2015). This particular theorem is influential because it gives the essentials for the Fermat's primality test. Moreover, the security implementation of this theorem is a generalization called Euler-Fermat generalisation and it states that for "any modulus 'n' and any integer 'a' which is coprime to ' n ' we

can conclude that: ‘ $a^{f(n)} \equiv 1 (\mathrm{mod}\, N)$ ’” (Kenneth Ireland; Michael Rosen, 1990 and Carl Gauss; Arthur A. Clarke, 1986).

The generalization of both theorems is fundamental in terms of creating the RSA encryption system.

2.2 Symmetric-Key Cryptography

The section represents the principles and functionality of the main topics regarding symmetric-key encryption algorithms. Moreover, the report aims to give a short presentation of: stream ciphers, block ciphers, the data encryption standard and modes of operation.

2.2.1 Stream Ciphers

Stream Ciphers are designed to encrypt plaintext into cipher text by using the combination of a cryptographic key and algorithm, applied to each binary digit of the plaintext, one bit at a time. In addition to that, stream ciphers use keystream generators that produce continues stream of bit known as keystream. Encryption and decryption both result from a ‘ XOR ’ operation.

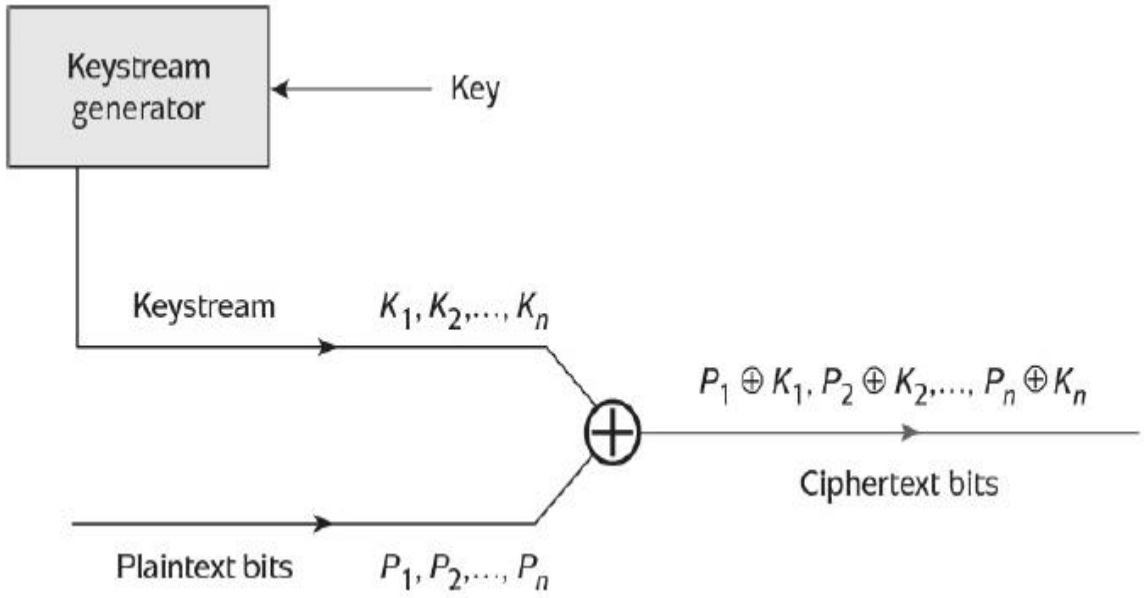


Figure 2.2.1 *operation of a simple stream cipher. Diagram source from (Keith M. Martin, 2017).*

Figure 2.2.1 demonstrates the operation of a simple stream cipher. The keystream generator using the secret key produces a pseudorandom sequence of bit (keystream). Encryption is achieved by a ‘ XOR ’ operation performed on each bit with each binary digit in the data stream. The result of this operation is the cipher text. Decryption happens as an exact reverse operation of encryption in stream ciphers.

The properties of stream ciphers include:

- **Speed:** Having encryption just being a simple ‘ XOR ’ operation allows stream ciphers to be very fast which gives them certain qualities and elements that can be exploited in certain systems and situations.

- **No Error Propagation:** Because of the fact that stream ciphers encrypt data one bit at a time, an error in one bit would not affect any of the other bit encrypted as it would with block ciphers.

- **Need for Synchronisation:** Encrypting one bit at a time requires synchronisation between sender and receiver given that one bit is lost in transmission would result fatal error when attempting to decrypt the cipher text, due to wrong keystream for each bit after the error occurred.

2.2.2 Block Ciphers

Block ciphers are constructed to encrypt blocks of plaintext into blocks of cipher text under the control of an encryption key and an algorithm. The length of the plaintext block is usually fixed and it is corresponding to the length of the produced cipher text block.

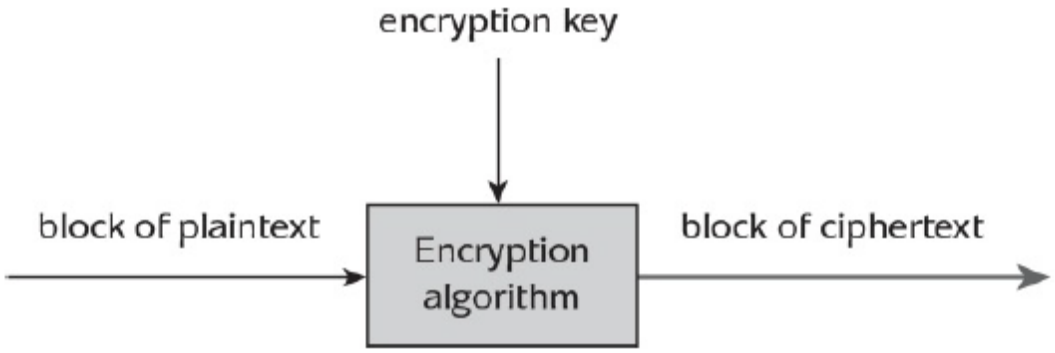


Figure 2.2.2 *operation of a simple block cipher. Diagram source from (Keith M. Martin, 2017).*

Figure 2.2.2 illustrates the operation of a simple block cipher. The fixed length plaintext block is inputted in the encryption algorithm which uses an encryption key and produces the block of cipher text. Encryption and decryption methods vary depending on the specific encryption algorithm used.

The properties of block ciphers include:

- **Functionality:** Block ciphers are widely spread and are the main method used for encryption. On the other hand, block ciphers are also used for other components such as MACs and hash functions.

- **Error Propagation:** This property is concerned specifically for a simple block cipher as the one illustrated above. In a situation where a one bit transmission error occurs it would only change one bit in the cipher text block. However, when decrypting, due to that issue half of the bit of the plaintext block would be decrypted wrong.

- **Need for Padding:** Due to block ciphers operating only with fixed block sizes, in practicality the length of most plaintexts would not be a multiple of the block size, leading to the use of padding.

2.2.3 Data Encryption Standard

The data encryption standard or also known as DES is a well-known block cipher, subject to a considerable amount of studies and cryptanalysis. Essentially DES is considered unbroken, however its block size is too small and exhaustive key search on DES has become feasible. The data encryption standard is a typical Feistel cipher. It has a block size of 64-bit, a key of length 64-bit and it is

operating with 16 rounds. Furthermore, to give a detailed representation of the key length of DES it actually has key length of 56-bit since the last 8-bit are used for check bit.

Feistel ciphers are related as a class of the modern iterative block ciphers with an internal function also known as a round function. The advantage of Feistel ciphers is in the analogy of encryption and decryption procedures, demanding only reversal of the key schedule.

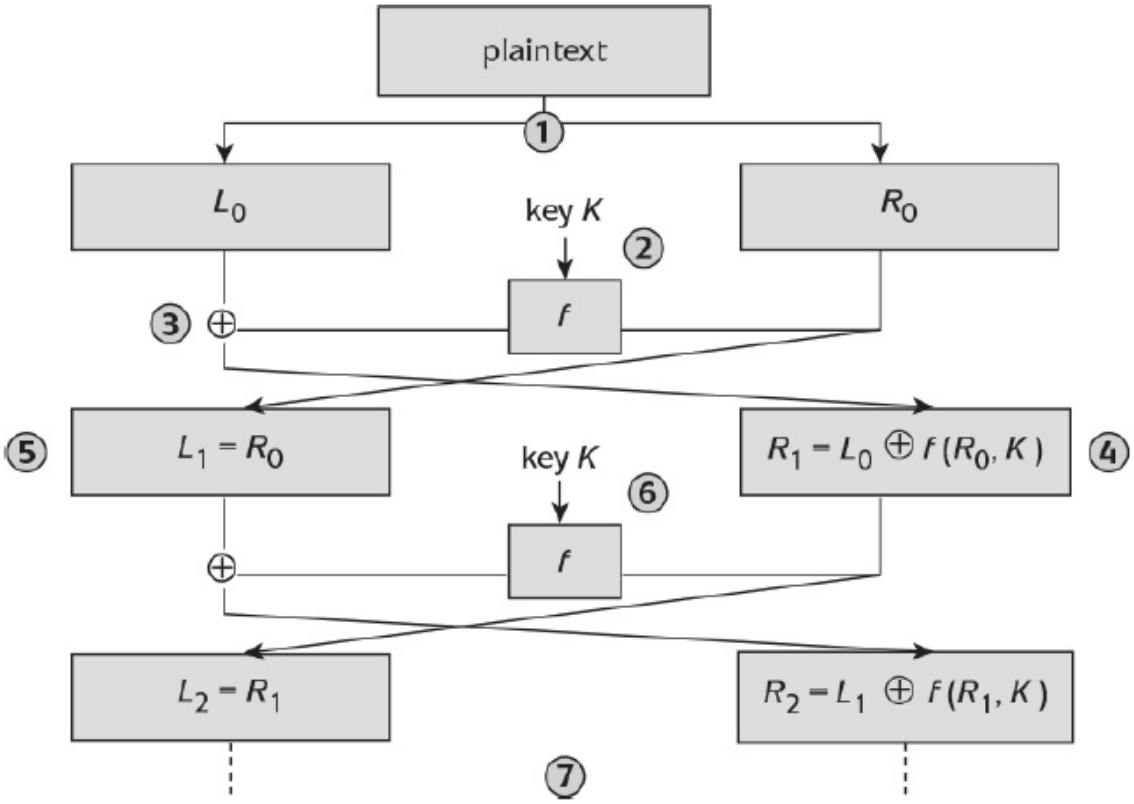


Figure 2.2.3 *operation of a Feistel cipher. Diagram source from (Keith M. Martin, 2017).*

Figure 2.2.3 shows the operation of a Feistel cipher. To give a further explanation of procedure behind encryption and decryption we consider that the figure above implements algorithm with block size of 64-bit.

Firstly, to begin with encryption the cipher operates in the following way (Keith M. Martin, 2017):

1. Splits the plaintext block into two equal parts: left 32-bit (‘ L_0 ’) and right 32-bit (‘ R_0 ’).

2. Implement a mathematical function also known as round function ‘ f ’ which takes as input the key ‘ K ’ and the right part of the block ‘ R_0 ’ and outputs the computation of ‘ $f(R_0, K)$ ’.
   * It is important to note that each round uses a specific round key generated from the key.

3. Performs a ‘ XOR ’ operation of the round function ‘ $f(R_0, K)$ ’ with the left part ‘ L_0 ’ to compute the new 32-bit sequence.

4. Let the new produces right part 32-bit ‘ R_1 ’ be ‘ Y ’.

5. Let the new left part 32-bit ' L_1 ' be the previous right part 32-bit ' R_0 '.

6. Repeat steps '**2.**' to '**5.**' for as many rounds the cipher is using.

7. Lastly, once the last round is completed the last left part 32-bit ' L_0 ' are combined with the last right part 32-bit- ' R_0 ' to form the 64-bit.

Secondly, describing decryption in Feistel ciphers is quite simple because decryption follows the same steps as encryption but obviously in a reverse manner.

Finally, due to the insufficient size of the block DES in the modern world with the striking increase of computational power the algorithm has become vulnerable to brute-force attacks. That is why DES doesn't find use in practicality today. However, Triple DES or (3DES) is an improvement of the old algorithm by applying DES algorithm three times to each data block and it is largely used in modern cryptography.

2.2.4 Advanced Encryption Standard

The advanced encryption standard or AES is the most used cipher algorithm in the world today and it was designed as a replacement of DES. The design of AES is different than DES from a point of view that it is not based on a certain model as DES on Feistel ciphers. Moreover, AES is designed on the basis of a design principle called substitution-permutation network. This type of design principle defines as replacement of inputs by specific called substitutions and changing bit also known as permutations, all of which procedures are performed as series of linked operations (Keith M. Martin, 2017).

It is believed that AES would deliver good security for the predictable future because of its effective process of design and flexibility regarding key length.

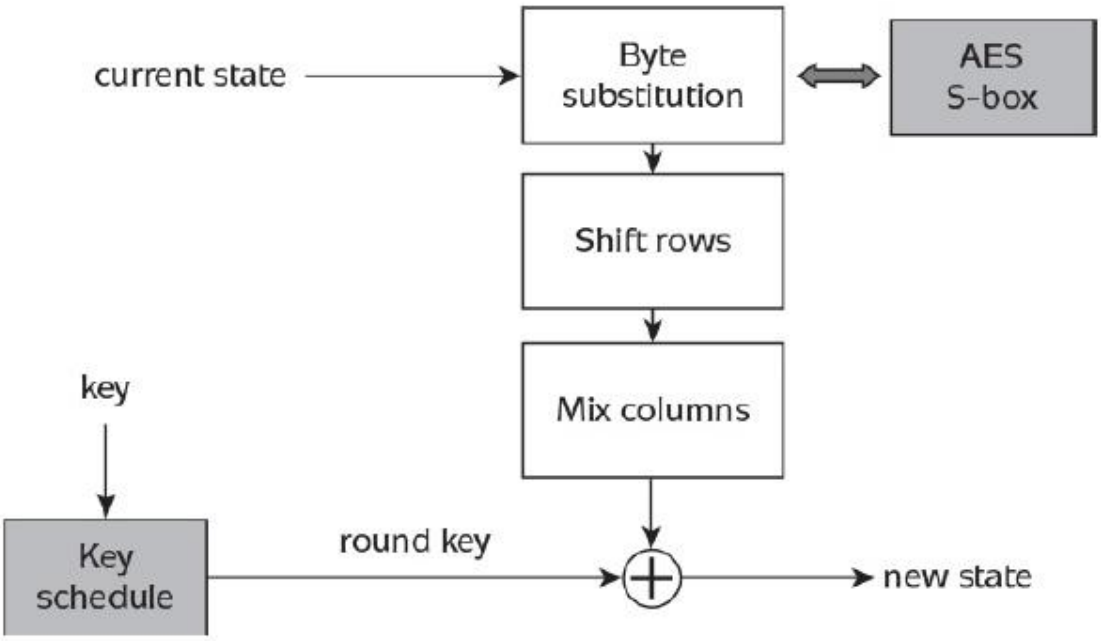


Figure 2.2.4 *operation of a single AES round. Diagram source from (Keith M. Martin, 2017).*

To begin with, it is important to state that AES works with bytes instead of bit, meaning a plaintext block of 128-bit would be represented as block of 16-bytes. Similar to DES, AES uses rounds and each round has a unique round key generated from the key, a round of AES works with 16-bytes of plaintext as input and outputs 16-bytes of cipher text. However, rounds in AES are not static as it is with DES, actually they are a variable that is dependent on the length of the key used. To be more specific, "AES uses 10 cycles of repetition for 128-bit keys, 12 cycles of repetition for 192-bit keys and 14 cycles of repetition for 256-bit keys" (Keith M. Martin, 2017).

Figure 2.2.4 demonstrates the operation of one encryption round in AES. Each round produces cipher text by executing the following steps:

1. **Byte substitution:** In this first step the input 16-bytes are replaced with new 16-bytes under the control of an 8-bit substitution box (S-box). Furthermore, the new 16-bytes are organized in a table that has four columns and four rows, used in the following steps of the process.

2. **Shift rows:** This operation is responsible for shifting bytes in each of the four rows by specific distance. The first row will be left unchanged, the second row will have shifted one byte to the left, the third row will have two bytes shifted to the left and the forth row will have three bytes shifted to the left. To conclude the whole operation restructures the bytes in the table.
   *It is important to state that any bytes that 'fall off' the table due to shifting to the left, will be inserted on the right.

3. **Mix columns:** The process of mixing columns is related to the use of a mathematical function which takes as input each column of four bytes and outputs four new distinct bytes which replace the old ones in the column.

4. **Add round key:** Finally, if this is the last round of AES the 16-bytes resulted from the previous step will be transformed into 128-bit and a 'XOR' operation will be performed with the 128-bit of the round key, resulting the cipher text. However, if this is not the last round of the cipher algorithm those 128-bit will be transformed back into 16-bytes and the process will start again from step one.

The whole process of encryption is just a complicated representation of 'XOR' operations with the combination of table lookups, making AES extremely fast in practice.

Decryption of AES follows as a reverse process of encryption following the same steps and procedures.

2.2.5 Modes of Operation

The need of modes of operation arises with the problem of having block ciphers being able to operate on a certain fixed-length block. Therefore, block ciphers are unable to apply cryptographic transformation on larger amounts of data than the specific block. However, a mode of operation defines "operational rules for a generic block cipher which each result in different properties being achieved when they are applied to plaintexts consulting of more than one block" (Keith M. Martin, 2017). Moreover, modes of operation allow a cipher's single-block operation to be used to handle cryptographic transformation of data that is larger than a single block.

ECB or Electronic Code Book mode is rather simple to implement, because it simply splits the plaintext into blocks having each block being encrypted separately. However, ECB mode is almost never considered the correct mode to use due to concerning problems regarding information leakage. Moreover, "a given block of plaintext is always encrypted in the same way to produce the same cipher text block" (Keith M. Martin, 2017). A conclusion can be made that identical plaintexts blocks are converted into identical cipher text blocks, in a sense the mode doesn't provide message confidentiality and it is considered dangerous from security point of view.

CBC or Cipher Block Chaining mode it operates as having "each cipher text block, as well as being sent to the receiver, serves as an input into the encryption process of the subsequent plaintext block" (Keith M. Martin, 2017). Therefore, this accomplishes all cipher text blocks computationally 'chained' together. It aims to obstruct information leakage of ECB mode. The idea behind CBC mode is that a 'XOR' operation is performed on each block of plaintext with the previous cipher text block before proceeding with encryption. In addition to that, regarding the first plaintext block an initialization vector is used to perform the 'XOR' operation. The 'XOR' technique creates randomness and eliminates some of the major issues in ECB mode.

CFB or Cipher Feedback mode is closely related to CBC mode, it transforms a block cipher into a "self-synchronizing stream cipher" (Keith M. Martin, 2017). It is important to note that a "self-synchronizing stream cipher" is different from a normal stream cipher and has several advantages for example no synchronisation is required. The other properties of the mode also include positional dependency, limited error propagation and no padding required.

CTR or Counter mode also turns a block cipher into a stream cipher. It is a "counter- based version of CFB mode without the feedback" (Keith M. Martin, 2017). The mode uses the block cipher to generate a keystream, which keystream is encrypted on the basis of a stream cipher using a 'XOR' operation. However, the mode doesn't include the decryption of the block cipher. The main disadvantage of the mode is the requirement of synchronous counter. If synchronisation is lost this would lead to plaintexts incorrectly recovered from their relevant cipher texts, On the other hand, CTR includes many positive features such as no requirement for padding, error propagation and parallelisation.

