# Secure File Storage on cloud using Hybrid Cryptography

**Project Report**

*Submitted in the partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**Cloud Computing**

**Submitted by:**

| | |
|---|---|
| **Shalin Dashora** | **20BCS4079** |
| **Milind Pal Singh Tanwar** | **20BCS4117** |
| **Abhinav Singh** | **20BCS4126** |
| **Navneet Gupta** | **20BCS4148** |

**Under the Supervision of: Ms. Ramneet Kaur**

**CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413,**

**PUNJAB**

**2023**

# Acknowledgment

We would like to take this incentive to convey gratitude to Ms. Ramneet Kaur our project supervisor, for providing us with the amazing chance to do this project on the topic Secure File Storage Hybrid Cryptography, which also assisted us in doing a great deal of research and learning about so many new concepts.

We are extremely grateful to them. Second, we’d like to thank all the people for their assistance in completing this project in such a short period of time. It was only because of them that we were able to develop our project and make it a fun and rewarding experience.

We are doing this project not only for the grades, but also to expand our knowledge.

# Abstract

Cloud computing originated from a commercial enterprise concept, and developed into a flourishing IT revolution due to its pay as per you go pricing model, yet there are concerns related to the security of the third-party data and that's why the customers are still reluctant to shift entirely from on premises to cloud. Security services provided by the existing cloud providers is not up to the mark to provide high level security to data on cloud. Secure file storage in a hybrid cloud using cryptography refers to the process of ensuring that files stored in a hybrid cloud environment are secure from unauthorized access, tampering, or theft. Hybrid cloud refers to a cloud computing environment that combines on-premises infrastructure with one or more public or private cloud providers. Cryptography is used to secure the files in transit and at rest. Cryptography involves the use of mathematical algorithms and protocols to ensure that data is protected from unauthorized access. This includes encryption, which converts the original data into an unreadable format, and decryption, which converts the encrypted data back to its original format. Overall, secure file storage in hybrid cloud using cryptography [3] is essential for protecting sensitive data and ensuring that it remains secure, even in a cloud computing environment. It requires a comprehensive approach that includes strong encryption, secure transmission, and robust access controls.

# Objective

The proposed work is aimed to carry out work leading to the development of an approach for Cloud based secure file storage system using Cryptography. Our main objective is very clear i.e., to provide the information about the security of the cloud and how different algorithms work to secure the customer's data. We have also analyzed different cryptographic algorithms and mentioned their pros and cons. We have used Python language to implement the algorithms and provide an idea of how encryption and decryption are carried out using these algorithms to secure our data.

The proposed aim will be achieved by dividing the work into the following objectives:

1. To give security to customers' crucial data.
2. Prevent data breaches and cyber-attacks.
3. Implementation of cryptography-based algorithms.
4. To analyze the performance of algorithms

# Table of Contents

Acknowledgment ........................................ 2
Abstract ........................................ 3
Objective ........................................ 4
Table of Contents ........................................ 5
Table of Figures ........................................ 6
Chapter-1 Introduction ........................................ 7
Cloud Computing ........................................ 8
Advantages of Cloud Computing ........................................ 9
Disadvantages of Cloud Computing........................................ 9
Cloud Deployment Models........................................ 10
Cloud Service Models ........................................ 12
Cryptography ........................................ 16
Types of Cryptography ........................................ 18
Chapter-2 Analysis of Cryptography Algorithms ........................................ 21
Data Encryption Standard (DES) Algorithm ........................................ 21
DES algorithm steps ........................................ 22
Advanced Encryption Standard (AES) Algorithm ........................................ 23
AES algorithm steps ........................................ 24
RSA (Rivest, Shamir, Adleman) Algorithm ........................................ 25
RSA algorithm steps ........................................ 26
Blowfish Algorithm ........................................ 27
Blowfish Algorithm steps ........................................ 28
Chapter-3 Proposed Design ........................................ 29
User Registration ........................................ 30
File Upload ........................................ 31
File Download ........................................ 31
Data Sharing ........................................ 32
Chapter-4 Implementation of Algortihms ........................................ 34
Advanced Encryption Standard (AES) Algorithm ........................................ 35
Data Encryption Standard (DES) Algorithm ........................................ 37
RSA (Rivest, Shamir, Adleman) Algorithm ........................................ 40
LSB Stanography ........................................ 42
Conclusion ........................................ 43
References ........................................ 44
Appendix........................................ 46

