

**A
PROJECT REPORT
ON
FILE SECURITY USING CRYPTOGRAPHY IN PYTHON**

A dissertation submitted in partial fulfilment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Submitted By

E.MADHURI	(17TQ1A0527)
M.AKRAM BAIG	(17TQ1A0556)
O.THARUN	(17TQ1A0565)
RAJASHEKAR	(16TQ1A0518)
T.ROJA	(16TQ5A0503)

Under the esteemed guidance of

Mrs.D.SHIRISHA Asst.Prof



Department Of Computer Science and Engineering
SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES
(Approved by AICTE,Affiliated to JNTUH,Accredited by NAAC)
Korremula Road, Narapally,Medchal-501301
2017-2021



SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES
(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NAAC)
Korremula Road, Narapally (V), Ghatkesar Mandal, Medchal-Dist:- 501301



CERTIFICATE

This is to certify that

E.MADHURI (17TQ1A0527)

M.AKRAM BAIG (17TQ1A0556)

O.THARUN (17TQ1A0565)

RAJASHEKAR (16TQ1A0518)

T.ROJA (16TQ5A0503)

has submitted a report on “**FILE SECURITY USING CRYPTOGRAPHY IN PYTHON**” in partial fulfillment of the requirements for award of the degree of **Bachelor of Technology in COMPUTER SCIENCE ENGINEERING** to the **Jawaharlal Nehru Technological University** is record of bonafide work carried out by them under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

PROJECT GUIDE

Mrs.D.Shirisha

PRINCIPAL

Dr.Y.Rajasree Rao

HEAD OF DEPARTMENT

Mrs. M. Sowjanya reddy

EXTERNAL EXAMINER

DECLARATION

We **E.MADHURI, M.AKRAM BAIG, O.THARUN, RAJASHEKAR, T.ROJA** hereby declare that the work which is being presented in the project entitled **“FILE SECURITY USING CRYPTOGRAPHY IN PYTHON”** in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology and submitted in the **Department of Computer Science and Engineering, Siddhartha institute of technology and sciences**, Narapally is an authentic record of our own work carried out during a period from 2020 to 2021 under the guidance of **Mrs.D.SHIRISHA**. The work presented in this project report has not been submitted by me for the award of any other degree of this or any other Institute/University

Date:

Place: Narapally

E.MADHURI	(17TQ1A0527)
M.AKRAM BAIG	(17TQ1A0556)
O.THARUN	(17TQ1A0565)
RAJASHEKAR	(16TQ1A0518)
T.ROJA	(16TQ5A0503)

ACKNOWLEDGEMENT

This is an acknowledgement of the intensive drive and technical computer of many individuals who have contributed to the success of my project.

I am heartily thankful to my principal **Dr.Y.Rajasree Rao** and Head of the Department, **Mrs. M. Sowjanya reddy** Computer Science and Engineering, for their constant support towards my project.

I am grateful to my guide **Mrs.D.Shirisha** , Assistant Professor, Computer Science and Engineering Department, who gave me the necessary motivation and support during the full course of my project.

I would like to express our immense gratitude and sincere thanks to all the faculty members of CSE department and my friends for their valuable suggestions and support which directly or indirectly helped me in successful completion of work.

E.MADHURI (17TQ1A0527)

M.AKRAM BAIG (17TQ1A0556)

O.THARUN (17TQ1A0565)

RAJASHEKAR (16TQ1A0518)

T.ROJA (16TQ5A0503)



SIDDHARTHA INSTITUTE OF TECHNOLOGY AND SCIENCES
(Approved by AICTE, Affiliated to JNTU Hyderabad, Accredited by NAAC)
Korremula Road, Narapally (V), Ghatkesar Mandal, Medchal-Dist:- 501301



Vision of the Department: To be a Recognized Center of Computer Science Education with values and quality research

Mission of the Department:

MISSION	STATEMENT
DM1	Impart High Quality Professional Training With An Emphasis On Basic principles Of Computer Science And Allied Engineering
DM2	Imbibe Social Awareness And Responsibility To Serve The Society
DM3	Provide Academic Facilities Organize Collaborated Activities To enable Overall Development Of Stakeholders

Programme Educational Objectives (PEOs)

- **PEO1:** Graduates will be able to synthesize mathematics, science, engineering fundamentals, laboratory and work- based experiences to formulate and to solve problems proficiently in Computer science and Engineering and related domains.
- **PEO2:** Graduates will be prepared to communicate effectively and work in multidisciplinary engineering projects following the ethics in their profession.
- **PEO3:** Graduates will recognize the importance of and acquire the skill of independent learning to shine as experts in the field with a sound knowledge.

ABSTRACT

Our Project Mainly Focuses on Data encryption, which translates data into another form, so that only people with access to a secret key(formally called a decryption key) or password can read it.Encrypted data is commonly referred to as cipher text, while unencrypted data is called plain text. Many protocols rely on asymmetric cryptography, including the transport layer security (TLS) and secure sockets layer (SSL) protocols, which make HTTPS possible. The encryption process is also used in software programs - - such as browsers -- that need to establish a secure connection over an insecure network like the Internet or need to validate a digital signature.

Currently, encryption is one of the most popular and effective data security methods used by organizations.Our Project is based on Asymmetric cryptography.The purpose of Our project is to protect digital data confidentiality as it is stored on computer systems and transmitted using the internet or other computer networks.Files are the most basic securable units of a repository. Often data is stored and shared as files and folders. Therefore, file security is a subset of data security that focuses on the secure use of files.

Key Words: Python, PyCrypto, Cybersecurity, Asymmetric Key Cryptography, Encryption and Decryption

The expected program outcomes are

PO1,PO2,PO3,PO4,PO5,PO6,PO7,PO8,PO9,PO10,PO11,PO12 & PSO1,PSO2

CONTENTS

Topic	Page No
Abstract	
List of Figures	1
List of Screens	1
CHAPTER-1 : INTRODUCTION	2
1.1 Introduction	2
1.2 Problem statement	3
1.3 Objective	3
CHAPTER-2 : LITERATURE SURVEY	4
2.1 Introduction	4
2.2 Communication	4
2.3 Existing System	4
2.3.1 The RSA Encryption Algorithm	5
2.3.1.1 Architecture of RSA Algorithm	7
2.3.1.2 Steps Involved in RSA Algorithm	8
2.3.1.3 Advantages and Disadvantages of RSA Algorithm	9
2.4 Proposed System	10
2.4.1 Advantages of Proposed System	10
CHAPTER-3 : ANALYSIS	11
3.1 Introduction	11

3.2 Functions and modules used	11
3.2.1 Methods Used	11
3.2.2 Modules Used	11
3.3 Software and Hardware Requirement	12
3.3.1 Hardware Requirement	12
3.3.2 Software Requirement	12
CHAPTER-4 : DESIGN	13
4.1 Introduction	13
4.2 UML Diagrams	13
4.2.1 Use Case Diagram	13
4.2.2 Sequence Diagram	14
4.2.3 Activity Diagram	15
4.2.4 Class Diagram	16
4.2.5 Flow Chart	17
CHAPTER - 5 : IMPLEMENTATION	18
5.1 Introduction	18
5.2 Types Of Files commonly Used	18
5.2.1 Audio File Formats	18
5.2.2 Compressed File Formats	18
5.2.3 Image File Formats	19

5.2.4 Video File Formats	19
5.2.5 Common Types Of Document Formatting	19
5.3 Types Of Cryptography	20
5.3.1 Symmetric Key Cryptography	21
5.3.2 Three Common Types of Symmetric Encryption Algorithms	23
5.4 Asymmetric Key Cryptography	27
5.4.1 Types of Asymmetric Cryptography Algorithms	33
5.5 Working	33
5.5.1 installing visual studio code	33
5.5.2 Installing python 2.7x Version	33
5.5.3 Installing Modules	34
5.5.4 Running The Program	34
5.6 Source Code	35
CHAPTER - 6 : Output Screens	40
CHAPTER - 7 : Conclusion	42
References	43

LIST OF FIGURES

S.NO	FIG	FIG NAME	PAGE NO
1.	2.3.1.1	RSA Architecture	8
2.	4.2.1	Use case Diagram	14
3.	4.2.2	Sequence Diagram	15
4.	4.2.3	Activity Diagram	16
5.	4.2.4	Class Diagram	17
6.	4.2.5	Flow Chart	18
7.	5.4	Asymmetric Key Cryptograhya	29
8.	5.5.4	Running The Program	35

LIST OF SCREENS

S. NO	SCREEN NAME	PAGE NO
1.	OutputScreen1	41
2.	OutputScreen2	42
3.	OutputScreen3	43
4.	OutputScreen4	44

CHAPTER-1

INTRODUCTION

1.1 Introduction

Previously, when RSA was developed in the late 1970's it was mainly to encrypt secret messages that are transferred from one place to another. In this project we mainly focused on Securing files which contain sensitive data or any confidential information.

We will be Using the RSA Algorithm to Decrypt and Encrypt Files. As RSA is an Asymmetric Cryptography we use both Public and private keys. These keys are used to encrypt/decrypt sensitive information stored in the Files. Cryptography is a process which is mainly used for safe and secure communication. It works on different mathematical concepts and algorithms to transfer the encoded data into a secret code which is difficult to decode.

It involves the process of encrypting and decrypting the data, for eg. If I need to send my personal details to someone over mail, I can convert the information using Encryption techniques and send it, on the other hand, the receiver will decrypt the information received using Decryption Techniques and there will be no data tampering in between.

The ciphertext is a data or text which is encrypted into a secret code using a mathematical algorithm, it can be deciphered using different mathematical Algorithms. Encryption is converting the text into a secret message, technically known as converting the plaintext to ciphertext and Decryption is converting the ciphertext back to plaintext so that only the authorized users can decipher and use the data. Generally, it uses a key that is known to both the sender and the receiver so that they can cipher and decipher the text.

Python has modules/libraries which are used for cryptography namely: Cryptography, Simple-Crypt, PYCRYPTODOME, Hashlib: MD5 & SHA1 (Most secured). Symmetric encryption is when a key is used to encrypt and decrypt a message, so whoever encrypted it can decrypt it.

The only way to decrypt the message is to know what was used to encrypt it; kind of like a password. Asymmetric cryptography, also known as public-key cryptography, is a process that uses a pair of related keys -- one public key and one private key -- to encrypt and decrypt a message and protect it from unauthorized access or use. A public key is a cryptographic key that can be used by any person to encrypt a message so that it can only

be deciphered by the intended recipient with their private key. A private key -- also known as a secret key -- is shared only with the key's initiator. When someone wants to send an encrypted message, they can pull the intended recipient's public key from a public directory and use it to encrypt the message before sending it.