2.3 Asymmetric-Key Cryptography

The need of asymmetric-key cryptography or also known as public-key cryptography arises due to the issues regarding the practicality and restrictive implications of the symmetric-key cryptography. Moreover, symmetric cryptography is based on the 'trust' between the sender and receiver, because of them sharing the same cryptographic key, meaning that the sender and receiver are strictly equal in terms of actions they can perform. In addition to that, "with symmetric cryptography the sender and receiver need to have access to a secure key establishment mechanism in order to be able to exchange a symmetric key in advance of the use of a symmetric cryptosystem" (Niels Fergusen; Bruce Schneier, 2003). Considering those two statements we can conclude that symmetric-key cryptography can be impractical in some scenarios and a different approach would be required.

Public-key cryptography is a type of cryptographic system that would exploit the use of two different pair of keys:

1. **Public Key:** Known and visible to anyone and used for encryption of plaintext.

2. **Private Key:** Known and visible only to the specific entity, which provides the public key and it is used for decryption of cipher text.

In asymmetric cryptography anyone can use the public key of the receiver with a purpose encryption of a message. However, only the receiver can decrypt the message under the control of the private key. To achieve security public key cryptography "relies on the computational impracticality for a properly generated private key to be determined from its corresponding public key" (Niels Fergusen; Bruce Schneier, 2003) and therefore preserve the confidentiality of the private key. Furthermore, the root of the security of asymmetric encryption relies on "mathematical problems that currently admit no efficient solution – practicality those inherit in certain integer factorization, discrete algorithm and elliptic curve relationships" (Niels Fergusen; Bruce Schneier, 2003).

The main purpose of public-key cryptography is the transfer of a session key that would provide symmetric encryption or alternatively, it can be used for small amounts of data. The reason

asymmetric cryptography is not used for larger data transfer is due to the high computational complexity of this type of encryption.

2.3.1 Diffie-Hellman

Diffie-Hellman key exchange is one of the first methodologies that implements public key exchange in cryptography. Its core is not based on encryption and decryption procedures but on mathematical functions that allow two parties to generate and use a shared secret key for achieving confidentiality of the information. Moreover, "it establishes a shared secret between two parties that can be used for secret communication for exchanging data over network" (Technet Microsoft, 2018).

The main idea behind Diffie-Hellman key exchange is that having two parties involved, each one generates a public value ' g ' and a large prime number ' p '. In addition to that, one party generates secret value ' x ' and respectively the second party generates a secret value ' y '. Both parties, using their generated secret values calculate the following public value, respectively for the first and second party: ' $g^x \pmod p$ ' and ' $g^y \pmod p$ ' and they exchange those public values. Following that, each party can now calculate the shared secret key using the other's party public value. The shared secret key itself is the same for both parties. The method works because there is no way for a third party to acquire the shared secret key, simply because there is no possible way of knowing the secret values ' x ' and ' y ' of each party (Technet Microsoft, 2018).

Even though the main principles of Diffie-Hellman key exchange were developed around 40 years ago it influences public key cryptography today. Furthermore, this basis defining key exchange is "widely used with varying technical details by Internet security technologies, such as IPsec and TLS in terms of providing secret key exchange for confidential online communications" (Technet Microsoft, 2018).

2.3.2 RSA

The RSA algorithm was one of the first proposed and still even today, the most extensively used public key cryptographic algorithm. The basis of RSA key exchange are dependent on having the "secret keys exchanged securely online by encrypting the secret key with the intended recipient's public key" (Technet Microsoft, 2018). Furthermore, only the expected recipient would be able to decrypt the secret key because that exact recipient controls the private key used for decryption. Therefore, preventing any third parties who could intercept the shared secret key to be able to use it effectively.

RSA being an example of asymmetric encryption is a rather slow algorithm and its main purpose is not related to encrypting user data. However, RSA is essentially used when passing encrypted shared keys for the use of symmetric cryptography which results a much higher speed in terms of encryption and decryption.

Operation of RSA algorithm is built around four fundamental phases:

- **Key generation:** The generation of the public and the private keys used for the public encryption scheme. Moreover, this phase is considered the most crucial part of RSA.

- **Key distribution:** A communication under RSA is established by using a public and private key between two entities ' A ' and ' B '. Furthermore, ' A ' needs to know ' B ' public key to encrypt the message it is sending. On the other hand, ' B ' uses its private key to decrypt the message. Finally to achieve this procedure, ' B ' is sending its public key ' (n, e) ' to ' A ', allowing ' A ' to send its encrypted messages, having the channel of communication being not essentially secured.

- **Encryption:** Once the first party ' A ' gains control of the public key ' e ' of ' B ' it can now send cipher text ' c ' to ' B '- ' $c \equiv m^e \pmod{n}$ '.

- **Decryption:** After ' B ' has received the cipher text from ' A ' it now can decrypt it by using its private key exponent ' d '- ' $c^d \equiv (m^e)^d \equiv m \pmod{n}$ '.

A much further and detailed explanation of RSA algorithm is given in **'Chapter: 3 The RSA Cryptosystem'**.

2.4 Data Integrity

Considered a fundamental component of information security, data integrity is "the maintenance of, and the assurance of the accuracy and consistency of, data over its entire life-cycle, and is a critical aspect to the design of any system that manipulates, retrieves and stores data" (Josiang, 2014 and K. Patterson, 2018). Defining the scope of data integrity mechanisms, regarding potential alteration of the data can be quite complicated. Therefore, for simplicity four different levels of data integrity are outlined:

- Accidental Errors
- Simple Manipulations
- Active Attacks
- Repudiation Attacks

Each of the four levels is representing different possible attacks, attacks difficulty increases with levels.

2.4.1 Hash Functions

Hash functions are considered one of the most practical cryptographic primitives. They play an irreplaceable role in many cryptographic applications, however hash functions are not that useful on their own. The main purpose of hash functions includes: strong one-way functions, data integrity, components of other cryptographic primitives and as a source of pseudo randomness. Furthermore, hash functions is a function that "is used to map data of arbitrary size to data of fixed size" (Josiang, 2014 and K. Patterson, 2018). The hash values are the values that a hash function outputs. A very wide spread example of the implementation of hash functions is hash tables, their purpose is to dramatically accelerate data lookup.

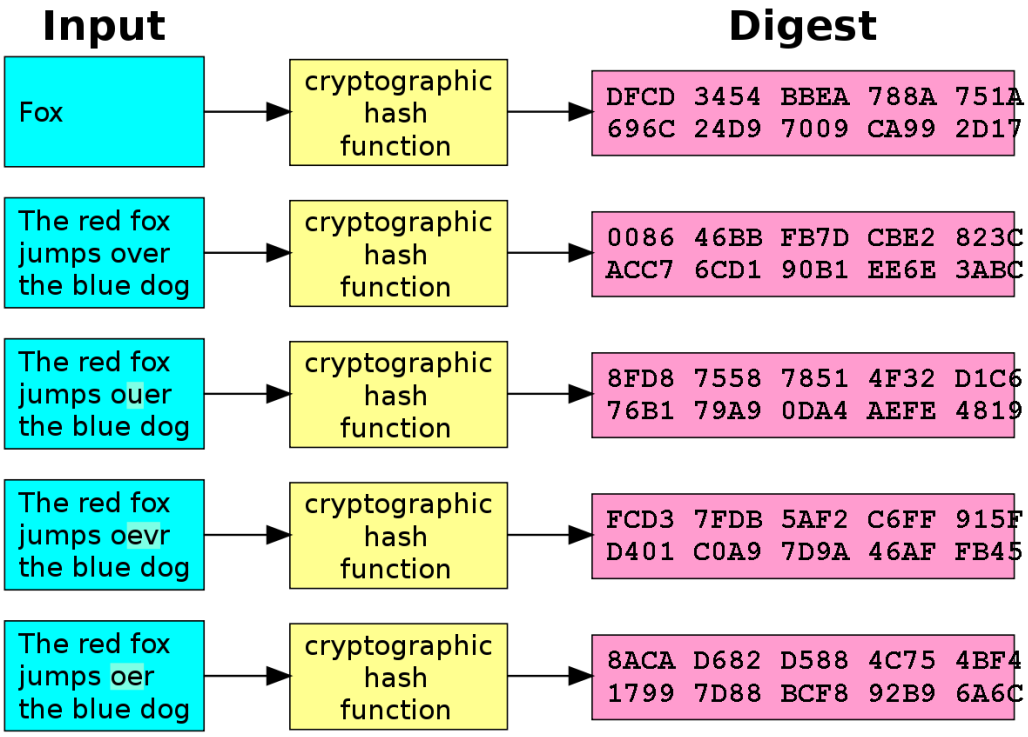


Figure 2.4.1 *basic model of a hash function. Diagram source from (Wikipedia, 2018).*

Figure 2.4.1 describes the basic model of a hash function. The message is being input into the cryptographic hash function which then outputs a hash value with fixed length.

The security properties of a hash function are extremely strict, considering that hash functions are used in almost every cryptographic application. A secure hash functions satisfies "two important practical properties and three security properties" (Keith M. Martin, 2017). Those properties take into consideration that hash functions do not use a key and that hash functions are publicly computable.

1. **Practical Property 1: Compresses arbitrary long inputs into a fixed length output**
   -no matter of amount of data input, the hash function creates output with same fixed length
   -output data is always less than input data
   -hash functions are often referred to as compression functions

2. **Practical Property 2: Easy to compute**
   -it is essential that hash functions are computed with efficiency and speed
   -hash functions wouldn't be that widely used if they weren't easy to compute

1. **Security Property 1: Preimage resistance**
   -a hash function should be very computationally hard to reverse
   -"in terms of complexity theory, security property 1 demands that reversing a hash functions involves a process which runs in exponential time" (Keith M. Martin, 2017)

2. **Security Property 2: Second preimage resistance**
   -"given an input and its hash, it should be hard to find a different input with the same hash" (Keith M. Martin, 2017)

3. **Security Property 3: Collision resistance**
   -two different inputs applied to a hash function shouldn't result the same hash computation
   -fully avoiding hash collisions is impossible, the goal is that those collisions are hard to find

2.4.2 Message Authentication Codes

Message authentication codes or commonly known as MACs are "symmetric cryptographic primitives designed to provide data origin authentication" (Keith M. Martin, 2017). Moreover, MACs confirm the authentication property of a message- if it has been altered or not. There are three algorithms involved in MACs: a key generation algorithm, a signing algorithm which returns a tag using the specific key and the message and a verifying algorithm which uses the tag and the message to verify the authenticity of the message. From security point of view it must be computationally infeasible to be able to produce a valid tag for a given message without the possession of the secret key.

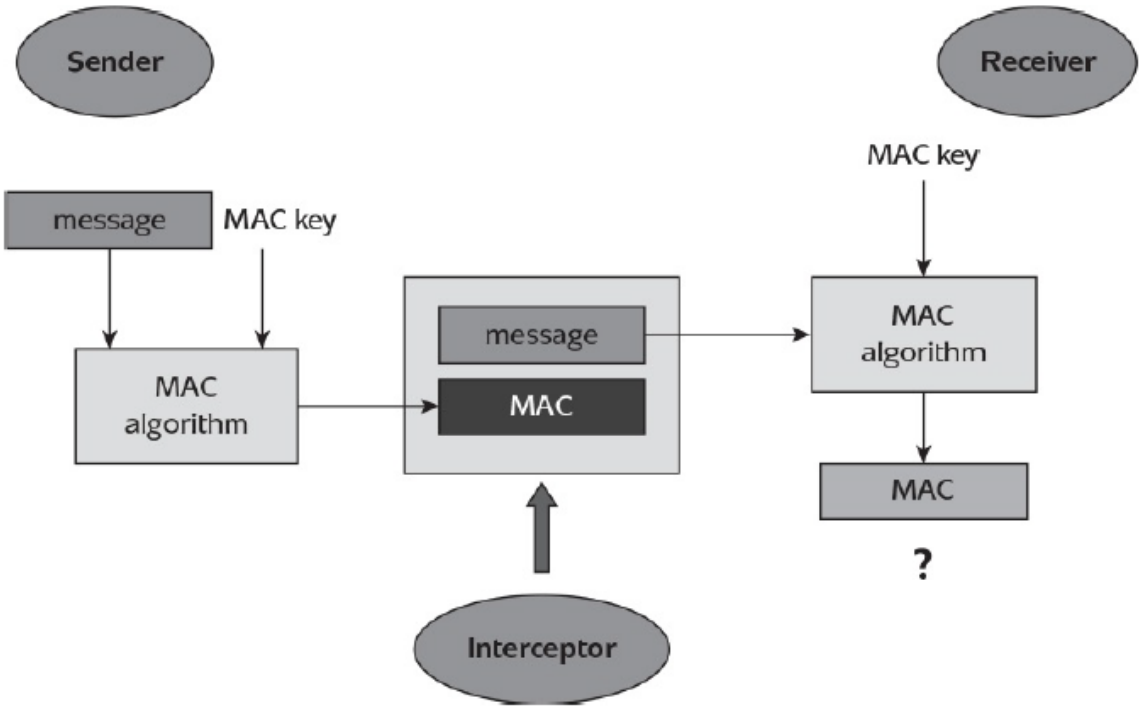


Figure 2.4.2 *basic model of a MAC. Diagram source from (Keith M. Martin, 2017).*

Figure 2.4.2 describes the basic model of a MAC. The sender and the receiver of the message both participate using a shared key K. The sender then inputs the message and the secret key into the MAC and afterwards transmits the message to the receiver with the computed MAC. On the other hand, upon receiving the message the receiver inputs the message into the MAC and checks if the MAC that he computed matches the one that was sent with the message from the sender. If the match is successful then data origin authentication of the message has been maintained.

The properties of a message authentication code include: compression, easily computational, preimage resistance, second preimage resistance and collision resistance.

2.4.3 Digital Signature Schemes

Digital Signature Schemes are mathematical schemes that aim to prove the credibility of messages. If a digital signature of a message is verified as valid it provides the receiver with the information that the message is indeed sent by the sender and the message wasn't modified during transit. There are three algorithms involved in digital signatures: a key generation algorithm, a singing algorithm which outputs a signature using the specific key and the message and a verifying algorithm that uses the signature and the message to verify the correctness of the message.

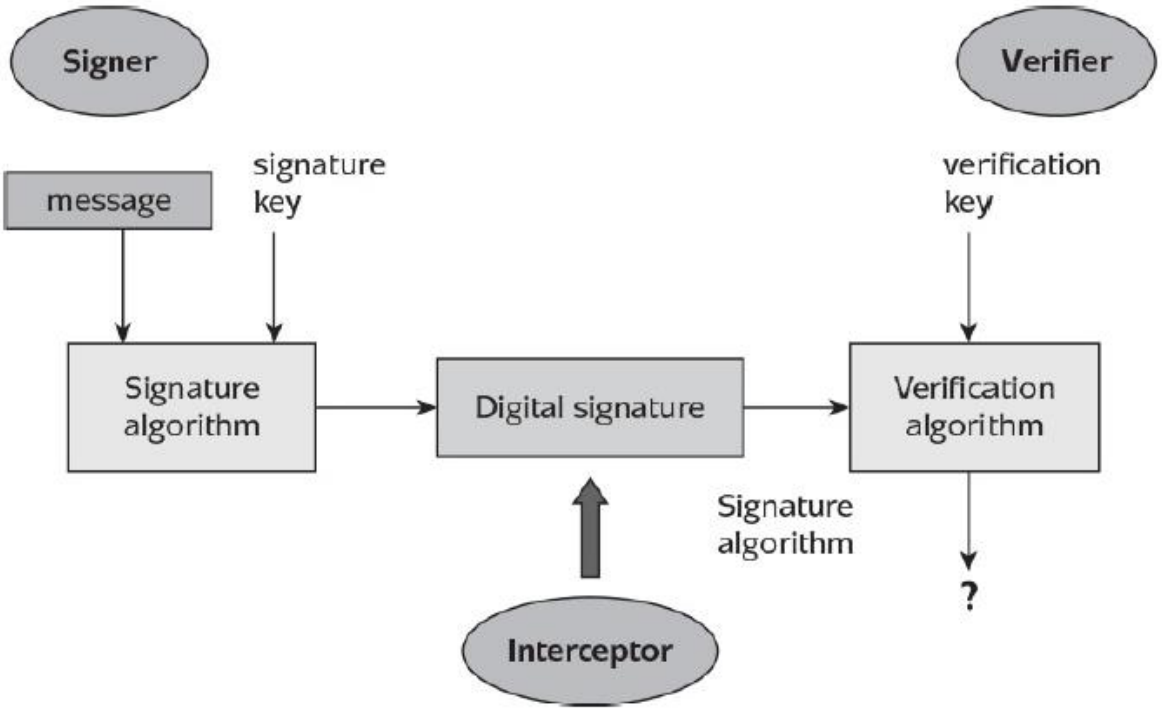


Figure 2.4.3 *basic model of a digital signature scheme. Diagram source from (Keith M. Martin, 2017).*

Figure 2.4.3 describes the basic model of a digital signature scheme. The message being signed and the key are being input into the signature algorithm and it outputs the digital signature. Furthermore, the digital signature is sent to the receiver or in this model the verifier. The verifier then uses the verification algorithm by inputting the digital signature and the key. Afterwards, the verifying algorithm outputs the message which should be the same as the one from the signer that was digitally signed. The verifier can now verify whether the message is valid or not.

The required properties of a signature scheme include: data origin authentication of the signer, non-repudiation, efficient signing of data and an active verifying of data and rather complicated to forge.

Chapter 3: The RSA Cryptosystem

The chapter aims to give an overview of how is the RSA cryptosystem used in practice

3.1 Key Generation

In public-key encryption schemes, key generation is considered the most important part simply because the core of the system relies on proper designed relationship between two keys. The process of generating a key pair in RSA is the following:

- **Generate the modulus ‘ n ’:** The procedure behind generating the modulus is rather simple, it requires two very large prime numbers to be found- ‘ p ’ and ‘ q ’. Therefore, to compute ‘ n ’ we multiply the large prime numbers ‘ p ’ and ‘ q ’, ‘ $n = p*q$ ’. To conclude, after multiplying the result ‘ n ’ is an even larger number, which is a key property for RSA security.

- **Generate the special number ‘ e ’:** The choice of the special number is not random and it has exact properties. “The precise mathematical property that ‘ e ’ must have is there can’t be any numbers dividing neatly into ‘ e ’ and into ‘ $(p-1)*(q-1)$ ’ except for 1” (Keith M. Martin, 2017).

- **Generating the public key ‘ pK ’:** The public key is generated from the first and the second step, moreover the public key is computed from ‘ (n,e) ’. The public key is available to anyone who requires it and therefore the values that generate it are vulnerable, therefore having ‘ n ’ being computed from ‘ p ’ and ‘ q ’ it is essential for the security of RSA that the prime numbers remain secret. The secrecy of the prime numbers is achieved by “the fact that multiplication of two primes is a one-way function, preventing the numbers being discovered by unauthorized entities” (Keith M. Martin, 2017).

- **Generating the private key ‘ d ’:** The private key is generated from the generated prime numbers and the chosen special number- ‘ (p,q,e) ’. It is important to state that there is a relation between the private and public keys- “given an ‘ n ’ and an ‘ e ’, there can be only one possible value ‘ d ’” (Keith M. Martin, 2017). Therefore, the security of the private key relies on the confidentiality of the special number ‘ e ’ and the generated prime numbers. To give a mathematical representation of the security when generating the private key the following rules are applied “‘d’ is a unique number that is less than ‘ $(p-1)*(q-1)$ ’, which when multiplied with ‘ e ’ equals ‘1 modulo ‘ $(p-1)*(q-1)$ ’” (Keith M. Martin, 2017).

