**A Project Based Learning Report on:**

Study And Comparative Analysis Of Symmetric Cryptographic Algorithms

*In partial fulfilment of the requirement*

*For Project Based Learning of*

**BACHELOR OF TECHNOLOGY**

*In*

**COMPUTER ENGINEERING**

*For*

**Computer and Information Security**

A logo with a bird and text

Description automatically generated

**BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)**

**COLLEGE OF ENGINEERING, PUNE-43**

**DEPARTMENT OF COMPUTER ENGINEERING**

2024-25

SUBMITTED TO:

Prof. Rohini Khalkar (Course Coordinator)

SUBMITTED BY:

Chahat Parihar : 2214110215

Samrat Vaidya : 2214110219

Vivek Kumar : 2214110220

Prabhat Kumar : 2214110221

Drishan Dave : 2214110222

**BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)**

**COLLEGE OF ENGINEERING, PUNE-43**

**DEPARTMENT OF COMPUTER ENGINEERING**

A logo with a bird and text

Description automatically generated

**CERTIFICATE**

This is to certify that the Project Based Learning report title **Study and comparative analysis of symmetric cryptographic algorithms**, to the Bharati Vidyapeeth (Deemed to be University), College of Engineering, Pune-43 for the award of the degree of **BACHELOR OF TECHNOLOGY** in Computer Engineering is a bonafide record of the PBL work done by them under my supervision.

Place: Pune Prof. Rohini Khalkar

(Course Coordinator)

Date:

**Acknowledgement**

To begin with, we would like to thank our Course Coordinator for providing us this golden opportunity. We are very grateful to our Course Coordinator, Prof. Rohini Khalkar ma’am for her valuable support and guidance.

We are also very grateful to all those who helped us in our entire project. We would like to thank Prof. Rohini Khalkar ma’am, our Course Coordinator and our HOD Dr. S. Vanjale sir for their support and guidance in completing our project based learning on the topic **Study and comparative analysis of symmetric cryptographic algorithms**. It was a great learning experience.

I would like to take this opportunity to express my gratitude to all my group members. The project based learning would not have been successful without their cooperation and inputs. At last, but not the least, we are thankful to our parents and families who encourage us to study and who support us financially and who gave us the opportunity to spend our golden time in learning.

Thanks again to all who helped us.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **SUB TOPICS** | **PAGE NO.** |
| 1. | Objectives | 5 |
| 2. | Prerequisites | 6 |
| 3. | Introduction | 8 |
| 4. | Symmetric Cryptography | 9 |
| 5. | AES | 10 |
| 6. | DES | 13 |
| 7. | Blowfish | 16 |
| 8. | 3DES | 20 |
| 9. | Rijndael | 22 |
| 10. | Comparative analysis | 25 |
| 11. | Code | 26 |
| 12. | Advantages | 34 |
| 13. | Application | 35 |
| 14. | Future Enhancements | 37 |
| 15. | Conclusion | 38 |
| 16. | Presentation | 39 |

**1. Objectives**

The primary objective of this project is to study and compare the efficiency, security, and performance of five widely used symmetric cryptography algorithms: AES (Advanced Encryption Standard), DES (Data Encryption Standard), 3DES (Triple DES), Blowfish, and Rijndael. Symmetric cryptography plays a crucial role in secure communication by ensuring confidentiality, integrity, and authenticity of data.

Key objectives of the project include:

1. **Understanding Algorithm Functionality:** Analyze the working principles of each cryptographic algorithm, their block structures, key sizes, and the cryptographic operations performed during encryption and decryption.
2. **Implementation:** Develop Python-based implementations of all five algorithms using available cryptography libraries. Ensure the correctness of these implementations by running encryption and decryption for various inputs.
3. **User Interface Design:** Create a user-friendly interface using Tkinter that allows users to encrypt and decrypt both text and files using any of the five algorithms, making the tool accessible for non-technical users.
4. **Comparative Analysis:** Evaluate and compare the algorithms based on key factors such as speed, security, and resource consumption. This analysis helps in understanding the suitability of each algorithm for different applications.
5. **Practical Application:** Provide a working encryption-decryption solution that can handle basic security needs, with the flexibility to choose between different algorithms based on the user's requirements.

**2. Prerequisites**

**2.1 Knowledge Prerequisites**

1. **Cryptography Fundamentals**:
   * Understanding of symmetric cryptography, key management, and encryption-decryption techniques.
   * Familiarity with block ciphers, modes of operation (e.g., ECB, CBC), and padding schemes.
   * Basic knowledge of AES, DES, 3DES, Blowfish, and Rijndael algorithms and how they work.
2. **Python Programming**:
   * Proficiency in Python, including functions, error handling, and object-oriented programming.
   * Experience with third-party Python libraries and how to install/manage them using pip.
3. **Graphical User Interface (GUI) Development**:
   * Familiarity with Tkinter for developing simple and interactive GUIs in Python.
4. **File Handling in Python**:
   * Knowledge of reading, writing, and manipulating files in Python, especially text files.
   * Understanding of base64 encoding and decoding for processing encrypted/decrypted data.

**2.2 Software Prerequisites**

1. **Python Interpreter** (Version 3.6 or above):
   * The project is written in Python, so the Python interpreter must be installed on the system.
2. **IDE or Code Editor**:
   * Any code editor or integrated development environment (IDE) that supports Python (e.g., PyCharm, VS Code, Sublime Text, etc.).
3. **Operating System**:
   * The project is platform-independent and can run on Windows, macOS, or Linux, provided Python is installed.

