Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Bachelor Thesis

Encryption Algorithms

Babisha Shrestha

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

abs. v. š. Babisha Shrestha

Systems Engineering and Informatics
Informatics

Thesis title

Encryption algorithms

Objectives of thesis

The thesis focuses on importance of encryption in information security to protect data. The main goal of the thesis is to describe commonly used encryption algorithms and scenarios when these algorithms are used in data security. The supporting goal is to implement a prototype application which will demonstrate practical use case of selected encryption algorithm.

Methodology

Methodology of this thesis is based on analysis and study of various information sources, with special emphasis on cryptography and encryption algorithms. Based on synthesis of gained knowledge a prototype application for demonstrating the properties and typical use case of selected algorithm will be developed using common software development methods.

The proposed extent of the thesis

40-50 pages

Keywords

30-50 pages

Recommended information sources

LOSHIN, Peter. Simple steps to data encryption: a practical guide to secure computing. Amsterdam: Elsevier Science, 2013. ISBN 978-0-12-411483-8.

PAAR, Christof a Jan PELZL. Understanding cryptography: a textbook for students and practitioners. New York: Springer, 2010. ISBN 978-3-642-04100-6.

SALOMON, D. Data privacy and security: Encryption and Information Hiding. New York: Springer, 2003. ISBN 03-870-0311-8.

Expected date of thesis defence

2019/20 WS – FEM (February 2020)

The Bachelor Thesis Supervisor

Ing. Petr Hanzlík, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 25. 11. 2019

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 25. 11. 2019

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 14. 03. 2021

Declaration

I declare that I have worked on my bachelor thesis titled "Encryption Algorithms" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on 15.03.2021 _____

Acknowledgement

I would like to thank Ing. Petr Hanzlík, Ph.D. for his advice and support during my work on this thesis.

Encryption Algorithms

Abstract

The thesis describes the importance of cryptography to protect data, history of cryptology, cryptoanalysis, goals of cryptography, symmetric and asymmetric encryption algorithms. The focus of the thesis is on the AES and the RSA encryption algorithms, explaining how they work in more detail.

Based on the literature review, a prototype application is developed using a hybrid encryption and decryption method that combines both AES and RSA algorithms together to compensate for their own weaknesses and achieve better security.

The design of the prototype application is illustrated using data flow diagrams and code snippets, which describe the main stages of program creation. The security that the developed prototype application provides is then discussed mainly in relation to the main goals of cryptography.

Keywords:

Cryptology, Cryptoanalysis, Cryptography, Encryption algorithm, Hybrid encryption algorithm, Symmetric algorithm, Asymmetric algorithm, AES, RSA, Public key, Private key.

Šifrovací algoritmy

Abstrakt

Práce popisuje význam kryptografie pro ochranu dat, historii kryptologie, kryptoanalýzu, cíle kryptografie, symetrické a asymetrické šifrovací algoritmy. Práce se zaměřuje na šifrovací algoritmy AES a RSA a podrobněji vysvětlují jejich fungování.
Na základě literární rešerše byla vyvinuta prototypová aplikace využívající hybridní metodu šifrování a dešifrování, která kombinuje algoritmy AES a RSA tak, aby kompenzovala jejich vlastní slabiny a dosáhla vyšší bezpečnosti.
Návrh prototypové aplikace je ilustrován pomocí diagramů datových toků a fragmentů kódu, které popisují hlavní fáze tvorby programu. Zabezpečení, které poskytuje vyvinutá prototypová aplikace, je diskutováno zejména ve vztahu k hlavním cílům kryptografie.

Klíčová slova: Kryptologie, kryptoanalýza, kryptografie, šifrovací algoritmus, hybridní šifrovací algoritmus, symetrický algoritmus, asymetrický algoritmus, AES, RSA, veřejný klíč, soukromý klíč.

Table of content

List of pictures

List of tables

List of abbreviations

DES - Data Encryption Standard
3DES/3DEA- Triple Data Encryption Standard / Triple Data Encryption Algorithm
AES - Advanced Encryption Standard
ECB - Electronic Code Book
CBC - Cipher Block Chaining
CFB -Cipher Feedback
OFB- Output Feedback
XOR- Exclusive OR Operation
Nb-Block Length
Nk- Key Length
Nr- Number of Rounds
W0- First column
NSA- National Security Agency
NIST- National Institute of Standards and Technology
S-Box- Substitution box

RSA- Rivest-Shamir-Adelman algorithm
Kpub,kpri - Public key, Private key
Gcd- Greatest common divisor
SSL/TLS-Secure Socket Layer/Trasport Layer Security
HTTPS- HypterText Transfer Protocol Secure

1 Introduction

Computer technology is growing faster in today's world and has become essential to our daily lives. People are using computers in their personal and professional lives. Computer technology is widely used in business, personal work, administrative work, business strategy, client personal data, banking transactions, etc. Before the time of computer technology, people used to store information on paper. The sensitive data was securely stored through physical protection using security doors, fences, security officers, surveillance cameras, and vaults. When you lock the door, an intruder may still break in, but they will need more effort to break the lock. Nowadays the same happens in the digital technology world. Intruders can effortlessly get access to the data if it remains available without any protection. Cryptography is like the lock on the door of the digital technology. With the encryption algorithms, intruders can't read the information even when they have access to it, if they don't know the encryption key. People feel more secure when they lock the door with the key rather than leaving it open.

For many years, cryptography was mainly used for government and military purposes. Cryptography was used to transfer secret information during wars. Later, with the development of digital technology, computer cryptography became widely used not only for government or military purposes but also for commercial and personal purposes.

People started using cryptography on the Internet, for example, SSL (Secure Socket Layer/TLS (Trasport Layer Security) is used to provide a secure connection between the client and the webserver. SSL/TLS connection is indicated by a padlock icon or HTTPS is displayed in the web address area in your browser instead of HTTP. Moreover, cryptography is used in wireless local area networks like WPA/WPA2 PSK, mobile telecommunications like GSM, payment transactions, video broadcasting like Pay-TV, identity cards, for example, Belgian eID card, in TOR to provide anonymous communication on the Internet, in digital currency like bitcoin, etc.(Keith, 2017)

The use of personal computers, laptops, smartphones, and tablets is increasing day by day. Many of us possess multiple devices to do our day-to-day activities, and the personal data we store and transfer every day has also increased. This leads to high risk of security concern. Encryption algorithms provide security on data when it is stored or transferred.

2 Objectives and Methodology

2.1 Objectives

The thesis focuses on importance of encryption in information security to protect data. The main goal of the thesis is to describe commonly used encryption algorithms and scenarios when these algorithms are used in data security. The supporting goal is to implement a prototype application which will demonstrate practical use case of selected encryption algorithm.

2.2 Methodology

Methodology of this thesis is based on analysis and study of various information sources, with special emphasis on cryptography and encryption algorithms. Based on synthesis of gained knowledge a prototype application for demonstrating the properties and typical use case of selected algorithm will be developed using common software development methods.

3 Literature Review

3.1 Overview of Cryptology

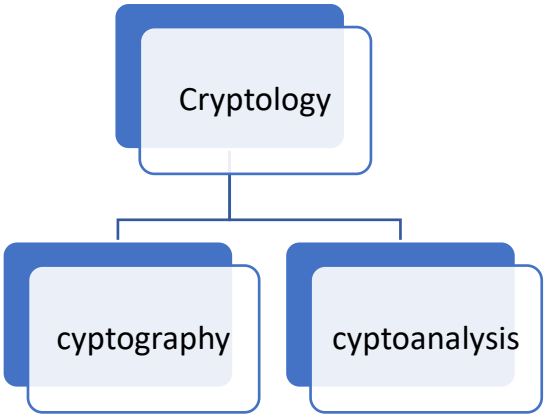

Figure 1.Classification of Cryptology (author)

Cryptology is the art and science of secret writing and breaking ciphers. It is divided into two branches(Paar and Pelzl, 2010):

1. Cryptography – the science of writing ciphers.
2. Cryptoanalysis - the science of breaking cryptosystems.

3.2 Cryptanalysis

Cryptoanalysis is the process of breaking a cryptosystem. It is a technique to decrypt the ciphertext without having access to the key used to encrypt the plaintext. It is a crucial part of the modern cryptosystem which helps to understand and create more secure algorithms.

The basic theory of cryptoanalysis was expressed by Auguste Kerchoffs, in the 19th century, which is also known as Kerchoff's principle. It assumed that all of the details of the cryptosystem along with its algorithms and their implementation are known to the antagonist, and therefore, the security of the cryptosystem should be built on secret keys.(Delfs and Knebl, 2007)

There have been many cryptography attacks over the years on the cryptographic algorithms. Attacks can be classified as passive attacks, where attackers only observe the communication

channels, threatening data confidentiality, and active attacks, where attackers try to delete, insert, or alter the transmission on the channels, affecting data confidentiality, authentication and integrity. Cryptanalytic attacks depend on the types of algorithms and characteristics of the plaintext. (Menezes, Oorschot and Vanstone, 1996)

There are many possible attack approaches, which are described below(Menezes, Oorschot and Vanstone, 1996):

1. ***Known Plaintext Analysis***.

   The attackers know some combination of plaintext and ciphertext that was already used and based on this knowledge they try to analyse the encryption key to decrypt the messages.

2. ***Chosen Plaintext Analysis***.

   Chosen plaintext analysis assures that the attackers can take the random plaintext to be encrypted and obtain the corresponding ciphertext and try to get the key. This attack can expose the secret information after calculating the secret key.

3. ***Ciphertext Only Analysis***.

   The attackers try to get the decryption key and plaintext by only observing the ciphertext. The encryption algorithms that fail to resist this attack are considered to be totally insecure.

4. ***Chosen ciphertext attack***.

   Attackers select the ciphertext and obtain the corresponding plaintext. To establish such an attack the intruder needs to get access to the tools used for decryption, but the main motive behind such attack is to later obtain the plaintext from a different ciphertext, when no longer having the access to the tools of decryption.

5. ***An Adaptive Chosen Plaintext Analysis***.

   It is a Chosen Plaintext Analysis where the choice of the plaintext depends on the obtained ciphertext from the earlier requests.

6. ***An Adaptive Chosen-ciphertext attack***.

   It is a Chosen Ciphertext Attack where the choice of the ciphertext depended on the plaintext obtained from the earlier requests.

3.3 Cryptography

The art of protecting information from readable format to unreadable format is called cryptography. In other words, it is a process of hiding confidential information from third parties so that they cannot get access to it. It is widely used in commercial sectors like banking, e-commerce, and many more.

In the ancient period, people used cryptography to write secret messages. The word cryptography came from the Greek words ‘Krypto’, means ‘hidden’, and ‘graphene’, means ‘writing’. Many people believe cryptography was born along with the art of writing. With the evolution of civilization, human beings created groups, tribes, kingdoms. This started the idea of power, battles, and politics. Because of that, there was a need to communicate in a secret way which helped to the development of cryptography over the times. Non-standard hieroglyphics were used by Egyptian scribes around 4000 years ago which was the first evidence of the use of cryptography. They communicated by messages written in hieroglyphs, and only scribes were aware of the secret key, who used to transfer the messages on behalf of their kings.(*Cryptography - Quick Guide - Tutorialspoint*, no date)

The earlier Roman cryptography system was known as the Caesar cipher, it is a monoalphabetic cipher with the idea of writing the letter by shifting the letter by the accepted number and then shifting the letter back by the same number to decode the message. This encryption method was simple to crack by comparing the beginning of the alphabet to each subsequent letter. Another way to break it was using the frequency analysis method. This method uses the idea that some letters in the English alphabet are repeated more often than others. Thus, a person could go through the message and look for repeated letters and try to replace them with often used letters. (*A Brief History of Cryptography - Inquiries Journal*, no date)

In the sixteenth century Vigenere constructed a cipher, which was probably the first cipher that used an encryption key. This cipher used multiple Ceaser chipers based on a key, and this cipher used each letter of the key to decide which Ceaser cipher shift to use. When all of the letters of the key had been used the cycle began again from the first letter of the key.Vigenere cipher was secure for 300 year until Kasiski and Kerchoff discovered methods

to break it. These methods were based on key and language repietitiveness. With the developenment of computer technology this cipher became even easier to crack and is not secure for today's standards.(Wilson and Garcia, 2006)