## Table of Figures

Figure 1Cloud Models ....................12
Figure 2Service Models ....................14
Figure 3Cloud Statistics ....................16
Figure 4Difference in cloud service models ....................16
Figure5SymmetricEncryption ……… ....................17
Figure6Asymmetric Encryption ....................18
Figure7Hash Function ....................18
Figure 8DES algorithm ....................19
Figure 9AES Algorithm ....................21
Figure 10Adding the round key ....................22
Figure 11RSA algorithm ....................23
Figure 12Blowfish Algorithm ....................25
Figure 13Cloud Model ....................27
Figure 14User Registration ....................27
Figure15File splitting ....................28
Figure 16Encryption ....................28
Figure 17File storage ....................28
Figure 18Decryption ....................29
Figure 19Original form ....................29
Figure 20Data sharing ....................30
Figure 21Key management ....................30

# Chapter-1 Introduction

Cloud computing is the fastest growing technology and offers different variety of services for different sizes of industries, it also provides different service models for different needs such as public, private and hybrid cloud with different types of cloud services Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS).

With different advantages that cloud brings there are also some major concerns related to security and confidentiality of data on cloud. One of the major concerns is the theft of data from the cloud server by malicious cyber-attacks. The customer is not in complete control of their sensitive data on the cloud and who can access it. To overcome these security concerns, security algorithms based on cloud cryptography serve the purpose of encrypting and decrypting user's data and help to build a secured cloud infrastructure. To achieve this there are a number of cryptographybased security algorithms such as Blowfish Encryption Algorithm, Advanced Encryption Standard (AES), Data Encryption Standard (DES), RivestShamirdleman encryption (RSA). Two operations performed by these algorithms are encryption and decryption. Symmetric algorithms use one key for both encryption and decryption whereas asymmetric algorithms use two different keys for them. Blowfish Encryption Algorithm is the first symmetric algorithm that uses a single encryption key to both encrypt and decrypt data. Rivest-Shamir-dleman encryption (RSA) algorithm uses mathematical concept of factorization of product of two large primes. It is best suited for verification and encryption. Advanced Encryption Standard (AES) is a symmetrical block cipher-based algorithm that takes block of plain text of 128 bits and transforms it into ciphertext of 128,192- and 256bits using keys. Data Encryption Standard (DES) is a symmetrical block cipher-based algorithm that takes block of plain text of 64 bits and transforms it into ciphertext of 64 bits. There is also an advanced method-Hybrid cryptography that combines one or more cryptography algorithms that strengthens every shape of encryption.

# Cloud Computing

Cloud computing allows you to rent rather than buy your IT. Companies choose to access their computational power over the internet, or the cloud, and pay for it as they use it, rather than investing extensively in databases, software, and hardware. Servers, storage, databases, networking, software, analytics, and business intelligence are now among the cloud services available. Cloud computing helps businesses to develop, innovate, and support commercial IT solutions with the speed, scalability, and flexibility that they require.

Cloud computing refers to any subscription-based or pay-per-use service that expands IT's existing capabilities in real time over the Internet. There is a large workload change in a cloud computing system. When it comes to executing apps, local PCs no longer have to do all of the heavy lifting. Instead, they are handled by the cloud's network of computers. The user's hardware and software requirements reduce. The user's machine simply has to be capable of running the cloud computing system's interface software, which can be as simple as a Web browser, and the cloud's network will handle the rest.

Cloud computing is a type of computing that uses shared computing resources rather than local servers or personal devices to handle applications. The term "cloud computing" refers to a sort of Internet-based computing in which various services, such as servers, storage, and applications, are supplied to an organization's computers and devices via the Internet.

Grid computing, a sort of computing that uses the unused processing cycles of all computers in a network to solve problems that are too complex for any single machine, is similar to cloud computing.

The goal of cloud computing is to use traditional supercomputing, or highperformance computing power, which is typically used by armed services and research institutions to perform tens of trillions of computations per second in user applications such as personalised information, financial portfolios, data storage, and the powering of large, immersive computer games.
With the projected quick and sustained growth of major worldwide cloud data centres, cloud computing will be even more significant in the future years.

## Advantages of Cloud Computing

- Cost efficient: The most compelling incentive for businesses to migrate to Cloud Computing is that it is significantly less expensive than any on-premise solution. Companies no longer need to save data on discs because the cloud provides unrestricted space and resources.
- Back-up and restore data: When data is kept on the cloud, it is much easier to back it up and retrieve it, which is a time-consuming process in traditional technology.
- High Speed: We can swiftly install the service with fewer clicks thanks to cloud computing. We can receive the resources we need for our system in minutes thanks to this speedy deployment.