The recipient of the message can then decrypt the message using their related private key. On the other hand, if the sender encrypts the message using their private key, then the message can be decrypted only using that sender's public key, thus authenticating the sender. These encryption and decryption processes happen automatically; users do not need to physically lock and unlock the message.

Many protocols rely on asymmetric cryptography, including the transport layer security (TLS) and secure sockets layer (SSL) protocols, which make HTTPS possible. The encryption process is also used in software programs -- such as browsers -- that need to establish a secure connection over an insecure network like the Internet or need to validate a digital signature.

1.2 Problem Statement

Security is one of the issue often face by web developer and software engineer. In today's life, computer and networking is essential to today's Communication and transferring of data is essential when doing online banking, data transferring, sharing and more. So in this project, RSA algorithm will be used as the base of the encryption. This Algorithm will be used to encrypt different kinds of files

1.3 Objective

The objectives of the research are to:

- i. Encrypt and Decrypt Files using RSA algorithm.
- ii. Developing a python program helps to encrypt the given file using secret keys.

CHAPTER - 2

LITERATURE SURVEY

2.1 Introduction

This literature review explains the concept of the ways to find out all information that will be used in order to develop this system. In this chapter, all the research that is related to this system will be analyzed.

2.2 Communication

In general security is the quality or state of being secure and free from danger. In other words, building protection against adversaries, from those who do harm, intentionally or otherwise, is objective. Computer security can be defined as technological and managerial procedures applied to computer systems to ensure the availability, integrity, and confidentiality, availability, and confidentiality of data and information from threats and vulnerabilities.

Confidentiality ensures that computer-related assets are accessed only by a authorized parties. That is, only who's who should have access to something will actually get that access. By "access", unauthorized users not only can read, but also view, print, or simply know that particular assets exist. Confidentiality is sometimes called secrecy and or privacy. Integrity means that assets can be modified only by authorized parties or only in authorized ways. In this context, modification includes writing, changing, changing status, deleting and creating. In distributed systems, the traffic between clients and servers is a new point of attack for would-be intruders. Vulnerabilities introduced by insecure communication links can naturally be counteracted by services and mechanisms for communication security.

2.3 Existing System

1.Security is the main Issue when it comes to sharing any file, Some Encryption Methods don't provide much security.

2.The Keys used in other methods for the Encryption Process are limited in size.

2.3.1 The RSA Encryption Algorithm

RSA is a popular encryption method that uses very large prime numbers to generate public and private keys. This algorithm was programmed by Ronald Rivest, Adi Shamir and Leonard Adelman. This algorithm is based on the concept of factoring, making it easy to encrypt but hard to decrypt. RSA is one of the hardest algorithms that is robust, and difficult to crack of the encryption standards currently used by applications for secure storage and transmission of data. RSA uses factors containing roughly 150 digits, making it not impossible to be calculated manually, even with a powerful computer, one cannot crack the encryption in a reasonable timeframe. RSA is a block encryption algorithm, which means when a chunk of data is encrypted. It is first broken into a number of blocks. Each block is treated as a sequence of bits, with the number of digits being just a little less.

RSA encryption is among the safest encryption algorithms currently used in application. The keys are everywhere between 1024-2048 bits long, making them quite difficult to crack with brute force. One disadvantage of RSA encryption, it is slower than other types of encryption. In RSA algorithm, it works by multiplying each other by a third number, for example, the two number X and Y, it can be calculated by multiplying each by a third number, N. if N is known only by the person who knows it, other who try to break the code will have a difficult time calculating X and Y. in addition, on RSA, N is a very large number, making the calculation even more difficult..

Indeed, RSA uses factors roughly 150 digits, making it hard. When the message is decrypted, the recipient uses another special number, K, 'where $(KE-1)$ is divisible by $(P-1)(Q-1)$. The value of k is chosen such that multiplying the encrypted message by itself K times, and then dividing the result by N, gives the original message as the remainder. To find out K, then, the values P and Q should be known. The values E and N constitute the public key, which can be freely distributed. The value K forms the private key, which should be kept secret.

The RSA algorithm is among the safest encryption algorithm currently used by application. Typically, RSA keys are anywhere between 1024-2048bits long, making them quite difficult to crack with brute force. The only vulnerability in the RSA encryption algorithm is man-in-the-middle impersonation attacks during the key

distribution stage. If the sender and destination systems are able to securely exchange the keys, then the RSA is quite secure.

Data communication is an important aspect of our living. So, protection of data from misuse is essential. A crypto system defines a pair of data transformations called encryption and decryption. Encryption is applied to the plain text i.e. the data to be communicated to produce cipher text i.e. encrypted data using encryption key. Decryption uses the decryption key to convert cipher text to plain text i.e. the original data.

Now, if the encryption key and the decryption key are the same or one can be derived from the other then it is said to be symmetric cryptography. This type of cryptosystem can be easily broken if the key used to encrypt or decrypt can be found. Any practical implementation of the RSA crypto system would involve working with large integers (in our case, of 1024 bits or more in size). One way of dealing with this requirement would be to write our own library that handles all the required functions.

While this would indeed make our application independent of any other third party library, we refrained from doing so due to mainly two considerations. First, the speed of our implementation would not match the speed of the libraries available for such purposes. Second, it would probably not be as secure as some available open source libraries are.

There were several libraries to consider for our application. There are three choices of libraries: the Big Integer library (Java), the GNU MP Arbitrary Precision library (C/C++), and the OpenSSL crypto library (C/C++). Of these, the GMP library (i.e. the GNU MP library) seemed to suit our needs the most.

Timings for 1024-bit RSA (without compiler optimization) All the times recorded below have been measured on a 733 MHz Pentium class processor, using the time measurement functions offered by the C library on a GNU/Linux platform (kernel 2.4.20).

Encrypting sensitive data and managing the encryption keys can be a complex task in an enterprise environment. RSA has extensive experience in solving hard encryption and key management problems for companies across the globe. RSA is currently used for many applications like RSA SecuriD, Digital Certificates, Smart Cards,

etc. This algorithm is considered computationally unbreakable. i.e. it would take a very long time to break the code.

Especially if we use large keys (1024 bits at least), it is almost impossible to find the private key to decode the cipher text. This is because the algorithm requires factoring two very large numbers. There are few security protocols that use RSA. Some of them are IPSEC/IKE for IP data security, TLS/SSL for transport data security, POP for email security, SSH for terminal connection security and SILC, the conferencing service security

2.3.1.1 Architecture of RSA Algorithm

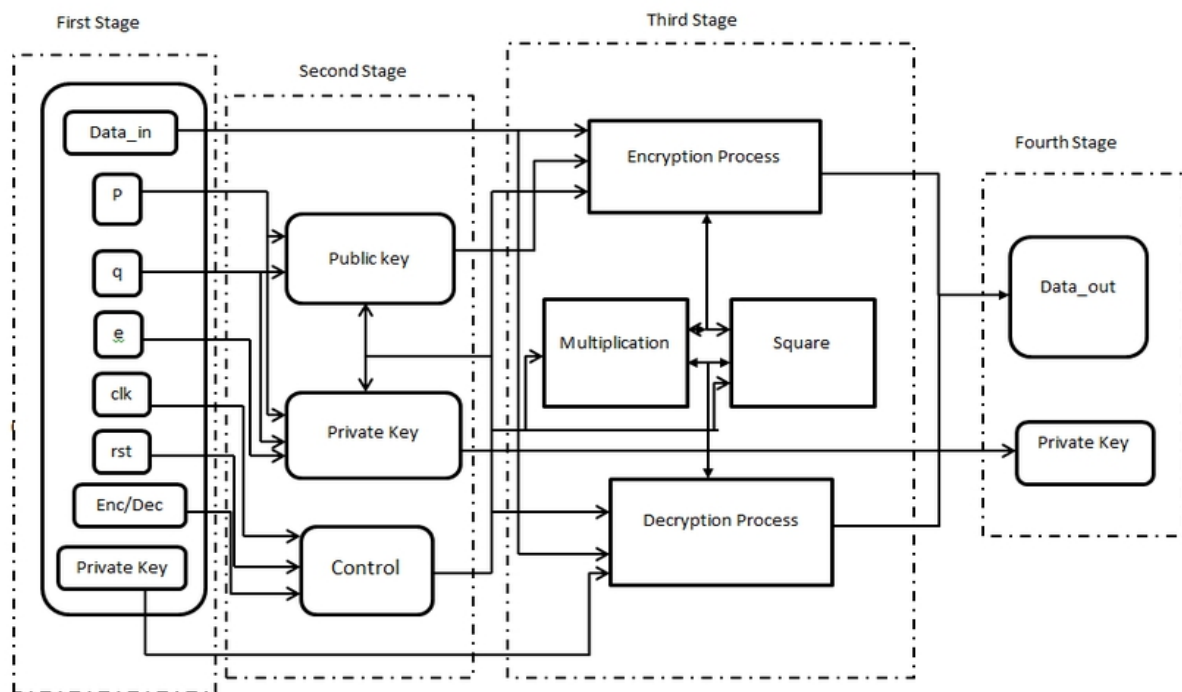


Fig 2.3.1.1 RSA Architecture

2.3.1.2 Steps Involved in RSA Algorithm

RSA encrypts messages through the following algorithm, which is divided into 3 steps:

1. Key Generation

I. Choose two distinct prime numbers p and q .

II. Find n such that $n = pq$.

n will be used as the modulus for both the public and private keys.

III. Find the totient of n , $\phi(n)$

$\phi(n) = (p-1)(q-1)$.

IV. Choose an e such that $1 < e < \phi(n)$, and such that e and $\phi(n)$ share no divisors other than 1 (e and $\phi(n)$ are relatively prime).

e is kept as the public key exponent.

V. Determine d (using modular arithmetic) which satisfies the congruence relation

$$de \equiv 1 \pmod{\phi(n)}.$$

In other words, pick d such that $de - 1$ can be evenly divided by $(p-1)(q-1)$, the totient, or $\phi(n)$.

This is often computed using the Extended Euclidean Algorithm, since e and $\phi(n)$ are relatively prime and d is to be the modular multiplicative inverse of e .

d is kept as the private key exponent.

The public key has modulus n and the public (or encryption) exponent e . The private key has modulus n and the private (or decryption) exponent d , which is kept secret.

2. Encryption

I. Person A transmits his/her public key (modulus n and exponent e) to Person B, keeping his/her private key secret.

II. When Person B wishes to send the message "M" to Person A, he first converts M to an integer such that $0 < m < n$ by using agreed upon reversible protocol known as a padding scheme.