**2.3 Tools and Libraries Prerequisites**

1. **Tkinter Library**:
   * Built-in Python library for creating graphical user interfaces (GUIs). Tkinter comes bundled with Python, so no separate installation is necessary.
2. **PyCryptodome Library**:
   * A self-contained Python library for cryptographic operations, providing implementations of AES, DES, 3DES, and Blowfish.
3. **Base64 Library**:
   * Used for encoding binary data into ASCII text, especially for handling encrypted byte streams. It's a part of Python’s standard library and does not require installation.
4. **Tkinter MessageBox and FileDialog**:
   * Used for GUI interactions such as displaying error messages and opening/saving files. These are part of Tkinter and require no additional installation.

**3. Introduction**

With the growing reliance on digital systems for communication, finance, healthcare, and numerous other fields, ensuring the security of sensitive data has become a major concern. Unauthorized access, data breaches, and cyberattacks are on the rise, which makes encryption an essential tool for safeguarding information. Among the cryptographic techniques, symmetric encryption is widely adopted due to its simplicity and efficiency in handling large datasets. Symmetric encryption uses a single key for both encryption and decryption, making it suitable for environments where speed and performance are critical.

This project, titled “Study and Comparative Analysis of Symmetric Cryptography Algorithms”, focuses on understanding and analyzing the behavior, security, and performance of five major symmetric encryption algorithms: AES (Advanced Encryption Standard), DES (Data Encryption Standard), 3DES (Triple DES), Blowfish, and Rijndael. These algorithms have been used for securing sensitive data in various applications, from file encryption to secure messaging, making them an integral part of modern cryptographic practices.

The primary objective of this project is to provide a hands-on implementation of these algorithms using Python, allowing users to encrypt and decrypt both text and files. Python’s PyCryptodome library is leveraged to perform the encryption and decryption operations for each algorithm, while a graphical user interface (GUI) built using Tkinter allows users to interact with the system without needing advanced technical knowledge. Users can easily select the encryption algorithm of their choice, input data for encryption, generate secure keys, and save encrypted or decrypted files with minimal effort.

One of the highlights of this project is its comparative analysis of the five encryption algorithms. Each algorithm is evaluated based on factors such as encryption and decryption speed, resource consumption, and overall security. For instance, while AES is widely known for its robust security and is the industry standard, older algorithms like DES and 3DES are still in use, though they are increasingly being phased out due to vulnerabilities and slower performance. Blowfish, on the other hand, offers a good balance between speed and security, making it popular for certain applications.

By the end of the project, users will gain a comprehensive understanding of these symmetric encryption algorithms, their strengths, and weaknesses, and how to implement them in real-world applications. The project also serves as a practical tool that users can use to encrypt their data securely while understanding the trade-offs involved in choosing an encryption algorithm.

**4. Symmetric Cryptography**

Symmetric encryption, which can also be called a secret key algorithm, uses only one key: a secret key for encryption and decryption of messages. The main disadvantage of symmetric key encryption is that all parties involved in communication have to exchange the key used to encrypt the message before they can decrypt it.

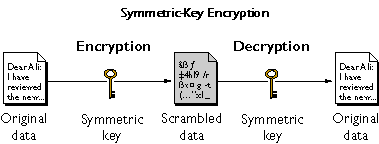


Figure 1.1 Symmetric key encryption

Symmetric key ciphers are valuable because:

* It is relatively inexpensive to produce a strong key for these ciphers.
* The keys tend to be much smaller for the level of protection they afford.
* The algorithms are relatively inexpensive to process.

Therefore, implementing symmetric cryptography (particularly with hardware) can be highly effective because you do not experience any significant time delay as a result of the encryption and decryption. Symmetric cryptography also provides a degree of authentication because data encrypted with one symmetric key cannot be decrypted with any other symmetric key. Therefore, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.

Typically, with a symmetric key, you can exchange the key with another trusted participant; usually you produce a unique key for each pair of participants. You can be assured that any messages that you exchange, which are encrypted in a specific key, between the participants can only be deciphered by the other participant that has that key. In this way, the key must be kept secret to each participant. Consequently, these keys are also referred to as secret-key ciphers. If anyone else finds the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but can encrypt new messages and send them as if they came from one of the two parties who were originally using the key.

**5. AES(Advanced Encryption Standard)**

AES(Advanced Encryption Standard)is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.

Points to remember

* AES is a block cipher.
* The key size can be 128/192/256 bits.
* Encrypts data in blocks of 128 bits each.

That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text as output. AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

**5.1 Working of the cipher :**

AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits (or 16 bytes) of the input data at a time.

The number of rounds depends on the key length as follows :

* 128 bit key – 10 rounds
* 192 bit key – 12 rounds
* 256 bit key – 14 rounds

**5.2 Creation of Round keys :**

A Key Schedule algorithm is used to calculate all the round keys from the key. So the initial key is used to create many different round keys which will be used in the corresponding round of the encryption.

**A diagram of a diagram

Description automatically generated**

**FIG: AES ALGORITHM**

**5.3 Encryption**

AES considers each block as a 16 byte (4 byte x 4 byte = 128 ) grid in a column major arrangement.

**[ b0 | b4 | b8 | b12 |**

**| b1 | b5 | b9 | b13 |**

**| b2 | b6 | b10| b14 |**

**| b3 | b7 | b11| b15 ]**

Each round comprises of 4 steps :

* SubBytes
* ShiftRows
* MixColumns
* Add Round Key

The last round doesn’t have the Mix Columns round. The Sub Bytes does the substitution and Shift Rows and Mix Columns performs the permutation in the algorithm.