## Disadvantages of Cloud Computing

- Vulnerability to attacks: Because all of a company's data is online on the cloud, storing it there can pose major risks of information theft. Even the most secure enterprises have experienced a security breach, and it's a danger that exists in the cloud too. Even though modern security measures are placed on the cloud, storing confidential data on the cloud might be risky.

- Vendor lock-in: Because of the disparities across vendor platforms, a corporation may face major issues while attempting to migrate from one cloud platform to another. Support concerns, configuration complications, and additional costs may arise if the present cloud platform's applications are hosted and run on another platform.

- Dependence on network connectivity: The Internet is absolutely necessary for cloud computing to work. Because of the direct connection to the Internet, a corporation must have a dependable and constant Internet service, as well as a fast connection and bandwidth, in order to reap the benefits of Cloud Computing.

## Cloud Deployment Models

A cloud deployment model is a type of cloud environment that is defined by who handles security, who has rights to handle data, and whether resources are pooled or reserved. The goal and characteristics of your cloud environment are also defined by the cloud deployment model.

There are four basic cloud-deployment models, each with its own set of features, requirements, and advantages.

**1. Public Cloud:** The public cloud is a suitable deployment solution for businesses that require immediate access without paying significant upfront fees. It's free and open to all sizes and types of organisations, and it's incredibly useful because of its unique capability of securely transferring data over the internet.Because its services are more commoditized, it is more costeffective. It's a pay-as-you-go system with a low initial cost. It's services are especially helpful for workloads that are only needed for a limited period of time, such as for an event or the early stages of a startup.

**Advantages:-**

- It's the most cost-effective model on the market, and it's not locationdependent.
- To effectively utilise a public cloud, you do not require infrastructure management by a professional in-house staff.
- Through Virtualization, it allows higher vertical scalability.

**Disadvantages:-**

- There are substantial concerns about its security and privacy. Because it lacks a strict data protocol, it attracts more focused attacks.
- Customization is limited on the public cloud. Clients can choose the operating system and the size of the virtual machine, but they can't change the reports, orders or networking.

**2. Private Cloud:** A private cloud is an infrastructure within a user's firewall that is dedicated to a single user. It restricts access to just authorised individuals, allowing businesses to have greater central control over privacy and data. The data centre might be on-premises or co-located. This is normally a single-tenant deployment, which implies the platform isn't shared. It can, however, have several clients per company department. Companies with specific requirements, such as confidential and sensitive data or the necessity for safe and reliable efficiency, should opt for a private cloud model.

**Advantages:-**

- Only authorised staff have access, which is perfect for safeguarding corporate data in accordance with a data protection policy.
- Companies can modify their solutions to meet specific needs.

**Disadvantages:-**

- Private cloud isn't a pay-as-you-go service—you pay for the entire stack, whether or not it's being used.
- It is managed in-house and requires a lot of maintenance.