III. Person B computes, with Person A's public key information, the ciphertext c corresponding to

$$c \equiv me \pmod{n}.$$

IV. Person B now sends message "M" in ciphertext, or c , to Person A.

3. Decryption

I. Person A recovers m from c by using his/her private key exponent, d , by the computation

$$m \equiv cd \pmod{n}.$$

II. Given m , Person A can recover the original message "M" by reversing the padding scheme.

This procedure works since

$$c \equiv me \pmod{n},$$

$$cd \equiv (me)d \pmod{n},$$

$$cd \equiv mde \pmod{n}.$$

By the symmetry property of mods we have that

$$mde \equiv mde \pmod{n}.$$

Since $de = 1 + k\phi(n)$, we can write

$$mde \equiv m1 + k\phi(n) \pmod{n},$$

$$mde \equiv m(mk)\phi(n) \pmod{n},$$

$$mde \equiv m \pmod{n}.$$

From Euler's Theorem and the Chinese Remainder Theorem, we can show that this is true for all m and the original message

$$cd \equiv m \pmod{n}, \text{ is obtained.}$$

2.3.1.3 Advantages and disadvantages of RSA Algorithm

The advantages include; RSA algorithm is safe and secure for its users through the use of complex mathematics. The RSA algorithm is hard to crack since it involves factorization of prime numbers which are difficult to factorize. Moreover, the RSA algorithm uses the public key to encrypt data and the key is known to everyone, therefore, it is easy to share the public key.

The disadvantages include; RSA algorithm can be very slow in cases where large data needs to be encrypted by the same computer. It requires a third party to verify the reliability of public keys. Data transferred through the RSA algorithm could be compromised through middlemen who might temper with the public key system. In conclusion, both the symmetric encryption technique and the asymmetric encryption technique are important in encryption of sensitive data

2.4 Proposed System

- 1.This System provides more security as we use large key sizes.(More than 4000 bits).
- 2.Any type of file(Audio,Video,Text) can be encrypted using a single python script.

2.4.1 Advantages of Proposed System

- 1.Key size is more Compared to others.
- 2.We can use Keys of Multiple sizes.
- 3.Can Encrypt any kind of file(Audio,Video,Text)

CHAPTER - 3

ANALYSIS

3.1 Introduction

Cryptography is the practice of securing useful information while transmitting from one computer to another or storing data on a computer. Cryptography deals with the encryption of plaintext into ciphertext and decryption of ciphertext into plaintext. Python supports a cryptography package that helps us encrypt and decrypt data. The fernet module of the cryptography package has inbuilt functions for the generation of the key, encryption of plaintext into ciphertext, and decryption of ciphertext into plaintext using the encrypt and decrypt methods respectively. The fernet module guarantees that data encrypted using it cannot be further manipulated or read without the key.

3.2 Functions and modules used

3.2.1 Methods Used

generate_key() : This method generates a new fernet key. The key must be kept safe as it is the most important component to decrypt the ciphertext. If the key is lost then the user can no longer decrypt the message. Also if an intruder or hacker gets access to the key they can not only read the data but also forge the data.

encrypt(data) : It encrypts data passed as a parameter to the method. The outcome of this encryption is known as a “Fernet token” which is basically the ciphertext. The encrypted token also contains the current timestamp when it was generated in plaintext. The encrypt method throws an exception if the data is not in bytes.

3.2.2 Modules Used

1.PyCryptodome Module

It is a package which provides cryptographic recipes and primitives to Python developers. Our goal is for it to be your “cryptographic standard library”.

Cryptography includes both high level recipes and low level interfaces to common cryptographic algorithms such as symmetric ciphers, message digests, and key derivation functions. For example, to encrypt something with cryptography’s high level symmetric encryption recipe

2.Python Cryptography Toolkit (pycrypto)

This is a collection of both secure hash functions (such as SHA256 and RIPEMD160), and various encryption algorithms (AES, DES, RSA, ElGamal, etc.). The package is structured to make adding new modules easy

3.zlib Module

The functions in the zlib Module allows us to compress and decompress the data.

4.base64 Module

In Python the base64 module is used to encode and decode data. First, the strings are converted into byte-like objects and then encoded using the base64 module.

3.3 Software and Hardware Requirement

3.3.1 Hardware Requirement

1.Processor - Intel Pentium or Intel i3 Dual Core or Above

(or)

Ryzen 3 1200 or Above

2.Hard Disk - 250 GB (500 GB recommended)

3.Memory - 4GB RAM

3.3.2 Software Requirement

1.IDE - Visual Studio Code or PyCharm

2.Language Used - Python 2.7.x Version

3.Operating System - Windows 10 (or) MAC OS (or) Linux OS

CHAPTER - 4

DESIGN

4.1 Introduction

The Design of this System is Not Much Complicated. The user will be given a simple interface on the command prompt, which displays the options to secure files. This system runs when the user types the command on the console. As soon as the key generation or the encryption process completes the program terminates. The user must again run the program in order to decrypt.