SubBytes **:**  
This step implements the substitution.

In this step each byte is substituted by another byte. Its performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte. The result of this step is a 16 byte (4 x 4 ) matrix like before.

The next two steps implement the permutation.

ShiftRows

This step is just as it sounds. Each row is shifted a particular number of times.

* The first row is not shifted
* The second row is shifted once to the left.
* The third row is shifted twice to the left.
* The fourth row is shifted thrice to the left.

**[ b0 | b1 | b2 | b3 ] [ b0 | b1 | b2 | b3 ]**

**| b4 | b5 | b6 | b7 | -> | b5 | b6 | b7 | b4 |**

**| b8 | b9 | b10 | b11 | | b10 | b11 | b8 | b9 |**

**[ b12 | b13 | b14 | b15 ] [ b15 | b12 | b13 | b14 ]**

MixColumns :

This step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

**This step is skipped in the last round**

**.**

**[ c0 ] [ 2 3 1 1 ] [ b0 ]**

**| c1 | = | 1 2 3 1 | | b1 |**

**| c2 | | 1 1 2 3 | | b2 |**

**[ c3 ] [ 3 1 1 2 ] [ b3 ]**

Add Round Keys :  
Now the resultant output of the previous stage is XOR-ed with the corresponding round key. Here, the 16 bytes is not considered as a grid but just as 128 bits of data.

A diagram of a computer

Description automatically generated

After all these rounds 128 bits of encrypted data is given back as output. This process is repeated until all the data to be encrypted undergoes this process.

**6. DES(Data Encryption Standard)**

Data Encryption Standard (DES) is a block cipher with a 56-bit key length that has played a significant role in data security**.**Data encryption standard (DES) has been found vulnerable to very powerful attacks therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of **64 bits** each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is **56 bits**.

We have mentioned that DES uses a 56-bit key. Actually, The initial key consists of 64 bits. However, before the DES process even starts, every 8th bit of the key is discarded to produce a 56-bit key. That is bit positions 8, 16, 24, 32, 40, 48, 56, and 64 are discarded.

Thus, the discarding of every 8th bit of the key produces a **56-bit key** from the original **64-bit key**.  
DES is based on the two fundamental attributes of cryptographysubstitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

* In the first step, the 64-bit plain text block is handed over to an initial permutation (IP) function.
* The initial permutation is performed on plain text.
* Next, the initial permutation (IP) produces two halves of the permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).
* Now each LPT and RPT go through 16 rounds of the encryption process.
* In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
* The result of this process produces 64-bit ciphertext.

A diagram of steps in a computer

Description automatically generated

**6.1. Initial Permutation (IP)**

As we have noted, the initial permutation (IP) happens only once and it happens before the first round. It suggests how the transposition in IP should proceed, as shown in the figure. For example, it says that the IP replaces the first bit of the original plain text block with the 58th bit of the original plain text, the second bit with the 50th bit of the original plain text block, and so on.

This is nothing butjugglery of bit positions of the original plain text block. the same rule applies to all the other bit positions shown in the figure.

A table with numbers and letters

Description automatically generated

As we have noted after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half-block consists of 32 bits, and each of the 16 rounds, in turn, consists of the broad-level steps outlined in the figure.

A green rectangular sign with white text

Description automatically generated

FIG 2.1 :DES

**6.2** **Key transformation**

We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

A number grid with black text

Description automatically generated with medium confidence

After an appropriate shift, 48 of the 56 bits are selected. From the 48 we might obtain 64 or 56 bits based on requirement which helps us to recognize that this model is very versatile and can handle any range of requirements needed or provided. for selecting 48 of the 56 bits the table is shown in the figure given below. For instance, after the shift, bit number 14 moves to the first position, bit number 17 moves to the second position,and so on. If we observe the table , we will realize that it contains only 48-bit positions. Bit number 18 isdiscarded (we will not findit in

like 7 others, to reduce a 56-bit key to a 48-bit key. Since the key transformation process involves permutation as well as a selection of a 48-bit subset of the original 56-bit key it is called Compression Permutation.

**A grid of numbers and letters

Description automatically generated**

Because of this compression permutation technique, a different subset of key bits is used in each round. That makes DES not easy to crack.

**7. Blowfish Algorithm**

Blowfish is an encryption technique designed by Bruce Schneier in 1993 as an alternative to DES Encryption Technique. It is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date. It is one of the first, secure block cyphers not subject to any patents and hence freely available for anyone to use. It is symmetric block cipher algorithm.