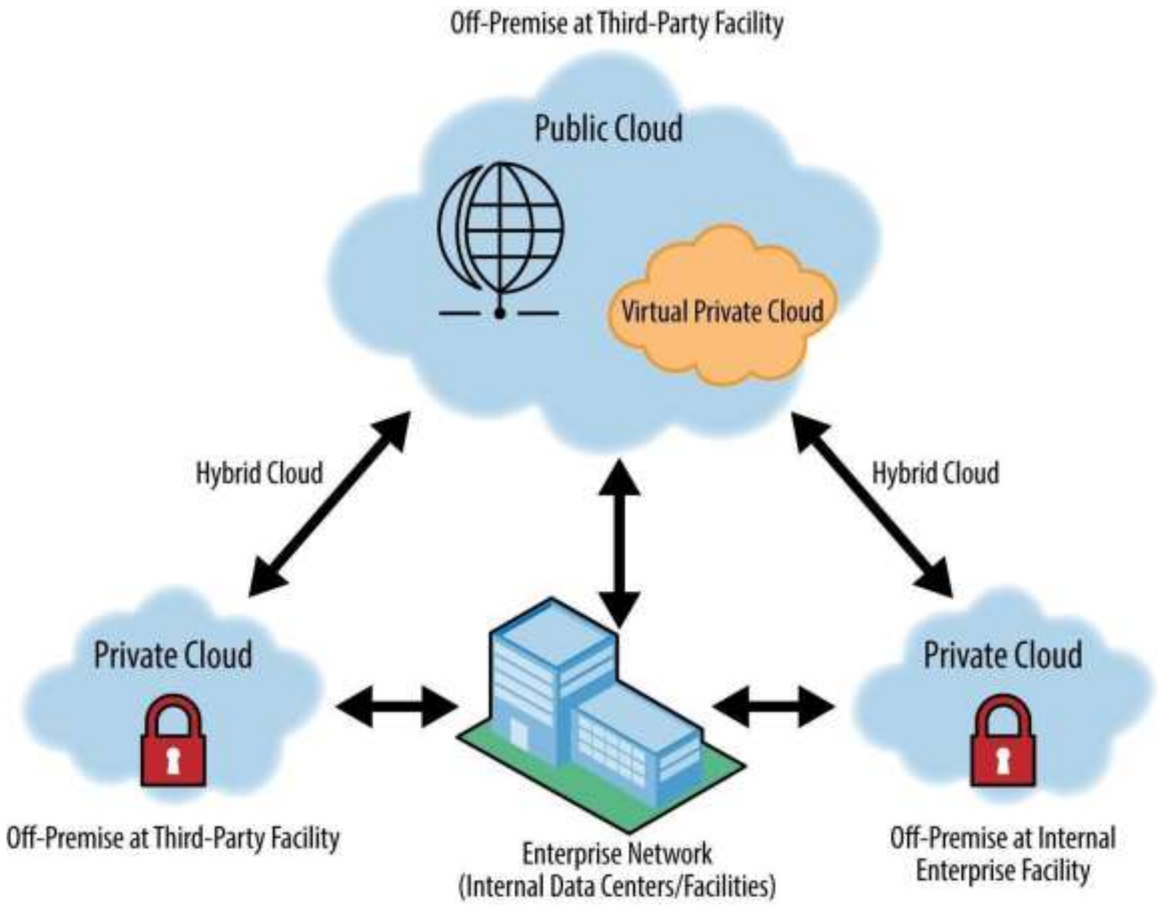


*Figure 1Cloud Models*

**3. Hybrid Cloud:** The term "hybrid cloud" refers to a cloud deployment paradigm that combines two or more cloud deployment techniques. They're all different, but they're all governed by the same set of rules. Cloud bursting is done using hybrid cloud models. Assume the client's application is primarily hosted in a private cloud. However, if the system encounters a spike, rapid surge, or excessive load, it can 'burst' into the public cloud to relieve the strain.

**Advantages:-**

- It lowers operational expenses and allows businesses to mix and match cloud workflow models.
- It's scalable because it uses a mix-and-match approach to operating and managing workloads.

**Disadvantages:-**

- Because you're combining two or more different cloud models, it's a complicated arrangement to handle.

## Cloud Service Models

Cloud computing can be divided into several components, each concentrating on a distinct aspect of the technology stack and a particular use case. There are three types of cloud service models:

i) Infrastructure as a service (Iaas) ii)
Platform as a service (Paas) iii)
Software as a service (Saas)

*Figure 2Service Models*

### 1. Infrastructure as a service (Iaas):

Infrastructure as a Service (IaaS) refers to the basic computing components that can be rented, such as physical or virtual servers, storage, and networking. Companies who want to develop applications from the ground up and control practically all of the aspects will find this appealing, but it does necessitate having the technical capabilities to coordinate services at that level. Hardware as a Service is another name for IaaS. It's a computer network that's managed over the internet. The fundamental benefit of adopting IaaS is that it saves consumers money and time by allowing them to avoid the cost and complexity of owning and managing physical servers**.**

There are the following characteristics of IaaS:

i) As a service, resources are offered**.** ii) The services are extremely scalable.

iii) API-based access and a dynamic and configurable GUI iv) Administrative tasks that are automated

Example: Digital Ocean, Linode, Amazon Web Services (AWS), Microsoft Azure, Google Compute Engine (GCE)

**2. Platform as a service (Paas):**

The next layer up is Platform as a Service (PaaS), which provides the tools and software that developers need to build applications on top of the underlying storage, networking, and virtual servers, such as middleware, database management, operating systems, and development tools. The PaaS cloud computing platform is designed to help programmers create, test, execute, and manage applications.

The following are the features of PaaS:

i) Various people can access the same development application.

ii) Integrates with databases and web services.

iii) Based on virtualization technology, resources can be readily scaled up or down to meet the needs of the enterprise.

iv) The ability to "Auto-scale" is included.

Example: AWS Elastic Beanstalk, Windows Azure, Google App Engine, Apache Stratos

**3. Software as a service (Saas):**

SaaS is a software delivery model that allows users to access data from any device that has an internet connection and a web browser. Software providers host and manage the servers, databases, and code that make up an application under this web-based approach. "On-demand software" is another term for SaaS. It's a type of software where the applications are hosted by a third-party cloud service provider. Users can use a web browser and an internet service to access these applications. The cloud-based paradigm is now so ubiquitous that more than 60% of software buyers who call Software Advice specifically request web-based products—only 2% specifically request on-premise software.