3.2 Encryption and Decryption

Encryption in RSA is rather straightforward compared to the key pair generation process. To encrypt a plaintext ‘ P ’, the public encryption scheme uses the generated public key ‘ n,e ’. However, it is essential for RSA that the plaintext ‘ P ’ is transformed into “a series of numbers less than ‘ n ’ ” (Keith M. Martin, 2017). Moreover, meaning that RSA doesn’t work with strings of bit but with numbers modulo ‘ n ’. After that process of transforming the plaintext is complete, encryption can proceed by “multiplying the transformed plaintext ‘ P ’ by itself, ‘ e ’ amount of times and then reduced modulo ‘ n ’” (Keith M. Martin, 2017). The representation of this explanation is supported

by the formula- ‘ $C = P^e \pmod n$ ’, where ‘ C ’ is the cipher text, ‘ P ’ is the transformed plaintext and ‘ e ’ is the special number included in the public key.

The process of decrypting the cipher text ‘ C ’ is also considered simple. To produce back the plaintext ‘ P ’ the holder of the private key has to raise the cipher text ‘ C ’ to the power of the private key ‘ d ’ reduced ‘modulo n’. Therefore, to support this claim the formula for decryption in RSA is the following- ‘ $P = C^d \pmod n$ ’.

3.3 Security

Security within RSA public key encryption algorithm essentially is the integer factorization problem. Moreover, integer factorization is based on the computational difficulty of finding two large numbers, having only their multiplication result. In RSA this is expressed as: “Given integer ‘ n ’ as the product of two distinct prime numbers ‘ p ’ and ‘ q ’, find ‘p’ and ‘ q ’” (Herongyang, 2018). If the given integer ‘ n ’ is not large enough, that would determine the factorization problem and the prime numbers ‘ p ’ and ‘ q ’ would be found, which moreover means that the algorithm is broken and the private key is not private anymore.

Considering what was mentioned in the previous paragraph it can be concluded that RSA requires large bit keys to provide security. RSA is commercially used today with 1024-bit keys, however that is not considered fully secure, especially with the fast growth and improvement of technology. Moreover, RSA is used with 2048-bit keys for a high-security applications or extremely sensitive data that requires to maintain its confidence levels and it is considered that would provide security for a few years until it is possible to break the key.

So if larger keys provide significant security to RSA, why not use as large as possible bit keys? Even thought, security is the main priority of the algorithm it also has to be relatively fast in order to be used in practice. Increasing the length of the key for example from 2048-bit to 4096-bit would notably reduce the speed of the algorithm, to be precise “with doubling of the RSA key length, decryption happens 6 to 7 times slower. This statement is supported by the figure below, “the timings are made on a 2GHz Pentium processor” (Neil Coffey, 2012).

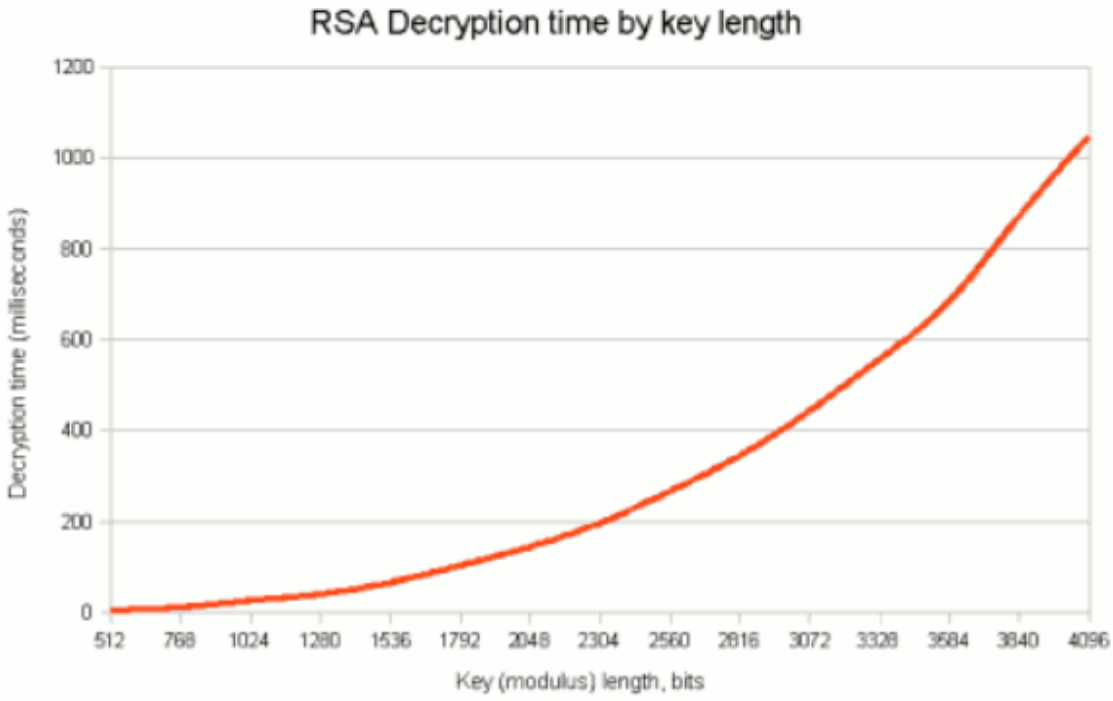


Figure 3.3 *RSA decryption time by key length. Diagram source from (Neil Coffey, 2012).*

The latest recorded largest number that has been successfully broken down with a brute force attack is RSA-768, which happened on December 12, 2009. Moreover, it took over two years for the attack to be complete. The computation power involved was a "collection of parallel computers amounted approximately to the equivalent of almost 200 years of computing on a single-core 2.2 GHZ AMD Opteron-based computer" (Neil Coffey, 2012).

Finally, to be able to give a conclusion about the security of RSA an approach considering the near future is essential. Shamir and Tromer made an estimation based on their hypothetical TWIRL device that "a few dozen million US dollars" (Neil Coffey, 2012) would be required to break a 1024-bit RSA encryption within 1 year, however it is important to note that this prediction was made 15 years ago. The research of Shamir and Tromer is equally important today because it provides the potential lifetime of data depending on the key size used. Moreover based on the analysis it can be concluded that a data would remain confidential until: "2010 for 1024-bit keys; 2030 for 2048-bit keys; 2031 and onwards for 3072-bit keys" (Neil Coffey, 2012).

3.4 Basic Attacks

Ever since the initial release of RSA, it has been an object of monitoring and analysis with one purpose, to find any possible vulnerabilities to the algorithm. The possible outlined attacks can be categorized in five sections, each section targeting different component of the algorithm.

3.4.1 Factoring Large Integers

This is the first and most simple attack that can be performed on the RSA algorithm. It concerns the factorization problem of the algorithm. Moreover, the attack is a brute-force attack that aims to achieve factorizing of the modulus. Currently the fastest algorithm is the "'General Number Field Sieve' with running time of '$\left((c+o(10)n^{1/3}\log^{2/3} n\right)$'" (Abdulaziz; Alrasheed; Fatima, 2018).

3.4.2 Elementary Attacks

These are attacks that are exploiting the misuse of the algorithm. One of the attacks is 'common modulus' the attack exploits that the modulus cannot be used by two entities. Moreover, if the modulus is the same for all users of a particular system every user can compute ' n ' using his own exponents, which compromises the security of the system. The second attack is 'binding' it is concerns one user being able to "obtain a valid signature on a message of his choice, by asking the other used to sign a random 'blinded message'" (Abdulaziz; Alrasheed; Fatima, 2018).

3.4.3 Low Private Exponent

Low private exponent attacks exploit the misuse of the private exponent in the algorithm during decryption. Furthermore, M. Wiener theorem proves that such actions can lead to an absolute break of the RSA cryptosystem. Moreover, the theorem states that if there is a modulus "' $n = p*q$ ' with ' $q < p < 2q$ '. Let ' $d < 1/3n1/4$ '. Given '(n, e)' with $ed = \mathrm{mod}\, f(n)$, an attacker can efficiently recover d." (Abdulaziz; Alrasheed; Fatima, 2018). In order to be able to resist such attack, if ' n = 1024-bit' then the exponent must be a minimum of ' d = 256-bit'.

3.4.4 Low Public Exponent

Low public exponent is often used with signature-verification time or encryption when attempting to reduce time during those operations. The lowest value that could be used is ' $e = 3$ ', however to be able to maintain the security of the system and avoid such attacks the recommended value is ' $e = 2^{16} + 1$ '.

Copper-smith theorem is the basis for most of the severely dangerous attacks concerning the exponent. The theorem states that if there is an "integer ' n ' and a monic polynomial of degree ' $f \in Z[x]*d$ '. Set ' $x = n1/d - \in$ ' for some ' $\in \geq 0$ '. Then, given '(n, f)' an attacker can efficiently find all integers '$| x0 | < x$' satisfying ' $f(x0) = 0(\mathrm{mod}\, n)$ '. In addition to that, the theorem also uses an algorithm that "efficiently finds all roots of modulo ' n ' that are less than ' $x = n1/d$ '. To conclude, the idea of the theorem is to find small roots of polynomials modulo a composite 'N'" (Abdulaziz; Alrasheed; Fatima, 2018).

3.4.5 Implementation Attacks

Those are attacks that are attempting to exploit the implementation of the RSA algorithm. Moreover, the attacks can be 'timing attacks'- relying on the exact time of decryption as an exploit to find the private decryption exponent 'd' and 'random faults attacks'- the attacker exploits faults that can occur that might occur when implementing the algorithm.

3.5 RSA in Practice

RSA in theory is completely different from how the algorithm is implemented in practice. Moreover, it is considered dangerous to deploy the textbook version of the algorithm for security purposes. The main difference between practical and textbook RSA would be that in the practical version randomisation is included within the encryption process. Moreover, in textbook RSA the encryption is deterministic. Meaning that "each time the same plaintext is encrypted using the same public key, the resulting cipher text will be the same" (Keith M. Martin, 2017). Therefore, the algorithm is extremely weak to an attack known as informed exhaustive plaintext search. However, this attack is considered infeasible in practice because of the use of probabilistic encryption. Furthermore, probabilistic encryption is a method which includes "a random number generated for a single use in a specific encryption" (Keith M. Martin, 2017) with the encryption process. Therefore, this achieves

different cipher texts for identical plaintexts encrypted with the same public key. This method completely stops the informed exhaustive plaintext search attack because the attacker has to attempt "all possible values of the random number for every guess of the plaintext" (Keith M. Martin, 2017).

3.6 Future of RSA

The future of RSA is determined by that if the algorithm would continue to be as secure and fast. However, this might be challenging, taking into account that technology is developing faster than ever, and dramatically increasing the possible computational power that can be applied. Moreover, hypothetically if conventional computers in the future and their computational power are not considered a threat. A real threat that RSA encryption might be facing is the development of quantum computers. Quantum computers provide enormous computational power compared to conventional computers. Therefore, quantum computers endanger the prime factorization property of RSA, simply by brute forcing even large bit keys. However, a suggestion has already been made to avoid that issue by using "Shor's algorithm to render the RSA cryptosystem unsalvageable" (Quanta Magazine, 2017). Moreover Nadia Heninger an assistant professor of information science at the University of Pennsylvania states: "RSA is not entirely dead even if quantum computers are practical" (Quanta Magazine, 2017).

A certain statement can be made about the feature of RSA and it is that the algorithm is heavily dependent on the development of technology. However, it is possible to adapt the algorithm and maintain its purpose- to be secure and fast.

Chapter 4: Proof-of-Concept Development

The chapter defines the construction progress towards the final program. Moreover, it describes the set of programs developed during the first term in order to complete the initial prototype.

4.1 RSA Encryption Program

4.1.1 Description

The first proof-of-concept program realizes a simple implementation of the RSA cryptosystem. The program has two core functionalities, the first one is the generation of two random prime numbers, at this initial stage those prime numbers are generated and primality checked using the 'BigInteger' Java class. Further, the second functionality of the program uses the previously generated prime numbers to set up the components of the RSA cryptosystem and therefore using those components, the program computes the public and private keys. Following that, the public and private keys are used by the program to perform encryption and decryption procedures over a data string, input by the user.

The RSA Encryption Program consists the following classes:

- A **'PublicKey.java'** class used to define the object public key.
- A **'PrivateKey.java'** class used to define the object private key.
- A **'KeyPairGenerator.java'** class performing generation of the components of the public and private keys, as well as saving those keys into files, which is required for the further functionality of the system.
- An **'Utilities.java'** class which contains public static methods used in the whole program. Moreover, in this current case the class involves the encryption and decryption algorithms for the RSA cryptosystem.

4.1.2 Class Diagram

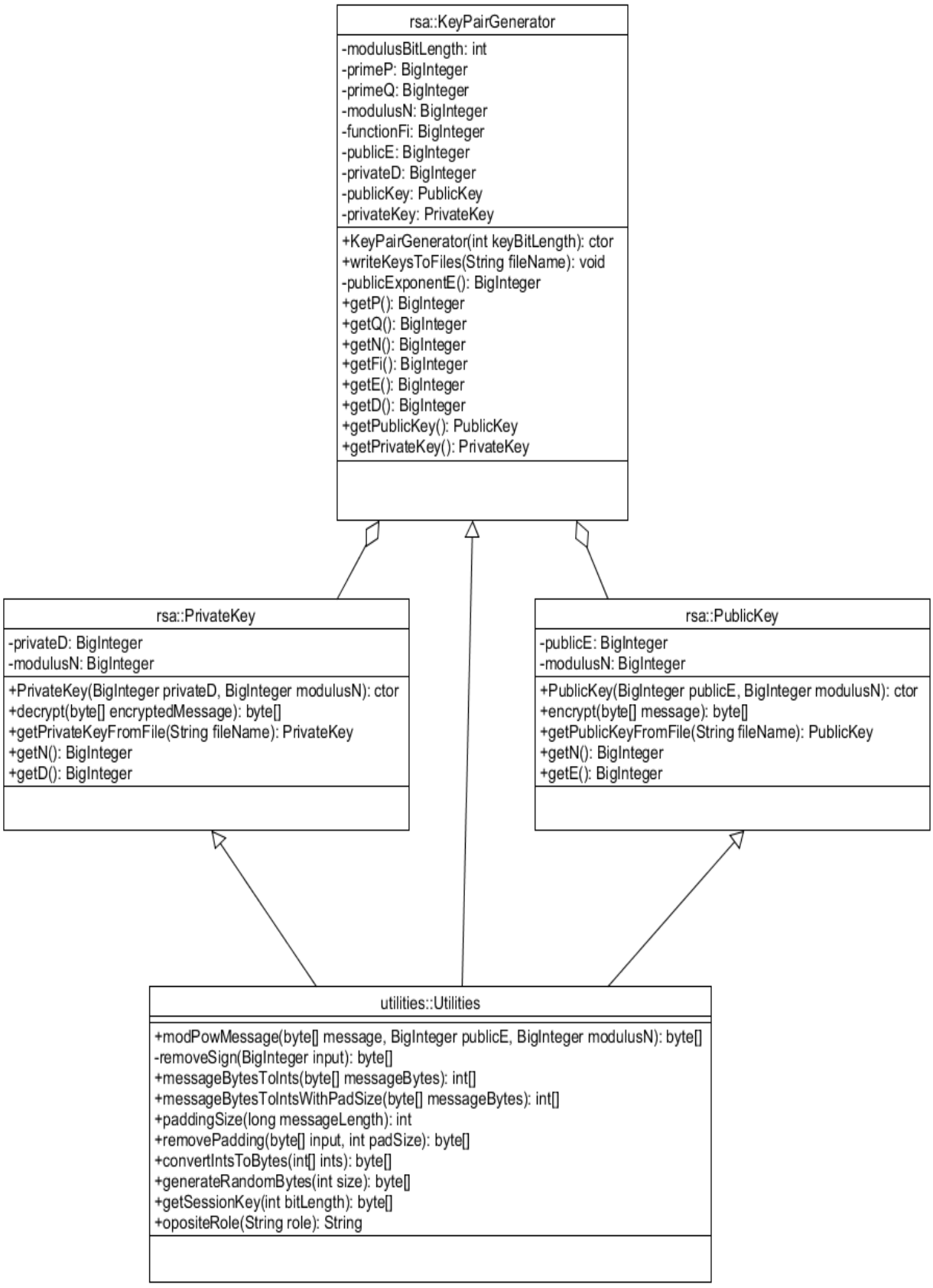


Figure 4.1.2 *represents the classes used in the 'RSA Encryption Program'.*

4.1.3 Development Process

To begin with, the development process of the program started with the generation of two large prime numbers and therefore checking if the generated prime numbers are actually prime. This was achieved by using the built in 'BigInteger' Java class which works on the basis of generating random numbers and simultaneously checking if those generated numbers are prime or not. Moreover, the probability that a 'BigInteger' returned by the method 'probablePrime' does not exceed 2^{-100}. The next step in the process was to calculate the components of the public and private keys for the RSA encryption. Those components are the modulus 'n'; public exponent 'e' and private exponent 'd'. Class 'KeyPairGenerator.java' was developed to generate the components of the public and private keys and initialize the objects of type 'PublicKey.java' and 'PrivateKey.java'. In addition to that, those two classes also contain methods for encryption and decryption. Finally, an additional class was created to simplify the structure of the program- this is the 'Utilities.java' class and that particular class is responsible for the delivery of methods and algorithms that are also required when computing the mentioned above components, those methods are of type public static and require no objects to be used.

The algorithm used in the first proof-of-concept program is the simple implementation of RSA encryption and decryption procedure explained in **'Chapter: 3 The RSA Cryptosystem'**.

4.1.4 Justification

The decision making behind the choice of this being the first proof-of-concept program is justified by the requirements gathering that led to the fact that the functionality implemented by the program is an absolute basis for further development. In addition to that, realizing asymmetric encryption in a program was essential to gain valuable knowledge in the work process of algorithms and cryptosystems. Finally, being able to achieve operating encryption and decryption of a data string is fundamental for the initial development of the end system.

4.1.5 Work Log

This section highlights the important milestones during the development of the program, the following information provided has been supported by the project diary:

September 23, 2017

"Completed the methods of 'PublicKey.java' and 'PrivateKey.java' classes."

September 24, 2017

"Managed to write the methods regarding the generation of the public and private keys, which methods are required by the 'KeyPairGenerator.java' class."

September 26, 2017

"Forwarding the graduate development to complete the first proof-of-concept program, the last class 'KeyPairGenerator.java' was completed."

September 30, 2017

"Adequate Junit tests and a main method were created to test the encryption and decryption of the program."

4.2 Feistel Block Cipher Encryption Program

4.2.1 Description

The second proof-of-concept program represents the implementation of a simple block cipher based on a Feistel network. The size of the block used in the application is 64-bit and the used cipher key is 256-bit, producing 8 32-bit round keys. Therefore, each block is encrypted in 8 rounds. Further, this simple implementation of a block cipher is used in ECB mode. Moreover, the algorithm applied allows encryption and decryption of blocks and their relevant transformation into strings. The first proof-of-concept program the 'RSA Encryption Program' serves as a distributor of the session keys that the 'Feistel Block Cipher Encryption Program' uses to perform efficient encryption and decryption over messages.

The proof-of-concept program consists the following classes:

- A **'BlockCipher.java'** class responsible for the encryption and decryption of blocks based on a Feistel network system.
- An **'EncryptDecryptMessage.java'** class works as a facade of 'BlockCipher.java' and implements methods for encryption and decryption of strings.
- An **'Utilities.java'** class which contains public static methods used in the whole program. Moreover, implements fundamental functionalities required by the 'BlockCipher.java' class.