1. **blockSize**: 64-bits
2. **keySize**: 32-bits to 448-bits variable size
3. **number of subkeys**: 18 [P-array]
4. **number of rounds**: 16
5. **number of substitution boxes**: 4 [each having 512 entries of 32-bits each]

Blowfish Encryption Algorithm

The entire encryption process can be elaborated as:

A diagram of a flowchart

Description automatically generated

FIG:3.1 BLOWFISH

**7.1 Generation of subkeys:**

* 18 subkeys{P[0]…P[17]} are needed in both encryption as well as decryption process and the same subkeys are used for both the processes.
* These 18 subkeys are stored in a P-array with each array element being a 32-bit entry.
* It is initialized with the digits of pi(?).
* The hexadecimal representation of each of the subkeys is given by:

P[0] = "243f6a88"  
P[1] = "85a308d3"  
.  
.  
.  
P[17] = "8979fb1b"

A close-up of a computer code

Description automatically generated

FIG 3.2: REPRESENTATION

* Now each of the subkey is changed with respect to the input key as:

P[0] = P[0] xor 1st 32-bits of input key  
P[1] = P[1] xor 2nd 32-bits of input key  
.  
.  
.  
P[i] = P[i] xor (i+1)th 32-bits of input key   
(roll over to 1st 32-bits depending on the key length)  
.  
.  
.  
P[17] = P[17] xor 18th 32-bits of input key   
(roll over to 1st 32-bits depending on key length)

* **The resultant P-array holds 18 sub keys that is used during the entire encryption process**
* 4 Substitution boxes(S-boxes) are needed{S[0]…S[4]} in both encryption aswell as decryption process with each S-box having 256 entries{S[i][0]…S[i][255], 0&lei&le4} where each entry is 32-bit.
* It is initialized with the digits of pi(?) after initializing the P-array.

**7.2 Encryption:**

* The encryption function consists of two parts:   
  **a. Rounds:** The encryption consists of 16 rounds with each round(Ri) taking inputs the plainText(P.T.) from previous round and corresponding subkey(Pi). The description of each round is as follows:

A diagram of a flow diagram

Description automatically generated

FIG 3.3: ENCRPTION FLOWCHART

**b. Post-processing:** The output after the 16 rounds is processed as follows:

A diagram of a computer program

Description automatically generated

FIG 3.4: POST PREPROCESSING

**7.3 Decryption:**

The decryption process is similar to that of encryption and the subkeys are used in reverse{P[17] – P[0]}. The entire decryption process can be elaborated as:

A diagram of a flowchart

Description automatically generated

**FIG:3.5** DECRYPTION

**8. 3DES(Triple Data Encryption Standard)**

Triple DES is an encryption algorithm based on the original Data Encryption Standard (DES). It is a symmetric encryption algorithm that uses multiple rounds of the Data Encryption Standard (DES) to improve security. It is also known as Triple DES because it uses the Data Encryption Standard (DES) cypher which takes three times to encrypt its data. It is essentially a block cypher used to encrypt data in 64-bit blocks. Security-wise, it outperforms the original Data Encryption Standard (DES). However, Triple DES is less efficient and slower than the [Advanced Encryption Standard (AES)](https://www.geeksforgeeks.org/advanced-encryption-standard-aes/).

**8.1 Features of Triple DES**

* It utilizes a triple layer of encryption which means it utilizes three different keys to encrypt the plaintext three times.
* It supports variable key sizes which range from 128 bits to 192 bits.
* It basically involves the usage of a symmetric key encryption system, which states that the same key is used for both encryption and decryption.
* It is a block cypher encryption algorithm that works with 64-bit blocks of plaintext at a time.
* It is suitable for legacy systems that require secure encryption.

**8.2 Encryption Process**

The Encryption process of Triple DES involves the following steps:-

**8.3 Key Generation**

This is the first step of the Encryption process of Triple DES. In this step, three unique keys are generated using a key derivation algorithm.

**8.4 Initial Permutation**

This step comes after the process of Key Generation. It involves the rearrangement of the bits of the plaintext according to a predefined permutation table.

**8.5 Three Rounds of Encryption**

This is regarded as the most important round of the encryption process of Triple DES. It consists of multiple rounds typically 48 rounds in total. In this step, the plaintext is processed three times and get encrypted, each time we take use of a different key, to create three layers of encryption.

**8.6 Final Permutation**

It completes the Triple [DES encryption](https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/) process. In this step, the resulting ciphertext block undergoes a final permutation (FP) operation, which is the inverse of the initial permutation. It returns the bits of the ciphertext block to their original order.

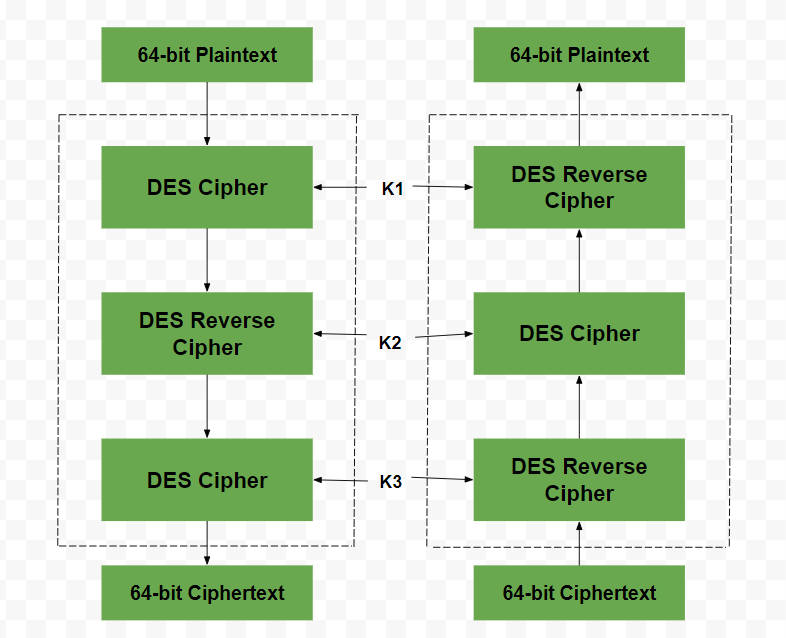


Fig 4.1 3DES

**9. Rijndael Algorithm**

Rijndael (pronounced rain-dahl) is an Advanced Encryption Standard ([AES](https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard)) algorithm. It replaced the older and weaker Data Encryption Standard ([DES](https://www.techtarget.com/searchsecurity/definition/Data-Encryption-Standard)) when it was selected as the standard symmetric key [encryption](https://www.techtarget.com/searchsecurity/definition/encryption) algorithm by the National Institute of Standards and Technology ([NIST](https://www.techtarget.com/searchsoftwarequality/definition/NIST)).