The first electromechanical cipher machine, called Mark I, was invented in 1915 by Edward Hebron. Two electric typewriters were connected by 26 cables that were used for monoalphabetic letter substitution. At first, the machine used only one rotor that scrambled typed letters, but later he redesigned it to use up to five rotors. The rotors rotate one or more positions as you give input and produce different substitutions of letters. Edward Hebron set up the first cipher machine company in the U.S. mainly targetting U.S. Army and Navy markets. In 1923 an encrypted message on this machine was broken by Herbert Yardley, who worked as a cryptographic clerk for the cryptologic bureau of the Navy. A year later Edwards company went bankrupt. Hebron made few attempts to restore his business but it was shut down for good after cancelation of his contract with the Navy in 1934.(Dooley, 2018)

In 1918 the Enigma machine was invented by the German engineer Arthur Scherbius. Its modified version was accepted by the German Navy in 1926 and two years later the it was accepted by the German Army. The machine was widely used by the Germans during World War II. It was an electromechanical cipher machine capable of both encryption and decryption. During its use by the Germans the number of the rotors the machine used has changed a few times. Originally, the machine only used three rotors in fixed positions, later the number of the rotors increased to five, where three rotors could be placed in any position, and by the end of the war the machine placed four rotors at a time. The Enigma had a weakness because it prevented letters from encrypting to themselves. In 1932 and 1933 Polish cryptanalysts made a breakthrough that enabled them to decrypt some of the messages of the German Army version of Enigma. It was the first analytical breach of the Enigma. In 1938 and 1939 Germans modified it and made the Polish solution unworkable.Later, Alan Turing played a crucial role in making another breakthrough in decrypting the code of an enigma machine.(Dooley, 2018)

3.3.1 Modern cryptography

Modern cryptography depends on various theories such as number theroy, computational-complexity, concepts of probability theories. It is the foundation of computer and communication security. The security of modern cryptography relies on publicly known algorithms. Therefore, even if someone knows the algorithms, they can not decrypt a particular message without having access to a secret key. (*Cryptography - Quick Guide - Tutorialspoint*, no date)

Modern cryptography is divided into two main parts, the symmetrical algorithm, where the sender and the receiver use the only key to encrypt and decrypt the data, for example, DES, AES, 3DES and the asymmetrical algorithm, where the sender uses the public key to encrypt the data, while the receiver uses the private key to decrypt data, for example, RSA, DH, DSA. Moreover, the symmetric algorithm is split into the block cipher and stream cipher which are shown in the Figure 2 below.

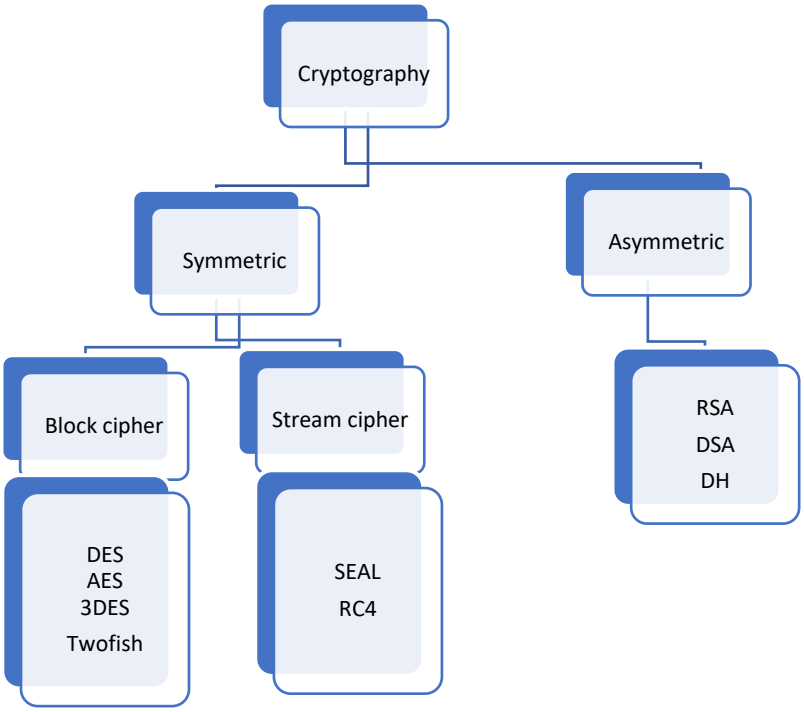

Figure 2. Classification of Cryptography (author)

3.3.2 Goals of modern cryptography

The essential goals of modern cryptography system are (Keith, 2017):

- ***Confidentiality***: It is a common aspect of information security. It provides confidentiality using encryption methods and prevents the third person to view and obtain the data. It allows authorized users to access sensitive and protected data. An unauthorized person could get access to the data sent over the internet, hence, various encryption algorithms have been implemented, so that even if the intruder got access to the data, he/she would not be able to read it without the private key.

- ***Integrity***: It makes sure the data has not been modified during transit, retrieval, or in storage by unauthorised way. The failure of integrity can lead to someone modifying data in-store or in transit. It provides the method to detect if data has been modified in an unauthorized way. For example, hash function.

- ***Authentication***: Data authentication helps receivers to know if data is sent from the actual users. It is mentioned as message authentication sometimes, since its primary purpose is to authenticate the data and not who we are communicating with at the time we get the data.

- ***Non-repudiation***: It prevents sender from denying participation in all the or part of the communication. Non-repudiation ensures that the sender cannot deny the fact that he/she sent the data. Cryptography can ensure non-repudiation using a digital signature, digital certificate, and public key infrastructure.

3.4 Symmetric Algorithm

The symmetric algorithm is also known as a secret key algorithm. It uses only one key to encrypt and decrypt the data. The sender uses the key to encrypt the data, and the receiver uses the same key to decrypt the data. The symmetric algorithm is very secure and fast. When it comes to encrypting and decrypting big data, the symmetric algorithm is more

suitable. Some of the widely used examples of symmetric algorithms are DES, 3DES, AES, Twofish, Serpent.

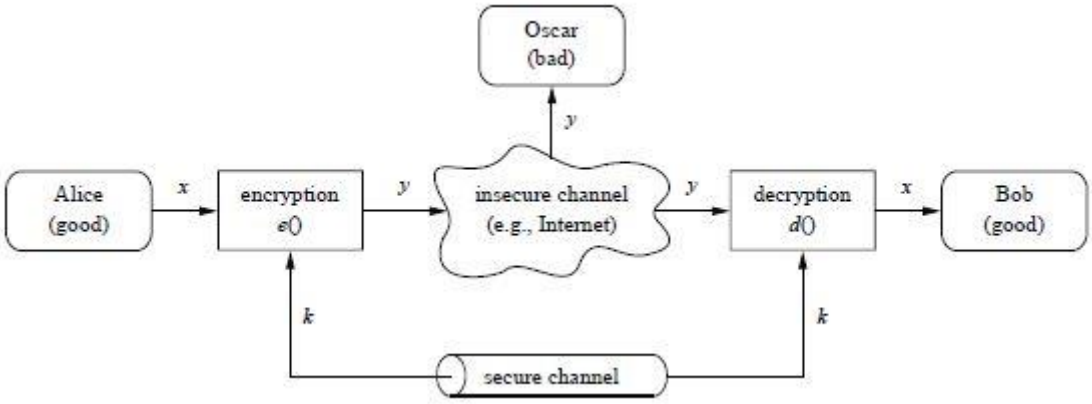

Figure 3. Symmetric Algorithm (Paar and Pelzl, 2010)

According to the Figure above, we have Alice, Bob, and Oscar. If Alice wants to communicate with Bob using insecure channels, Oscar might get access to the data. Therefore, Alice and Bob need to communicate securely through the channels. The symmetric algorithm is one of the solutions to this problem. Using the key ***k***, Alice can encrypt the plaintext denoted by ***x*** to ciphertext denoted by ***y***, and then Bob can decrypt the ciphertext ***y*** to get the plaintext ***x*** using the same key. Both encryption and decryption algorithms are generally publicly known. Therefore, if Oscar got to know the secret key, then he could easily get access to the data. If the algorithm was keept secret, then such encryption system would be harder to break, but with the untested algorithm, it would be difficult to find out whether the algorithm is secure or not. To test it, we need to make the algorithm public for cryptoanalysis from different cryptographers.(Paar and Pelzl, 2010)

The symmetric algorithm is divided into two categories, which are block cipher and stream cipher. The difference between a block cipher and a stream cipher is described in Table 1 below.

Block Cipher	Stream Cipher
• It encrypts and decrypts one block of plaintext at a time.	• It encrypts or decrypts one bit of plaintext at a time.
• Block ciphers use confusion (encryption operation where the relation between key and ciphertext is hidden) like substitution in both AES and DES. Diffusion (one plaintext is spread over many ciphertexts for the purpose of hiding statistical properties of plaintext) for instance in DES uses bit permutation and AES uses more Advanced Mix column(Paar and Pelzl, 2010)	• Stream cipher only uses confusion.
• Block Ciphers are versatile (not only used for encryption also used for MACs and hash functions) • It is compatible, which means it is used in many applications. • It is adaptive and is implemented in a different modes of operations to obtain different properties.(Keith, 2017)	• no error distribution: It encrypts the data 1 bit at a time, therefore, error on 1-bit transformation effect only 1 bit of plaintext. It is famous for the protection of mobile communications. • Less versatile: it is normally used for only encryption.(Keith, 2017)
• Block cipher is slower than stream cipher.	• Stream cipher is fast to operate, which makes it more suitable for real time encryption of data. For example, Mobile Telecommunication.
• Most of the block ciphers have a block size of 64 bits like DES, TDES or 128 bits like AES.	• 8 bits at a time is converted into stream cipher.
• It uses Electronic Code Book (ECB), Cipher Block Chaining (CBC) algorithm modes.	• It uses algorithm modes like Cipher Feedback (CFB), Output Feedback (OFB)
• Block ciphers are more popular in most domains such as internet security.	• Stream cipher requires fewer resources for implementation. The cell phone is a relevant environment for a stream cipher.
• DES, AES, TDES, TwoFish	• RC4, Vernam One Time Pad, A5/1,

Table 1. Difference between block cipher and stream cipher

3.4.1 DES (Data Encyption Standard)

A small radical change happened in cryptography in 1972. There was a request for a proposal for a standardized encryption algorithm in the USA by NBS (National Bureau of Standards) also known as the National Institute of Standards and Technology (NIST) nowadays. The concept of standardized encryption was to find a secure and single algorithm that could be used in differrent commercial applications, such as banking. In 1974, NBS received a candidate from IBM who could fulfil the requirement. IBM had proposed an algorithm that was based on the Lucifer algorithm, which was earlier developed by Horst Feistel and was the first example of a block cipher on digital data. Lucifer encrypted blocks of 64 bits of data using a 128 bits key.

The proposed algorithm was designed to survive different cryptanalysis, which was unknown to the public until 1990. To investigate the security of a submitted encryption algorithm, NBA requested help from NSA (National Security Agency). It was unclear whether IBM had gained knowledge about cryptanalysis themselves or they were influenced by the NSA that convinced IBM to reduce the key length from 128 bits to 56 bits, which showed weaker encryption to brute force attack. The NSA contribution worried some people that a mystery secret entrance, i.e., a numerical property with which DES could be broken was known to NSA. There was a major complaint on the decrease of the key estimate. Some people evaluated that NSA would be able to look through the key of 2^56 and break it, utilizing brute force attack, but these stresses turned out to be aimless.

Despite all the feedback and worries, in 1977, the NBS has published a modified version of IMB encryption as a Data Encryption Standard to the public. The cipher was executed at the bit level. In the standard, the substitution boxes were never officially released. In the 1980s, the rapid growth of the use of personal computers made all the specifications of DES to be freely accessible and it became easier to analyze its internal structure. By that time, many researchers had done various researches and went through DES security but they did not find any severe weaknesses in the algorithm. It was the most used encryption algorithm until the year 1999, and finally, AES replaced it.

DES is a symmetric algorithm that uses one secret key to encrypt and decrypt the data. It is a block cipher with 64 bits of plaintext, and the size of the key is 56 bits. It has iterative

algorithm encryption of 16 rounds, which all perform a similar operation but different subkeys are used in every round. It uses Feistel's cipher. It can be a strong cipher if it is designed carefully. Feistel's cipher for encryption and decryption uses a similar operation.