4.2.2 Class Diagram

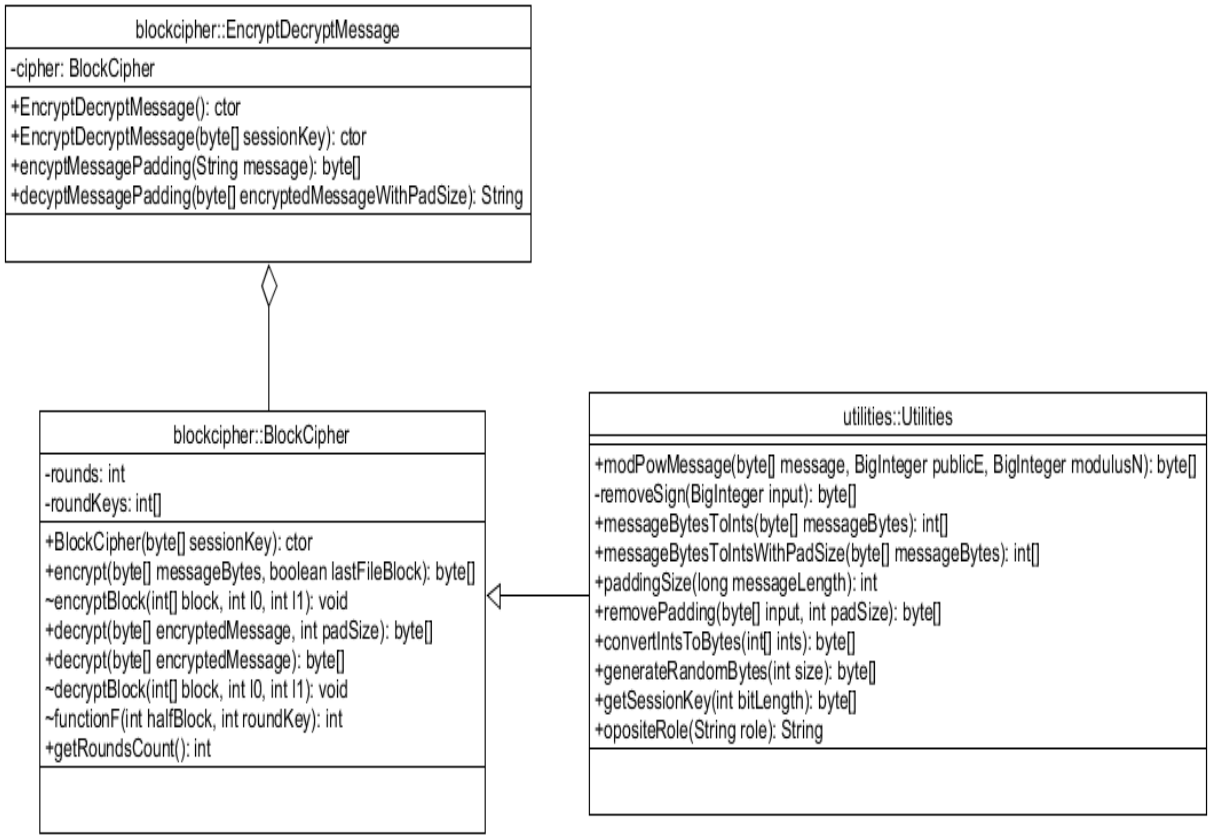


Figure 4.2.2 *represents the classes used in the 'Feistel Block Cipher Encryption Program'.*

4.2.3 Development Process

The realization of the block cipher was developed in the class 'BlockCipher.java'. On the other hand, the class 'EncryptDecryptMessage.java' is used as a facade of the 'BlockCipher.java' class. It implements methods that encrypt and decrypt strings. Further, the system works with 64-bit block sizes and a number of rounds, determined by the bit length of the session key, for example a 256-bit session key is used to produce round keys by splitting the key into 8 32-bit round keys, required for each round of encryption. During encryption, each message is converted to a byte array and converted to integers, having each four bytes (32-bit) equal to one integer, in this case each block in the cipher is represented by two integers. Therefore, the encryption procedures are performed over integers. Encryption uses the algorithm of a simple Feistel block cipher, having a cryptographic function and a 'XOR' operator, moreover its implementation explained in chapter **'Chapter: 2 Overview of Modern Cryptography'**. After encrypted, integer blocks are returned back to byte array structure. In addition to that, the last block of the message is padded to complete the block. The decryption algorithm implemented works as a reverse process of the encryption algorithm.

4.2.4 Justification

The described program is chosen as the second proof-of-concept program because it is the natural continuation of the 'RSA Encryption Program'. The combination of both proof-of-concept systems provides the foundation of practical and efficient encryption and decryption of given data string. In addition to that, the achieved functionality allows the further development of a chat program that would be based on this exact hybrid system between asymmetric and symmetric encryption.

4.2.5 Work Log

This section highlights the important milestones during the development of the program, the following information provided has been supported by the project diary:

October 01, 2017

"Started working on 'BlockCipher.java' class, which implements the use a simple block cipher encryption scheme."

October 02, 2017

"Completed the methods of 'BlockCipher.java' and in addition to that completed the relevant public static methods in the 'Utilities' class."

October 03, 2017

"Finalized the class 'BlockCipher.java' and developed the facade of the class- 'EncryptDecryptMessage.java'."

October 03, 2017

"Established the communication between the first proof-of-concept program and the second one described, where the asymmetric encryption is used to provide the session keys for the symmetric encryption."

4.3 Chat Communication Program

4.3.1 Description

The program is represented by a simple familiar chat user interface which aims to produce a communication between two parties, including both message and file transfer. At this current version the program is based on a peer-to-peer connection, using Java sockets and implements object output and object input streams in order to achieve the message or file transfer. The program uses messages as serializable objects. In addition to that, each object message can be a user or a system message, depending on which different actions are performed. When started the interface opens two windows one for the server and one for the client part, allowing live chat between both parties, further the live chat is supported by the use of threads in Java.

The proof-of-concept program consists the following classes:

- A **'ChatController.java'** class responsible for the connection and determines the current role (client or server).
- A **'ListenForMessage.java'** class implements a thread to listen for messages in the application.
- A **'MessageAndFileTransport.java'** class provides methods that transfer object messages and files using object input and output streams.
- A **'MessageObject.java'** class creates the message objects which are transferred in the chat interface.
- A **'SendMessage.java'** class which implements a thread for sending messages in the application.
- A **'SecureChat.java'** class is the main class and it is used to run the application.
- A **'SecureChatView.java'** class describes the components and application of the GUI.

4.3.2 Class Diagram

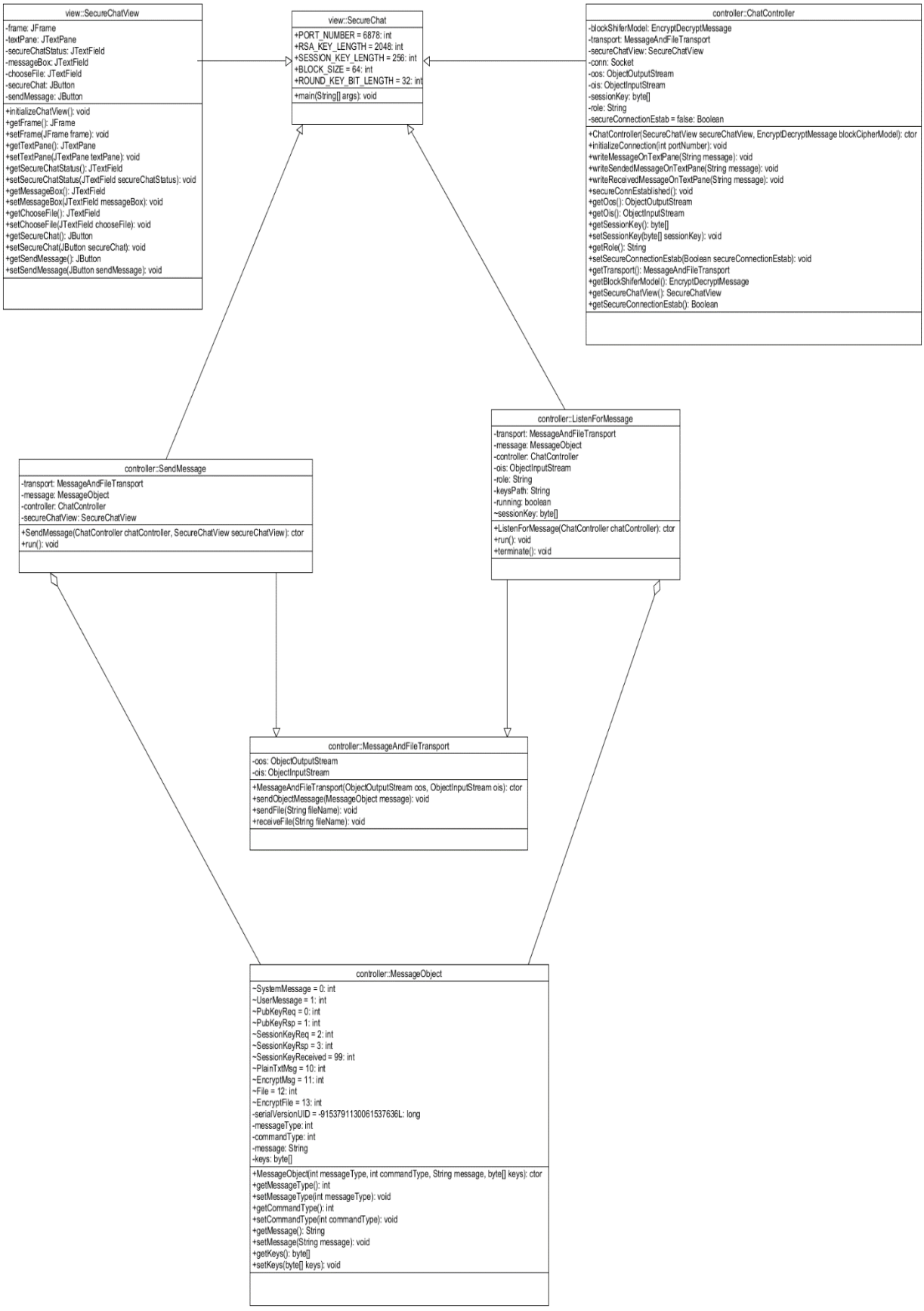


Figure 4.3.2 *represents the classes used in the 'Chat Communication Program'.*

4.3.3 Development Process

The development process began with researching how peer-to-peer connections are implemented with Java sockets. Further, the realization of this requires that the client knows the IP and port of the server in order to establish connection. Therefore, from a practical point of view this method dramatically complicates the relation between a client and server because of the NAT traversal issue. For simplicity of the proof-of-concept program the communication between client and server was narrowed down to using a single local host. The program is based on individual stand-alone application and behaves both as the client and the server roles. Moreover, when the application is started initially it runs as the server and when ran again it initializes the client as well- establishing the communication. In addition to that, with the start of the program the server opens a server socket and waits for a possible incoming client. Further, when a client connects to a server the socket communication is settled, therefore the conversation between client and server now is based on a protocol compiled in advance.

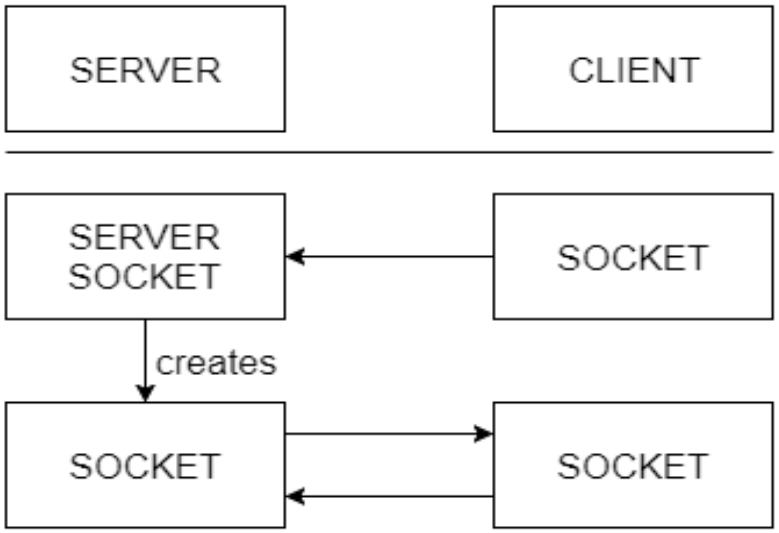


Figure 4.3.3.1 *represents the Java sockets communication.*

The messages in the program are dealt with in the 'MessageObject.java' class, moreover the messages realization is represented as objects and they are serializable during communication. Further, the structure of a message has three man components: message type, command type and payload. The message type specifies if a message is either of type 'system' or 'user' behaving different in each of both conditions, on the other hand the command type element is used to distribute the requests for the session keys between client and server and finally the payload is represented by a string or a byte array. On the other hand, the transportation (sending and receiving) of messages is performed by the classes 'SendMessage.java' and 'ListenForMessage.java' both of which are started as parallel threads when the program is executed.

messageType	commandType	sender	recipient	message	keys

Figure 4.3.3.2 *represents the structure of a message object.*

The interclass communication in the proof-of-concept program is accomplished using the MVC pattern, moreover the methodology splits the program in three main parts: model, view and controller.

4.3.4 Justification

This third proof-of-concept program was the final component in terms of establishing the initial prototype. The chat system is required in order to demonstrate the workflow of the previously developed two proof-of-concept programs. Moreover, the combination and connection between all three programs would result a working prototype that serves as an evidence of a working encrypted chat system between two users, allowing encrypted message and file transfer. Furthermore, the prototype showcases an early version of the end system, allows user testing and gives a clear understanding of what the core functionalities of the end system would be.

4.3.5 Work Log

This section highlights the important milestones during the development of the program, the following information provided has been supported by the project diary:

November 03, 2017

"Created the initial design of the user interface using Java swing."

November 11, 2017

"Developed a small program that performs a chat communication between two parties based on Java sockets"

November 13, 2017

"Refactored the chat program so it complies with the modern software engineering principles, moreover implementing the 'MVC' design pattern."

November 14, 2017

"Preparing the completed proof-of-concept programs for a merge forming the initial prototype."

4.4 Initial Prototype Program

4.4.1 Description

The initial prototype served not only as a key milestone showcasing the progress made but also supported the already established structure about the project and outlined the core components related to the design of the end system. Moreover, the resulted product is the logical connectivity between the three previously described proof-of-concept programs and implements a simple chat interface that allows communication between two entities based on a peer-to-peer connection using Java sockets. Further, it is the combination of classes, objects and methods of the proof-of-concept programs, modified to work as a whole system. In addition to that, the program allows encrypted communication between two parties using the combination of asymmetric cryptography as the key distributor with symmetric cryptography as the main encryption method to achieve performance and efficiency. The prototype being complete also allowed number of tests to be performed, indicating the correctness of the: encryption and decryption procedures, message transfer procedures, tests regarding the standards of human-computer interaction of the system and usability.

The structure of the 'Initial Prototype Program' is represented by 6 Java packages:

- An **'rsa'** package containing the classes responsible for the RSA algorithm that was implemented in this current state of the prototype.

- A **'blockcipher'** package representing the classes and methods that are required to execute the symmetric encryption during live chat.

- A **'utilities'** package an ancillary package supporting the 'rsa' and 'blockcipher' packages with the necessary methods required by the encryption packages.

- A **'controller'** package implementing the 'MVC' pattern of the chat GUI and containing the classes and methods related to the creation of message objects and their transfer using object input and output streams. In addition to that the classes contained are responsible for the connection establishment between client and server.

- A **'view'** package which is also part of the 'MVC' pattern implementation, which consists the class that generates the GUI interface itself and the main method that starts the program.

- A **'junittests'** package involved in the testing procedure of the program. Moreover, the main tests created are regarding the correctness of encryption and message transfer.

4.4.2 Class Diagram

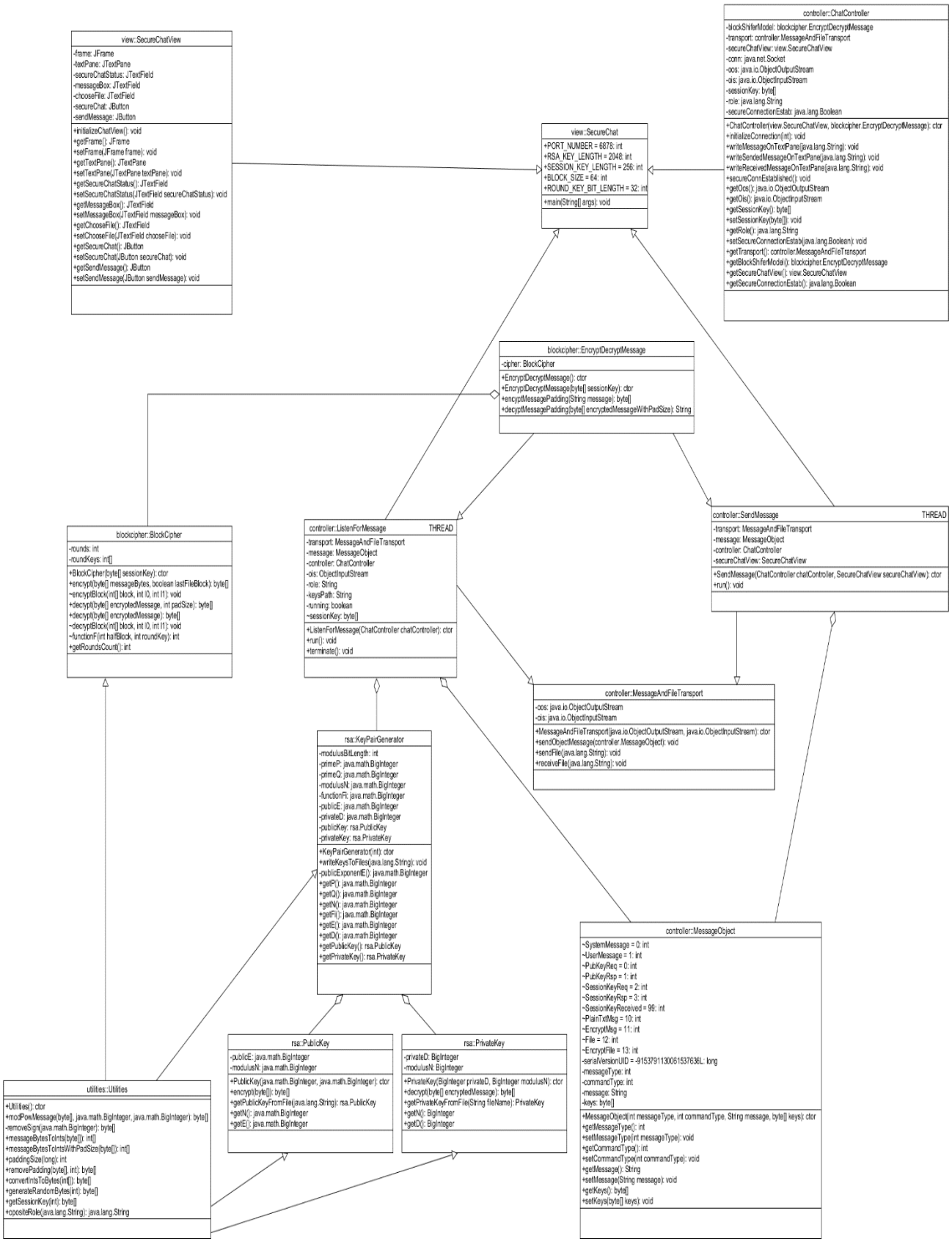


Figure 4.4.2 *represents the classes used in the 'Initial Prototype Program'.*

4.4.3 User Stories

The identified user stories and their fully or partially implementation, represent the importance of the 'Initial Prototype Program' to the final system.