There are the following characteristics of SaaS –

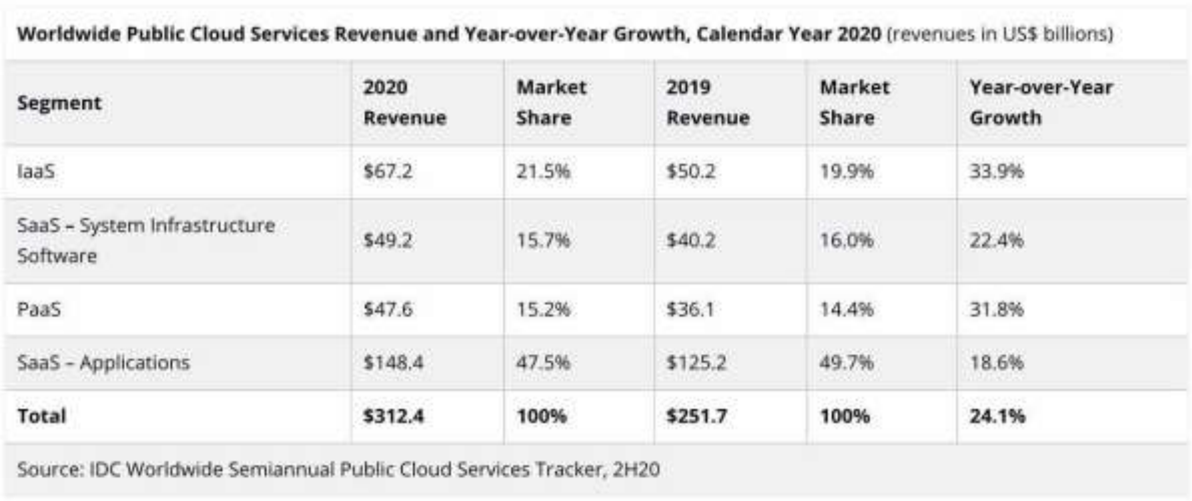
i) Directed from a single place ii) The website is operated on a remote server.

iii) Easily attainable via the internet

iv) Updates to hardware and software are not the responsibility of users. Updates are downloaded and installed automatically.

v) The solutions are accessible on a pay-as-you-go basis.

A suitable analogy for the SaaS model is a bank, which preserves each customer's privacy while offering a service that is dependable and secure—on a large scale. Customers of a bank can utilize the same financial technologies and systems without being concerned about unauthorized access to their personal records.



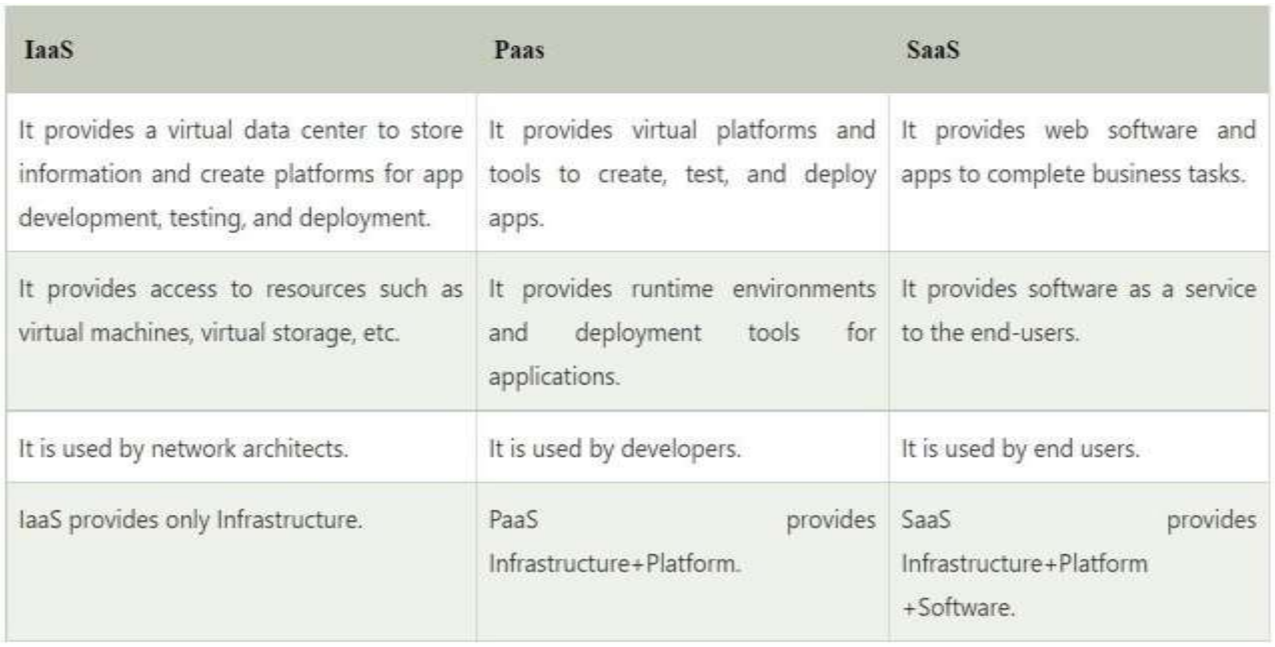
**Worldwide Public Cloud Services Revenue and Year-over-Year Growth, Calendar Year 2020** (revenues in US$ billions)