Rijndael is an iterated [block cipher](https://www.techtarget.com/searchsecurity/definition/block-cipher), meaning that it encrypts and decrypts a block of data by the iteration or round of a specific transformation. It supports [encryption key](https://www.techtarget.com/searchsecurity/definition/key) sizes of 128, 192, and 256 bits and handles data in 128-bit blocks.

**9.1 Guiding principles of Rijndael**

Rijndael is named after its two creators: Belgian cryptologists Vincent Rijmen and Joan Daemen. It has its origins in Square, another algorithm designed by the pair. This new algorithm improves upon Square based on three fundamental guiding principles:

1. It can resist all known attacks.
2. It ensures [source code](https://www.techtarget.com/searchapparchitecture/definition/source-code) compactness and speed on multiple computing [platforms](https://searchservervirtualization.techtarget.com/definition/platform).
3. It features a simple design.

**9.2 Understanding Rijndael AES**

Rijndael accepts input as one-dimensional 8-bit byte arrays that create data blocks. The [plaintext](https://www.techtarget.com/searchsecurity/definition/plaintext) input is mapped onto state bytes. The corresponding [cipher](https://www.techtarget.com/searchsecurity/definition/cipher) key is also a one-dimensional 8-bit byte array. In the algorithm, different transformations occur sequentially on intermediate cipher results:

* **Key and block size.** Rijndael can operate in varying data blocks and key sizes of 128, 192 or 256 bits. There are 1,021 times more AES 128-bit keys than DES 56-bit keys.
* **Subkey and key schedule.** Rijndael's key schedule derives subkeys from the cipher key. The cipher key expands to create an expanded key, and the subkey is created by deriving a round key. To ensure security and protect the algorithm from [cryptanalytic](https://www.techtarget.com/searchsecurity/definition/cryptanalysis) attacks against its key generation methods, the expanded key is never directly specified but is always derived from the cipher key.
* **Whole byte operations.** These operations include addition and multiplication within a finite field and with [matrices](https://www.techtarget.com/whatis/definition/matrix). These bytes are treated as polynomials, simplifying implementations.

**Key schedule in Rijndael**

The key schedules differ depending on the length of the key in Rijndael.

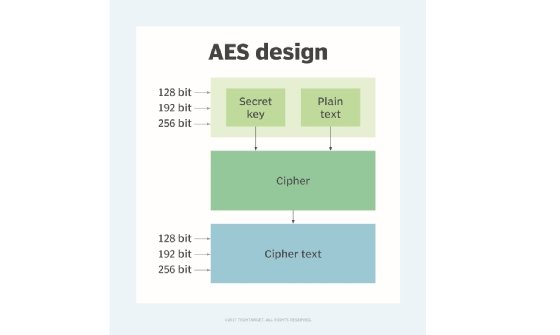
**Keys with 128-bit and 192-bit lengths**

The subkey material consists of all the round keys in order, the original key and stretches. Each stretch consists of 4-byte words, which is the same length as the original key. Each word is the [XOR](https://www.techtarget.com/whatis/definition/logic-gate-AND-OR-XOR-NOT-NAND-NOR-and-XNOR) of the preceding 4-byte word.

The first word in the stretch is rotated 1 byte to the left. This is followed by byte transformation using the S-box from the Byte Sub step and then a round-dependent constant that is XORed to the first byte.

**Keys with 256-bit lengths**

Here, the S-box from the Byte Sub step alone is applied to the word from the preceding stretch for the fifth word in a stretch.



As an Advanced Encryption Standard (AES) algorithm, Rijndael uses 128-, 192- or 256-bit keys to encrypt and decrypt data.

**9.3 Working of Rijndael**

In Rijndael, encryption happens through a series of matrix transformations or rounds. The number of rounds are variable, depending on the key or block sizes used:

* 128 bits = 9 rounds
* 192 bits = 11 rounds
* 256 bits = 13 rounds

The Rijndael algorithm is based on byte-by-byte replacement, swap and XOR operations. The procedure is as follows:

* The algorithm generates 10 128-bit keys from the 128-bit key, which are stored in 4x4 tables.
* The plaintext is divided into 4x4 tables, each of 128-bit sizes.
* Each 128-bit plaintext piece goes through a variable number of rounds as mentioned above. The code is generated after the 10th round.

Each round consists of four steps:

1. **Byte Sub.** Each byte of the block is replaced by its substitute in the S-box.
2. **Shift Row.** In a block made of bytes 1 to 16, bytes are arranged in a rectangle and shifted according to block sizes.
3. **Mix Column.** Here, matrix multiplication is performed, where each column is multiplied by the matrix. The bytes being multiplied are treated as polynomials, not as numbers. When results have more than 8 bits, the extra bits are cancelled out by XORing the binary 9-bit string 100011011 with the result. This technique is similar to what is used in in [cyclic redundancy checks](https://www.techtarget.com/searchnetworking/definition/CRC-4).
4. **Add Round Key.** Here, the subkey for the current round is XORed.