User Story- Message Encryption and Decryption:

As a user I would like to be able to encrypt and decrypt information no matter of its size, because I would like to be able to communicate efficiently.

User Story- Chat Communication:

As a user I would like to be able to chat with another user in real time.

User Story- Fast Encrypted Communication:

As a user I would like to be able to communicate with another user using a chat interface which provides quick encryption and decryption of the messages sent, because I would like the chat communication to be established in real time.

4.4.4 Development Process

The development of the prototype is characterized by the combination of the three proof-of-concept programs and their relevant modifications and adaptations to interact properly between each other. The merge of the three programs was compliant with the implementation of the MVC pattern. Moreover, the model component of the pattern is responsible for the cryptographic encryption and decryption procedures, the view component is representing the graphical user interface of the program, and finally the controller realizes the protocol of the communication and the connection between the other two components- model and view.

After both instances of the program are started, one as server and one as client and the socket connection is established, the execution of the communication protocol begins. Further, the initiation of the protocol performing the secure communication, can be started both by the client or server, no matter of the order (both sides have equal rights).

Let 'USER A' begin by executing the secure communication protocol. Firstly, 'USER A' sends a public key request to 'USER B' for its public key. Following that, 'USER B' accepts the request and sends its public key to 'USER A'. After receiving 'USER B' public key, 'USER A' generates the session key and encrypts it using 'USER B' public key. Next, 'USER A' sends the encrypted session key to 'USER B', which second user would decrypt that session key using its own private key, and therefore the session keys between 'USER A' and 'USER B' will be distributed. Further, this session key would be used in the symmetric encryption of the messages and files sent between both users.

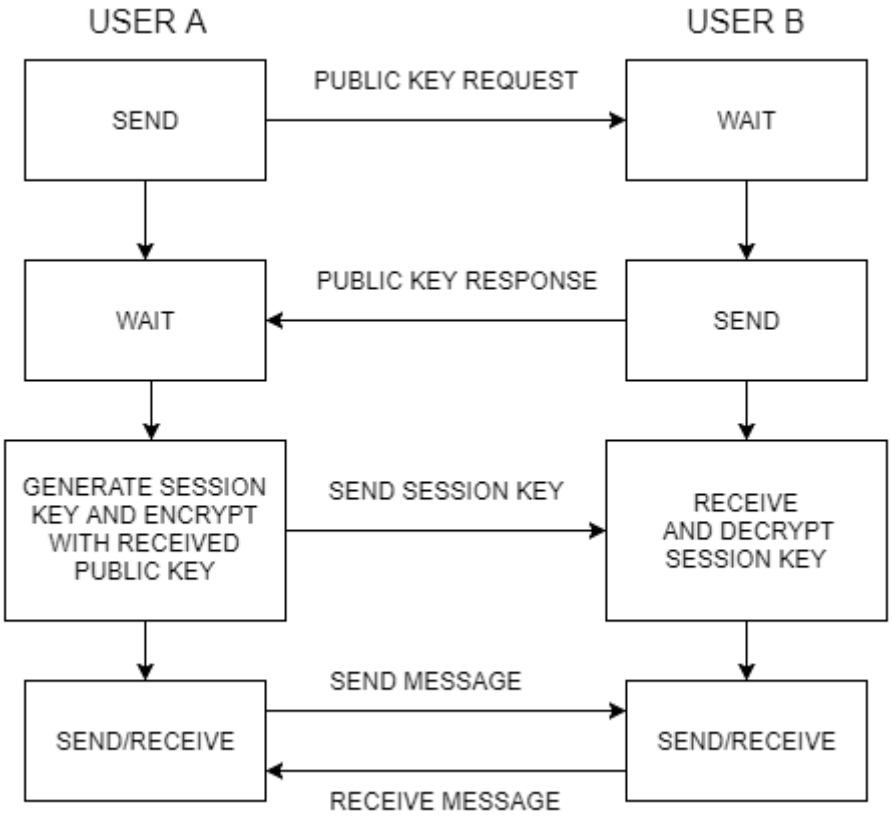


Figure 4.4.4 *represents the session key distribution process.*

4.4.5 Justification

An early development of a prototype was supported by the project specification requirements. Moreover, the prototype allowed prior establishment of the body of the end system and defined the most important characteristics and components that the final product must have. In addition to that, the prototype satisfied the most of the final deliverables and gave an opportunity for further development in terms of additional features and updates of the system. On the other hand, the creation of the program is also justified by the fact that it allowed user tests to be performed and outlined possible problems and issues that the system might have regarding user experience and usability. Finally, the prototype was a good model to be presented at the end of the first term as a model of what the end system might look like.

4.4.6 User Guide

The section provides the information required for any user to run and test the prototype.

Installation:

1. A working version of ‘Eclipse IDE for Java Developers’ is required to run the program (the prototype was developed and tested on ‘Version: Oxygen.2 Release (4.7.2)’).

2. Import the project in ‘Eclipse’ using the build in import feature and selecting the import method to be ‘projects from folder or archive’

3. The already imported project would contain two main folders: ‘src’ and ‘test’ respectively for the source code and the Junit tests of the project.

4. To start the program simply navigate to ‘Secure Chat Prototype\src\view’ and run twice (once for server and once for the client) the ‘SecureChat’ class as a Java application.

5. The program would recognise that both the server and client are started properly and it would establish the connection between both interfaces.

6. To run the tests of the program navigate to ‘Secure Chat Prototype\tests\junittests’ and run the desired test as a Junit test.

Usage:

- **Sending and Receiving Messages:**

  Type anything in the field on the right of the ‘Send Message’ button and click the button to send the message typed.

  Both messages sent and received are displayed in the history text field of the interface, identifying both the origin of the message whether it was the server or the client that sent the message.

- **Sending and Receiving Files:**

  Click on the field right of the ‘Send File’ button to select the file desired to be send, afterwards click the button to send the file to the other party of the communication.

  If a file was sent from another user, the save file dialog would automatically appear to the receiver.

- **Enabling Encryption:**

  To begin an encrypted chat session, simply click the ‘Secure Chat’ button located on the left top corner of the interface.

4.4.7 Work Log

This section highlights the important milestones during the development of the program, the following information provided has been supported by the project diary:

November 15, 2017

“Established connection between the three proof-of-concept programs.”

November 17, 2017

“Refactored most of the code structure in order to comply with modern software engineering principles.”

November 19, 2017

“Wrote adequate tests to confirm the correct workflow of the prototype”.

November 23, 2017

“Performed user experience tests based on the interface and functionalities developed, results were recorded.”

Chapter 5: End System Development

5.1 Development Preparation

The process of the development preparation began with an early requirements gathering during the summer, which was represented by the analysis of the project specification provided. Moreover, following the analysis of tasks a plan was established, representing the importance, difficulty and estimated time until completion of each of the identified tasks. In addition to that, the outlined tasks led to initial developmental focus on the proof-of-concept programs, gradually progressing towards the initial prototype. All of which proof-of-concept development was justified by design documentation. Moreover, the purpose of the initial requirements analysis was not just to establish a good work and time management structure but also dramatically improve my work efficiency.

On the other hand, the early establishment of requirements resulted a faster proof-of-concept development which on its own proceed with an early prototype being produced by the end of first term, being beneficial in terms of user tests and further requirements gathering. Further, apart from having established the core components of the system represented by the prototype, the development of the end system enforced additional research in terms of possible enchantments and upgrades based on other similarly operating systems.

The whole process of requirements gathering was formerly translated into early sequence diagrams, which established the initial design of the system, some of which remained through the entire development process towards the end system, while others required further refactoring and adaptation to be able to correspond to the changes made. In addition to that, an early advancement of class diagrams of the proof-of-concept programs contributed in terms of understanding and establishing the initial structure of the system.

All of the early design documentation justified the choice of the three proof-of-concept programs and their relevance as the core components of the end system.

5.1.1 Class Diagrams

'Server' Application

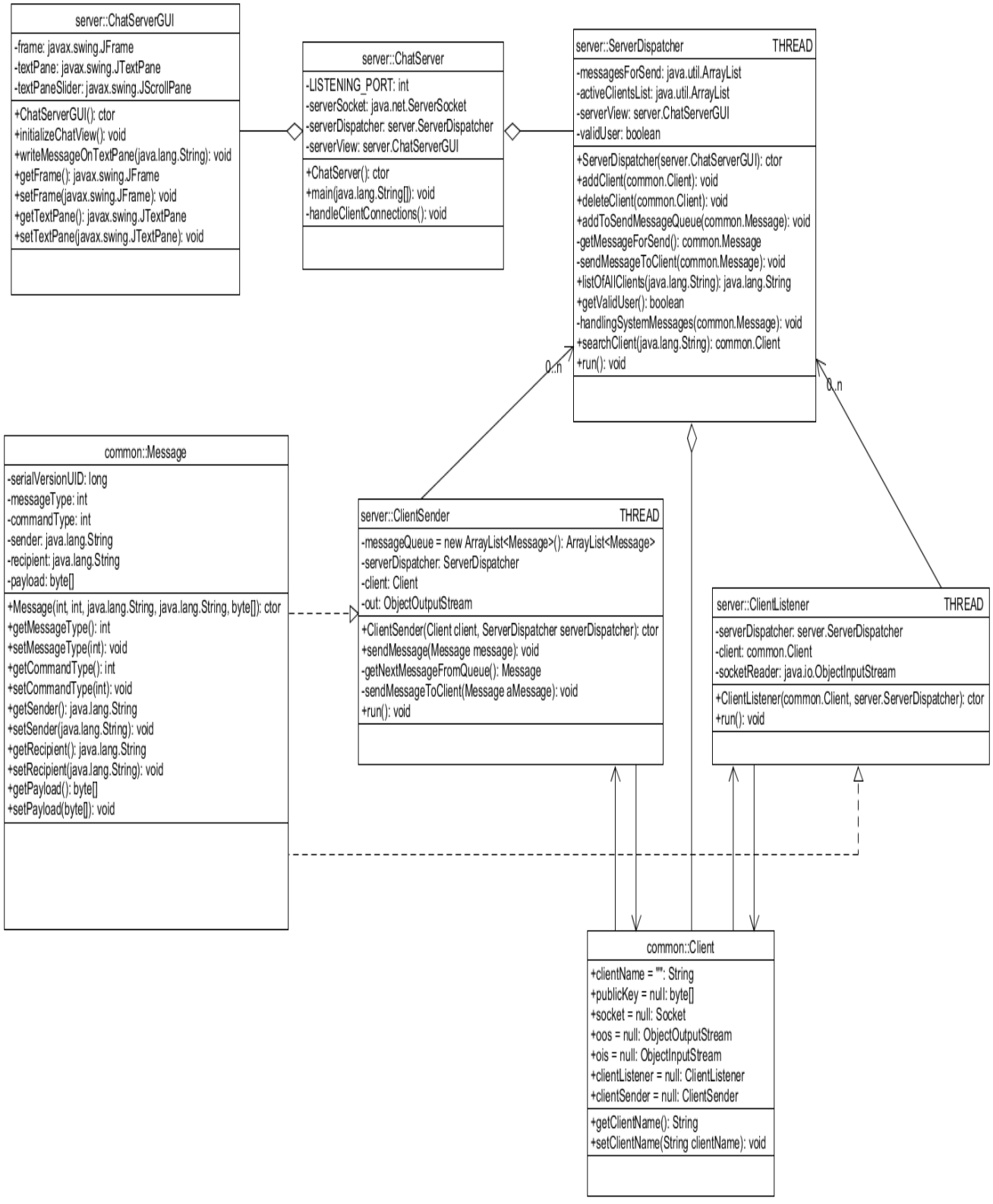


Figure 5.2.1.1 *represents the classes used in the 'ChatServer' application.*

‘Client’ Application

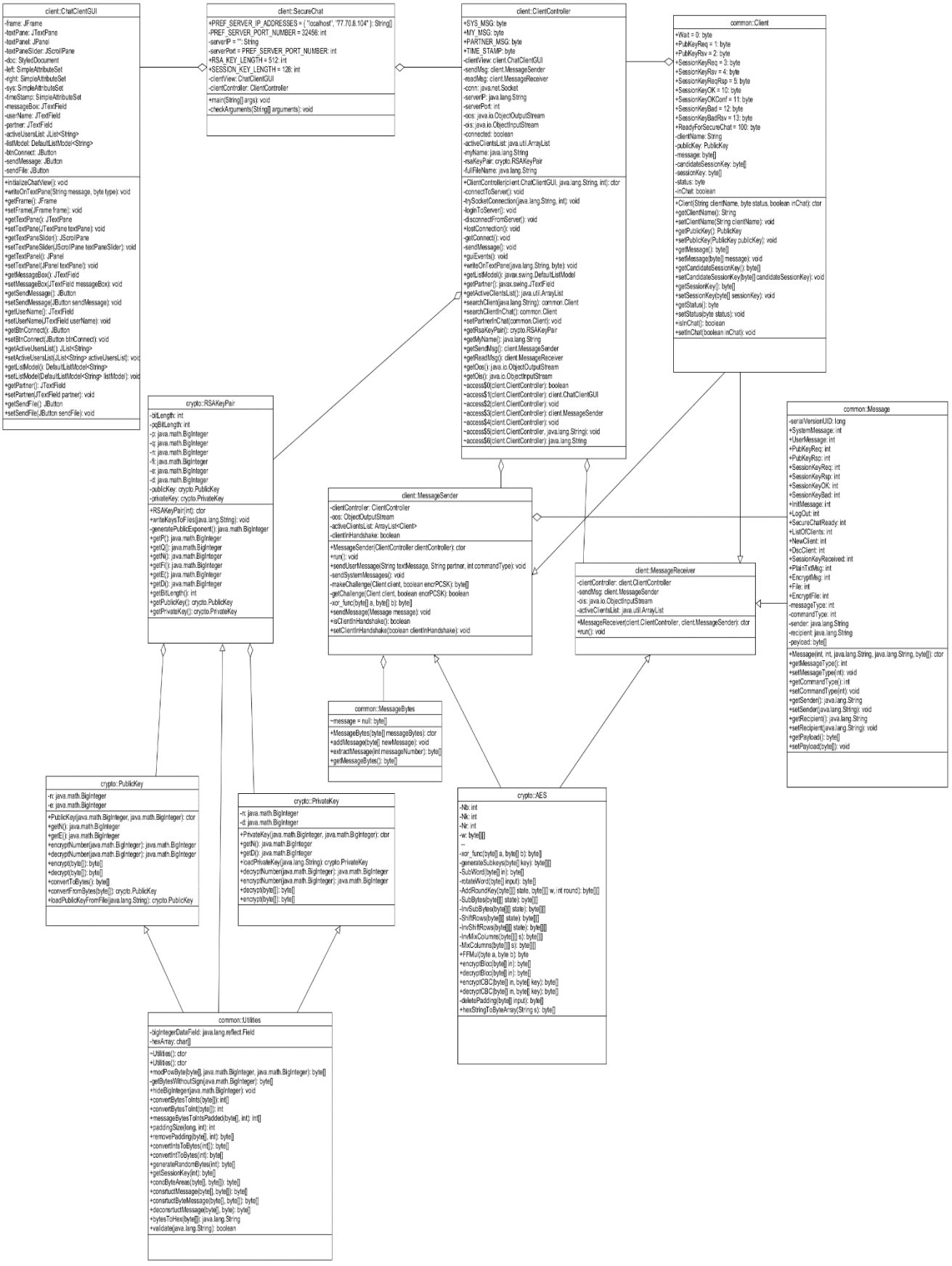


Figure 5.2.1.2 *represents the classes used in the ‘ChatClient’ application.*

5.1.2 User Stories

The identified user stories and their full implementation, represent the importance of the final system.

User Story- Message Encryption and Decryption:

As a user I would like to be able to encrypt and decrypt information no matter of its size, because I would like to be able to communicate efficiently.

User Story- File Encryption and Decryption:

As a user I would like to be able to encrypt and decrypt different type of files.

User Story- Chat Communication:

As a user I would like to be able to chat with another user in real time.

User Story- Fast Encrypted Communication:

As a user I would like to be able to communicate with another user using a chat interface which provides quick encryption and decryption of the messages sent, because I would like the chat communication to be established in real time.

User Story- Choose Chat Partner:

As a user I would like to see all other online users in the chat system and choose who I want to communicate with.

User Story- Multi-Chat:

As a user I would like to be able to send and receive messages from all the active parties involved in the chat system.

5.2 Prototype Analysis

The progress towards achieving the final product began with detailed analysis of the prototype, identifying both any possible problems and issues concerning mostly the security and efficiency of the program.

Peer-to-Peer Communication:

This is one of the main problems encountered because of the limitations that this type of communication provides. Further, those limitations are supported by the requirement that the first party that is trying to establish the communication has to know the IP address of the second party. Therefore, the practicality of the program would be severely damaged because in most cases the IP address would be always different, making it difficult to establish a connection. In addition to that, usually the partner a user is attempting to connect is behind NAT, meaning a configuration on the router has to be made on the port forwarding rules in order to be able to begin the communication. Further, NAT traversal problems have their solutions in terms of maintain the peer-to-peer communication (TCP hole punching), however all of the solutions require complex techniques which are beyond the scope of the project.

To avoid the listed above complications, a transition was made from a peer-to-peer based connection to a client/server based one. This decision solves all of the related problems because the server works

as a mediator between both communicating parties. In addition to that, this methodology also allows the realization of a multi-chat system and communication over the internet.

Authentication Absence:

This is a considerable weakness of the prototype, especially taking into consideration the fact that the main priority of the system is security. Moreover, one of the main elements of cryptography are authentication and data integrity, which are not implemented in the prototype making it vulnerable to different type of attacks, the most well-known one being “man in the middle”.

The solution of this problem required the implementation of an authentication and singing algorithm.

Simple Block Cipher:

The initially implemented block cipher in the prototype is not secure due to the simplicity of the cipher. Moreover, the cipher implemented is in ECB mode and it is vulnerable to brute force and statistical attacks making the chat system insecure.

To resolve the issue another much more advanced block cipher was implemented- AES in CBC mode which improves the resistance of the program dramatically against the possible attacks outlined above.

All of the identified problems above led to absolute redesign of the prototype structure and scope in order to achieve a good quality end product. Moreover, those changes stressed primarily the security aspect of the system but also focused on feasibility and design of the end system.

5.3 Development Process

The start of the final development process was characterized with the prototype analysis. Moreover, the identified issues in the prototype served as a backlog of the initial problems that required solutions. Therefore, development begin with the transition to a server/client based model by writing the code for two applications for the server and the client roles.

‘ChatServer’ Application

The server is realized as a multi-threaded application, using Java sockets for the communication. It has two main functions: registers all of the participating users in the system using their public key and username; transits messages between the registered users. Following that, the server also alerts if any changes happened to the list of users, moreover new users connecting to the system or current users disconnecting. It is also important to note that the server does not participate in any encryption and decryption procedures at all.

When the program is executed, the main class of 'ChatServer.java' creates a thread 'ServerDispatcher.java', starts it and opens a TCP socket, constantly listening for new clients. Following that, when a new client appears, the 'ChatServer.java' class creates an object of type client' and two threads respectively for 'ClientListener.java ' and 'ClientSender.java ', the purpose of those two threads allowing the client to send and receive messages. In addition to that, to finalize the process of adding a new client the 'ServerDispatcher.java ' saves in an object the socket and the two threads of the clients, adding that object in the list of clients using the 'ServerDispatcher.java ' thread.