Talking about security, the DES key size was not large enough, and therefore, was vulnerable against brute force attacks. Also, because the substitution boxes were never released, only the designers of the algorithm would know if there existed an analytical attack that could exploit the mathematical properties of the substitution boxes. Despite the serious cryptanalysis of DES, there were not very successful attacks over the lifespan of DES. Nonetheless, with exhaustive key-search attacks the DES can be easily broken, therefore, for most applications DES is not appropriate anymore.(Paar and Pelzl, 2010)

3.4.2 Triple DES

Triple DES, also known as TDEA (Triple Data Encryption algorithm), was developed to overcome the weaknesses of DES without re-designing the whole cryptosystem. It was widely used in the different financial sectors, biometric and payment systems. It uses the three DES keys of 56 bits in size, which are **k1**, **k2**, **k3**, and all three keys are referred to as a bundle. Triple DES is a symmetric key block cipher that implements the triplicate DES cipher by encrypting the first key **k1**, decrypting the second key **k2**, and encrypting the third key **k3**. Decryption of the Triple DES consists of the reverse process, first it uses **k3** to decrypt the data, then **k2** to encrypt the data, and **k1** to decrypt the data.(Keith, 2017)

NIST first began to devalue 64-bit 3DES when it showed serious security vulnerabilities against the cryptoanalytic attacks made by Karthikeyan Bhargavan and Gaëtan Leurent, proving great vulnerabilities for protocols like TLS, IPsec, and HTTPS. It was also known that when 2^32 blocks were encrypted with the single key bundle, the ciphertext collision would likely occur. These security flaws in the implementation of the TDEA influenced the necessity for the 128-bit block cipher. As a result NIST planned to reduce the maximum amount of plaintext allowed to encrypt under a single 3DEA key bundle from 2^32 to 2^20 blocks and urged all TDEA users to move to AES as soon as possible. (*Update to Current Use and Deprecation of TDEA | CSRC*, no date)

TDEA comes with two variations: 2 key TDEA and 3 key TDEA. 3 key TDEA is stronger than 2 key TDEA. The latest version already disallows the use of 2 key TDEA for cryptography protection. NIST has restricted the 3 key TDEA to apply no more than 2^20 blocks using a single key bundle. NIST has released the draft proposal of encryption 3DES deprecated through December 31, 2023, using the approved encryption modes.(Barker and Roginsky, 2019)

Algorithms	**Status**
Two key 3DES encryption	Disallowed (algorithm and key are no longer accept)
Two key 3DES decryption	Legacy use (algorithm and key can use only to process alreadyprotected information)
Three key 3DES encryption	Deprecated(alorithm and key may use but user has to take some security risk) through 2023 Disallowed after 2023
Three key 3DES decrytion	Legacy use

Table 2. Status of TDEA (Barker and Roginsky, 2019)

3.4.3 AES (Advanced Encryption Standard)

It is mostly used for symmetric encryption nowadays. The term “Standard” refers to US government applications. AES is used in many commercial standards. For instance, Internet Security standard IPsec, TLS, WiFi encryption standard 802.11i, Shell Network protocol (SSH) Secure Shell.

In 1997 NIST requested a proposal with requirements to block cipher, 128 bits of block size, key lengths must support 128, 192, 256 bits, and the algorithm must be efficient with software and hardware. In 1998, there were 15 candidates submitted different algorithms from different countries. NIST announced five finalists among them after the selection process of algorithms in the year 1999. They were **Mars**, by **IBM**

cooperation, **RC6** by **RSA laboratories**, **Rijndael**, by **Joan Daemen and Vincent Rijmen**, **Serpent**, by **Ross Anderson, Eli Biham and Lars Knudsen**, and **Twofish**, by **Bruce Schneier, John Kelsey, Dough Whiting, David Wagner, Chris Hall, and Niels Ferguson**.

Eventually, in 2000 NIST announced that they had chosen Rijendael as AES, and it was accepted by the US federal standard and published as a final standard (FIPS PUB 197) in 2001. In 2003, The US National Security Agency (NSA) announced that it allows AES to encrypt classified documents up to the level secret for all key lengths and up to the top-secret for the key of either 192 or 256 bits.(Paar and Pelzl, 2010)

Outline of AES

The design of the AES is referred to as a substitution-permutation network that means it relies on a series of connections or operations like replacing input with specific output, shifting bits around. AES does all the computations on bytes. The AES encryption is dependent on series of lookup tables and XOR operations, which are very fast to perform on a computer.(Keith, 2017)

Encryption is the process that converts plaintext into ciphertext, and decryption is an inverse of encryption where we convert cipher to plaintext. AES is a symmetric algorithm. It is a widely used block cipher which encrypts 128 bits of data at a time. It treats 16 bytes as a 4 by 4 matrix. Data that are longer than 128 bits are broken into blocks of 128 bits. If the data are not divisible by the block length, then the padding will be added. AES encryption is a process of encrypting the plaintext using a secret key and getting the ciphertext, and AES decryption is a process of using the same secret key to decrypt the ciphertext into original plaintext.

State

It is a two-dimensional array of bytes having four rows and Nb (number of columns, or number of 32-bit words). State is the output from the XOR operations between plaintext and key.

Input and output

In AES, input and output consist of sequences of 128 bits which are also called blocks. AES standard only calls for a block size of 128 bits which is represented by Nb=4, where Nb is a number of 32-bit words or a number of columns in the state.

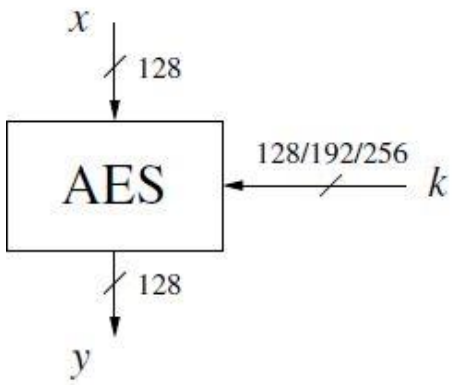

Figure 4. Input/output parameters (Paar and Pelzl, 2010)

Key length

The length of the cipher key in AES is 128, 192, or 256 bits. The length of the key is represented by Nk = 4, 6, 8, where Nk is the number of 32 bits words in the cipher key. The size of the key determines how many rounds are to be performed. If Nk=4 then Nr=10, if Nk=6 then Nr=12, Nk=8 then Nr=14, where Nr is the number of rounds. (*Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES)*, 2001)

No of bits in key	Key length (in words) Nk	Key rounds Nr	Block length (in words) Nb
128	4	10	4
192	6	12	4
256	8	14	4

Table 3. Combination of keys, blocks and rounds (authors work), inspired by (*Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES)*, 2001)

Figure 5 indicates different stages during the AES encryption process. Firstly, we have **keys** and the **key expansion** process, which will be discussed below. **Key expansion** and **Add Round key** are the initial steps in AES. After that, a series of rounds are performed, which is also known as transformation. The number of rounds depends on the size of the key. There will be four transformation processes in each round, which are **Sub Bytes**, **Shift Rows**, **Mix Columns**, and **Add Round Key**, but there won't be **Mix Columns** transformation in the final round.

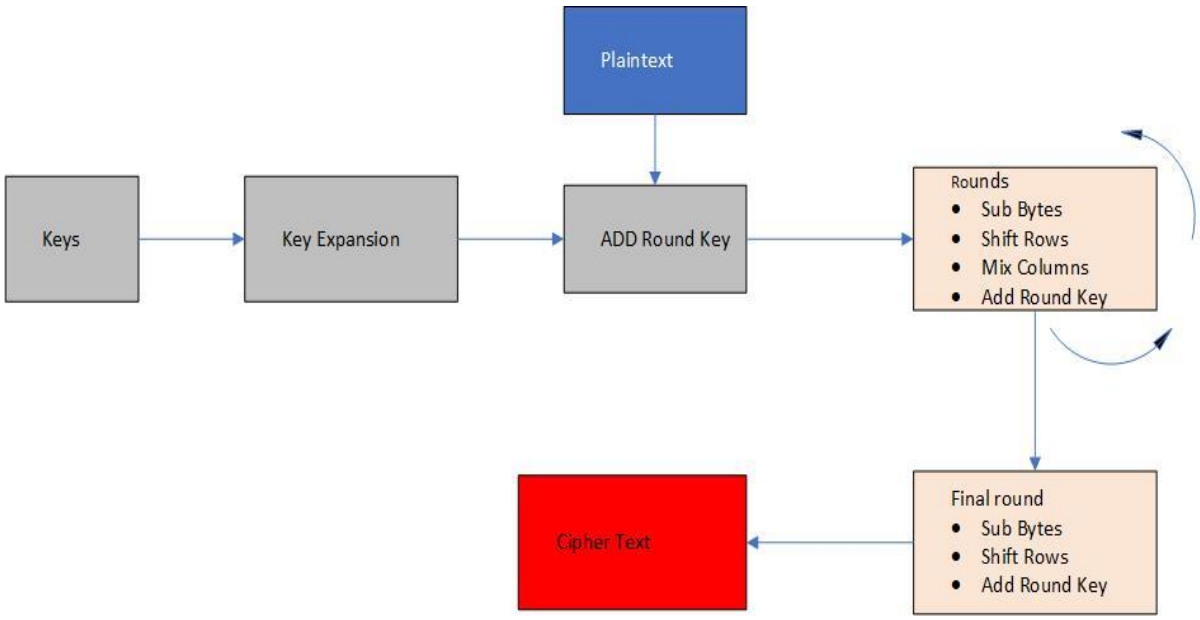

Figure 5. Stages in AES (author)

AES Key Expansion

We have one original key, and each round performs modification of a key, which is called Key Expansion. It switches the key in such a way that it is different for each round.

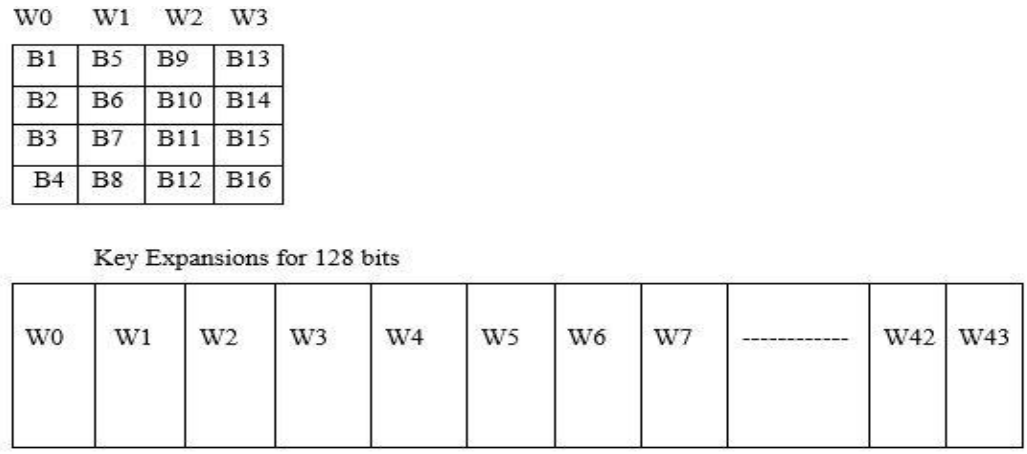

Figure 6. AES key Expansion (author)

Let us consider we have the key in the form of matrix 4*4. The first column is called a word (1 word = 4 bytes), represented by **W0**, the second column is represented by **W1**, the third is represented by **W2**, and the fourth is represented by **W3**, respectively. For 128 bits size of the key, the number of rounds performed is 10. Using **W0**, **W1**, **W2**, **W3**, which is subkey 0 (original key), we can generate **W4**, **W5**, **W6**, **W7**, which is subkey1, and so on. We follow the procedure below to expand the key.

Firstly, to get **W4**, we can transfer **W3** to **g**, which represents the complex function, and XOR it with **W0**. Once we obtain **W4**, we can XOR it with **W1** to get **W5**, and so on.

W4 = W0 XOR g(W3)

W5= W1 XOR W4

W6= W2 XOR W4

W7= W3 XOR W5 and so on.

We calculate g using the following process.

W3	Rot word(x1)	Sub word (y1)	y1 XOR rcon=g(W3)
b13	b14	b'14	g(B13)
b14	b15	b'15	g(B14)
b15	b16	b'16	g(B15)
b16	b13	b'13	g(B16)

Table 4. Process to calculate g (author)

The rot word performs a one-byte circular shift to the left, where the subword performs a byte substitution on each byte of its input word using s-box, and finally, we do XOR operation with y1 and r-con (round constant), to get the result of **g (w3)**.

AES Encryption Process

According to the Figure 7, the plain text is divided into a block of 128 bits, and there are 128 bits of subkey 0, which is also the original key. We XORed plaintext with subkey 0, and this operation is called Add Round Key. The result state array is then sent to the **round 1**, which is also called the transformation process. There are different processes within the rounds: **Sub bytes**, **Shift Rows**, **Mix Columns**, and **Add Round Key**.
The Key Expansion process uses the subkey 0 to generate subkey 1, and subkey 2 is then generated from subkey 1, and so on.