| Segment | 2020 Revenue | Market Share | 2019 Revenue | Market Share | Year-over-Year Growth |
|---|---|---|---|---|---|
| IaaS | $67.2 | 21.5% | $50.2 | 19.9% | 33.9% |
| SaaS – System Infrastructure Software | $49.2 | 15.7% | $40.2 | 16.0% | 22.4% |
| PaaS | $47.6 | 15.2% | $36.1 | 14.4% | 31.8% |
| SaaS – Applications | $148.4 | 47.5% | $125.2 | 49.7% | 18.6% |
| **Total** | **$312.4** | **100%** | **$251.7** | **100%** | **24.1%** |

Source: IDC Worldwide Semiannual Public Cloud Services Tracker, 2H20

*Figure 3Cloud Statistics*

The below table shows the difference between IaaS, PaaS, and SaaS –



| IaaS | Paas | SaaS |
|---|---|---|
| It provides a virtual data center to store information and create platforms for app development, testing, and deployment. | It provides virtual platforms and tools to create, test, and deploy apps. | It provides web software and apps to complete business tasks. |
| It provides access to resources such as virtual machines, virtual storage, etc. | It provides runtime environments and deployment tools for applications. | It provides software as a service to the end-users. |
| It is used by network architects. | It is used by developers. | It is used by end users. |
| IaaS provides only Infrastructure. | PaaS provides Infrastructure+Platform. | SaaS provides Infrastructure+Platform +Software. |

*Figure 4Difference in cloud service models*

## Cryptography

Cryptography is a method of safeguarding information and communications by encoding it in a way that only the people who need to know can interpret and process it. As a result, unwanted access to information is prevented. The suffix graphy means "writing" and the word "crypt" implies "hidden." The procedures used to safeguard information in cryptography are derived from mathematical principles and a set of rule-based calculations known as algorithms that change signals in ways that make them difficult to decode. These algorithms are used to generate cryptographic keys, digitally sign documents, verify data privacy, browse the internet, and protect sensitive transactions.
In the age of computers, cryptography is frequently associated with the conversion of plain text into cypher text, which is text that can only be decoded by

the intended recipient. This process is known as encryption. Decryption is the process of converting encrypted text into plain text.

## Common Terms Used in Cryptography

- Plaintext: The original and understandable text. As an instance, 'Y' needs to transmit a “Computer” message to 'Z'. Here, “Computer” is the plaintext or the original message.
- Ciphertext: The text that cannot be understood by way of anybody or a gibberish text, example “A@$&J9.”
- Encryption: A process of changing clear text into unclear text. The manner of encipherment needs an encipherment algorithm and a key. Encipherment occurs on the sender side.
- Decryption: A reverse method of encode. It is a manner of converting ciphertext into plaintext.
- Key: A key is character, number, or a special character. It is used at the time of encipherment on the original text and at the time of decode on the ciphertext