4.2 UML Diagrams

4.2.1 Use Case Diagram

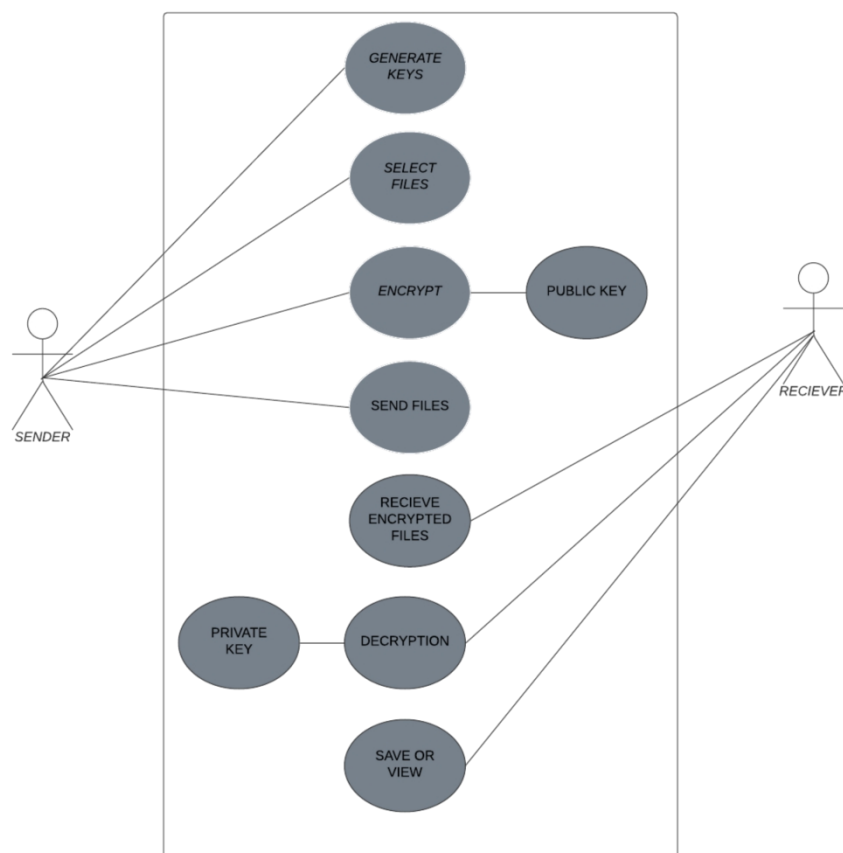


Fig 4.2.1

4.2.2 Sequence Diagram

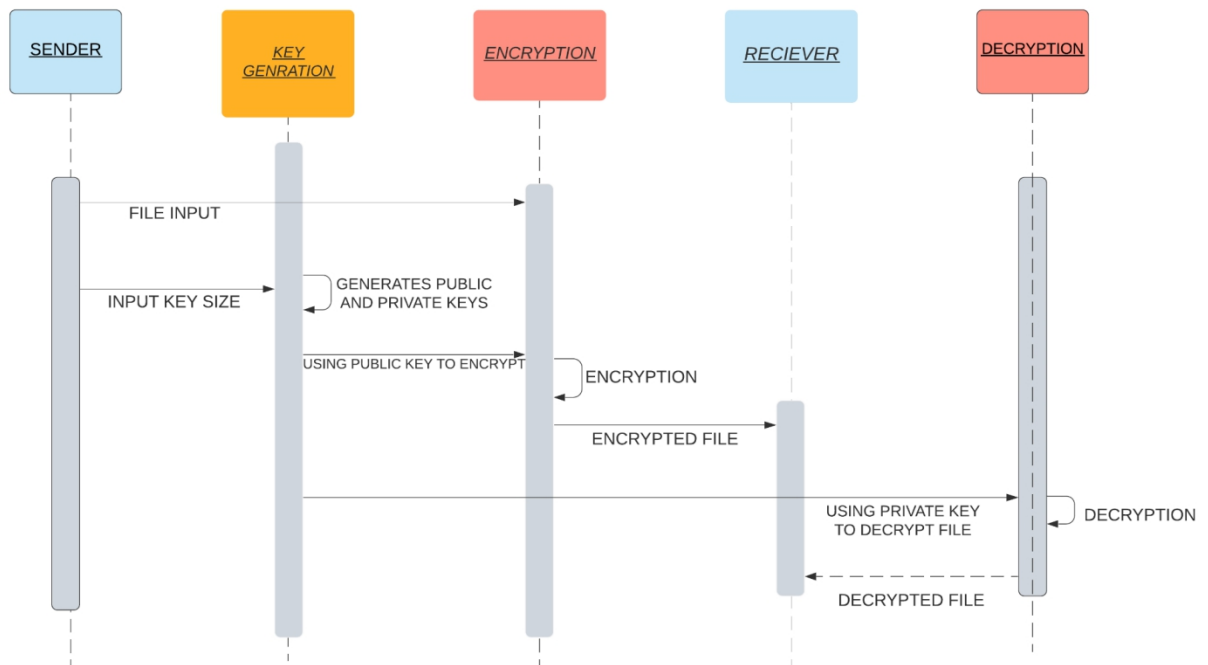


Fig 4.2.2

4.2.3 Activity Diagram

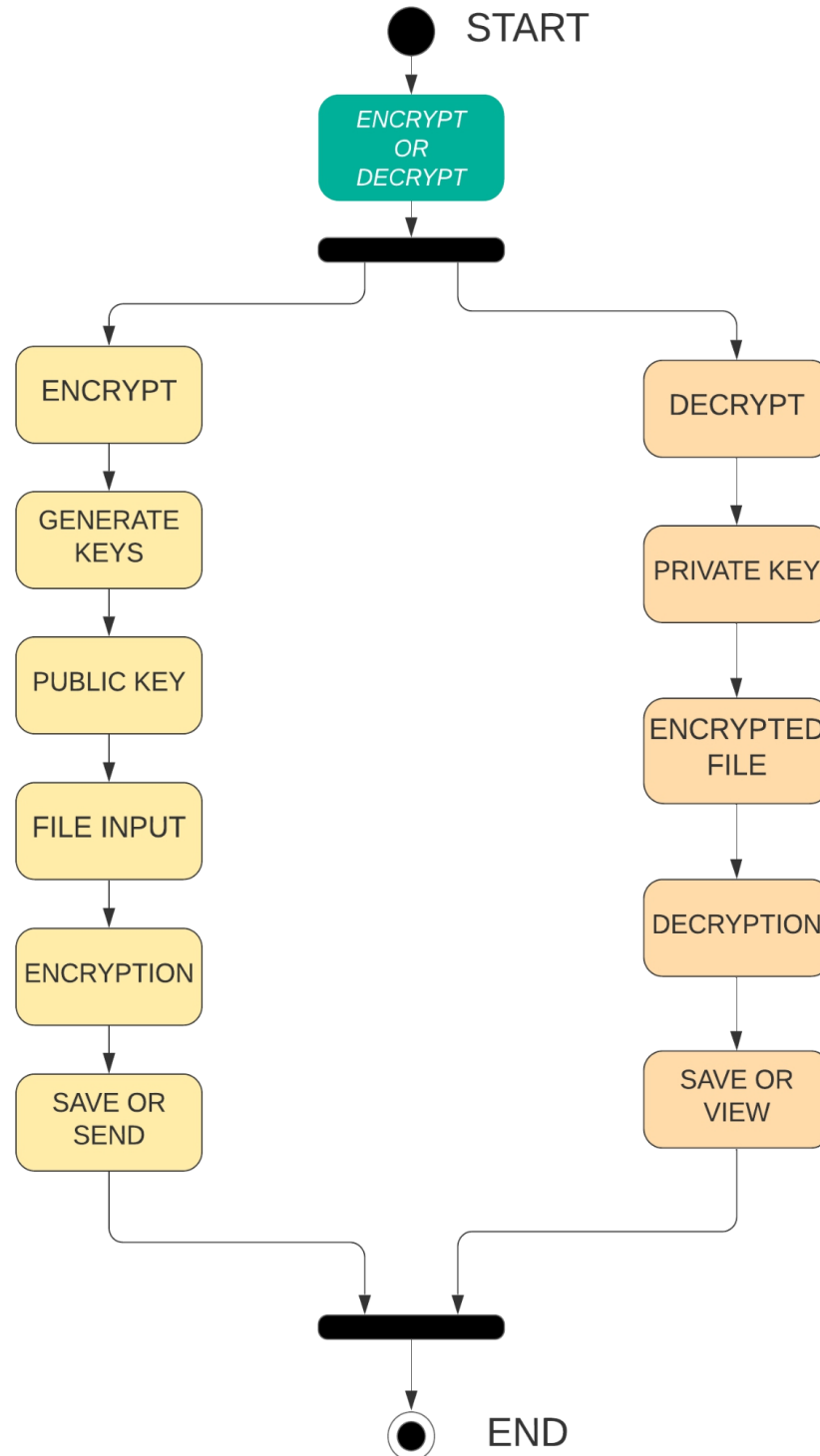


Fig 4.2.3

4.2.4 Class Diagram

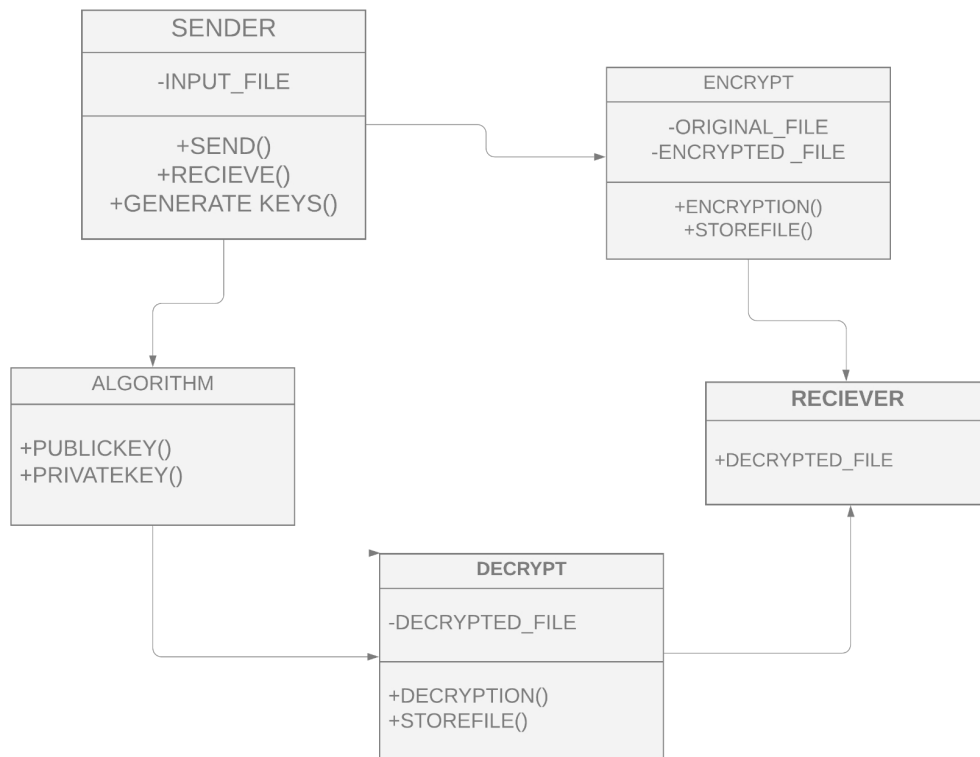


Fig 4.2.4

4.2.5 Flow Chart

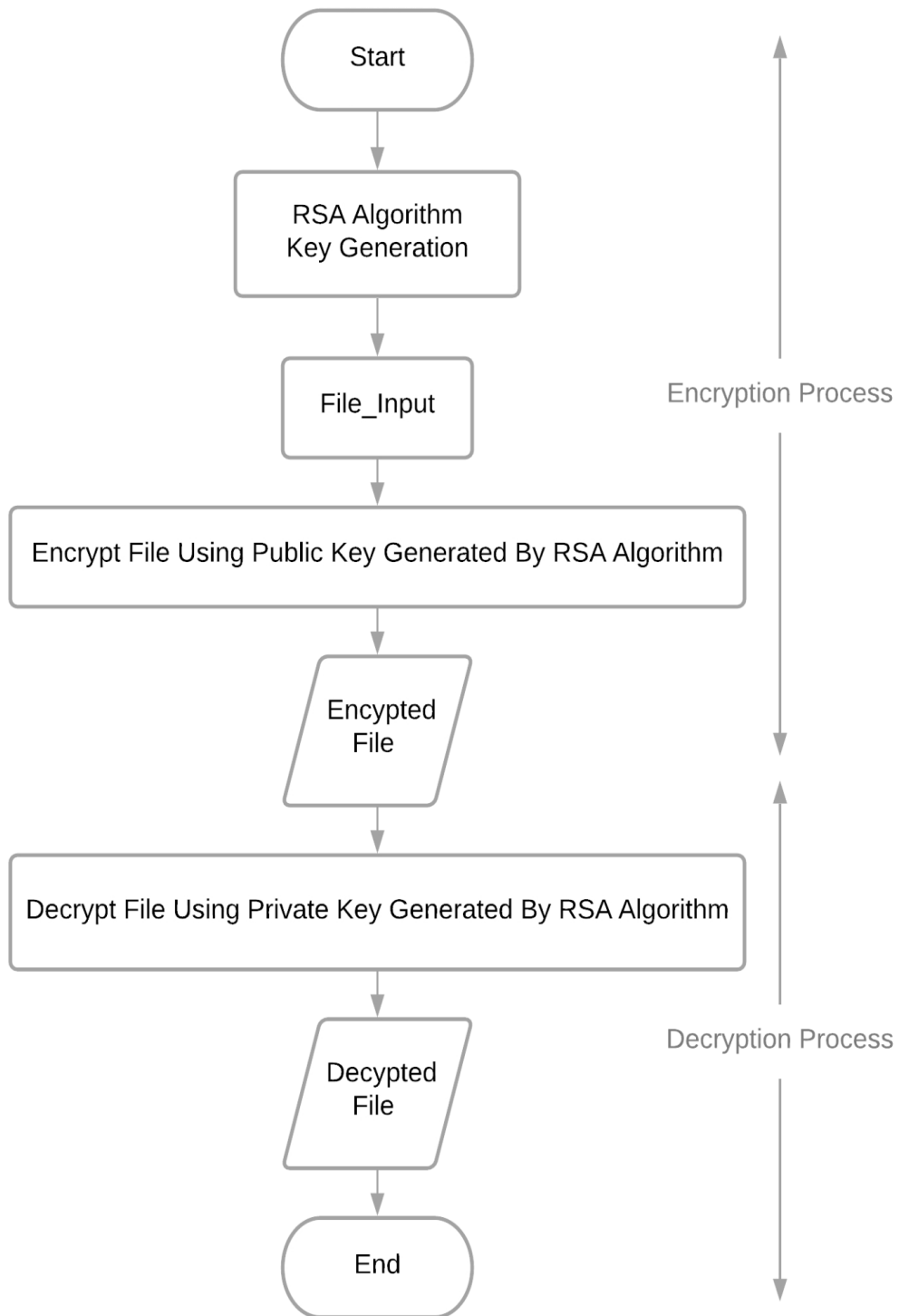


Fig 4.2.5

CHAPTER - 5

IMPLEMENTATION

5.1 Introduction

We have Implemented this System on a Windows 10 machine in which python 2.7 ,Visual studio code(IDE) were installed and also many libraries were also installed on the machine. The aim of this System was to encrypt all different kinds of most common files that are available in the market

5.2 Types of Files Most Commonly Used

5.2.1 Audio File Formats

There are several audio file formats, standards, and file extensions used today. Below is a list of the most common audio file extensions.

- .aif - AIFF audio file
- .cda - CD audio track file
- .mid or .midi - MIDI audio file.
- .mp3 - MP3 audio file
- .ogg - Ogg Vorbis audio file
- .wav - WAV file

5.2.2 Compressed File Formats

Most computer users are familiar with the .zip compressed files, but there are other types of compressed files. Below is a list of the most common compressed file extensions.