The output of the initial step moves to round 1 and performs all the operations and uses subkey 1 for the Add Round Key, which is transferred to round 2, and then round 2 performs similar methods and uses subkey2 for Add Round Key and so on. There will be ten rounds for 128 bits of the size of the key, the process will repeat ten times, and finally, we get the output as a ciphertext.

Figure 7 demonstrates the process for encryption of 128 bits block of plaintext using the 128 bits of the key. The rounds vary, depending on the size of the key.

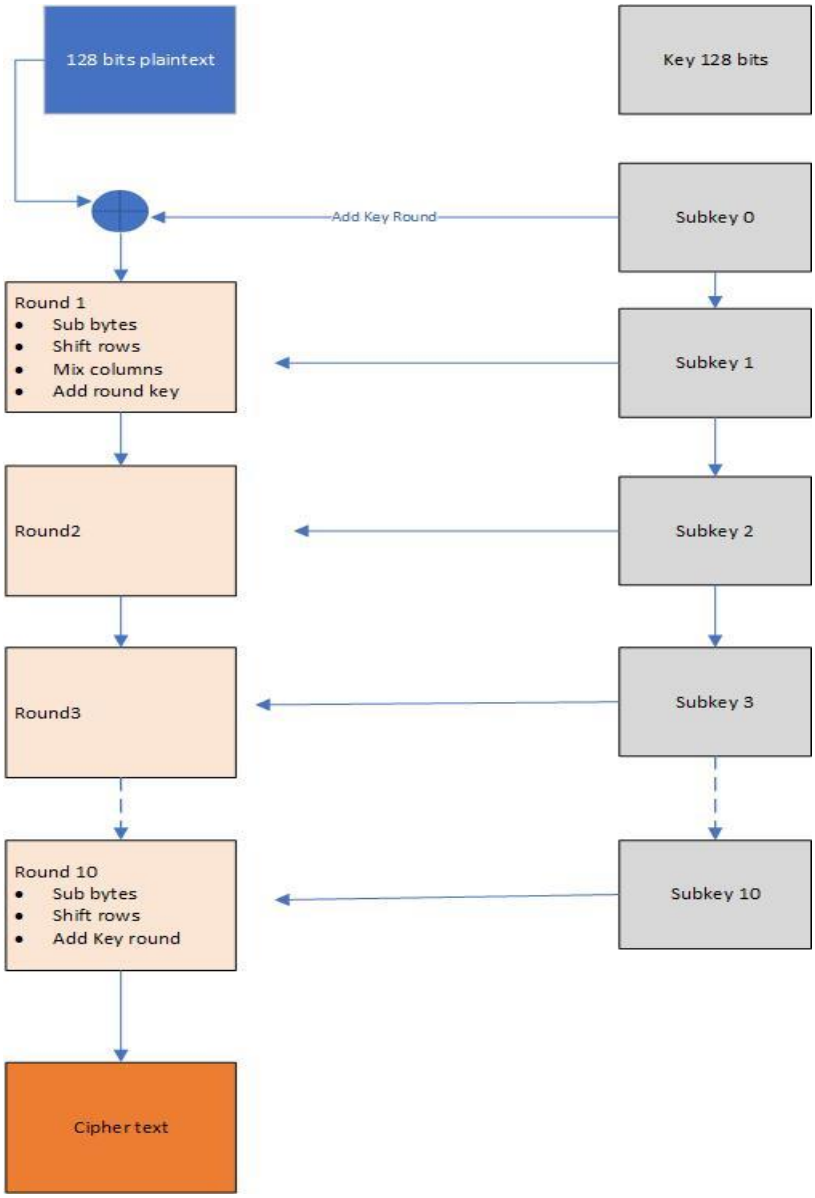

Figure 7. General overview of AES encryption (author)

Figure 8 shows what happens in single rounds. The same process repeats for each round, except the final round, where will not be a **Mix column**.

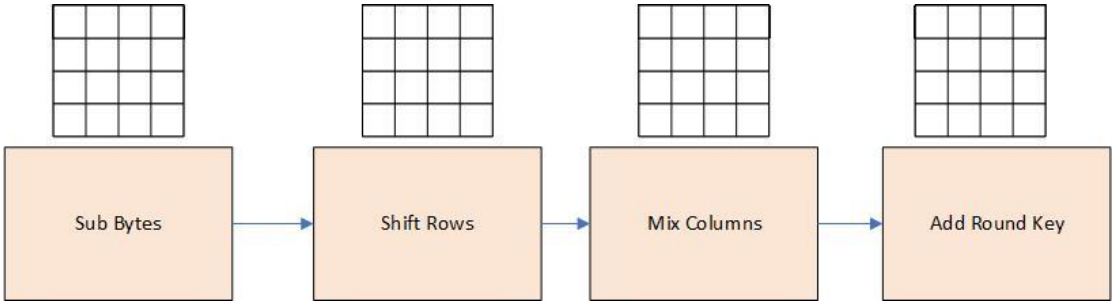

Figure 8. Single round generalization (author)

Add round key (at the initial step)

Suppose we have a plain text of 128 bits, we represent it as 4*4 matrix of 16 bytes, and then XOR it with the subkey0 that is also represented in a form of 4*4 matrix. The output matrix is called the state array and it is sent to round 1. Add Round key will then be performed at the end of all next rounds.

Substitute bytes (Sub bytes)

It is a non-linear substitution, operating each byte independently. S-box is invertible. Each byte of the state is replaced with another byte depending on the key. Figure 10 shows the Rijndael s-box that consists of 256 bytes arranged in 16*16 matrix.

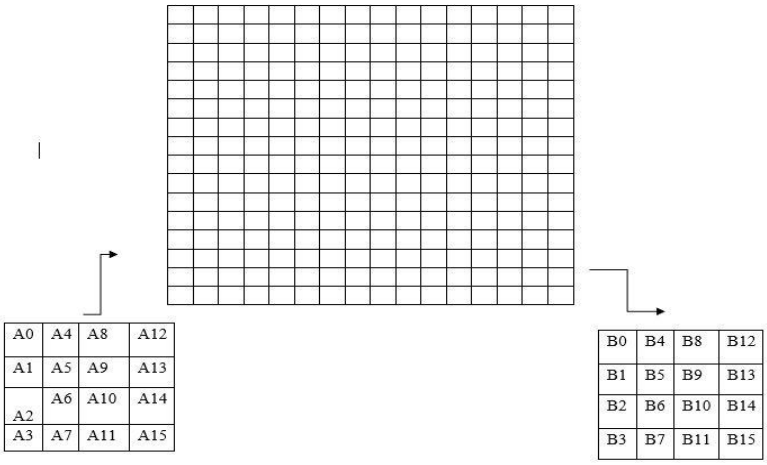

Figure 9. Substitution bytes (author)

Let us suppose we have A0=(C7) hex, which we consider as (x, y), where x represents rows and y represents the columns.

B0= (C6) hex

If A5= 28 then, B5=34

A14=F7 then, B14=68, etc.

We substitute using the substitution box down below.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 10. S-box (*Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES)*, 2001)

Shift Rows

This method is also called permutation, which is a shift in a byte. AES permutation is done at the byte level. The rules of shift row in encryption are described below:

- Shifting to the left
- The number of the shift depending on the number on the rows of the state
- Row0 = no shifts
- R1 = 1, byte cyclic shift to the left
- R2 =2, byte cyclic shifts to the left
- R3 =3, byte cyclic shifts to the left

Figure 11. Shift rows (author)

Mix Columns

Mix Columns is the third crucial step in transformation. The columns of the state are considered to be polynomials over finite field GF(2^8), and multiplied modulo x^4+1 with a fixed polynomials C(x)=03x^3+01x^2+01x+02.(Daemen and Rijmen, 1999)

Mix Column operates on each column individually. Each byte is mapped into a new value that is a function of all four bytes in that column. Take each word/column that is 4 bytes or 4 by 1 matrix and multiply with a constant matrix. You will get a new state matrix. It is a central **diffusion** process in AES.

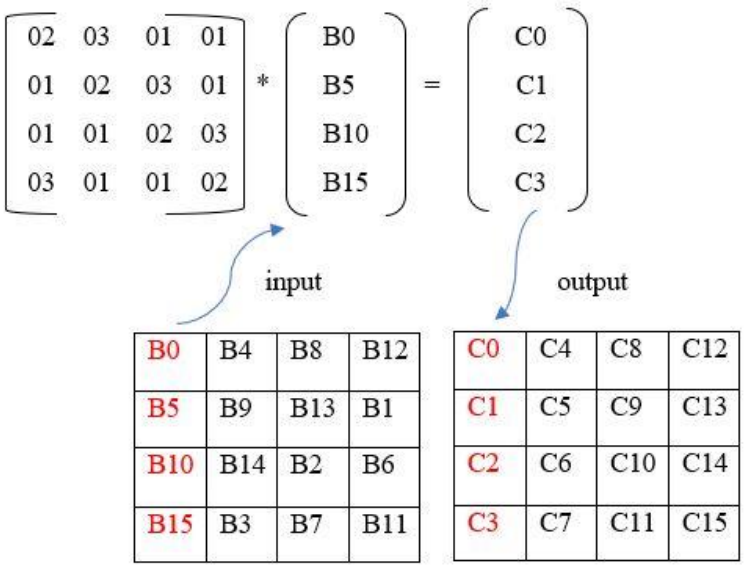

Figure 12. Mix columns(author)

Add Round key (in the inner loop of the rounds)

Add Round Key that occurs within the rounds is the last process of each round. The process is the same as at the initial step. 128 bits state array is XORed with a corresponding 128 bits subkey. It proceeds one column at a time. It adds the subkey word with each state column matrix. It is an important stage in AES. Both the key and input data are structured in a 4 by 4 matrix of bytes. The Figure 13 shows the Add Round key process of round 1, where state array is XORed with subkey1.

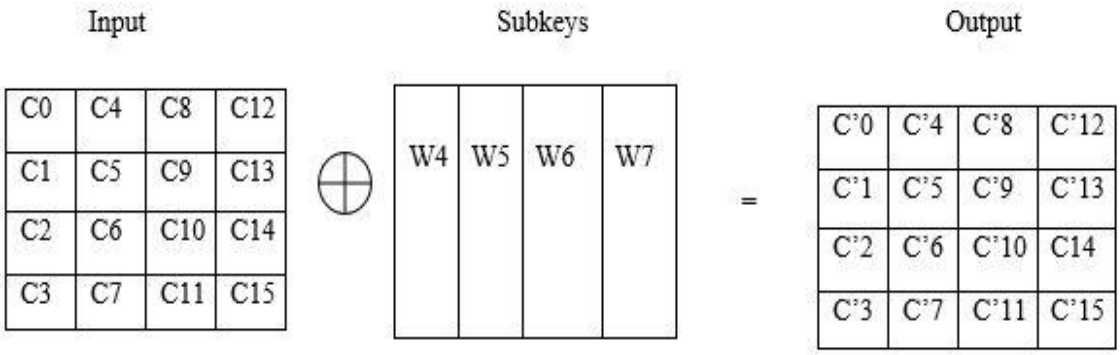

Figure 13. Add round key in inner loop (author)

Decryption AES

The decryption process is like encryption but in a reverse way. We take the ciphertext and Add Round Key, and get the state array, that state array sent to round 1, then we perform the transformation for the decryption processes like Invsub bytes, InvShift Rows, InvMix Columns, and Add Round Key. The encryption transformation can be inverted and implemented in reverse order to get decryption. The inverse mix columns is not required in the last round of decryption.