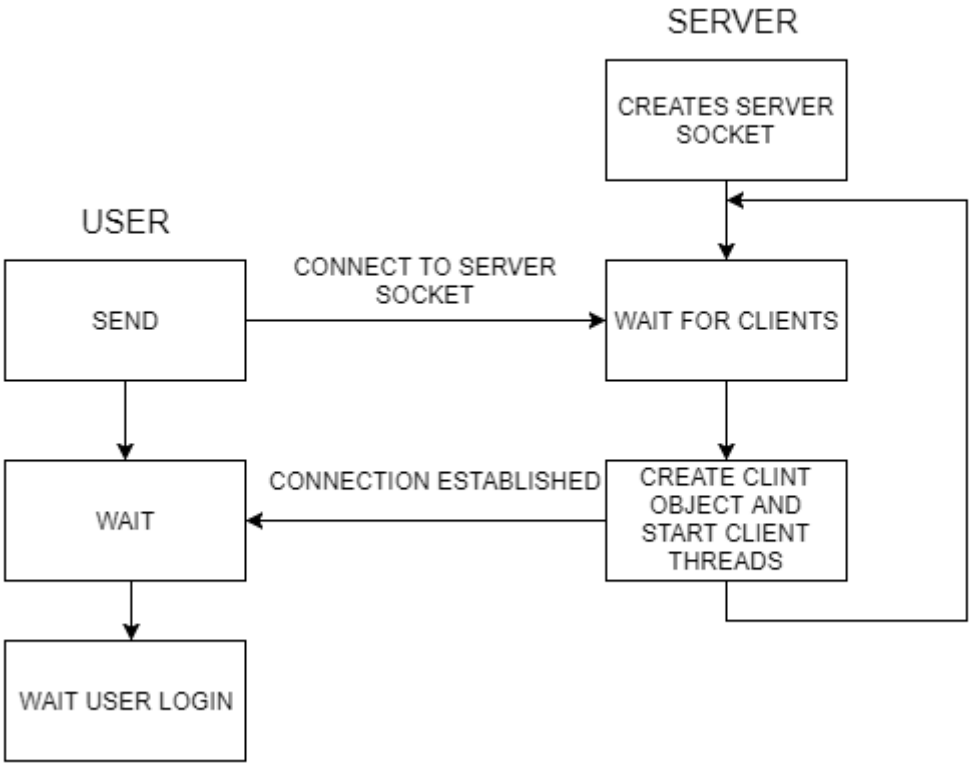


Figure 5.3.1 *represents the establishment of a socket connection between client and server.*

Thread 'ClientListener.java ' spends most of its time sleeping, while constantly listens for messages coming from clients, when the thread gets a message it sends it to the 'ServerDispatcher.java ' thread which is responsible for the transportation of the message to a specific client- contained in the message object.

Thread 'ClientSender.java' uses the producer-consumer pattern and the thread is responsible for the delivery of the message to its relevant client.

Thread 'ServerDispatcher.java' transitions all of the messages that it receives to their relevant users that are currently connected to the server. Further, at any given time ‘ServerDispatcher.java ' maintains a list of active users and repeatedly updates that list so it represents the current online users in the program. This is realized by using a queue that collects all of the received messages that are yet not sent. Moreover, when the queue is not empty, the messages are sent to the users, however when the que is empty the 'thread' goes to 'sleep' and waits.

The communication between the client and server is based on message transfer using a custom protocol. Further, this is showcased by the following diagrams below:

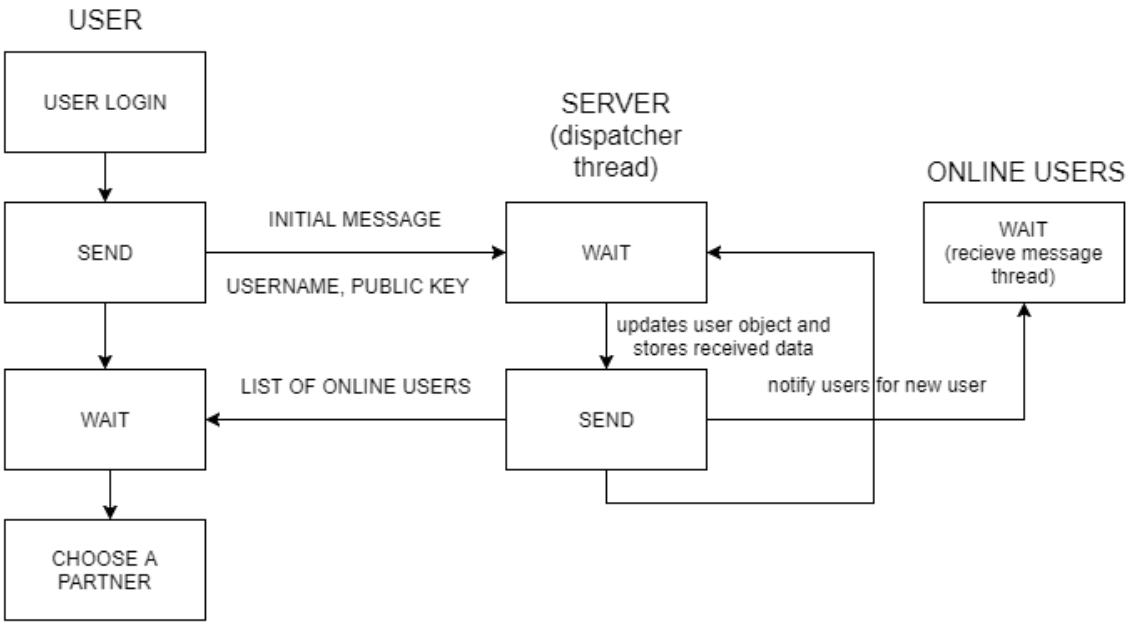


Figure 5.3.2 *represents the process of a user logging in to the server.*

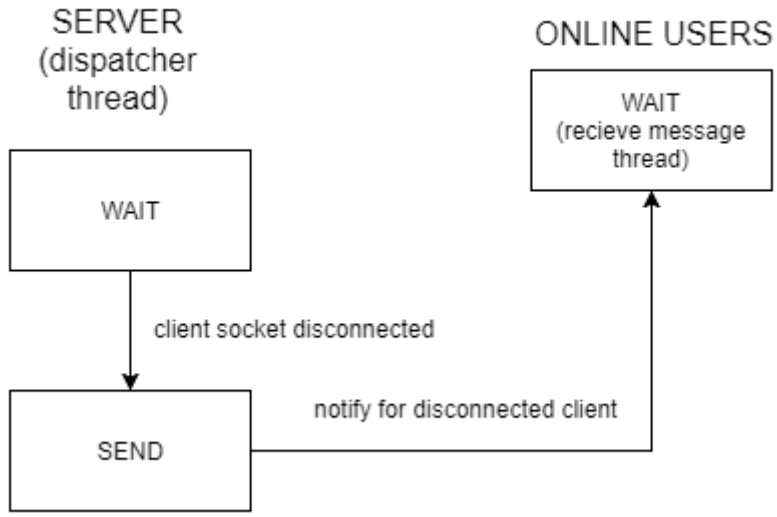


Figure 5.3.3 *represents the process of a user disconnecting from the server.*

'ChatClient' Application

The client application is developed as a multi-threaded program with the communication aspect being based on Java sockets. It is responsible for the registration of a user on the server and for the message and file transfer between already registered users. To guarantee authentication and data integrity of the transfer, the session key is exchanged using a cryptographic protocol. Following, the session key exchange AES in CBC mode is used to perform fast symmetric encryption of data.

When the client application is executed it attempts to establish a socket connection with the chat server program. Further, the program allows the input of an IP and port using the command prompt or using a prefixed list of addresses of servers. Following that, after a successful socket connection with the server is established, the program generates the RSA public/private key pair, starts two threads- 'MessageReciever.java' and 'MessageSender.java'- responsible for the message transfer

and waits for a username to be input in order to register the user in the server with that username and the generated public key. In addition to that, the client receives a response from the server containing a list of the currently online users in the system. Therefore, the recently connected user can choose a partner from that list to begin the communication. This process of communication between client and server is described by ***Figure 5.4.2***.

When a partner is chosen, the program begins the process of the session key exchange which can be described in the following steps:

1. 'USER A' (user initiating the communication) sends a request to the server for the public key of 'USER B'

2. Server responds with the public key of 'USER B'

3. 'USER A' prepares and sends the session key request.

   3.1. 'USER A' generates the candidate session key (SKa)

   3.2. 'USER A' encrypts the generated session key with the public key of 'USER B', represented by the formula '$\{SKa\}Kb$'

   3.3. 'USER A' calculates '$\{\{H(SKa)\}K-a\}Kb$'

   - 'H' is the cryptographic hash (SHA-256)
   - 'SKa' is the generated session key from 'USER A'
   - '$K-a$' is the private key of 'USER A'
   - 'Kb' is the public key of 'USER B'

   Moreover, this formula implies a hash over the generated session key, encrypts the resulted hash with the private key of 'USER A' (signing) and this second result is encrypted with 'USER B' public key.

4. 'USER B' receives the session key request and begins solving the challenge

   4.1. 'USER B' decrypts the session key 'SKa' (from **'3.2.'**) and calculates SHA-256 over result

   - If 'USER B' doesn't have 'USER A' public key, it requests the key from the server

   4.2. Decrypts challenge (from **'3.3.'**) applying reverse procedures of encryption

   - First decrypts using 'USER B' private key and then using 'USER A' public key
   - The result is '$H(SKa)$'

   4.3. 'USER B' compares results of **4.1** and **4.2**

   If the compared results are the same, the system proceeds with the next step, however if not the same the session key exchange is denied ('USER B' sends a message to 'USER A': 'SessionKeyBad').

5. 'USER B' prepares and sends the session key response

   5.1. 'USER B' generates the candidate session key (SKb)

5.2. 'USER B' encrypts the generated session key with the public key of 'USER A', represented by the formula '$\{SKb\}Ka$'

5.3. 'USER B' calculates '$\{\{H(SKb)\}K-b\}Ka$'

- 'H' is the cryptographic hash (SHA-256)
- 'SKb' is the generated session key from 'USER B'
- '$K-b$' is the private key of 'USER B'
- 'Ka' is the public key of 'USER A'

5.4. 'USER B' calculates '$SKab$' as 'XOR' over 'SKa' and 'SKb'

5.5. 'USER B' encrypts 'SKa' using the AES applying key '$SKab$'

5.6. 'USER B' sends to 'USER A':

- '$\{SKb\}Ka$'
- '$\{\{H(SKb)\}K-b\}Ka$'
- '$AES(SKa)SKab$'

6. 'USER A' receives the response from 'USER B'

6.1. 'USER A' decrypts the session key 'SKb' (from **5.2.**) and calculates SHA-256 over result

6.2. Decrypts challenge (from **'5.3.'**) applying reverse procedures of encryption

- First decrypts using 'USER A' private key and then using 'USER B' public key
- The result is '$H(SKb)$'

6.3. 'USER A' compares results from **'6.1.'** and **'6.2.'** if the results are equal proceed to step **'6.4.'** , otherwise the session key exchange is rejected ('USER A' sends a message to 'USER B': 'SessionKeyBad')

6.4. 'USER A' calculates '$SKab$' as 'XOR' over 'SKa' and 'SKb' and decrypts the received in **'5.6.'** '$AES(SKa)SKab$'

6.5. 'USER A' compares results of **'6.4'** with 'SKa' if equal then '$SKab$' is the final session key of the communication

6.6. 'USER A' encrypts 'SKb' with '$SKab$' applying the AES cipher ($AES(SKb)SKab$) and sends it to 'USER B'

7. 'USER B' receives '$AES(SKb)SKab$' from **6.6.** decrypts it using '$SKab$' and AES and compares the result with 'SKb' if equal then accept '$SKab$' as the final session key and send to 'USER A' a message stating that the session key is appropriate ('SessionKeyOk'), otherwise send message ('SessionKeyBad')

8. 'USER A' receives message from 'USER B' which confirms or denies the final session key

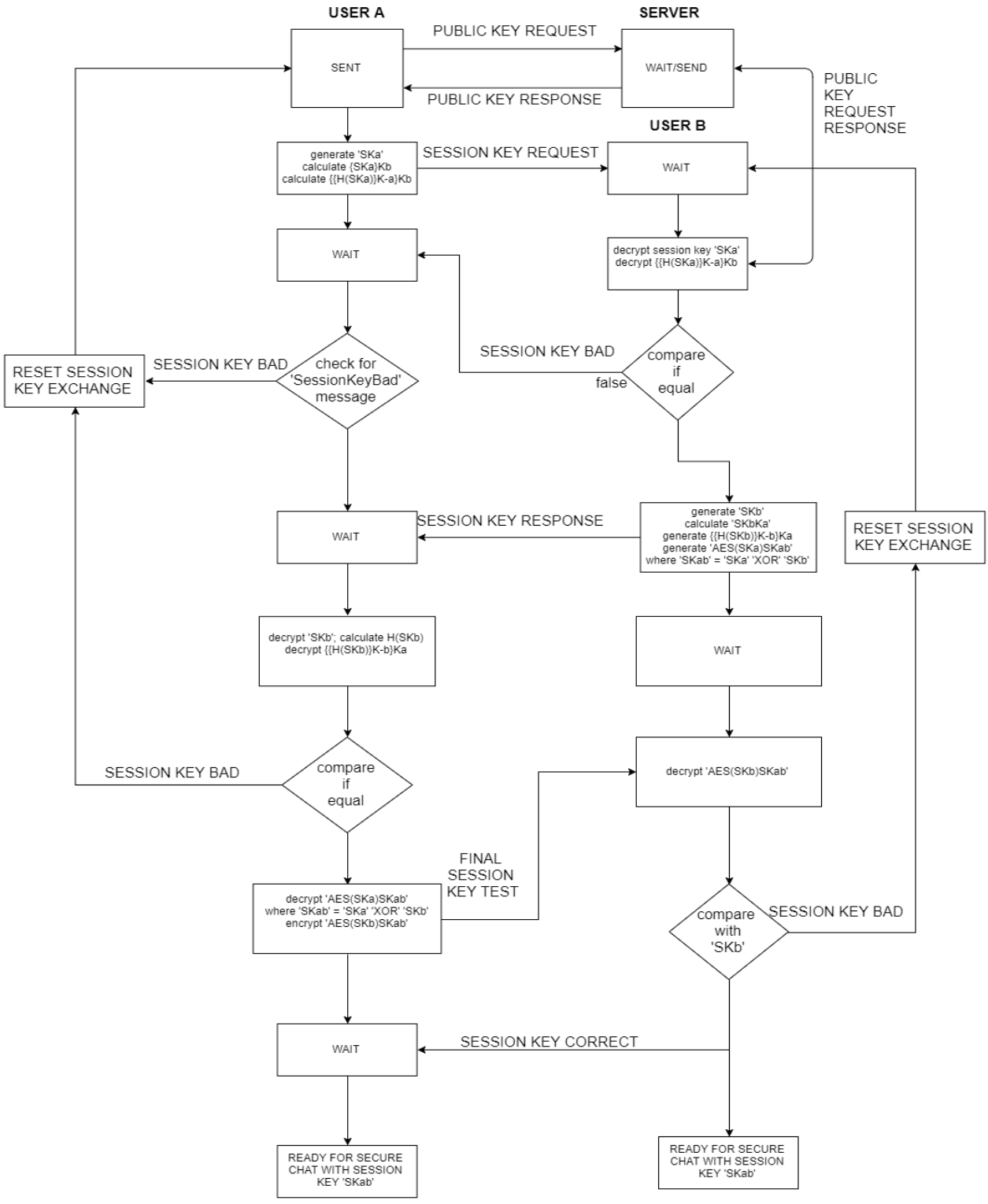


Figure 5.3.4 *represents the session key exchange process.*

After completing the session key exchange, the system proceeds the communication between ‘USER A’ and ‘USER B’ implementing symmetric encryption (AES).

The ‘Advanced Encryption Standard’ is realized in class AES, moreover implementing all of the principle functions of the encryption algorithm. Further, it allows the usage of 128-bit, 192-bit and 256-bit keys. The application is set to work with 128-bit key by default because it is sufficient in terms of the chat application.

The principle of the applied AES algorithm in the program can be demonstrated with the following pseudocode (for 128-bit key):

```
PROCEDURE AES128(plaintext[16]:byte, ciphertext[16]:byte,
w[44]:word)
BEGIN
  state[4,4]:byte
  state = plaintext
  AddRoundKey(state, w[0, 3])
  FOR round = 1 to 10
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*4, (round+1)*4-1])
  END FOR
  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[40, 43])
  ciphertext = state
END PROCEDURE
```

Figure 5.3.5 *represents the 128-bit key AES encrypting pseudocode.*

The number of encrypting repetitions of the plaintext is dependent on the length of the cipher key. Further, 10 cycles are performed using a 128-bit key, each cycle executes several consecutive steps, each step containing five alike procedures, where the cipher key is used in just one of the five procedures. AES performs a number of rounds, each round requires a round key which is generated from the cipher key, this process is also known as key expansion (in the chat application this is achieved in method 'generateSubkeys').

```
FOR i = 4 to 43
  temp = w[i-1]
  IF i = 4, 8, 12, 16, ... , 40 THEN //multiply of 4
    // Operation 'Rotate' is performed on temp
    Each byte is changed (using SubBytes)
    temp=temp XOR Rcon[i] // each byte from temp is accumulated
with the constant value of Rcon
  END IF
  w[i] = w[i-4] XOR temp
END FOR
```

Figure 5.3.6 *represents the key expansion procedure.*

The other 4 left procedures: 'SybBytes', 'ShiftRows', 'MixColumns' and 'AddRoundKey' are accomplished using the standard for the AES algorithm and are further described in **'Chapter: 2 Overview of Modern Cryptography'**.

Further, decryption procedures are the exact opposite of encryption. This is supported by the pseudocode below:

```
PROCEDURE AES128
(ciphertext[16]:byte,plaintext[16]:byte,w[44]:word)
BEGIN
  state[4,4]:byte
  state = ciphertext
  AddRoundKey(state, w[40, 43])
  FOR round = 9 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round*3, (round+1)*3-1])
InvMixColumns(state)
  END FOR
  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, 3])
  plaintext = state
END PROCEDURE
```

Figure 5.3.7 *represents the 128-bit key AES decrypting pseudocode.*

Further, the AES encryption is realized in Cipher Block Chaining mode. The workflow of CBC mode can be described using the following two figures representing encryption and decryption:

5.4 Features of the System

Username Choice

The system allows the user to choose a username, which would represent the user in front all of the other current online users on the system, moreover users choose who to communicate with based on the usernames.

Chat Partner Choice

The system grants the user the right to choose who he is communicating with at any time simply by choosing the username he wants to chat with.

Sending and Receiving Messages

The message transfer is represented by the chat interface. Further, users can send messages to any other users that are presently online in the chat system, and therefore receive messages from any users that desire to send a message.

Sending and Receiving Files

The file transfer feature allows a user to send files to any user that is online in the system and receive files from other online users.

Multi-Chat

The multi-chat component allows a user to send messages to multiple users at the same time, and in addition to that receive messages from different online users simultaneously.

Flexible Communication

The system allows both communication using a local host which is used mostly for demonstration purposes and communication over a network (using a server as a mediator), which represents the practical implementation of the system. Moreover, connection to the server allows users from different locations to communicate between each other in real time.

Secured Communication

The application supports fully encrypted conversation between all parties participating in the program. Moreover, this is achieved by the use of modern encryption algorithms and procedures that allow fast and reliable secure communication.