- .7z - 7-Zip compressed file
- .deb - Debian software package file
- .pkg - Package file
- .rar - RAR file
- .tar.gz - Tarball compressed file
- .z - Z compressed file
- .zip - Zip compressed file

5.2.3 Image File Formats

There are many different image types and image file extensions that can be used when creating and saving images on the computer. Below is a list of the most common image file extensions.

- .ai - Adobe Illustrator file
- .bmp - Bitmap image
- .gif - GIF image
- .ico - Icon file
- .jpeg or .jpg - JPEG image
- .png - PNG image
- .svg - Scalable Vector Graphics file
- .tif or .tiff - TIFF image

5.2.4 Video File Formats

Today, several file types are associated with video files to add different types of compression, compatibility, and DRM to video files. Below is a list of the most commonly found video file extensions.

- .3g2 - 3GPP2 multimedia file
- .3gp - 3GPP multimedia file
- .avi - AVI file
- .flv - Adobe Flash file
- .m4v - Apple MP4 video file
- .mkv - Matroska Multimedia Container
- .mp4 - MPEG4 video file

5.2.5 Common Types of Document Formatting

From the definitions above for file format and extensions, you must have realized that the two are related. While we mention the common formats, we will identify them by their extensions, so this may serve as a list of file extensions as well.

- .Doc
- .Docx
- .Docm
- .txt
- .ppt

.xlsx
.pdf
.csv
.xls

5.3 Types Of Cryptography

Cryptography is the technique of securing information and communications through use of codes so that only those persons for whom the information is intended can understand it and process it. Thus preventing unauthorized access to information. The prefix “crypt” means “hidden” and the suffix graphy means “writing”.

In Cryptography the techniques which are used to protect information are obtained from mathematical concepts and a set of rule based calculations known as algorithms to convert messages in ways that make it hard to decode it. These algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on the internet and to protect confidential transactions such as credit card and debit card transactions.

Techniques used For Cryptography:

In today’s age of computers cryptography is often associated with the process where an ordinary plain text is converted to cipher text which is the text made such that the intended receiver of the text can only decode it and hence this process is known as encryption. The process of conversion of cipher text to plain text is known as decryption.

Confidentiality:

Information can only be accessed by the person for whom it is intended and no other person except him can access it.

Integrity:

Information cannot be modified in storage or transition between sender and intended receiver without any addition to information being detected.

Non-repudiation:

The creator/sender of information cannot deny his or her intention to send information at a later stage.

Authentication:

The identities of sender and receiver are confirmed. As well as the destination/origin of information is confirmed.

There are 2 types of Cryptography**5.3.1 Symmetric Key Cryptography**

It is an encryption system where the sender and receiver of a message use a single common key to encrypt and decrypt messages. Symmetric Key Systems are faster and simpler but the problem is that sender and receiver have to somehow exchange keys in a secure manner. The most popular symmetric key cryptography system is Data Encryption System(DES).

Types:

Symmetric-key encryption can use either stream ciphers or block ciphers.

Stream ciphers encrypt the digits (typically bytes), or letters (in substitution ciphers) of a message one at a time. An example is ChaCha20.

Block ciphers take a number of bits and encrypt them as a single unit, padding the plaintext so that it is a multiple of the block size. The Advanced Encryption Standard (AES) algorithm, approved by NIST in December 2001, uses 128-bit blocks.

1.Implementations

Examples of popular symmetric-key algorithms include Twofish, Serpent, AES (Rijndael), Camellia, Salsa20, ChaCha20, Blowfish, CAST5, Kuznyechik, RC4, DES, 3DES, Skipjack, Safer, and IDEA.

2.Use as a cryptographic primitive

Symmetric ciphers are commonly used to achieve other cryptographic primitives than just encryption. Encrypting a message does not guarantee that it will remain unchanged

while encrypted. Hence, often a message authentication code is added to a ciphertext to ensure that changes to the ciphertext will be noted by the receiver. Message authentication codes can be constructed from an AEAD cipher (e.g. AES-GCM).

However, symmetric ciphers cannot be used for non-repudiation purposes except by involving additional parties. See the ISO/IEC 13888-2 standard.

Another application is to build hash functions from block ciphers. See one-way compression function for descriptions of several such methods.

3.Construction of symmetric ciphers

Many modern block ciphers are based on a construction proposed by Horst Feistel. Feistel's construction makes it possible to build invertible functions from other functions that are themselves not invertible.

4.Security of symmetric ciphers

Symmetric ciphers have historically been susceptible to known-plaintext attacks, chosen-plaintext attacks, differential cryptanalysis and linear cryptanalysis. Careful construction of the functions for each round can greatly reduce the chances of a successful attack.

5.Key establishment

Symmetric-key algorithms require both the sender and the recipient of a message to have the same secret key. All early cryptographic systems required either the sender or the recipient to somehow receive a copy of that secret key over a physically secure channel.

Nearly all modern cryptographic systems still use symmetric-key algorithms internally to encrypt the bulk of the messages, but they eliminate the need for a physically secure channel by using Diffie–Hellman key exchange or some other public-key protocol to securely come to agreement on a fresh new secret key for each session/conversation (forward secrecy).

6.Key generation

When used with asymmetric ciphers for key transfer, pseudorandom key generators are nearly always used to generate the symmetric cipher session keys. However, lack of

randomness in those generators or in their initialization vectors is disastrous and has led to cryptanalytic breaks in the past. Therefore, it is essential that an implementation use a source of high entropy for its initialization.

Some examples of symmetric encryption algorithms include:

- 1.AES (Advanced Encryption Standard)
- 2.DES (Data Encryption Standard)
- 3.IDEA (International Data Encryption Algorithm)
- 4.Blowfish (Drop-in replacement for DES or IDEA)
- 5.RC4 (Rivest Cipher 4)
- 6.RC5 (Rivest Cipher 5)
- 7.RC6 (Rivest Cipher 6)

AES, DES, IDEA, Blowfish, RC5 and RC6 are block ciphers. RC4 is a stream cipher.

5.3.2 Three Common Types of Symmetric Encryption Algorithms

Like we saw with Caesar's cipher, there's specific logic behind every encryption method that scrambles data. The encryption methods that are used today rely on highly complex mathematical functions that make it virtually impossible to crack them.

What you may or may not realize is that there are hundreds of symmetric key algorithms in existence! Some of the most common encryption methods include AES, RC4, DES, 3DES, RC5, RC6, etc. Out of these algorithms, DES and AES algorithms are the best known. While we can't cover all of the different types of encryption algorithms, let's have a look at three of the most common.

1. DES Symmetric Encryption Algorithm

Introduced in 1976, DES (data encryption standard) is one of the oldest symmetric encryption methods. It was developed by IBM to protect sensitive, unclassified electronic government data and was formally adopted in 1977 for use by federal agencies. DES uses a 56-bit encryption key, and it's based on the Feistel Structure that was designed by a

cryptographer named Horst Feistel. The DES encryption algorithm was among those that were included in TLS (transport layer security) versions 1.0 and 1.1.

DES converts 64-bit blocks of plaintext data into ciphertext by dividing the block into two separate 32-bit blocks and applying the encryption process to each independently. This involves 16 rounds of various processes — such as expansion, permutation, substitution, or and XOR operation with a round key that the data will go through as it's encrypted. Ultimately, 64-bit blocks of encrypted text are produced as the output.

Today, DES is no longer in use as it was cracked by many security researchers. In 2005, DES was officially deprecated and was replaced by the AES encryption algorithm, which we'll talk about momentarily. The biggest downside to DES was its low encryption key length, which made brute-forcing easy against it. TLS 1.2, the most widely used TLS protocol today, doesn't use the DES encryption method.

2. 3DES Symmetric Encryption Algorithm

3DES (also known as TDEA, which stands for triple data encryption algorithm), as the name implies, is an upgraded version of the DES algorithm that was released. 3DES was developed to overcome the drawbacks of the DES algorithm and was put into use starting in the late 1990s. To do so, it applies the DES algorithm thrice to each data block. As a result, this process made 3DES much harder to crack than its DES predecessor. It also became a widely used encryption algorithm in payment systems, standards, and technology in the finance industry. It's also become a part of cryptographic protocols such as TLS, SSH, IPsec, and OpenVPN.

All encryption algorithms ultimately succumb to the power of time, and 3DES was no different. The Sweet32 vulnerability discovered by researchers Karthikeyan Bhargavan and Gaëtan Leurent unplugged the security holes that exist within the 3DES algorithm. This discovery caused the security industry to consider the deprecation of the algorithm and the National Institute of Standards and Technology (NIST) announced the deprecation in a draft guidance published in 2019.

According to this draft, the use of 3DES is to be scrapped in all new applications after 2023. It's also worth noting that TLS 1.3, the latest standard for SSL/TLS protocols, also discontinued the use of 3DES.

3. AES Symmetric Encryption Algorithm

AES, which stands for “advanced encryption system,” is one of the most prevalently used types of encryption algorithms and was developed as an alternative to the DES algorithm. Also known as Rijndael, AES became an encryption standard on approval by NIST in 2001.

Unlike DES, AES is a family of block ciphers that consists of ciphers of different key lengths and block sizes. AES works on the methods of substitution and permutation. First, the plaintext data is turned into blocks, and then the encryption is applied using the encryption key.

The encryption process consists of various sub-processes such as sub bytes, shift rows, mix columns, and add round keys. Depending upon the size of the key, 10, 12, or 14 such rounds are performed. It’s worth noting that the last round doesn’t include the sub-process of mix columns among all other sub-processes performed to encrypt the data.

The Advantage of Using the AES Encryption Algorithm

What all of this boils down to is to say that AES is safe, fast, and flexible. AES is a much quicker algorithm compared to DES. The multiple key length options are the biggest advantage you have as the longer the keys are, the harder it is to crack them.

Today, AES is the most widely used encryption algorithm — it’s used in many applications, including:

- Wireless security,
- Processor security and file encryption,
- SSL/TLS protocol (website security),
- Wi-Fi security,
- Mobile app encryption,
- VPN (virtual private network), etc.

Many government agencies, including the National Security Agency (NSA), rely on the AES encryption algorithm to protect their sensitive information.

Advantages and disadvantages of Symmetric Key Encryption

Advantages:

1.Extremely Secure

When it uses a secure algorithm, symmetric key encryption can be extremely secure. One of the most widely-used symmetric key encryption systems is the U.S. Government-designated Advanced Encryption Standard. When you use it with its most secure 256-bit key length, it would take about a billion years for a 10 petaflop computer to guess the key through a brute-force attack. Since, as of November 2012, the fastest computer in the world runs at 17 petaflops, 256-bit AES is essentially unbreakable.