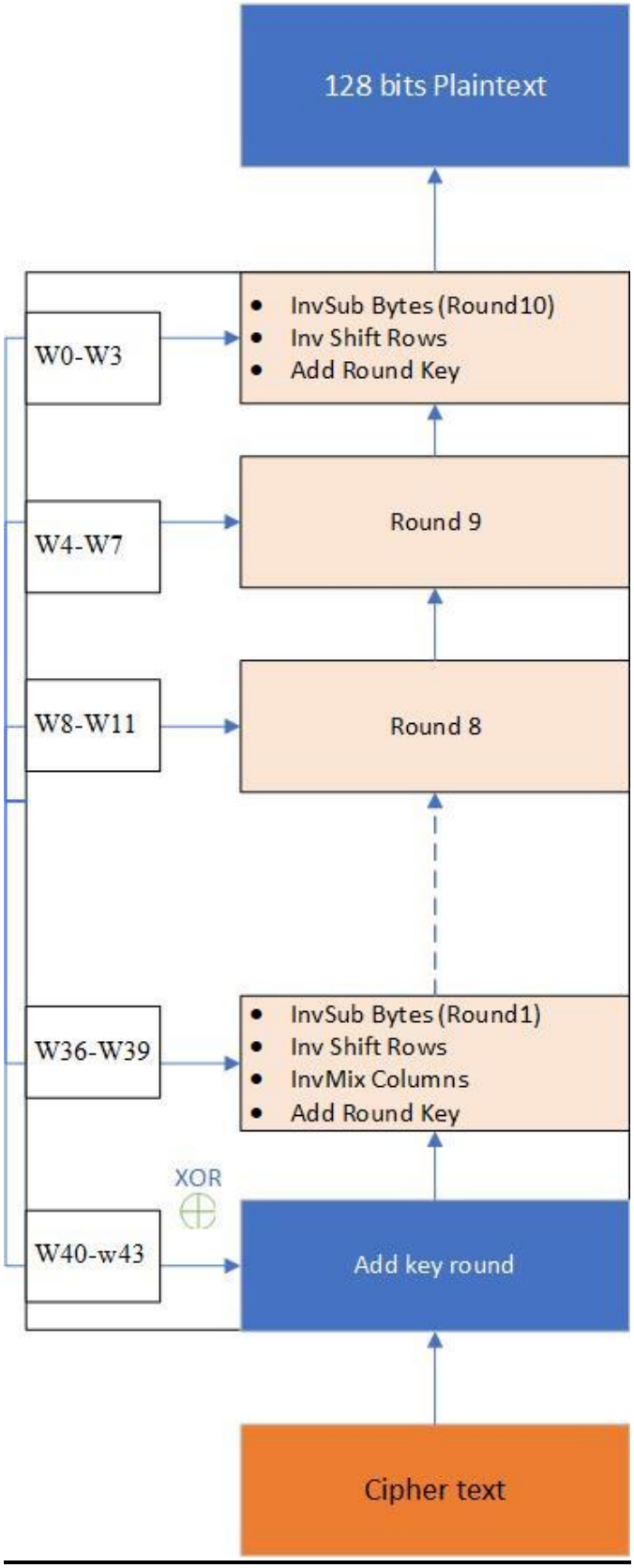

Figure 14. Decryption process (source)

Invsubbyte

AES s-box is a bijective that is one to one mapping. The inverse of the affine transformation can be obtained by taking the multiplicative inverse in GF(2^8). It is the inverse of the byte substitution.(*Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES)*, 2001)

For example:

Ai=S-1(Bi) , Ai= c7 then Bi=31

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 15. Inv s-box(*Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES)*, 2001)

Invshiftrows transformation

It is the inverse of shift rows transformation. The bytes of the last three rows are shifted to the right. The number of the shifts depends on the number of the rows of the state.

- Row0= no shifts
- R1= 1 cyclical shift to the right
- R2= 2 cyclical shifts to the right
- R3= 3 cyclical shifts to the right

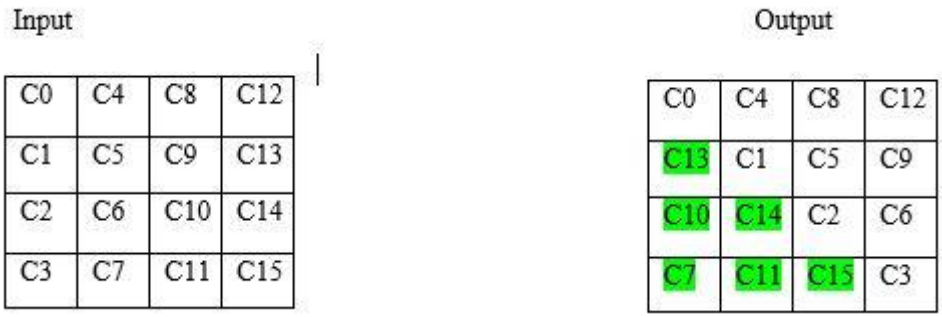

Figure 16. Inverse shift rows transformation (author)

Invmixcolumns

To reverse the operation of the Mix Columns, the inverse of the matrix is used. The input 4 bytes of a column of the state is multiplied by the constant inverse 4*4 matrix.

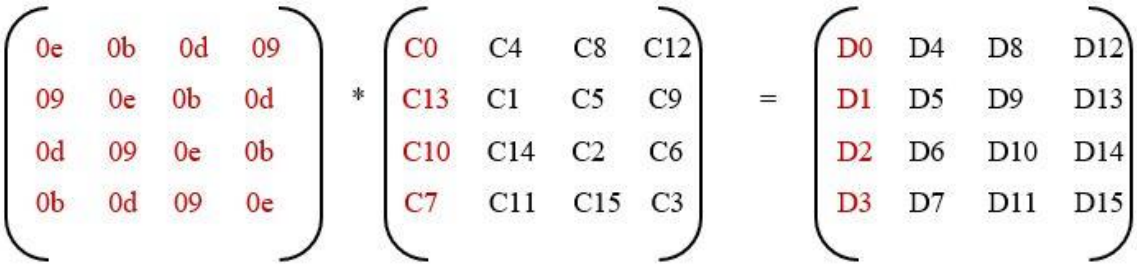

Figure 17. inv mix columns (author)

Add round key

The output from mix column is XORed with (w36-w39) 4*4 matrix and the result is sent to the next round. The same process will repeat for 9 rounds apart from the last 10^{th} round, where the inverse mix columns will be excluded.

3.5 Asymmetric Algorithm

The asymmetric algorithm is a relatively new and complex method of encryption. It was introduced by **Whitfield Diffie**, **Martin Hellman**, and **Ralph Markle** in 1976. The symmetric key is very secure, fast, and widely used, but there are some drawbacks like safe key exchange, fraud against the sender. Diffie, Hellman and Markle had a revolutionary proposal to overcome these problems.

Figure 18 shows the basic principle of how Public-key cryptography works. We have Alice as a sender and Bob as a receiver. At first, Bob generates public and private keys, and he sends a public key to Alice. She then uses the public key to encrypt the plain text. Once Alice encrypts the plaintext, she will send it back to Bob, and he will use his private key to decrypt the message.

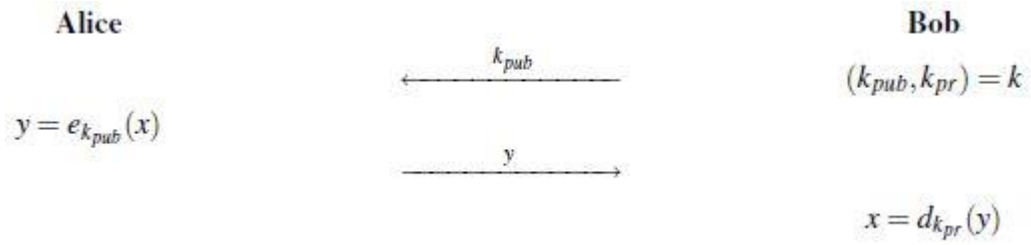

Figure 18. Basic protocol of Asymmetric Algorithm(Paar and Pelzl, 2010)

The Public key is available to everyone but the private key is kept secret. The sender uses the public key to encrypt the data, while the receiver uses the private key to decrypt the data. There are hundreds of symmetric algorithms that have been introduced, a lot of them have been found unsecure, but there still exist many symmetric algorithms which are secure and are used in different fields. As for asymmetric algorithms there are only three main public-key families based on practical significance, which are:

- ***Integer Factorization Schemes***: Based on the fact that it is difficult to factor large integers. Example: RSA algorithm.

- ***Discrete Logarithm Schemes***: Based on discrete logarithm problem in finite field. Examples: DH, DSA algorithms.

- ***Elliptical curve schemes***: The generalization of discrete logarithm algorithm. Algorithm examples: ECDH, ECDSA.

There are also other public-key algorithms, but some of them are not secure, poorly implemented, and not efficient.(Paar and Pelzl, 2010)

3.5.1 RSA Algorithm (Ronald Rivest, Adi Shamir, Leonard Adleman)

RSA was developed in 1977 by **Ronald Rivest**, **Adi Shamir**, and **Leonard Adleman.** The acronym RSA is made from the initial letters of their surnames Rivest, Shamir, and Adleman. It is the most popular asymmetric algorithm. RSA is based on one-way functions that are easy to calculate but hard to invert. In particular, it is easy to calculate the product of two large prime numbers, but it is difficult to factor output of the product. RSA is more appropriate for encrypting small size of data, key exchanges, and digital signatures. RSA does not replace symmetric encryption like AES, but it rather compliments it and helps to create a safer system. RSA is slower than symmetric encryption because of the many calculations involved while implementing it. RSA is often used together with symmetric encryption such as AES, where the symmetric encryption is used for the actual encryption of large size data.

The RSA uses a public key (known to all the users in the network) and a private key (secret key known to the only receiver) to encrypt and decrypt the message. To generate the public and private key, the receiver generates two primes *p* and *q*, and multiplies *p*q* to get the output n, then calculate $\emptyset(n)=(p-1)(q-1)$. After that, the receiver selects *e* that is relatively prime to $\emptyset(n)$, and the pair of numbers (e, n) is the public key, which is later sent to the sender to encrypt the message. Eventually, the receiver needs to find the inverse of *e* to calculate the private key, which is denoted by *d.* The private key is a crucial part of the decryption process and only the receiver should know it. (Paar and Pelzl, 2010)

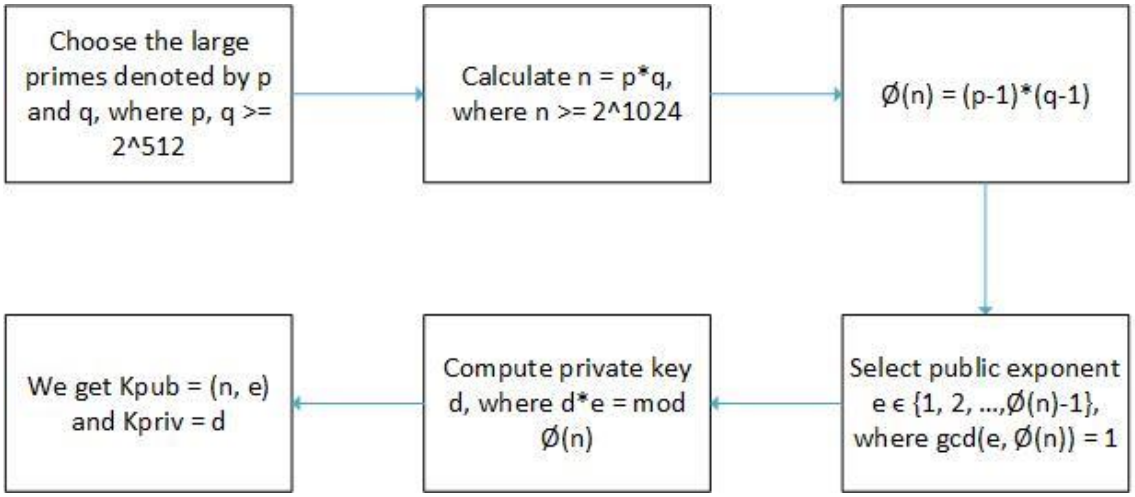

Figure 19. Steps to generate RSA keys, inspired by (Paar and Pelzl, 2010)

Once we generate the key using the steps above, we can encrypt and decrypt the data with the following mathematical formulas.

3.5.2 RSA Encryption

Once we generate the key using the steps above, the sender can encrypt the data with the following mathematical formulas.

Given the public key **Kpub** = (**n**, **e**), plaintext **x,** and integer ring **Zn**={0,1,….n-1}, where **x** in a bit string representation is an element in **Zn** and so the binary value of the plaintext must be less than **n**, the RSA encryption function **y** is calculated as:

$$y = ek_{pub}(x) = x^e \ mod \ n$$

where x, y ∈ Zn, and e is an encryption exponent.

3.5.3 RSA Decryption

Given the private key **Kpri** = **d** and ciphertext **y**, and integer ring **Zn**={0,1,….n-1}, where **y** in a bit string representation is an element in **Zn** and so the binary value of the ciphertext must be less than **n,** the RSA decryption function **x** is calculated as:

$$x = dk_{pri}(x) = y^d \, mod \, n$$

where x, y ∈ Zn, and d is a decryption exponent.