## Purpose of Cryptography

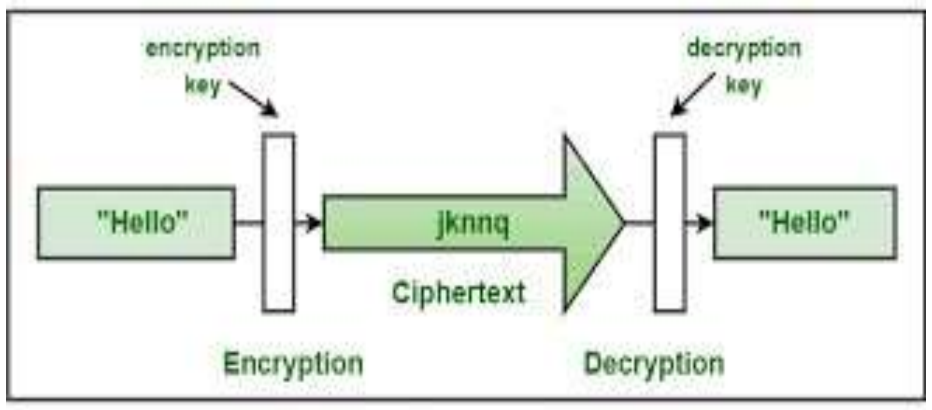
- Authentication: The potential of a system to test the identity of the sender.

- Confidentiality: Information transmitted ought to be accessed handiest by using legal parties and not through anyone else.

- Integrity: Only the authorized parties are permitted to alter on transmitted information.

- Non-repudiation: Is the guarantee that someone cannot deny the validity of something.

• Access Control: Just the authorized persons are capable to get right of entry to the given information.



Cryptography

## Types of Cryptography

• **Symmetric Key Cryptography:** This method, also known as private-key cryptography, necessitates the sender and receiver having access to the same key. As a result, before the communication can be decoded, the recipient must hold the key. Closed systems with a low danger of third party intrusion perform best with this method.

In this type of encryption, the sender and the receiver agree on a secret (shared) key. Then they use this secret key to encrypt and decrypt their sent messages.

Suppose, we have Node A and Node B. So, Node A and B first agree on the encryption technique to be used in encryption and decryption of communicated data. Then they agree on the secret key that both of them will use in this connection. After the encryption setup finishes, node A starts sending its data encrypted with the shared key, on the other side node B uses the same key to decrypt the encrypted messages.

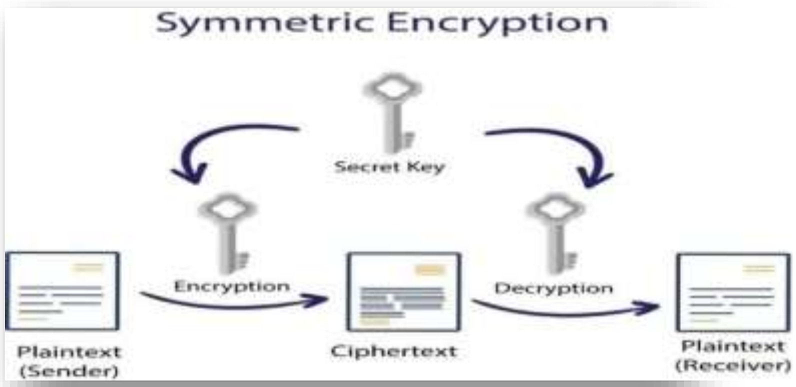


*Figure 5Symmetric Encryption*

• **Asymmetric Key Cryptography:** This technology, also known as public key cryptography, encrypts data using two keys: a public and a private key that are mathematically connected. One key is used for encryption and the other is used for decryption. In public key encryption, data integrity and confidentiality are also guaranteed.

Asymmetric encryption is the other type of encryption where two keys are used.
To explain more, what Key1 can encrypt only Key2 can decrypt, and vice versa. It is also known as Public Key Cryptography (PKC), because users tend to use two keys: public key, which is known to the public, and private key which is known only to the user.

Suppose, we have node A and node B. After agreeing on the type of encryption to be used in the connection, node B sends its public key to node A. And Then, Node A uses the received public key to encrypt its messages. Then when the encrypted messages arrive, node B uses its private key to decrypt them.