When Rijndael is performed several times with different round keys, its security increases significantly.

**10. Comparative Analysis**

A screenshot of a computer screen

Description automatically generated

Fig 6.1 Comparison table

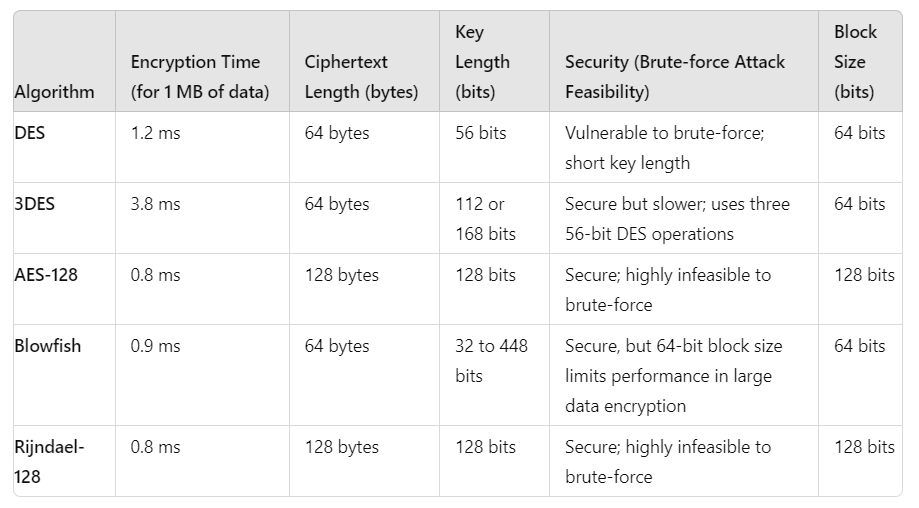


Fig 6.2 Test Cases Comparison

**11. Code**

from Crypto.Cipher import AES, DES, DES3, Blowfish

from Crypto.Random import get\_random\_bytes

from base64 import b64encode, b64decode

import tkinter as tk

from tkinter import ttk, messagebox, filedialog

import os

BLOCK\_SIZE = 16

# Padding and Unpadding Functions

def pad(text, block\_size):

    padding\_length = block\_size - len(text) % block\_size

    return text + (chr(padding\_length) \* padding\_length)

def unpad(text):

    padding\_length = ord(text[-1])

    return text[:-padding\_length]

# Encryption Function

def encrypt(plain\_text, key, algorithm):

    try:

        if algorithm == 'AES':

            cipher = AES.new(key, AES.MODE\_ECB)

            padded\_text = pad(plain\_text, BLOCK\_SIZE)

            encrypted\_bytes = cipher.encrypt(padded\_text.encode('utf-8'))

            return b64encode(encrypted\_bytes).decode('utf-8')

        elif algorithm == 'DES':

            if len(key) != 8:

                raise ValueError("DES key must be 8 bytes long")

            cipher = DES.new(key, DES.MODE\_ECB)

            padded\_text = pad(plain\_text, DES.block\_size)

            encrypted\_bytes = cipher.encrypt(padded\_text.encode('utf-8'))

            return b64encode(encrypted\_bytes).decode('utf-8')

        elif algorithm == '3DES':

            if len(key) != 16 and len(key) != 24:

                raise ValueError("3DES key must be either 16 or 24 bytes long")

            cipher = DES3.new(key, DES3.MODE\_ECB)

            padded\_text = pad(plain\_text, DES3.block\_size)

            encrypted\_bytes = cipher.encrypt(padded\_text.encode('utf-8'))

            return b64encode(encrypted\_bytes).decode('utf-8')

        elif algorithm == 'Blowfish':

            cipher = Blowfish.new(key, Blowfish.MODE\_ECB)

            padded\_text = pad(plain\_text, Blowfish.block\_size)

            encrypted\_bytes = cipher.encrypt(padded\_text.encode('utf-8'))

            return b64encode(encrypted\_bytes).decode('utf-8')

        elif algorithm == 'Rijndael':

            cipher = AES.new(key, AES.MODE\_ECB)

            padded\_text = pad(plain\_text, BLOCK\_SIZE)

            encrypted\_bytes = cipher.encrypt(padded\_text.encode('utf-8'))

            return b64encode(encrypted\_bytes).decode('utf-8')

    except Exception as e:

        messagebox.showerror("Error", f"Encryption failed: {str(e)}")

        return None

# Decryption Function

def decrypt(cipher\_text, key, algorithm):

    try:

        decoded\_data = b64decode(cipher\_text)

        if algorithm == 'AES':

            cipher = AES.new(key, AES.MODE\_ECB)

            decrypted\_text = unpad(cipher.decrypt(decoded\_data).decode('utf-8'))

            return decrypted\_text

        elif algorithm == 'DES':

            cipher = DES.new(key, DES.MODE\_ECB)

            decrypted\_text = unpad(cipher.decrypt(decoded\_data).decode('utf-8'))

            return decrypted\_text

        elif algorithm == '3DES':

            cipher = DES3.new(key, DES3.MODE\_ECB)

            decrypted\_text = unpad(cipher.decrypt(decoded\_data).decode('utf-8'))

            return decrypted\_text

        elif algorithm == 'Blowfish':

            cipher = Blowfish.new(key, Blowfish.MODE\_ECB)

            decrypted\_text = unpad(cipher.decrypt(decoded\_data).decode('utf-8'))