5.5 Runtime and Analysis

The section represents the runtime of the RSA and AES encryption and decryption algorithms implemented in the program. The tests are performed on two different computers. The measured data in the tables below is in milliseconds.

The RSA encryption is made over 256-bit array (32-bytes).

The AES encryption is made over 4096-bit data (512-bytes).

Test 1 (Intel Pentium N3700 - 4 Core 1.6GHz):

RSA Test / Key Length	512	1024	2048	4096
RSA key pair generation	122	203	2795	24466
RSA encryption	9	8	36	411
RSA decryption	2	10	60	444

AES Test / Key Length	128	192	256
AES encryption	10	4	4
AES decryption	8	4	4

Test 2 (Intel i5 4210H - 8 Core 2.9GHz):

RSA Test / Key Length	512	1024	2048	4096
RSA key pair generation	215	145	2761	6769
RSA encryption	2	8	14	102
RSA decryption	1	5	19	131

AES Test / Key Length	128	192	256
AES encryption	3	2	2
AES decryption	3	2	2

A conclusion can be made that both the RSA and AES encryption procedures have their speed performance increased with the increase of the CPU power, however the RSA encryption is much more dependent on the computational power of the processor than the AES encryption.

5.6 User Guide

5.6.1 Installation and Preparation

The program can be run either from Eclipse or as a JAR file from the command prompt.

Installation Eclipse:

1. A working version of 'Eclipse IDE for Java Developers' is required to run the program (the prototype was developed and tested on 'Version: Oxygen.2 Release (4.7.2)').

2. Import both projects in 'Eclipse' ('ChatServer' and 'ChatClient') using the build in import feature and selecting the import method to be 'projects from folder or archive'

3. The already imported two projects would contain a single main folder: 'src', for the source code of the project.

4. To start the server instance of the program simply navigate to 'ChatServer\src\server' and run class 'ChatServer.java' to start the server of the application using local host.

5. To start the client instance of the program simply navigate to 'ChatClient\src\client' and run class 'SecureChat.java' to start a single interface of the program, however for testing purposes run 'SecureChat.java' twice to start two interfaces which would represent the communication between two users.

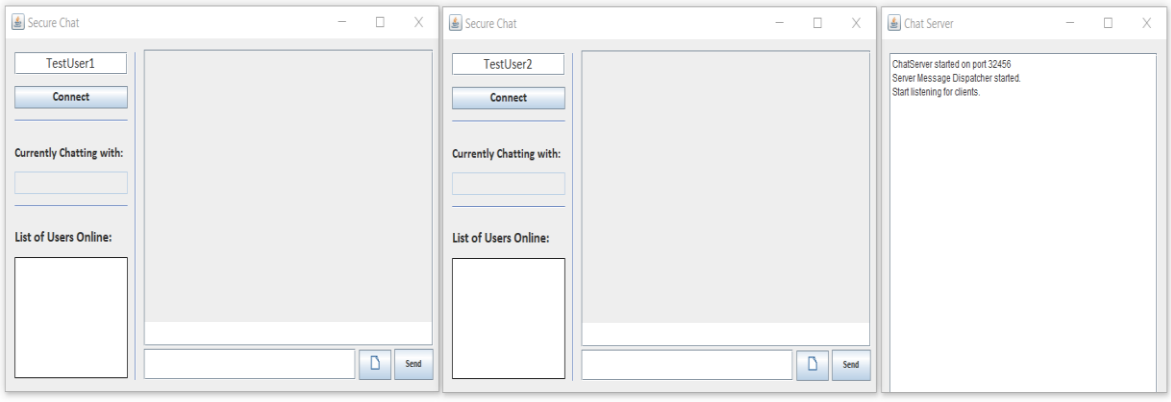
6. The program would recognise that both the server and client are started properly and it would establish the connection between both interfaces.

7. To run the tests of the program navigate to 'ChatClient\test' and run the desired test as a Junit test.

*The program also does read command line arguments if the user wishes to manually input port for server and IP and port for client.

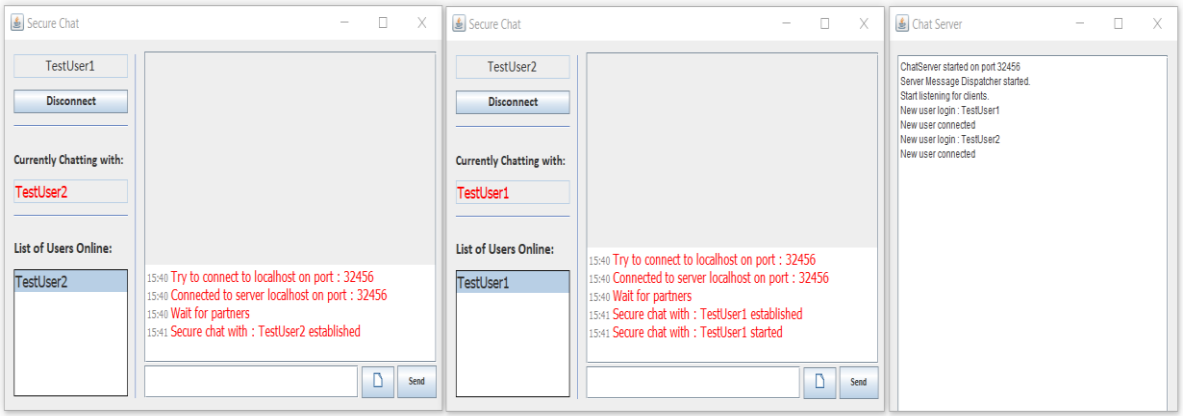
5.6.2 Usage

- **Connecting to Server:**



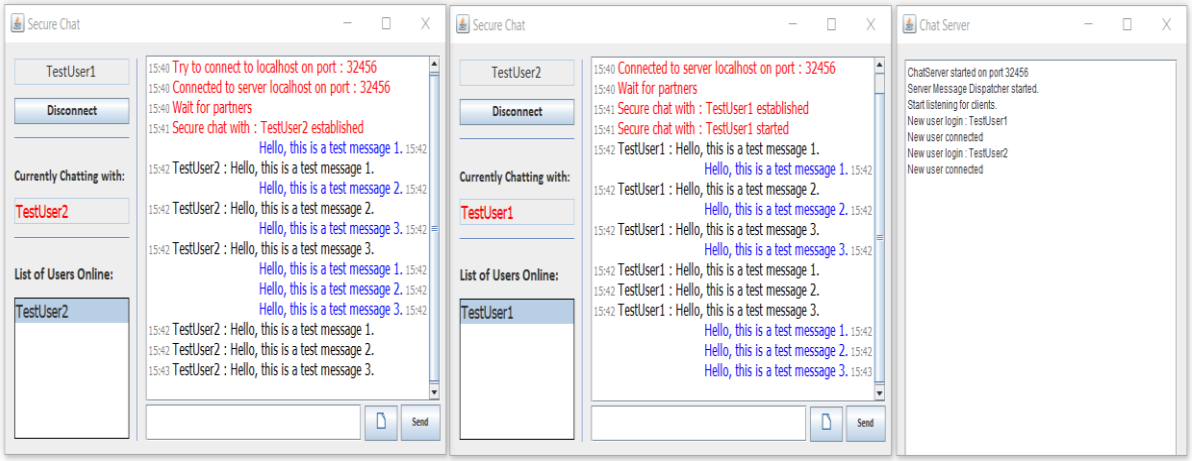
Choose the desired username and press the ‘Connect’ button to connect to the server.

- **Choosing a Chat Partner:**



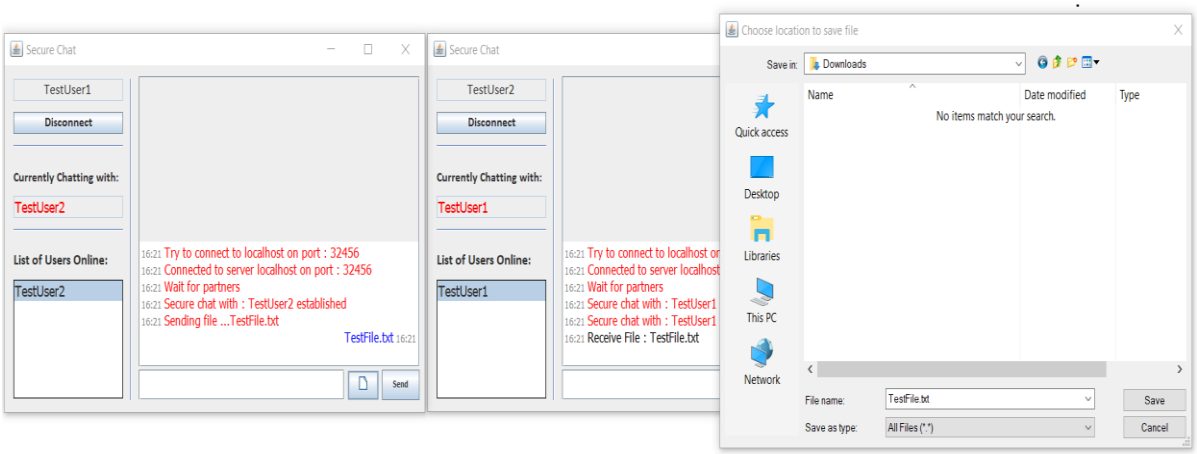
Currently active users would appear in ‘List of Users Online:’ box, simply click on the user you are going to chat with to establish the communication.

- **Sending and Receiving Messages:**



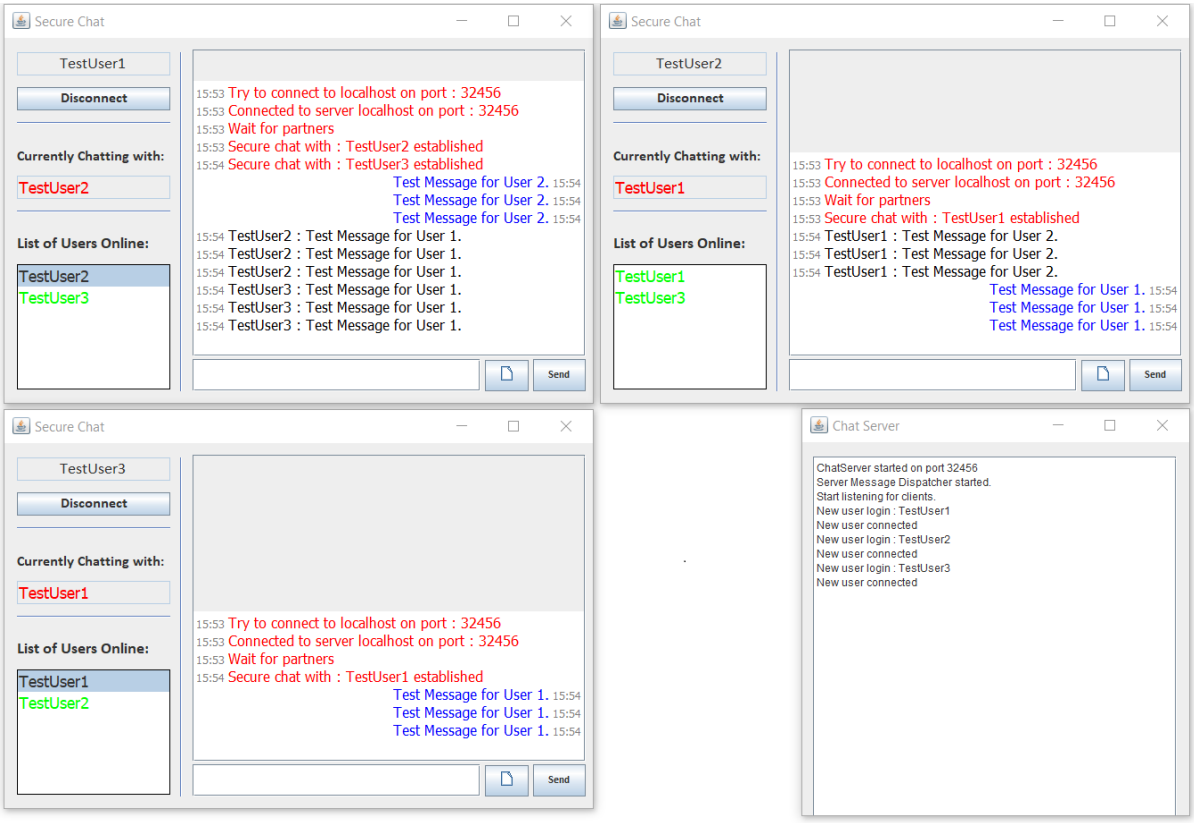
After the connection between both users is established, moreover meaning that the session key exchange was successful, the chat now works in a fully encrypted mode. Users can type messages in the bottom text field and send them securely using the ‘Send’ button or the ‘Enter’ key. Receiving messages happens automatically and both sent and received messages appear in the chat history field.

- **Sending and Receiving Files:**



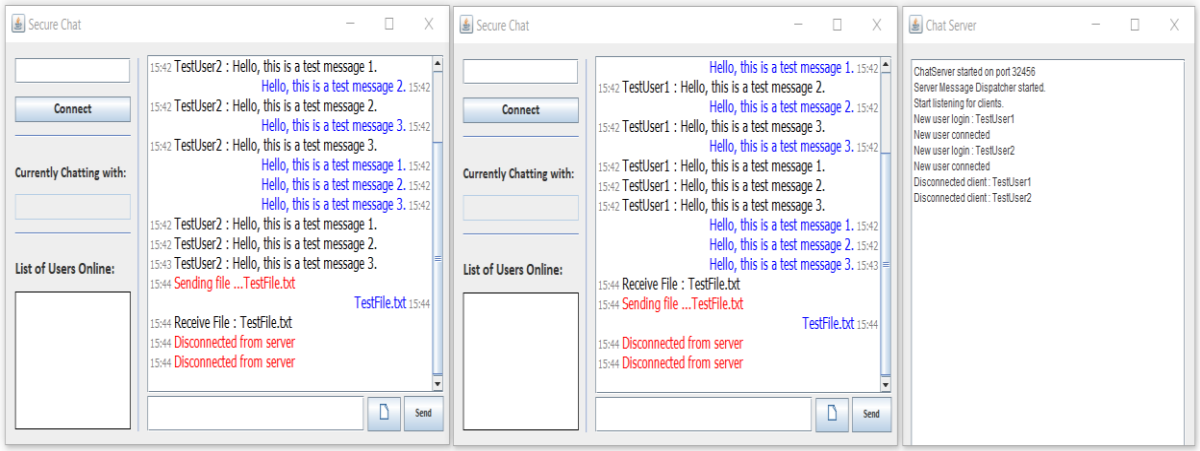
To send a file a user must click the 'File' icon next to the 'Send' button and choose the desired file to be sent. Following that, to receive the file a user must choose where that file is going to be saved in the file transfer dialog box.

- **Multi-chat**



The application supports multi-chat- meaning that more than two users can be communicating simultaneously. Users can constantly swap between who they are communicating with, using the 'List of Users Online:' box. In addition to that, a single user can also receive messages from all online users that have chosen to chat with him.

- **Disconnect from Server:**



To disconnect from the server, simply click the disconnect button, a message will be shown in the chat history field that the user has successfully disconnected from the server.

5.7 Future Enhancements

The possible future extensions of the system are based upon user testing and can be categorized in two main topics.

The first topic being improvements regarding the interface of the system. Those enhancements are responsible for the better user experience and are regarding the change of the design of the application. Moreover, those changes can be regarding the current look and layout of the different fonts, colours, components and functionalities that the system performs. Based on that information, different changes are planned to be made in order to improve the user experience. For example, user testing outlined that a conversation chat between 3 or more users simultaneously would be a strong improvement. Moreover, another additional features such as user block communication and user profiles are planned.

The second topic of enhancements is concerning the security of the system. Moreover, different security techniques can be implemented to make the code of the final program to be more resistant to different type of attacks. In addition to that, numerous improvements can be made regarding the encryption and decryption algorithms implemented and their efficiency.

5.8 Work Log

This section highlights the important milestones during the development of the program, the following information provided has been supported by the project diary:

January 02, 2018

“Research programs performing similar actions to the system, collect information and ideas for the final product.”

January 07, 2018

“Began the development of the final product based on the previously carried research.”

January 22, 2018

"Converted the program to use a client/server connection instead of the peer-to-peer one used before."

January 25, 2018

"Major overall code refactored to comply properly with the client/server connection."

February 04, 2018

"Relevant comments were made on all of the developed code and also configured the code structure with 'Google' check style."

February 10, 2018

"Finished the AES Java implementation research, the outlined information crucial in terms of how the algorithm should be applied."

February 17, 2018

"Improved the security of the system by transitioning from using a simple Feistel block cipher to the Advanced Encryption Standard (AES) as a tool for symmetric encryption."

February 27, 2018

"Extensive refactoring of the code was done, stressing error handling and bad code smells."

March 02, 2018

"Concluded the project documentation."

March 05, 2018

"Observed, analysed and registered the runtime of the encryption and decryption procedures with different key lengths."

March 12, 2018

"Concentrated the work on completing the chapter in the report regarding the development of the final product."

March 17, 2018

"In depth testing of the code, preparing for the final submission."

March 21, 2018

"Fixed any issues and errors that occurred during the testing period and finalized the program."

Chapter 6: Software Engineering Process

6.1 Methodology

Considering the required deliverables and deadlines by the university, the most appropriate software development process for the project is agile. Developing in agile allows working software to be produced early which satisfies the requirement of early deliverables of the project. Moreover, in this project agile is a perfect choice because of the gradually and evolutionary development of the system being very convenient with the support of the arranged supervisor meetings. On the other hand, agile is extremely flexible, meaning that redesign is always possible even in the core components of the system, granting the developer the ability to perfect the end system. In comparison if waterfall methodology was chosen, those arguments wouldn't be applicable.

To support the agile software development process, an informal representation of scrum framework was adopted. The main reason behind the choice of scrum is the fact that I already have some experience working with scrum because of the team project course taken last year. In addition to that, the scrum workflow represented by sprints each two weeks, coincides with the supervisor meetings allowing retrospective of the work that has been done over the sprint. To conclude, this combination of software engineering methodologies resulted a progressive software development, allowing constant improvement by adding functionalities to the system until the end system is developed.

6.2 Testing

The process of creating the program was supported by test driven development. The development of tests allowed me to detect and prevent unexpected errors, which led to a much better overall code quality. The testing process involved: Junit tests, prototyping, user testing. Moreover, the tests performed considered the core functionalities and usage of the program.

Unit Testing:

Junit testing was performed on most of the code, achieving more than 80% coverage. Further, testing all of the functionalities in the program, while concentrating on the correct generation of prime numbers and proper implementation of encryption and decryption algorithms. Testing provided guarantee that encryption and decryption on certain data was completed appropriate. In addition to that, important tests also stressed the success of communication between two users, represented by the message objects transfer.

Prototyping:

The completion of the prototype offered a chance to observe how the gathered requirements transferred into the development process. The prototype showcased the most important essential functionalities that the system had to implement. Moreover, this led to both the identification of possible problems with the structure of the system and the recognition of further enhancements. On the other hand, the prototype was also in terms of gathering user feedback, which also resulted numerous changes to be made.

User Testing:

User testing was crucial in terms of seeing how the user interacts with the program. Results of all user tests were observed and analysed. Therefore, this led to the change of the design of different components of the system. Following that, a much better user experience, moreover the final design of the system was heavily influenced by the carried user testing.

6.3 Techniques and Tools

Following the correct principles and implementations to produce good quality code, number of techniques and tools were used in order to achieve that.