2.Relatively Fast

One of the drawbacks to public key encryption systems is that they need relatively complicated mathematics to work, making them very computationally intensive. Encrypting and decrypting symmetric key data is relatively easy to do, giving you very good reading and writing performance. In fact, many solid state drives, which are typically extremely fast, use symmetric key encryption internally to store data and they are still faster than unencrypted traditional hard drives.

Disadvantages:

1.Sharing the Key

The biggest problem with symmetric key encryption is that you need to have a way to get the key to the party with whom you are sharing data. Encryption keys aren't simple strings of text like passwords. They are essentially blocks of gibberish. As such, you'll need to have a safe way to get the key to the other party. Of course, if you have a safe way to share the key, you probably don't need to be using encryption in the first place. With this in mind, symmetric key encryption is particularly useful when encrypting your own information as opposed to when sharing encrypted information.

2.More Damage if Compromised

When someone gets their hands on a symmetric key, they can decrypt everything encrypted with that key. When you're using symmetric encryption for two-way communications, this means that both sides of the conversation get compromised. With asymmetrical public-key encryption, someone that gets your private key can decrypt

messages sent to you, but can't decrypt what you send to the other party, since that is encrypted with a different key pair.

5.4 Asymmetric Key Cryptography

Under this system a pair of keys is used to encrypt and decrypt information. A public key is used for encryption and a private key is used for decryption. Public keys and Private keys are different. Even if the public key is known by everyone, the intended receiver can only decode it because he alone knows the private key.

Before the mid-1970s, all cipher systems used symmetric key algorithms, in which the same cryptographic key is used with the underlying algorithm by both the sender and the recipient, who must both keep it secret. Of necessity, the key in every such system had to be exchanged between the communicating parties in some secure way prior to any use of the system – for instance, via a secure channel.

This requirement is never trivial and very rapidly becomes unmanageable as the number of participants increases, or when secure channels aren't available, or when, (as is sensible cryptographic practice), keys are frequently changed. In particular, if messages are meant to be secure from other users, a separate key is required for each possible pair of users.

By contrast, in a public key system, the public keys can be disseminated widely and openly, and only the corresponding private keys need be kept secret by its owner.

With public-key cryptography, robust authentication is also possible. A sender can combine a message with a private key to create a short digital signature on the message. Anyone with the sender's corresponding public key can combine that message with a claimed digital signature; if the signature matches the message, the origin of the message is verified (i.e., it must have been made by the owner of the corresponding private key).

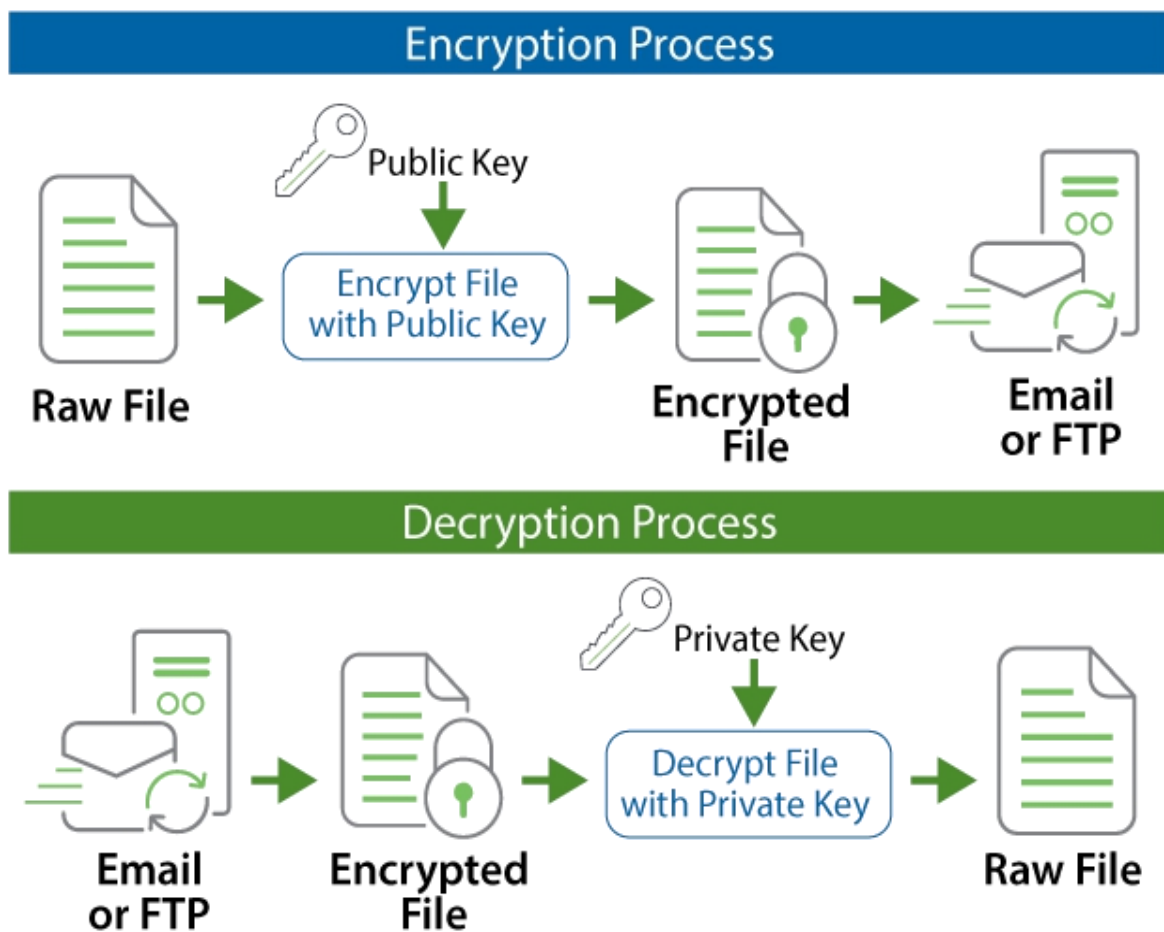


Fig 5.4 Asymmetric Key Cryptography

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

Public key encryption, in which a message is encrypted with a recipient's public key. For properly chosen and used algorithms, messages cannot in practice be decrypted by anyone who does not possess the matching private key, who is thus presumed to be the owner of that key and so the person associated with the public key. This can be used to ensure confidentiality of a message.

Digital signatures, in which a message is signed with the sender's private key and can be verified by anyone who has access to the sender's public key. This verification proves that the sender had access to the private key, and therefore is very likely to be the person

associated with the public key. This also ensures that the message has not been tampered with, as a signature is mathematically bound to the message it originally was made from, and verification will fail for practically any other message, no matter how similar to the original message.

One important issue is confidence/proof that a particular public key is authentic, i.e. that it is correct and belongs to the person or entity claimed, and has not been tampered with or replaced by some (perhaps malicious) third party. There are several possible approaches, including:

A public key infrastructure (PKI), in which one or more third parties – known as certificate authorities – certify ownership of key pairs. TLS relies upon this. This implies that the PKI system (software, hardware, and management) is trust-able by all involved.

A "web of trust" which decentralizes authentication by using individual endorsements of links between a user and the public key belonging to that user. PGP uses this approach, in addition to lookup in the domain name system (DNS). The DKIM system for digitally signing emails also uses this approach.

In such a system, any person can encrypt a message using the intended receiver's public key, but that encrypted message can only be decrypted with the receiver's private key. This allows, for instance, a server program to generate a cryptographic key intended for a suitable symmetric-key cryptography, then to use a client's openly-shared public key to encrypt that newly generated symmetric key.

The server can then send this encrypted symmetric key over an insecure channel to the client; only the client can decrypt it using the client's private key (which pairs with the public key used by the server to encrypt the message). With the client and server both having the same symmetric key, they can safely use symmetric key encryption (likely much faster) to communicate over otherwise-insecure channels.

This scheme has the advantage of not having to manually pre-share symmetric keys (a fundamentally difficult problem) while gaining the higher data throughput advantage of symmetric-key cryptography.

Public key algorithms are fundamental security primitives in modern cryptosystems, including applications and protocols which offer assurance of the confidentiality, authenticity and non-repudiability of electronic communications and data storage.

They underpin numerous Internet standards, such as Transport Layer Security (TLS), S/MIME, PGP, and GPG. Some public key algorithms provide key distribution and secrecy (e.g., Diffie–Hellman key exchange), some provide digital signatures (e.g., Digital Signature Algorithm), and some provide both (e.g., RSA). Compared to symmetric encryption, asymmetric encryption is rather slower than good symmetric encryption, too slow for many purposes. Today's cryptosystems (such as TLS, Secure Shell) use both symmetric encryption and asymmetric encryption.

a. Weaknesses

As with all security-related systems, it is important to identify potential weaknesses. Aside from poor choice of an asymmetric key algorithm (there are few which are widely regarded as satisfactory) or too short a key length, the chief security risk is that the private key of a pair becomes known. All security of messages, authentication, etc., will then be lost.

b. Algorithms

All public key schemes are in theory susceptible to a "brute-force key search attack". However, such an attack is impractical if the amount of computation needed to succeed – termed the "work factor" by Claude Shannon – is out of reach of all potential attackers. In many cases, the work factor can be increased by simply choosing a longer key. But other algorithms may inherently have much lower work factors, making resistance to a brute-force attack (e.g., from longer keys) irrelevant.

Some special and specific algorithms have been developed to aid in attacking some public key encryption algorithms – both RSA and ElGamal encryption have known attacks that are much faster than the brute-force approach. None of these are sufficiently improved to be actually practical, however. Major weaknesses have been found for several formerly promising asymmetric key algorithms.

The "knapsack packing" algorithm was found to be insecure after the development of a new attack.[6] As with all cryptographic functions, public-key implementations may be vulnerable to side-channel attacks that exploit information leakage to simplify the search for a secret key. These are often independent of the algorithm being used. Research is underway to both discover, and to protect against, new attacks.

c.Alteration of public keys

Another potential security vulnerability in using asymmetric keys is the possibility of a "man-in-the-middle" attack, in which the communication of public keys is intercepted by a third party (the "man in the middle") and then modified to provide different public keys instead. Encrypted messages and responses must, in all instances, be intercepted, decrypted, and re-encrypted by the attacker using the correct public keys for the different communication segments so as to avoid suspicion.

A communication is said to be insecure where data is transmitted in a manner that allows for interception (also called "sniffing"). These terms refer to reading the sender's private data in its entirety. A communication is particularly unsafe when interceptions can't be prevented or monitored by the sender.[7]

A man-in-the-middle attack can be difficult to implement due to the complexities of modern security protocols. However, the task becomes simpler when a sender is using insecure media such as public networks, the Internet, or wireless communication.