            return decrypted\_text

        elif algorithm == 'Rijndael':

            cipher = AES.new(key, AES.MODE\_ECB)

            decrypted\_text = unpad(cipher.decrypt(decoded\_data).decode('utf-8'))

            return decrypted\_text

    except Exception as e:

        messagebox.showerror("Error", f"Decryption failed: {str(e)}")

        return None

# Key generation function

def generate\_key(algorithm):

    if algorithm == 'AES':

        return get\_random\_bytes(32)  # AES uses 16, 24, or 32 bytes

    elif algorithm == 'DES':

        return get\_random\_bytes(8)  # DES uses 8-byte (64-bit) key

    elif algorithm == '3DES':

        return get\_random\_bytes(24)  # 24-byte key for 3DES

    elif algorithm == 'Blowfish':

        return get\_random\_bytes(16)  # Blowfish key can be between 4-56 bytes, 16 is typical

    elif algorithm == 'Rijndael':

        return get\_random\_bytes(32)  # Rijndael with 32 bytes for AES-256 equivalent

    else:

        return None

# Function to open and read a file

def open\_file():

    file\_path = filedialog.askopenfilename(filetypes=[("Text Files", "\*.txt")])

    if file\_path:

        with open(file\_path, 'r') as file:

            text\_input.delete("1.0", tk.END)

            text\_input.insert(tk.END, file.read())

        messagebox.showinfo("Success", "File uploaded successfully!")

# Function to save the result to a file

def save\_file(result\_text):

    file\_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "\*.txt")])

    if file\_path:

        with open(file\_path, 'w') as file:

            file.write(result\_text)

        messagebox.showinfo("Success", "File saved successfully!")

# Function to handle text encryption/decryption

def process\_text(action):

    plain\_text = text\_input.get("1.0", tk.END).strip()  # Now defined correctly

    key = get\_key()

    algorithm = algorithm\_choice.get()

    if plain\_text and key:

        if action == 'encrypt':

            result = encrypt(plain\_text, key, algorithm)

        elif action == 'decrypt':

            result = decrypt(plain\_text, key, algorithm)

        if result:

            result\_output.delete("1.0", tk.END)

            result\_output.insert(tk.END, result)

            save\_file(result)

# Function to generate and display a new key

def generate\_new\_key():

    key = generate\_key(algorithm\_choice.get())

    key\_display.delete(0, tk.END)

    key\_display.insert(0, b64encode(key).decode('utf-8'))

# Function to get the key from input field

def get\_key():

    key\_input = key\_display.get().strip()

    if len(key\_input) == 0:

        messagebox.showwarning("Warning", "Key is missing! Either enter a key or generate one.")

        return None

    try:

        return b64decode(key\_input)

    except Exception as e:

        messagebox.showerror("Error", f"Invalid key: {str(e)}")

        return None

# Function to reset fields

def reset\_fields():

    text\_input.delete("1.0", tk.END)

    key\_display.delete(0, tk.END)

    result\_output.delete("1.0", tk.END)

# Main application function

def create\_app():

    root = tk.Tk()

    root.title("Encryption & Decryption")

    root.geometry("600x500")  # Set a default size for the window

    root.configure(bg='#f0f0f0')

    # Notebook (Tabs) for encryption and decryption

    notebook = ttk.Notebook(root)

    notebook.pack(fill='both', expand=True)

    # Encryption Page

    encryption\_frame = ttk.Frame(notebook, padding=10)

    notebook.add(encryption\_frame, text="Encrypt")

    # Encryption page content

    tk.Label(encryption\_frame, text="Encryption", font=("Arial", 16), background='#f0f0f0').pack(pady=10)

    tk.Label(encryption\_frame, text="Select Algorithm:", background='#f0f0f0').pack(pady=5)

    global algorithm\_choice

    algorithm\_choice = ttk.Combobox(encryption\_frame, values=["AES", "DES", "3DES", "Blowfish", "Rijndael"])

    algorithm\_choice.current(0)

    algorithm\_choice.pack(pady=5)

    tk.Button(encryption\_frame, text="Upload File", command=open\_file).pack(pady=5)

    tk.Label(encryption\_frame, text="Enter or Generate Key:", background='#f0f0f0').pack(pady=5)

    global key\_display

    key\_display = tk.Entry(encryption\_frame, width=50)

    key\_display.pack(pady=5)

    tk.Button(encryption\_frame, text="Generate New Key", command=generate\_new\_key).pack(pady=5)

    global text\_input  # Now defined correctly

    tk.Label(encryption\_frame, text="Enter Text to Encrypt:", background='#f0f0f0').pack(pady=5)

    text\_input = tk.Text(encryption\_frame, height=5, width=50)

    text\_input.pack(pady=5)

    tk.Button(encryption\_frame, text="Encrypt", command=lambda: process\_text('encrypt')).pack(pady=5)

    tk.Button(encryption\_frame, text="Reset", command=reset\_fields).pack(pady=5)

    global result\_output

    tk.Label(encryption\_frame, text="Result:", background='#f0f0f0').pack(pady=5)

    result\_output = tk.Text(encryption\_frame, height=5, width=50)

    result\_output.pack(pady=5)

    # Decryption Page

    decryption\_frame = ttk.Frame(notebook, padding=10)

    notebook.add(decryption\_frame, text="Decrypt")

    tk.Button(decryption\_frame, text="Upload File", command=open\_file).pack(pady=5)

    tk.Button(decryption\_frame, text="Decrypt", command=lambda: process\_text('decrypt')).pack(pady=5)

    root.mainloop()

if \_name\_ == "\_main\_":

    create\_app()

**12. Advantages**

**Comprehensive Cryptography Understanding:**

The project provides an in-depth understanding of symmetric encryption algorithms like AES, DES, 3DES, Blowfish, and Rijndael, enabling users to grasp their functionality, performance, and real-world applications in securing data.