3.5.4 Strength and weakness of RSA

The RSA became very popular due to the use of two keys, known as public and private keys. Any key of the pair could be used for encryption of data, as long as the remaining unused key would be used to decrypt it. It provides confidentiality, authenticity, integrity, and non-reputability of data. It is crucial to use two large prime numbers to calculate modulus n, since weak keys make RSA vulnerable to the attackers. The RSA algorithm is very effective, because while multiplying two primes is easy, it is very difficult to factor large integers into primes. The Federal Information Processing Standards Publication specified three choices of length of modulus ***n*** as being secure, which are 1024 bits, 2048 bits, and 3072 bits. (Nisha and Farik, 2017)

3.6 Hybrid Algorithm

We can encrypt the data only by using a symmetric algorithm, but we need more effort on key management and less effort to provide a non-repudiation function. On the other hand, the public key can provide all the necessary security, but it is very computationally intensive and very slow compared to the symmetric key algorithm. Considering all the drawbacks, it is best to use both algorithms together, which are called hybrid algorithms, where symmetric and asymmetric algorithms are used together to make the system secure. (Paar and Pelzl, 2010)

There are many studies on Symmetric and Asymmetric algorithms about how they can work better together, compensating each other's weaknesses. Symmetric and asymmetric algorithms have their advantages and disadvantages, and to overcome their flaws, a hybrid encryption has been introduced, where two or more encryptions are merged for better results and strength of security. For instance, using RSA you cannot encrypt a block of information that is bigger than the size of its key, while AES can solve this problem, it still has an issue

with safe key exchange over an insecure channel like the Internet.(Chudiwale, Mungse and Chaudhari, 2018)

Al-Shabi (Al-Shabi, 2019) discussed several important symmetric and asymmetric encryption algorithms, and compared them in terms of speed and security to determine whether or not each encryption algorithm was good. According to the research, the comparative analysis was based on the pros and cons of symmetrical and asymmetrical algorithms. Considering factors such as battery consumption, time consumption, block size, round, structure, types of attack, software and hardware implementation, it was determined that the symmetric algorithm AES performed the best in terms of speed of encryption and decryption, structure, length of the key, and usability. While the comparison between symmetric and asymmetric algorithms showed that symmetric algorithms were faster .(Al-Shabi, 2019)

4 Practical Part

The practical part consists of demonstration of application of file encryption/decryption using AES and RSA algorithms. These two algorithms complement each other and provide better and safer solutions. This application was focused on providing the confidentiality between sender and receiver. The application simulates interaction of Alice (sender) and Bob (receiver) based on the literature review. It consists of the various stages and methods used. The communication over the Internet is provided by emails using a SMTP server. The processes of the application are described through the Data Flow Diagrams and each process is then described. The application is then presented, and its user interface is explained.

4.1 Programming language

I have decided to develop a desktop application in the .NET framework using the high-level, object-oriented programming language C#. The .NET framework provides all the neccessary libraries to work with files, to implement AES and RSA algorithms, to connect to mail servers and send emails with attachments for key exchanges and encrypted data transfer. To build the program I have used the Visual Studio Community version, which is free for students.

4.2 Design

The following application consists of a combination of symmetric and asymmetric algorithms which are described as a hybrid algorithm in literature review. It uses AES to encrypt the data and uses RSA to share the key. The application is broken down into the following stages or event list:

- Sender logs in to email account.
- Sender sends an email to the receiver requesting him/her to generate RSA keys.
- Receiver logs in to his/her email account and reads the sender's request.
- Receiver creates the private and public RSA keys.
- Receiver stores the private RSA key locally and sends the RSA public key to the sender.
- Receiver sends public key to sender by email.
- Sender logs in to email account.
- Sender acquires the RSA public key by email.
- Sender uploads the file to the program.
- Sender generates AES Key and encrypts it using the RSA public key.
- Sender encrypts the source file using AES encryption.
- Sender sends the encrypted file to receiver by email.
- Receiver logs in to his/her email account.
- Receivers acquires encrypted file from email.
- Receiver decrypts the key using the private RSA key to obtain secret AES key and decrypts the AES encrypted file.

4.2.1 Data Flow Diagram level 1

Figure 20 shows the level 1 data flow diagram, which represents the flow of data through the main functional areas of the information system.

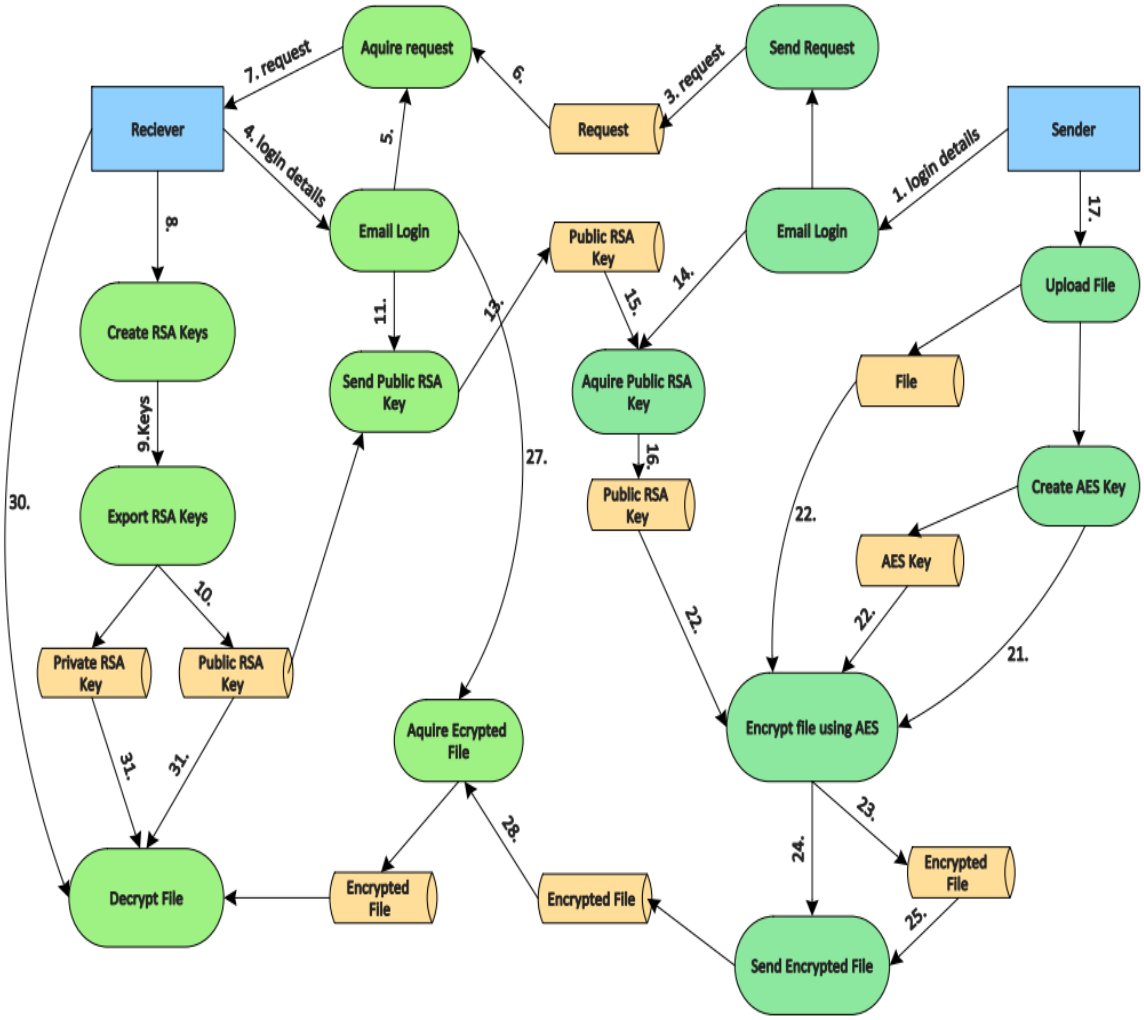

Figure 20. Level 1 DFD (author)

4.2.2 Requesting to generate RSA keys

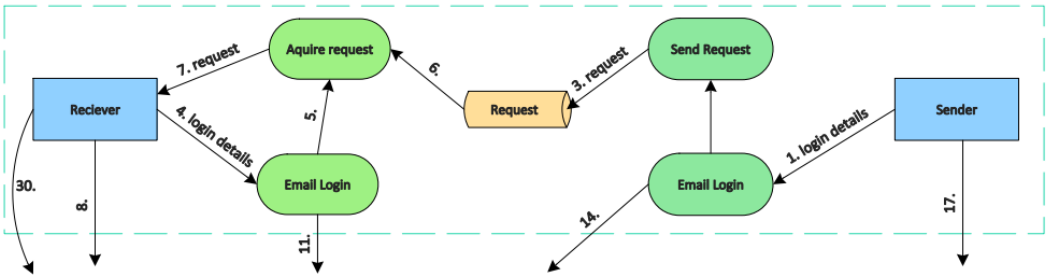

Figure 21. Sending and receiving request by email (author)

This process starts from the sender, who connects to an email server using his/her own account and requests the receiver by email to generate RSA keys. The receiver acquires the request sent from the sender by checking his/her own email. A built-in library of .NET framework provides support for sending emails using SMTP servers. In C# this can be achieved with the help of System.Net.Mail namespace. In the program I used Gmail accounts to connect to Google's SMTP mail server.

```
MailMessage mail = new MailMessage();
SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
mail.From = new MailAddress(EmailFrom.Text);
mail.To.Add(EmailTo.Text);
mail.Subject = EmailSubject.Text;
mail.Body = EmailBody.Text;

System.Net.Mail.Attachment attachment;
attachment = new System.Net.Mail.Attachment(filename);
mail.Attachments.Add(attachment);

SmtpServer.Port = 587;
SmtpServer.Credentials = new System.Net.NetworkCredential(username, password);
SmtpServer.EnableSsl = true;

SmtpServer.Send(mail);
```

Figure 22. Connecting to SMTP server and sending email (author)

The Figure 22 shows how the connection is made to the smtp.gmail.com server using user's account details and how the mail is sent to another email account. At this stage, the program did not send any attachments yet, only the email notification of sender's desire to send an encrypted file.

4.2.3 Generating and exporting of RSA keys

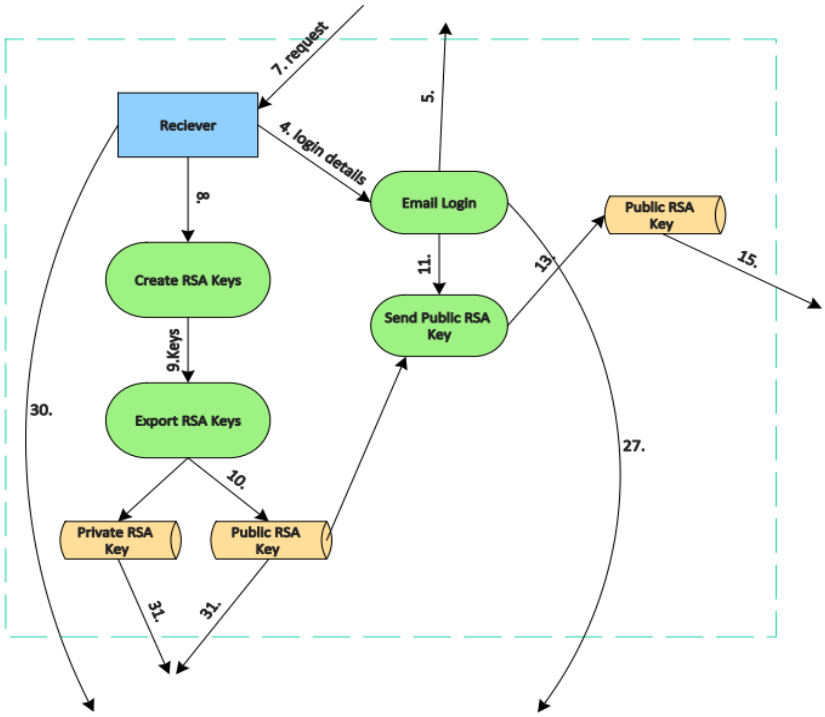

Figure 23. Generating, exporting RSA keys and sending the public key by email(author)

Figure 23 shows the data flow diagram of the second stage, where on the receiver's side the RSA keys were generated and exported to separate files, after obtaining a request from the sender. The private RSA key then safely stayed with the receiver and the public RSA key was sent to the sender by email, using the earlier described technique of connecting to Google's SMTP server. The private key will later play a crucial role in decrypting the data. The public key is sent over the internet without fear of anyone getting an unauthorised access to it, because even if someone acquired the public key, he/she would not be able to use it later to decrypt the data.