To begin with the main code redactor used is Eclipse Java Oxygen, October 2017. The choice of this software is justified by the fact that I have already experienced working with it during the course. In addition to that, eclipse is extremely powerful when developing in Java because of its simple interface and build in classes, methods and libraries.

The use of a file repository was also a vital part of the development process. Moreover, the technique was supported by the university SVN repository, providing version control over the developed code. This technique allowed me to avoid problems and critical errors regarding code development.

Google-check-style is also fully supported in the code provided. Following the standards in principles of code writing is an extremely good practice to achieve clean and good code quality. Moreover, that practice contributes to better visibility and understanding of the code when examined by another party which was not involved in the development.

Another crucial tool used in the project is an encryption program called Pretty Good Privacy or PGP. The program provides cryptographic privacy and authentication for data communication. Moreover, in practice the program can be used for encrypting and decrypting data and the application itself covers all of the modern principles of cryptography. Therefore, the program supports confidentiality, data integrity, non-repudiation and authentication. PGP served as an example of what a good secure cryptosystem should look like and what functionalities it must perform. Further, this precedent was valuable when defining the end system design, structure and components.

A few other supportive programs were used to complete the project, moreover those applications are responsible for: UML diagrams, work flow diagrams, charts.

6.4 Documentation

The design documentation process of the project has been consistent and represents the current state of the system at each period of the development. Moreover, this process is characterized with a lot of changes between the initial and final versions of the system. This is because the project required a lot of adaptations and changes in order to achieve a good overall product. The documentation of the project design describes the changes that took place and demonstrates the progress achieved.

On the other hand, the code documentation is supported by the use of modern software engineering practices. Extensive commenting has been done over most of the code and tests developed. Again, the detailed documentation serves as a clear indicator of the work completed and how the project was developed.

6.5 Professional Issues

Professional issues in information security can stress different financial, legal, social and organizations aspects of the IT industry. An information security specialist "plays an important role in an organization's approach to managing liability for privacy and security risks" (Cengage, 2018). Therefore this section represents an analysis of the professional issues that are most relevant to this project. Furthermore, this being an information security related project the most relevant outlined topics are regarding privacy and legal issues.

Privacy

Clearly, privacy is probably one of the most important topics in information security. In the past "many organizations are collecting personal information as a commodity, and many people seek protection of their privacy from the governments" (Cengage, 2018). Today with the rise of the pressure for privacy protection, the meaning of privacy is described not by an absolute freedom but by "the state of being free from unsanctioned intrusion" (Cengage, 2018). Different regulations are implemented responsible for the privacy of people, the Electronic Communications Privacy Act (1986) is an example for such regulation. It is responsible for the control of "the interception of wire, electronic and oral communications, protecting individuals from unlawful actions" (Cengage, 2018).

Looking into the past there are numerous examples of privacy breach in different chat systems. An example can be given with the thousand 'Bloomberg' terminal users that had their identities revealed while participating in an anonymous chat room in 2017 (NY Post, 2017). This is just a single example of the many information leakages that happen in our modern world. Moreover, the damage done in this particular example is not that severe compared to other cases that happened in the past. Thus, data protection and privacy are top priorities for a system performing a chat based communication, therefore their maintenance is essential to avoid causing any damage.

This specific project is closely related to privacy because the main functionality implemented is an encrypted chat system which allows the communication between two users. A close relation can be made with the BSC Code of Conduct which states that "You shall have due regard for public health, privacy, security and wellbeing of others and the environment" (BSC Code of Conduct, 2018), stressing the issue of privacy in a chat system. Furthermore, the main functions of a secure chat system would be prioritizing the complete implementation of confidentiality. This is achieved in this project using different techniques and principles that maintain the privacy of third party users. For example, the server instance of the program does not deal with any kind of encryption procedures over message, therefore the server maintains the privacy of those messages. On the other hand, security was also accomplished by the maintained good code principles, testing and error handling. This was done to mitigate possible attacks that would lead to information leakage in the program.

A lot of effort has been put in understanding the privacy regulations and adapting the system in such way that it would maintain confidentiality and be compliant with the BSC Code of Conduct.

Chapter 7: Self-Evaluation

7.1 Term 1

To begin with, the main focus of the project during the first term was to establish a good theoretical and practical understatement of what the project is about and outline the project requirements. Moreover, my work during the first term primary included an extensive research on prime numbers and the main encryption algorithms related to the project, specifying how to use them in a real environment.

The first term is also characterized with a lot of planning and time management. It was essential to me that I was able to fully plan the key aspects of the entire project from beginning to end, obviously a lot of changes had to be made over the initial plan, however those changes were not related to any of the core components of the project. A well organized and strict plan allowed me to be able to manage my time efficiently, mitigate any possible last minute issues and avoid any possible stress over deadlines.

Another very important tool I had to learn how to utilize were the supervisor meetings, which during the first 2 months were happening every two weeks. Moreover, this allowed me to implement a semi-scrum work methodology which I have learned about during my second year of the course. The meetings served as a clear indicator of progress made during a single sprint of two weeks and most importantly my advisor suggested me different approaches and features that I could include in my project.

Taking into consideration that the system is a combination of research and implementation, my priority during the first term was to understand the topic of the project as much as I can. Moreover, that included a large-scale analysis of what are the specific requirements that are necessary to achieve good quality in the end product. My research began with a brief introduction to modern cryptography, including all possible currently known techniques and principles most of which information was obtained from "Everyday Cryptography: Fundamental Principles and Applications**"** (Keith M. Martin, 2017). This was my first initial step in my planning process because I found it essential to know the basics of cryptography before I get started with the specific project requirements. Therefore, the next major milestone was to fully understand the topics: prime numbers, RSA and block ciphers. Moreover, those where the key fields that were required in the initial development of the two proof-of-concept programs during the first term. In addition to that, the first two proof-of-concept programs established the key functionalities of the end product: encryption and decryption of an input text using a combination of symmetric and asymmetric encryption. Furthermore, after developing the first two proof of concept programs I had to establish a chat user interface that at that stage was using a peer to peer connection between two parties, simply performing a chat between each other. Finally, to complete the early prototype that I had to present during the first term, both initially developed programs performing encryption and decryption had to be connected to the chat interface, resulting a fully working prototype that encrypts and decrypts messages sent between two users of the chat interface.

7.2 Term 2

Before the second term has started, following my plan that I created during the first term I managed to exploit the free time during the winter break. Moreover, I was able to complete my theoretical research and therefore finish the entire abstract part of the project, allowing me to focus only on the development of the final application.

The second term was definitely more challenging and required a set of new skills, apart from the ones, previously mentioned in the section about the first term. As already mentioned, most of the work done during the second term was concentrated on the development of the program. The process began with an analysis of the early prototype, focusing on the changes and improvements that can be done. The process of achieving the final system was represented by a lot of code restructure and refactoring. Further, the prototype was modified from using a peer-to-peer connection in to using a client/server one which gave the program a practical edge and also supported the authentication and data integrity mechanisms of the application by implementing a secure session key exchange procedure. Following that, security was also majorly improved by the substitution of the simple Feistel block cipher algorithm used in the prototype with a Java implementation of the Advanced Encryption Standard. In addition to that, the additional features implemented enforced the change of the initial code structure and organization of the prototype. Therefore, during the middle of the second term the final application structure and functionality was already established. Moreover, this was a key milestone in the development process, serving as a clear indicator of the progress made between the initial prototype developed during the first term and the end system. Finally, the conclusion of the second term required the major refactor of the code, regarding error handling, bad code smells or any unexpected errors that might occur. This closing process allowed me to polish the project and conclude the final version of the program.

The end product required a lot of planning, time management, research and developmental skills, all of which, stressing different aspects of the project. The tremendous knowledge gained from the project improved dramatically my skills as a developer and increased my confidence as an academic writer. Moreover, now being experienced with both team and individual projects, I feel much more confident about my future career.

Chapter 8: Conclusion

The effort behind achieving the end product required a lot of diverse skills and knowledge. Moreover, the program enforced that a vast analysis of the specification and requirements of the project had to be done before being able to begin. Therefore, two main tasks were outlined, the first task included an overall research on cryptography and the specific project related topics: prime numbers and the RSA cryptosystem, and the second one was regarding the development of a software that allows two parties to perform an encrypted live chat between each other.

The theoretical research of the project demanded the understatement of the basics of cryptography- from symmetric and asymmetric cryptography to modes of operation and data integrity. In addition to that the research had to mostly focus on the main topics of the project- prime numbers and the RSA cryptosystem. Moreover, mastering prime numbers, their generation, implementation and properties in combination with the proper interpretation of RSA key generation, encryption and decryption procedures was the first step of the development process of the end system. In addition to that, the other few crucial milestones in the expansion process included the creation of a user interface and the implementation of a block cipher to achieve efficient encryption. The whole development progress of the system was compliant with modern software engineering principles. Moreover, developing in object oriented fashion and implementing design patterns such as MVC and facade for the connectivity between user interface and functionalities. In addition to that, the extensive Junit testing of the core components of the system was more than necessary considering that this is a security based system and there is zero fault tolerance allowed. Followed by that, good code principles were maintained in terms of commenting the code in detail and preserving an adequate clean code style. To achieve maximum possible security the final product had to be tested against several well-known attacks that are often related to chat systems.

On the other hand, the supervisor meetings played an important role in terms of professionalism and served as progress meetings such as in a real business project environment, allowing me to implement the scrum methodology that I have learned about in the second year team project. In addition to that, the gradual progress made was supported by the diary and SVN version control system, summarizing the workflow and the evolution of the end product. Apart from what was just mentioned it is important to state that the success of the project wouldn't be possible without the proper planning and time management, especially when considering the fact that the scale and timeframe of the project.

Apart from the current state of the system, different extensions and features are planned to be implemented regarding the current interface design and most importantly the functionalities of the system. Different components are planned to be completed which would further enhance the security of the application and also dramatically improve the user experience of the system. For example, a set of test attacks are to be carried out in order to strengthen the current code and improve the security of the software against malicious parties. On the other hand, functionalities such as blocking communication from other users are also further to be implemented.

To finalize, the project allowed me to gain valuable experience both in academic and practical aspects regarding not just prime numbers and cryptosystems, but also stressing on crucial topics of modern cryptography, while strengthening my skills as a Java developer and an academic writer. In addition to that, the project gave me a first-hand experience of what an actual professional project would require from beginning to end. Moreover, the combination of skills and techniques learned, I consider being an extremely positive component in any further career involvement as an information security specialist.

Bibliography

[1] "A Classical Introduction to Modern Number Theory (2nd Ed.)" Kenneth Ireland; Michael Rosen (1990) [Book]. [Accessed 10 Mach 2018]

[2] "A Course in Number Theory and Cryptography (2nd Ed.)", Neal Koblitz (1987) [Book]. [Accessed 10 Mach 2018]

[3] "AES Class Documentation" (2018) [Online]. Available at: https://msdn.microsoft.com/en-us/library/system.security.cryptography.aes.aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-2 [Accessed 10 Mach 2018]

[4] "An Introduction to the Theory of Numbers (6th Ed.)", H. G. Hardy; M. E. Wright (2008) [Book]. [Accessed 10 Mach 2018]

[5] "An Overview of Modern Cryptography", Ahmed Al-Vahed; Haddad Sahhavi (2011) [Online]. Available at: https://pdfs.semanticscholar.org/46d6/fee601a4f89b448deff8af7fce9c52d68501.pdf [Accessed 10 Mach 2018]

[6] "Basic Components of Modern Cryptography" (2018) [Online]. Available at: https://technet.microsoft.com/en-us/library/cc962028.aspx [Accessed 10 Mach 2018]

[7] "BCS Code of Conduct", BCS (2018) [Online] Available at: http://www.bcs.org/upload/pdf/conduct.pdf

[8] "Biclique Cryptanalysis of the Full AES", Andrey Bogdanov; Dmitry Khovratovich; Christian Rechberger (2018) [Online]. Available at: https://www.webcitation.org/68GTcKdoD?url=http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf [Accessed 10 Mach 2018]

[9] "Big Data Breach Unmasks Bloomberg Chat Room Users", NY Post (2017) [Online] Available at: https://nypost.com/2017/08/04/data-breach-unmasks-bloomberg-terminal-chat-room-users/ [Accessed 10 Mach 2018]

[10] "BigInteger Class Documentation" (2018) [Online]. Available at: https://docs.oracle.com/Javase/7/docs/api/Java/math/BigInteger.html [Accessed 10 Mach 2018]

[11] "CS2800 - Software Engineering", D. Cohen (2012) [Online]. Available at: https://moodle.royalholloway.ac.uk [Accessed 10 Mach 2018]

[12] "Cryptographic Hash Function Diagram" (2018) [Online]. Available at: https://en.wikipedia.org/wiki/Cryptographic_hash_function [Accessed 10 Mach 2018]

[13] "Cryptography and Network Security: Principles and Practice", William Stallings (2010) [Book]. [Accessed 10 Mach 2018]

[14] "Cryptography Theory and Practice", Doug Stinson (1995) [Book]. [Accessed 10 Mach 2018]

[15] "Disquisitiones Arithemeticae (2nd Ed.)", Carl Gauss; Arthur A. Clarke (1986) [Book]. [Accessed 10 Mach 2018]

[16] "Elementary Number Theory", David Burton (2005) [Book]. [Accessed 10 Mach 2018]

[17] "Elementary Number Theory and its Applications (3rd Ed.)", H. Kenneth Rosen (1992) [Book]. [Accessed 10 Mach 2018]

[18] "Everyday Cryptography: Fundamental Principles and Applications", Keith M. Martin (2017) [Book]. [Accessed 10 Mach 2018]

[19] "Fundamental Data Structures", Josiang (2014) [Book]. [Accessed 10 Mach 2018]

[20] "Handbook of Applied Cryptography (5th Ed.)", Alfred J. Menezes; Scott A. Vanstone (2001) [Book]. [Accessed 10 Mach 2018]

[21] "How Secure Is RSA Algorithm" (2018) [Online]. Available at: http://www.herongyang.com/Cryptography/RSA-Algorithm-How-Secure-Is-RSA-Algorithm.html [Accessed 10 Mach 2018]

[22] "How RSA Works With Examples" (2012) [Online]. Available at: http://doctrina.org/How-RSA-Works-With-Examples.html [Accessed 10 Mach 2018]

[23] "Important Information about PGP & Encryption" (2018) [Online]. Available at: http://www.proliberty.com/references/pgp/ [Accessed 10 Mach 2018]

[24] "Introduction to Cryptography with Open-Source Software", Alasdair McAndrew (2016) [Book]. [Accessed 10 Mach 2018]

[25] "Introduction to Java Programming", Svetlin Nakov (2017) [Book]. [Accessed 10 Mach 2018]

[26] "Introduction to Java Programming (9th Ed.)", Y. Daniel Liang (2012) [Book]. [Accessed 10 Mach 2018]

[27] "Introduction to Modern Cryptography", Jon Katz; Y. Lindell (2007) [Book]. [Accessed 10 Mach 2018]

[28] "Introduction to Modern Cryptography", Mihir Bellare; Phillip Rogaway (2005) [Online]. Available at: http://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf [Accessed 10 Mach 2018]

[29] "IY2760 – Introduction to Information Security", K. Markantonakis (2017) [Online]. Available at: https://moodle.royalholloway.ac.uk [Accessed 10 Mach 2018]

[30] "IY3660 - Applications of Cryptography", K. Paterson (2017) [Online]. Available at: https://moodle.royalholloway.ac.uk [Accessed 10 Mach 2018]

[31] "Java How to Program (9th Ed.)", Paul Deitel; Harvey Deitel (2011) [Book]. [Accessed 10 Mach 2018]

[32] "Legal, Ethical, and Professional Issues in Information Security", Cengage (2018) [Online] Available at: https://www.cengage.com/resource_uploads/downloads/1111138214_259148.pdf [Accessed 10 Mach 2018]

[33] "Modern Cryptography" (2018) [Online]. Available at: https://www.tutorialspoint.com/cryptography/modern_cryptography.htm [Accessed 10 Mach 2018]

[34] "Modern Higher Algebra", Adrian A. Albert (2015) [Book]. [Accessed 10 Mach 2018]

[35] "Modular Arithmetic" (2018) [Online]. Available at: https://brilliant.org/wiki/modular-arithmetic/ [Accessed 10 Mach 2018]

[36] "Overview of Modern Cryptography", M. Kundalakesi; M. Phill; Sharmathi R. (2018) [Online]. Available at: http://ijcsit.com/docs/Volume%206/vol6issue01/ijcsit2015060175.pdf [Accessed 10 Mach 2018]

(37] "Practical Cryptography", Niels Fergusen; Bruce Schneier (2003) [Book]. [Accessed 10 Mach 2018]

[38] "Prime Factorization" (2018) [Online]. Available at: https://learncryptography.com/mathematics/prime-factorization [Accessed 10 Mach 2018]

[39] "Prime Number Theory" (2018) [Online]. Available at: https://bg.khanacademy.org/computing/computer-science/cryptography/comp-number-theory/a/comp-number-theory-introduction [Accessed 10 Mach 2018]

[40] "Prime Numbers", Margaret Rouse (2018) [Online]. Available at: http://whatis.techtarget.com/definition/prime-number [Accessed 10 Mach 2018]

[41] "Prime Numbers and Computer Methods for Factorization", Hans Riesel (1994) [Book]. [Accessed 10 Mach 2018]

[42] "Primes is in P", Manindra Agrawal; Neeraj Kayal; Nitin Saxena (2004) [Online]. Available at: http://annals.math.princeton.edu/wp-content/uploads/annals-v160-n2-p12.pdf [Accessed 10 Mach 2018]

[43] "Public-Key Cryptography: Theory and Practice", Abhijit Das; Veni Madhavan (2009) [Book]. [Accessed 10 Mach 2018]

[44] "RSA and Fermat's Little Theorem Relation" (2011) [Online]. Available at: https://crypto.stackexchange.com/questions/388/what-is-the-relation-between-rsa-fermats-little-theorem [Accessed 10 Mach 2018]

[45] "RSA Attacks", Abdulaziz; Alrasheed; Fatima (2018) [Online]. Available at: https://www.utc.edu/center-information-security-assurance/pdfs/course-paper-5600-rsa.pdf [Accessed 10 Mach 2018]

[46] "RSA and Quantum Computers" (2017) [Online]. Available at: https://www.quantamagazine.org/why-quantum-computers-might-not-break-cryptography-20170515/ [Accessed 10 Mach 2018]

[47] "RSA Key Lengths", Neil Coffey (2012) [Online]. Available at: https://www.Javamex.com/tutorials/cryptography/rsa_key_length.shtml [Accessed 10 Mach 2018]

[48] "The Design of Rijndael", Daemen; Rijmen (2000) [Book]. [Accessed 10 Mach 2018]

[49] "The Laws of Cryptography with Java Code", Neal R. Wagner (2003) [Book]. [Accessed 10 Mach 2018]

[50] "The Math behind RSA" (2002) [Online]. Available at: https://math.berkeley.edu/~kpmann/encryption.pdf [Accessed 10 Mach 2018]

[51] "The Mathematics of the RSA Public-Key Cryptosystem", Burt Kaliski (2000) [Online]. Available at: http://www.mathaware.org/mam/06/Kaliski.pdf [Accessed 10 Mach 2018]

[52] "The RSA Cryptosystem: History, Algorithm, Primes", Michael Calderbank (2007) [Online]. Available at: http://www.math.uchicago.edu/~may/VIGRE/VIGRE2007/REUPapers/FINALAPP/Calderbank.pdf [Accessed 10 Mach 2018]

[53] "Unbalanced Feistel Networks and Block Cipher Design", Bruce Schneier; John Kelsey (1996) [Book]. [Accessed 10 Mach 2018]