**Hands-On Implementation:**

By implementing these encryption algorithms using Python, users get practical experience in cryptographic operations, which enhances their programming skills and understanding of how encryption works in real-life applications.

**User-Friendly Interface:**

The Tkinter-based graphical user interface (GUI) simplifies encryption and decryption, allowing non-technical users to easily encrypt or decrypt files and text without needing deep cryptography or programming knowledge.

**Real-World Application:**

This project can be applied in everyday scenarios where users need to securely encrypt sensitive information, such as personal documents or messages, offering practical utility beyond academic learning.

**Algorithm Comparison:**

The project allows users to compare different symmetric encryption algorithms based on performance, speed, and security, helping them make informed decisions about which algorithm suits their specific needs.

**Key Management and Security:**

The project includes key generation and management features, which are crucial for secure encryption, demonstrating how cryptographic keys work in practice and providing a complete understanding of secure communication practices.

**Educational Resource:**

This project serves as a valuable educational tool, providing a comprehensive learning experience for students and professionals who want to explore cryptography, implement secure systems, and understand the practical differences between various encryption algorithms.

**13. Applications**

**13.1 Secure File Storage:**

Users can encrypt sensitive files and documents before storing them on local or cloud storage, ensuring that unauthorized individuals cannot access the data even if the storage medium is compromised.

**13.2 Confidential Communication:**

The project can be used to encrypt messages exchanged between individuals or organizations, safeguarding sensitive information during transmission and ensuring that only intended recipients can decrypt and read the messages.

**13.3 Data Protection in Applications:**

Developers can integrate the encryption algorithms into software applications to protect sensitive user data, such as passwords, personal information, and financial details, enhancing the security of the application.

**13.4 Secure Backup Solutions**:

The project can be applied in backup systems to encrypt data before creating backups, ensuring that backups are secure and cannot be easily accessed by unauthorized users.

**13.5 Digital Rights Management (DRM):**

The encryption algorithms can be utilized in DRM systems to protect digital content, such as music, videos, and eBooks, preventing unauthorized copying and distribution while allowing legitimate users to access the content.

**13.6 Healthcare Data Security:**

Healthcare providers can use this project to encrypt sensitive patient data, ensuring compliance with regulations like HIPAA and protecting patients' privacy while maintaining data accessibility for authorized personnel.

**13.7 Secure Messaging Applications:**

Developers can build messaging applications that utilize the encryption algorithms to secure user communications, ensuring that conversations remain private and confidential against potential eavesdropping.

**13.8 Authentication Systems:**

The project can enhance user authentication systems by encrypting credentials and tokens, protecting them during transmission and storage, and thereby reducing the risk of unauthorized access to user accounts.

**13.9 IoT Device Security:**

The algorithms can be implemented in Internet of Things (IoT) devices to secure communication between devices and servers, ensuring that sensitive data transmitted across networks remains protected from interception.

**13.10 Financial Transactions:**

The project can be applied in financial applications to encrypt sensitive transaction data, enhancing the security of online payments and banking, and protecting users from fraud and identity theft.

**14. Future Enhancements**

**14.1 Inclusion of Asymmetric Algorithms:**

Incorporating asymmetric encryption algorithms, such as RSA or ECC, can provide users with options for secure key exchange and digital signatures, enhancing overall security and functionality.

**14.2 Integration of Hashing Techniques:**

Adding hashing algorithms like SHA-256 or bcrypt would allow users to securely store passwords and verify data integrity, complementing the encryption functionalities.

**14.3 User Authentication Features:**

Implementing multi-factor authentication (MFA) can improve security, requiring users to provide additional verification beyond just a password or encryption key.

**14.4 Improved User Interface:**

Enhancing the user interface for better accessibility and usability, including responsive design for mobile and tablet devices, would make the application more versatile and user-friendly.

**14.5 Real-Time Encryption/Decryption:**

Developing real-time encryption capabilities for streaming data (e.g., video calls or live messaging) would allow users to secure communication without noticeable delays.

**15. Conclusion**

In conclusion, this project successfully demonstrates the practical application of symmetric cryptography algorithms through a user-friendly interface. By implementing AES, DES, 3DES, Blowfish, and Rijndael, users can easily encrypt and decrypt sensitive data, enhancing information security. The comparative analysis highlights each algorithm's strengths and weaknesses, enabling informed decisions based on specific security needs. This project not only serves as an educational tool for understanding cryptographic principles but also has real-world applications across various domains, such as secure communication and data protection. Overall, it contributes significantly to promoting data security in today's digital landscape.

**16. Presentation**

A computer monitor with text on it

Description automatically generated

A close-up of a document

Description automatically generated

A close-up of a computer code

Description automatically generated

A close-up of a document

Description automatically generated

A white rectangular box with black text

Description automatically generated

A computer screen with text and hands on keyboard

Description automatically generated

A diagram of a computer code

Description automatically generated with medium confidence

A close-up of several text

Description automatically generated

A close-up of a text

Description automatically generated

A diagram of a medical procedure

Description automatically generated with medium confidence