To implement RSA and later AES algorithms in C#, I used System.Security.Cryptography namespace. In my program I used the default RSA key size of 1024 bits, but the built-in library allows to make changes to the size of the generated keys, so it is possible to make RSA encryption even stronger.

```
CspParameters parameters = new CspParameters();
RSACryptoServiceProvider RSA;

public Form1()...

private void CreateRSAkeys_Click(object sender, EventArgs e)
{
    parameters.KeyContainerName = keyName;
    RSA = new RSACryptoServiceProvider(parameters);
    RSA.PersistKeyInCsp = true;
}
```

Figure 24. Creation of RSA keys, inspired by (*Walkthrough: Creating a Cryptographic Application | Microsoft Docs*, no date)

Figure 24 shows C# code for RSA key creation. The Boolean variable PersistKeyInCsp allows or disallows the keys to be persisted to local storage. In my program the variable is set to allow keys to be persisted, because the initial goal of the application is to work with encrypted data through email service, which means that the receiver needs to send the public key over the internet and needs to store the private key to local storage, since there are no time constraints on the sender to send the encrypted data.

Using StreamWriter and a ToXmlString function of the System.Security.Cryptography namespace, both keys were saved to separate XML files. The XML file of the public key was then attached to an email and sent to the sender.

4.2.4 Encryption processes

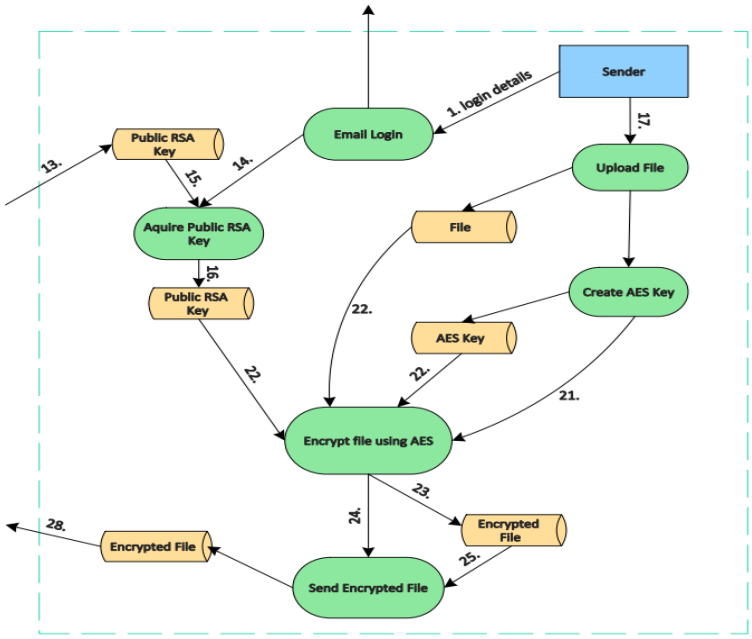

Figure 25. Encryption stage (author)

Once the sender obtains an email with an attached file containing RSA public key, he/she can start encrypting any chosen file using AES encryption algorithm. AES uses only one key to encrypt and decrypt the data. Using the program, the sender uploads any file of his/her desire that is then encrypted using AES encryption algorithm with the generated AES key. The AES key itself gets encrypted with the RSA algorithm using the earlier obtained RSA public key. The encrypted output is then sent to the receiver by email.

In the literature review it was described that such scenario where RSA compliments AES creates a safer system. As good as AES algorithm is, it has one safety issue, that is the problem of sending the key over the internet. If the key gets obtained by the third party, then it can be used to successfully decrypt the data.

RSA, as mentioned before, is slower than AES, but it is good at encrypting small size of data, so it can be used effectively in key exchanges. The size of an AES key used in my program was 128 bits.

```
Aes aes = Aes.Create();
ICryptoTransform transform = aes.CreateEncryptor();
byte[] keyEncrypted = rsa.Encrypt(aes.Key, false);
```

Figure 26. AES key generation and encryption, inspired by (*Walkthrough: Creating a Cryptographic Application | Microsoft Docs*, no date)

Figure 26 shows C# code how AES key and initialization vector (IV) are generated by CreateEncryptor() method and then the key is encrypted using RSA encryption algorithm with the imported public RSA key from the XML file sent earlier by the receiver.

In the literature review it was explained that AES is a block cipher which encrypts 128 bits (or 16 bytes) of data at a time. The process for encryption of 128 bit block of plaintext was described step by step. But in real world practice and also in my own application there is a need to encrypt the data that is bigger than 16 bytes. This poses a security problem, because it allows repetition of patterns that may be recognized by the attacker.

If the key doesn't change throughout encryption process then identical ciphertext blocks will be generated from identical plaintext blocks. Therefore, it is best to avoid pattern repetition in different ciphertexts. This is done by implementing an initialization vector (IV) that adds randomness to encryption, making it probabilistic instead of deterministic. Use of initialization vector helps prevent exploiting weaknesses of deterministic encryption and increases the security of block ciphers such as DES, 3DES and AES. Initialization vector is some randomly chosen number that is usually used only once. If IV is used only once, then it is also called a nonce. Use of IV makes sure that encryption of the same inputs will result in different outputs. Initialization vector can be freely transmitted between communicating parties and doesn't have to be kept secret. (Paar and Pelzl, 2010)

The application of IV can be demonstrated with one of the modes of operation called Cipher Block Chaining mode(CBC). The first block of plaintext is XORed with IV and the result is then encrypted using the key. The resulting ciphertext is then XORed with the next block of plain text, hence, the output from the previous block is always fed into the present block. With such approach the ciphertext of each block becomes unique even if the plaintext is the

same. Using the key and IV the process can be reversed and the ciphertext blocks can be decrypted.

Figure 27 shows how using the CryptoStream Class it is possible to output a file that will consist of an encrypted AES key, initialization vector IV and the encrypted source file.

```
using (FileStream outFs = new FileStream(outFile, FileMode.Create))
{
    outFs.Write(LenK, 0, 4);
    outFs.Write(LenIV, 0, 4);
    outFs.Write(keyEncrypted, 0, lKey);
    outFs.Write(aes.IV, 0, lIV);

    using (CryptoStream outStreamEncrypted = new CryptoStream(outFs, transform, CryptoStreamMode.Write))
    {
        int count = 0;
        int offset = 0;
        int blockSizeBytes = aes.BlockSize / 8;
        byte[] data = new byte[blockSizeBytes];
        int bytesRead = 0;
        using (FileStream inFs = new FileStream(inFile, FileMode.Open))
        {
            do
            {
                count = inFs.Read(data, 0, blockSizeBytes);
                offset += count;
                outStreamEncrypted.Write(data, 0, count);
                bytesRead += blockSizeBytes;
            }
            while (count > 0);
            inFs.Close();
        }
        outStreamEncrypted.FlushFinalBlock();
        outStreamEncrypted.Close();
    }
    outFs.Close();
```

Figure 27. Encryption (*Walkthrough: Creating a Cryptographic Application | Microsoft Docs*, no date)

To read the source file and at the same time encrypt it or decrypt it, the program used the FileStream class combined with the CryptoStream class. The contents of a source file were read and encrypted in specified number of bytes at a time. This approach made file encryption more efficient, allowing the program to work with larger files. Earlier encrypted AES key (by RSA) and the generated initialization vector were also added at the beginning of the output file, along with their length values. As a result, the final encrypted output file consisted of an encrypted AES key and its length value, IV and its length value, and the encrypted source file. The encrypted output was then sent by an email to the receiver.

4.2.5 Decryption processes

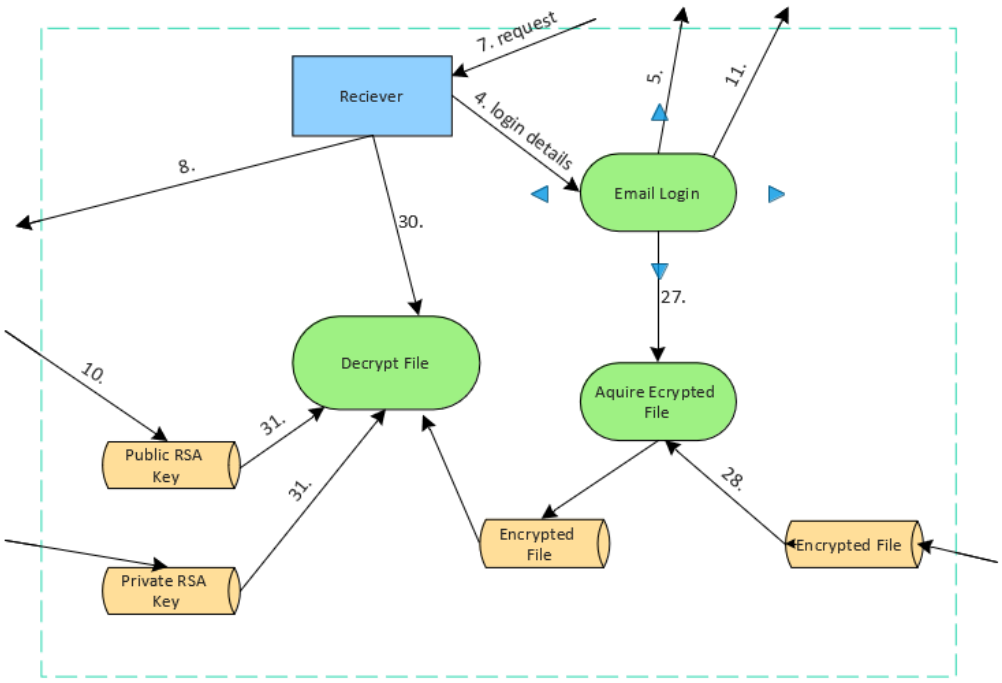

Figure 28. Decryption (author)

The Figure 28 illustrates the final stage, where the receiver obtains an encrypted file from email and decrypts it.

The encrypted bundle consisted of an RSA encrypted AES key and its length value, IV and its length value, and AES encrypted source file. Using the FileStream class the program read the lengths of the initialization vector and of the encrypted AES key from the received bundle file and then extracted the key and IV. The extracted encrypted AES key was then decrypted using previously locally stored private RSA key.

The decryption of the source file was based on the same algorithm that encrypted it. The program used the FileStream and the CryptoStream classes to read the contents of an encrypted file in the same number of bytes at a time. Using the obtained AES key and IV, the encryption was reversed, and the source file was successfully decrypted.

4.3 Application

This section will show and describe the created desktop application for encryption and decryption of files using AES and RSA methods from the receiver's and the sender's point of view.