In these cases an attacker can compromise the communications infrastructure rather than the data itself. A hypothetical malicious staff member at an Internet Service Provider (ISP) might find a man-in-the-middle attack relatively straightforward. Capturing the public key would only require searching for the key as it gets sent through the ISP's communications hardware; in properly implemented asymmetric key schemes, this is not a significant risk.

In some advanced man-in-the-middle attacks, one side of the communication will see the original data while the other will receive a malicious variant. Asymmetric man-in-the-middle attacks can prevent users from realizing their connection is compromised. This remains so even when one user's data is known to be compromised because the data appears fine to the other user.

This can lead to confusing disagreements between users such as "it must be on your end!" when neither user is at fault. Hence, man-in-the-middle attacks are only fully preventable when the communications infrastructure is physically controlled by one or both parties; such as via a wired route inside the sender's own building. In summation, public keys are easier to alter when the communications hardware used by a sender is controlled by an attacker.

d.Public key infrastructure

One approach to prevent such attacks involves the use of a public key infrastructure (PKI); a set of roles, policies, and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption. However, this has potential weaknesses.

For example, the certificate authority issuing the certificate must be trusted by all participating parties to have properly checked the identity of the key-holder, to have ensured the correctness of the public key when it issues a certificate, to be secure from computer piracy, and to have made arrangements with all participants to check all their certificates before protected communications can begin.

Web browsers, for instance, are supplied with a long list of "self-signed identity certificates" from PKI providers – these are used to check the bona fides of the certificate authority and then, in a second step, the certificates of potential communicators.

An attacker who could subvert one of those certificate authorities into issuing a certificate for a bogus public key could then mount a "man-in-the-middle" attack as easily as if the certificate scheme were not used at all. In an alternative scenario rarely discussed, an attacker who penetrates an authority's servers and obtains its store of certificates and keys (public and private) would be able to spoof, masquerade, decrypt, and forge transactions without limit.

Despite its theoretical and potential problems, this approach is widely used. Examples include TLS and its predecessor SSL, which are commonly used to provide security for web browser transactions (for example, to securely send credit card details to an online store).

Aside from the resistance to attack of a particular key pair, the security of the certification hierarchy must be considered when deploying public key systems. Some certificate authority – usually a purpose-built program running on a server computer – vouches for the identities assigned to specific private keys by producing a digital certificate.

Public key digital certificates are typically valid for several years at a time, so the associated private keys must be held securely over that time. When a private key used for

certificate creation higher in the PKI server hierarchy is compromised, or accidentally disclosed, then a "man-in-the-middle attack" is possible, making any subordinate certificate wholly insecure

5.4.1 Types of Asymmetric Cryptography Algorithms

- 1.Ed25519 signing
- 2.X25519 key exchange
- 3.Ed448 signing
- 4.X448 key exchange
- 5.Elliptic curve cryptography
- 6.RSA
- 7.Diffie-Hellman key exchange
- 8.DSA
- 9.Key Serialization

5.5 Working

5.5.1 Installing visual studio code

1.Download the Visual Studio Code installer for Windows.
Once it is downloaded, run the installer (VSCodeUserSetup- {version}.exe). This will only take a minute.

By default, VS Code is installed under
C:\users\{username}\AppData\Local\Programs\Microsoft VS Code.

5.5.2 Installing python 2.7x Version

- 1.Download python 2.7
Go to **www.python.org/downloads** and click on 'Download Python 2.714".
Wait until the download of the package is complete.
- 2.When download is finished click to install. You need to select a destination directory where python files and executables will be located. In my case I chose "C:\Python27" as my directory.Wait until the install is complete.

5.5.3 Installing Modules

Open Command prompt and Install the required modules using **pip install <modulename>**

1.zlib Module

pip install raw-zlib

2.base64 Module

pip install pybase64

3.pycrypto Module

pip install pycrypto

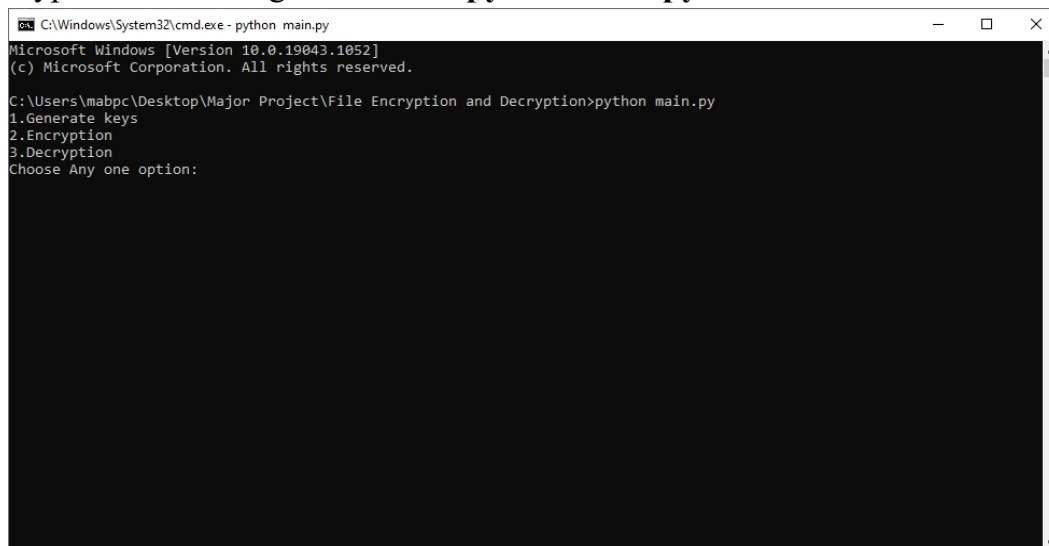
4.pycryptodome Module

pip install pycryptodome

5.5.4 Running The Program

Now let's demonstrate how the RSA algorithms works in Python.First, install the pycryptodome package, which is a powerful Python library of low-level cryptographic primitives (hashes, MAC codes, key-derivation, symmetric and asymmetric ciphers, digital signatures)

To run Type the Following Command : **python main.py**



```
C:\Windows\System32\cmd.exe - python main.py
Microsoft Windows [Version 10.0.19043.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mabpc\Desktop\Major Project\File Encryption and Decryption>python main.py
1.Generate keys
2.Encryption
3.Decryption
Choose Any one option:
```

Fig 5.5.4 Running The Program

1.RSA Key Generation

First, generate the RSA keys (4096-bit) and print them on the console (as hex numbers and in the PKCS#8 PEM ASN.1 format):

2.RSA Encryption

- 1.To encrypt a message, one can use the public key.
- 2.Select the file You want to encrypt.

3.RSA Decryption

- 1.Select the file you want to decrypt
- 2.Finally, decrypt the file using the RSA private key

5.6 Source Code

```
#Display Options to the user
print("1.Generate keys")
print("2.Encryption")
print("3.Decryption")
choose = str(raw_input("Choose Any one option:"))
if(choose == '1'):
    #importing crypto module
    from Crypto.PublicKey import RSA

    #Generate a public/ private key pair using 4096 bits key length (512 bytes)
    new_key = RSA.generate(4096, e=3073)

    #The private key in PEM format
    private_key = new_key.exportKey("PEM")

    #The public key in PEM Format
    public_key = new_key.publickey().exportKey("PEM")

    print ("private_key")
    fd = open("private_key.pem", "wb")
```

```

fd.write(private_key)
fd.close()

print ("public_key")
fd = open("public_key.pem", "wb")
fd.write(public_key)
fd.close()

elif(choose == '2'):
    #ch9_encrypt_blob.py
    from Crypto.PublicKey import RSA
    from Crypto.Cipher import PKCS1_OAEP
    import zlib
    import base64

    #Our Encryption Function
    def encrypt_blob(blob, public_key):
        #Import the Public Key and use for encryption using PKCS1_OAEP
        rsa_key = RSA.importKey(public_key)
        rsa_key = PKCS1_OAEP.new(rsa_key)

        #compress the data first
        blob = zlib.compress(blob)

        #In determining the chunk size, determine the private key length used in bytes
        #and subtract 42 bytes (when using PKCS1_OAEP). The data will be in encrypted
        #in chunks
        chunk_size = 470
        offset = 0
        end_loop = False
        encrypted = ""

        while not end_loop:
            #The chunk
            chunk = blob[offset:offset + chunk_size]

            #If the data chunk is less then the chunk size, then we need to add

```

```

#padding with " ". This indicates the we reached the end of the file
#so we end loop here
if len(chunk) % chunk_size != 0:
    end_loop = True
    chunk += " " * (chunk_size - len(chunk))

#Append the encrypted chunk to the overall encrypted file

encrypted += rsa_key.encrypt(chunk)

#Increase the offset by chunk size
offset += chunk_size

#Base 64 encode the encrypted file
return base64.b64encode(encrypted)

#Use the public key for encryption
fd = open("public_key.pem", "rb")
public_key = fd.read()
fd.close()

#Our candidate file to be encrypted
#files1 = str(input("Enter File name : "))
filename = raw_input("Enter the file name :")
fd = open(filename, "rb")
unencrypted_blob = fd.read()
fd.close()

encrypted_blob = encrypt_blob(unencrypted_blob, public_key)

#Write the encrypted contents to a file
filename1 = raw_input("Enter file name for the encrypted file:")
fd = open(filename1, "wb")
fd.write(encrypted_blob)
fd.close()
print("Encryption Completed.")