4.3.1 User interface

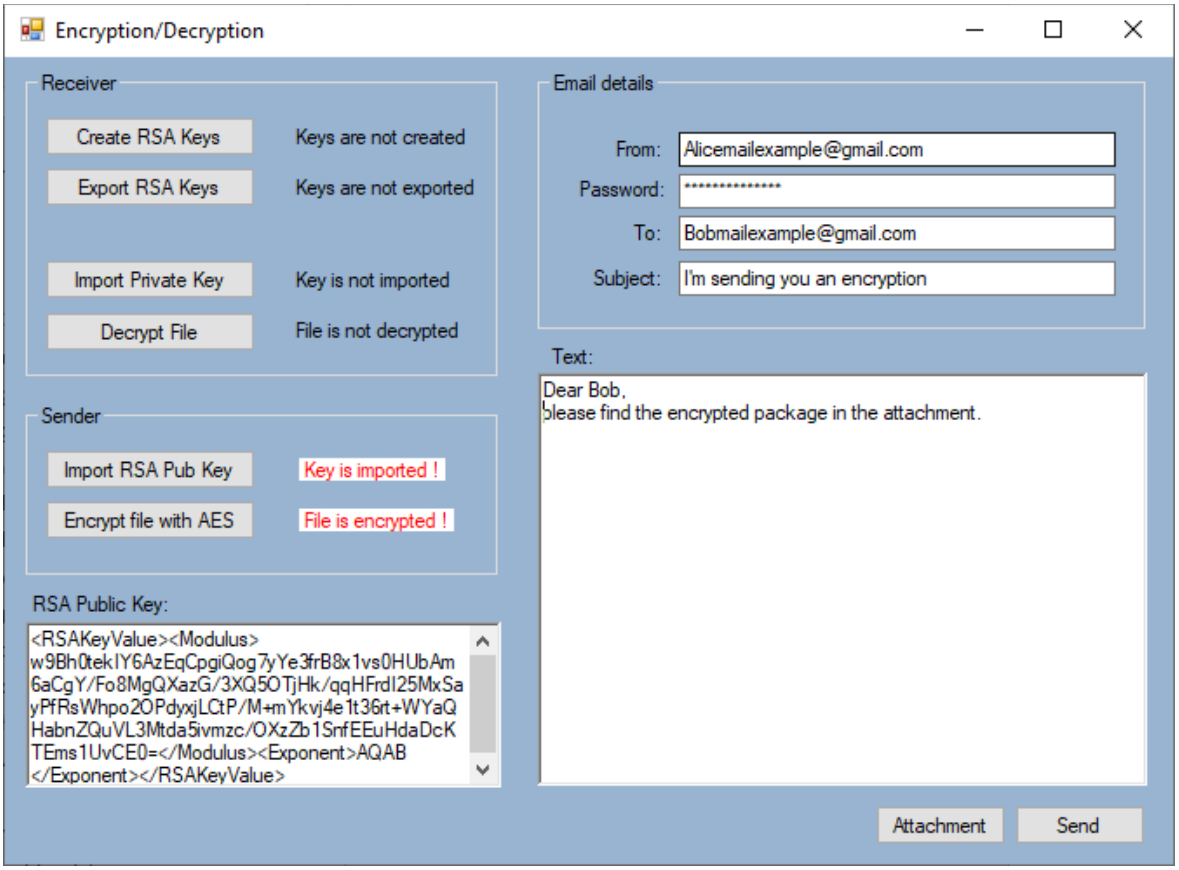

Figure 29. User Interface of the developed application (author)

In Figure 29 is shown a full user interface of the created application. The desktop application consists of only one window form, where its content is visually separated into sections. The right half of the window is dedicated to entering email details, composing an email, and sending it to the recipient. On the left side of the window there is a section called "Receiver" and a section called "Sender". The user can assume

the role of a receiver or a sender depending on the tasks he/she needs to perform. In the bottom left corner, the RSA Public Key is displayed when it is exported by clicking on a button called “Export RSA Keys” in the receiver’s section or when it is imported in the sender’s section by clicking on a button called “Import RSA Pub Key”. It is more just a visual feature. The key can be manually copied from the bottom left corner and sent as part of email text, but such action is not required, since the RSA public key can be attached to email by clicking on the button called “Attachment”.

4.3.2 Receiver section

The UI section is devoted to the receiver operations.

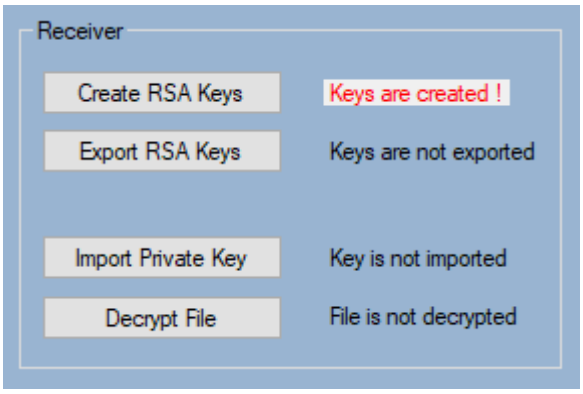

Figure 30. Receiver section of the User Interface (author)

The receiver section consists of four buttons and four corresponding text fields that indicate if the processes were completed by pressing the buttons. In Figure 30 is shown an example where the button “Create RSA Keys” was pressed and the corresponding text field was updated from “Keys are not created” to “Keys are created”, highlighted with red colour. In this section a user can create and export RSA keys, import private RSA key and decrypt the source file sent by the sender.

4.3.3 Sender section

The UI section is devoted to the sender operations.

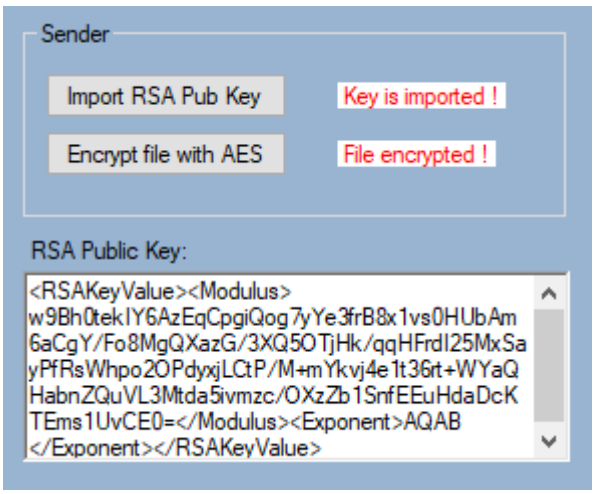

Figure 31. Sender section of the User Interface (author)

The sender section consists of two buttons and two corresponding text fields that tell if the processes were completed after the buttons have been pressed. The button “Import RSA Pub Key” opens the file browser window and lets the user import the public RSA key that was earlier sent by the receiver. The key is then loaded into the program memory. The button “Encrypt file with AES” also opens the file browser window and lets the user select a file that he/she wants to encrypt. After the file is selected the rest of the processes are performed automatically. The AES and IV are generated, the AES key is encrypted using RSA key and the source file is encrypted using AES encryption. The resulting encryption package is saved in local storage as one file.

4.3.4 Email section

The UI section is devoted to the email operations.

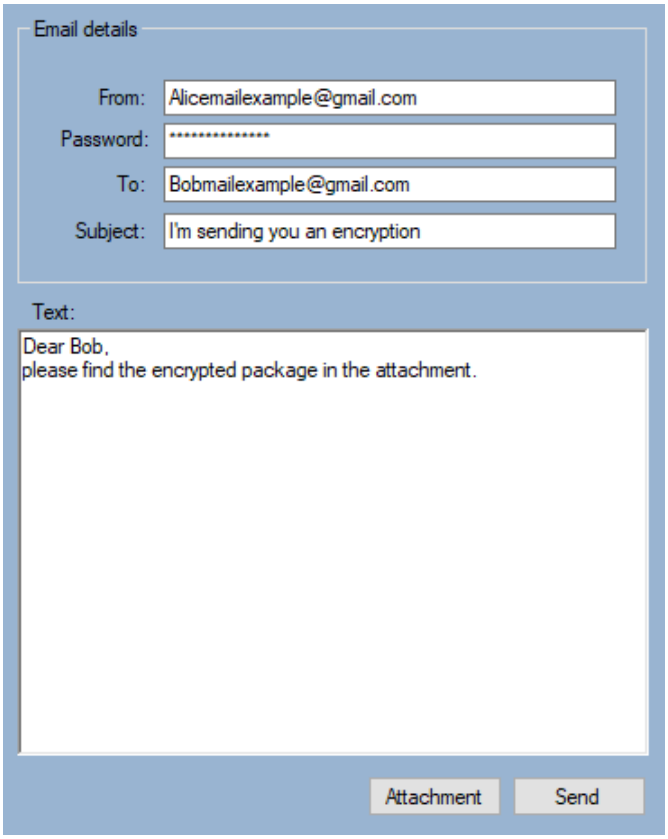

Figure 32. Email section of the User Interface (author)

The email section lets the user type in his own Gmail account and password, email address of a recipient, compose an email, and add file attachment. Here the sender can request the receiver to generate RSA keys or send him/her the encrypted data. The receiver uses this section to send RSA public key to the sender.

5 Results and Discussion

The prototype application has been successfully tested using different types of files of different sizes. The encryption algorithm used its own preset file extension when encrypting the data and did not store information about the original extension of the source file, hence, the sender needed to send this information to the receiver by email. Encryption and decryption algorithms were tested on audio, video, image, xml, and text files. As a result, the confidentiality of the data was assured. Even if the sender's or receiver's email accounts would get compromised and a hacker would get access to the decrypted files, he/she would not be able to decrypt them, because the private RSA key has never been transmitted over the Internet. As to authentication, the program used client's authentication(credentials) to connect to Gmail SMTP server, which required a username and a password. This way the origin of the data was assured through identification of the creator of the email, even if the email itself was sent by the service provider, unless the account got stolen. Regrettably, an integrity check was not implemented in the application, so there was no way to tell if the data was modified during the transit. This security weakness could be removed by implementing a hash function inside the program, an algorithm that would map the encrypted data to a fixed and unique output. This way the receiver would be able compare the hash values of the received data and tell if the data has been modified.

The program used the default settings when generating RSA and AES keys, but the built-in library allows to make changes to the size of the generated keys, so it is possible to further improve the encryption, making it even stronger. The user could be offered more options for selection of key sizes. For the RSA algorithm there could be given more choices, such as using RSA keys of 1024-bits, 2048-bits, or 3072-bits in size. AES encryption/decryption options could include 128-bit, 192-bit, and 256-bit keys. Using bigger key sizes would make the encryption more secure and harder to break, but it would also slow down the encryption and decryption processes.

Conclusion

The research conducted for this thesis was focused on the commonly used symmetric encryption algorithms such as DES, TDES, AES and an asymmetric algorithm RSA. The special emphasis was put on the detailed explanation of the RSA and the AES algorithms. The importance of data security has been studied through the perspective of two branches of cryptology: cryptoanalysis and cryptography. Study of the cryptanalytic attacks and goals of cryptography lead to a better understanding of strengths and weaknesses of the developed prototype application concerning data security. Based on the study of various information sources related to computer encryption, a prototype application was developed using a C# programming language. In the practical part, the design of the prototype application was illustrated using Data Flow Diagrams and different stages and processes of the application were explained. The program implemented a hybrid encryption method that incorporated both AES and RSA algorithms to encrypt and decrypt files of any given extension type and size. The application has also provided communication over the Internet by the use of Gmail SMTP server, which enabled transfer of encrypted data and key exchanges by emails. As a result, the program turned out to be successful in assuring confidentiality of transmitted data and its origin. The absence of the integrity check in the program, and the lack of the choice given to the user regarding the key size selection left room for further improvements in the future work.

6 References

Al-Shabi, M. A. (2019) ‘A Survey on Symmetric and Asymmetric Cryptography Algorithms in information Security’, *International Journal of Scientific and Research Publications (IJSRP)*, 9(3), pp. 576–589. doi: 10.29322/ijsrp.9.03.2019.p8779.

Barker, E. and Roginsky, A. (2019) ‘Transitioning the Use of Cryptographic Algorithms and Key Lengths’, *NIST Special Publication 800-131A Revision 2*, (March), pp. 17–18. Available at: https://doi.org/10.6028/NIST.SP.800-131Ar2.

Chudiwale, G., Mungse, A. and Chaudhari, R. (2018) ‘Hybrid Encryption/Decryption Technique Using Public and Symmetric Key Algorithm’, *International Journal of Advanced Innovative Technology In Engineering (IJAITE)*, 3(2), pp. 20–23.

Cryptography - Quick Guide - Tutorialspoint (no date). Available at: https://www.tutorialspoint.com/cryptography/cryptography_quick_guide.htm (Accessed: 12 January 2021).

Daemen, J. and Rijmen, V. (1999) *AES Proposal:Rijndael*. doi: 10.9783/9780812294491-001.

Delfs, H. and Knebl, H. (2007) *Introduction to Cryptography principles and Applications*. 2nd edn. Springer-Verlag Berlin Heidelberg.

Dooley, J. (2018) *History of Cryptography and Cryptanalysis*. Springer International Publishing AG, part of Springer Nature 2018. doi: https://doi.org/10.1007/978-3-319-90443-6_1.

Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES) (2001). Available at: http://csrc.nist.gov/csor/.

Keith, M. (2017) *EverydayCryptography:Fundamental Principles and Applications*. 2nd edn, *Oxford University Press*. 2nd edn. Oxford University Press.

Menezes, A., Oorschot, P. and Vanstone, S. (1996) *Applied Cryptography*, *CRC Press*. doi: 10.1.1.99.2838.

Paar, C. and Pelzl, J. (2010) *Understanding Cryptography*, *Springer-Verlag Berlin Heidelberg*. doi: 10.1007/978-3-642-04101-3.

Walkthrough: Creating a Cryptographic Application | Microsoft Docs (no date). Available at: https://docs.microsoft.com/en-us/dotnet/standard/security/walkthrough-creating-a-cryptographic-application (Accessed: 15 March 2021).

Wilson, P. and Garcia, M. (2006) ‘A Modified Version of the Vigenère Algorithm’,

International Journal of Computer Science and Network Security (IJCSNS), 6(3), pp. 140–143. doi: 10.1.1.385.6615.