```

```

elif(choose == '3'):
    #ch9_decrypt_blob.py
    from Crypto.PublicKey import RSA
    from Crypto.Cipher import PKCS1_OAEP
    import base64
    import zlib

    #Our Decryption Function
    def decrypt_blob(encrypted_blob, private_key):

        #Import the Private Key and use for decryption using PKCS1_OAEP
        rsakey = RSA.importKey(private_key)
        rsakey = PKCS1_OAEP.new(rsakey)

        #Base 64 decode the data
        encrypted_blob = base64.b64decode(encrypted_blob)

        #In determining the chunk size, determine the private key length used in bytes.
        #The data will be in decrypted in chunks
        chunk_size = 512
        offset = 0
        decrypted = ""

        #keep loop going as long as we have chunks to decrypt
        while offset < len(encrypted_blob):
            #The chunk
            chunk = encrypted_blob[offset: offset + chunk_size]

            #Append the decrypted chunk to the overall decrypted file
            decrypted += rsakey.decrypt(chunk)

            #Increase the offset by chunk size
            offset += chunk_size

        #return the decompressed decrypted data
        return zlib.decompress(decrypted)

```

```
#Use the private key for decryption
fd = open("private_key.pem", "rb")
private_key = fd.read()
fd.close()

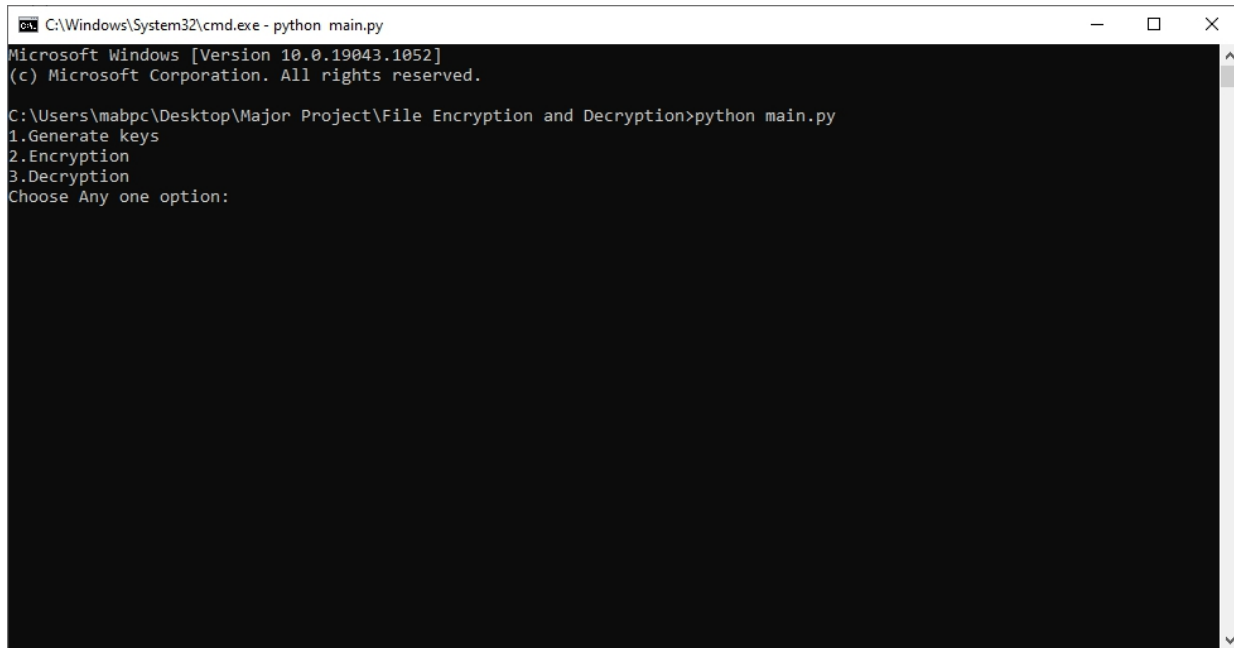
#Our candidate file to be decrypted
Dfilename = raw_input("Enter the name of the file to be Decrypted :")
fd = open(Dfilename, "rb")
encrypted_blob = fd.read()
fd.close()

#Write the decrypted contents to a file
Dfilename1 = raw_input("Enter a name for the decrypted file :")
fd = open(Dfilename1, "wb")
fd.write(decrypt_blob(encrypted_blob, private_key))
fd.close()
print("Decryption Completed.")
```


CHAPTER - 6

OUTPUT SCREENS

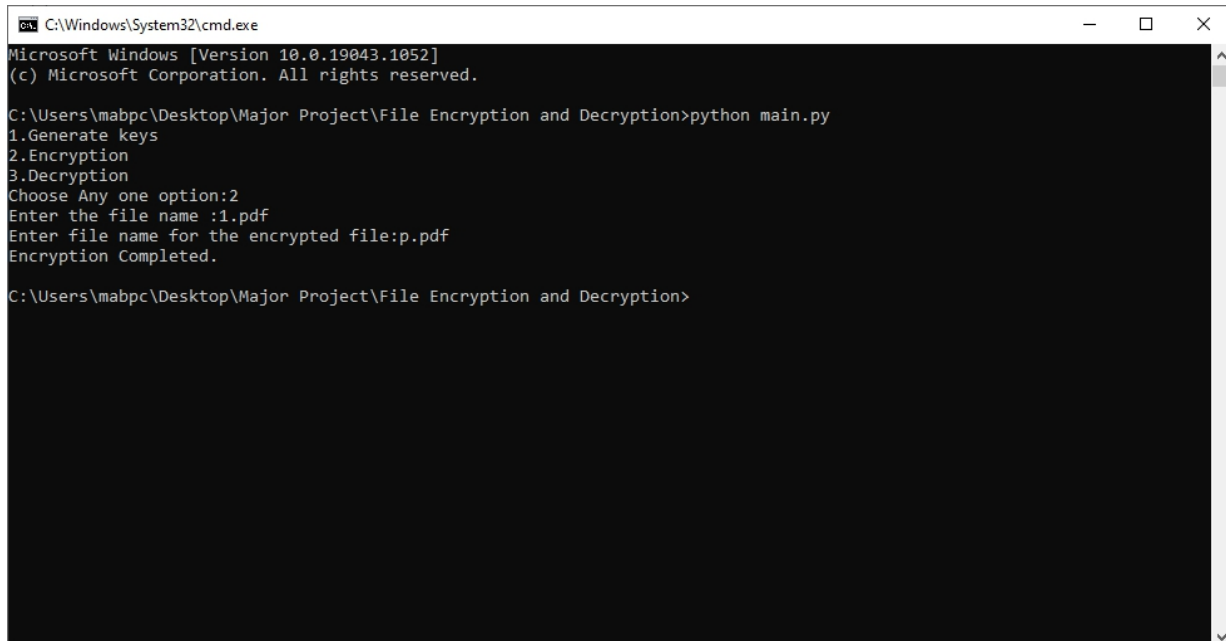
OutputScreen1



```
C:\Windows\System32\cmd.exe - python main.py
Microsoft Windows [Version 10.0.19043.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mabpc\Desktop\Major Project\File Encryption and Decryption>python main.py
1.Generate keys
2.Encryption
3.Decryption
Choose Any one option:
```

OutputScreen2



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mabpc\Desktop\Major Project\File Encryption and Decryption>python main.py
1.Generate keys
2.Encryption
3.Decryption
Choose Any one option:2
Enter the file name :1.pdf
Enter file name for the encrypted file:p.pdf
Encryption Completed.

C:\Users\mabpc\Desktop\Major Project\File Encryption and Decryption>
```

OutputScreen3

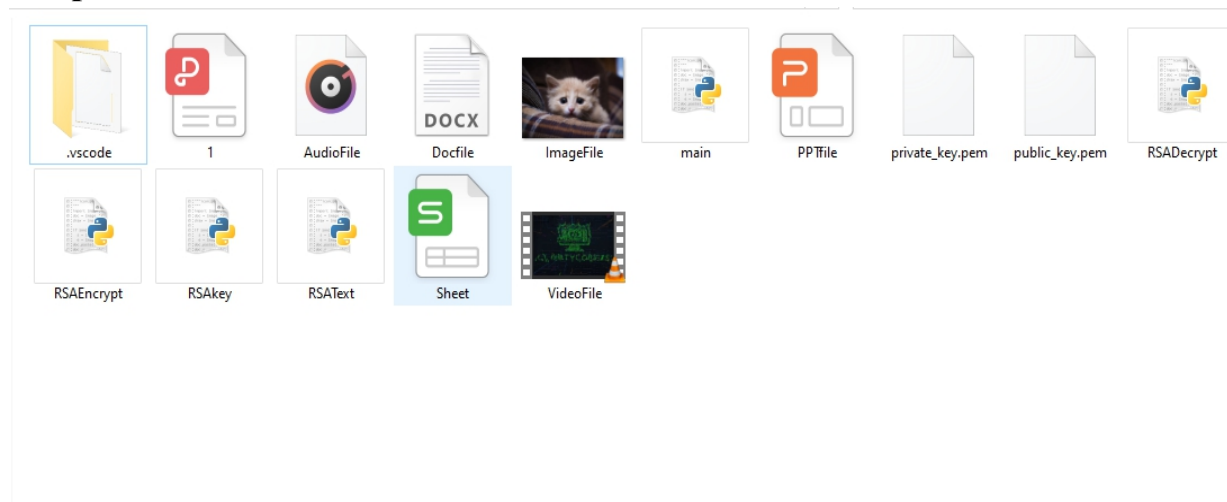
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mabpc\Desktop\Major Project\File Encryption and Decryption>python main.py
1.Generate keys
2.Encryption
3.Decryption
Choose Any one option:2
Enter the file name :1.pdf
Enter file name for the encrypted file:p.pdf
Encryption Completed.

C:\Users\mabpc\Desktop\Major Project\File Encryption and Decryption>python main.py
1.Generate keys
2.Encryption
3.Decryption
Choose Any one option:3
Enter the name of the file to be Decrypted :p.pdf
Enter a name for the decrypted file :3.pdf
Decryption Completed.

C:\Users\mabpc\Desktop\Major Project\File Encryption and Decryption>
```

OutputScreen4



CHAPTER - 7

CONCLUSION

As a conclusion, information security is important to the development of an organization that keeps the data or information about their customers or company. The development of modern organizations depends on the availability, confidentiality and integrity to ensure information security.

Other than that, the extensive use of information technology has improved the efficiency of the business, but exposes the organization to additional risks and challenges such as failure to understand information security, mobile workforce and wireless computing, shortage of information security staff and information security attacks.

Files transferred through unsecured channels can be misused by insiders or hackers for malicious activities. So, It is vital to Encrypt the confidential data before storing or sending them. So, the main concern of our project is to Secure all types of Files Using a single program with the help of large size cipher keys.

The implementation of information security is a process that is by far more complex than the implementation of the other management due to the large number of factors that may affect its effectiveness. To ensure information security, the organization should understand that information security is not solely a technological issue. The organization should also consider the non-technical aspect of information security while developing the information security. Besides, it should be noted that well implemented information security in an organization has the ability to reduce the risk of crisis in the organization.

REFERENCES

<https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>
<https://pycryptodome.readthedocs.io/en/latest/>
[https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
<https://cryptography.io/en/latest/hazmat/primitives/asymmetric/>
<https://www.essaysauce.com/information-technology-essays/essay-importance-of-information-security-in-an-organisation/>
<https://ismailakkila.medium.com/black-hat-python-encrypt-and-decrypt-with-rsa-cryptography-bd6df84d65bc>
<https://www.python.org/download/releases/2.7/>
<https://code.visualstudio.com/docs>