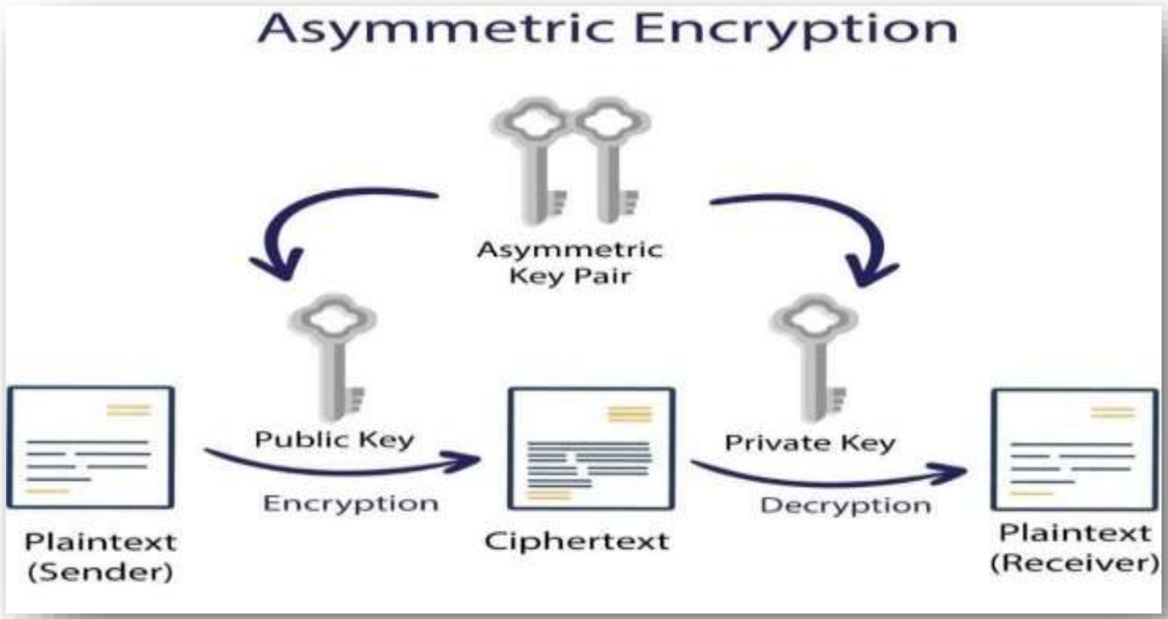


*Figure 6 Asymmetric Encryption*

- **<u>Hash Functions:</u>** Hashing creates a fixed-length unique signature for a data set or text. Each message has its own hash, allowing minor modifications to the data to be easily tracked. Data that hashing encrypts cannot be decrypted or reversed back to its initial form. As a result, hashing is only utilised as a data verification mechanism.

A mathematical operation known as a hash function compresses one input numerical value into another. Although the hash function's output is always a defined length, its input can be of any length.

Message digests or just hash values are terms used to describe values that a hash function returns. The hash function was demonstrated in the following image.

The typical features of hash functions are −

- Fixed Length Output (Hash Value) ○ Hash function coverts data of arbitrary length to a fixed length. This process is often referred to as hashing the data. ○ Since a hash is a smaller representation of a larger data, it is also referred to as a digest. ○ Hash function with n bit output is referred to as an n-bit hash function. Popular hash functions generate values between 160 and 512 bits.

• Efficiency of Operation ○ Generally, for any hash function h with input x, computation of h(x) is a fast operation.
○ Computationally hash functions are much faster than a symmetric encryption.

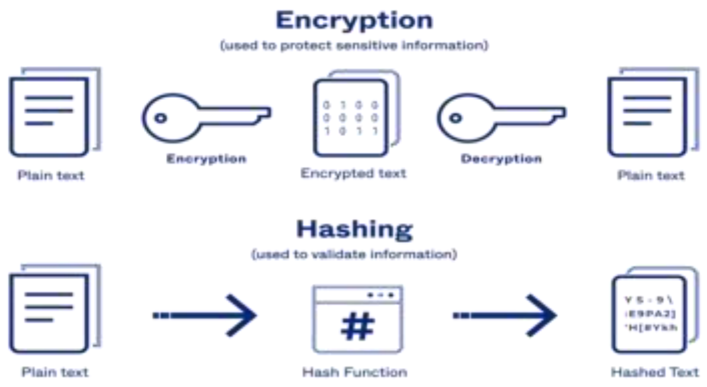


*Figure 7Hash Function*

# Chapter-2 Analysis of Cryptography Algorithms

## Data Encryption Standard (DES) Algorithm

The DES algorithm (Data Encryption Standard) is a symmetric-key block cypher developed by an IBM team in the early 1970s and adopted by the National Institute of Standards and Technology (NIST). The algorithm turns plain text into ciphertext using 48-bit keys after splitting it into 64-bit blocks. As it's a symmetric-key technique, the data is encrypted and decrypted using the same key.

DES employs a highly sophisticated algorithm, or key, that the US government has declared unbreakable. There are at least 72,000,000,000,000,000 (72 quadrillion) encryption keys to choose from. Each 64-bit block of data is given a 56-bit key. This process entails 16 rounds of operations that combine the data and key utilising permutation and substitution operations. The ultimate result is data and key that are fully jumbled, with every piece of the ciphertext relying on every bit of the key (a 56-bit quantity for DES)

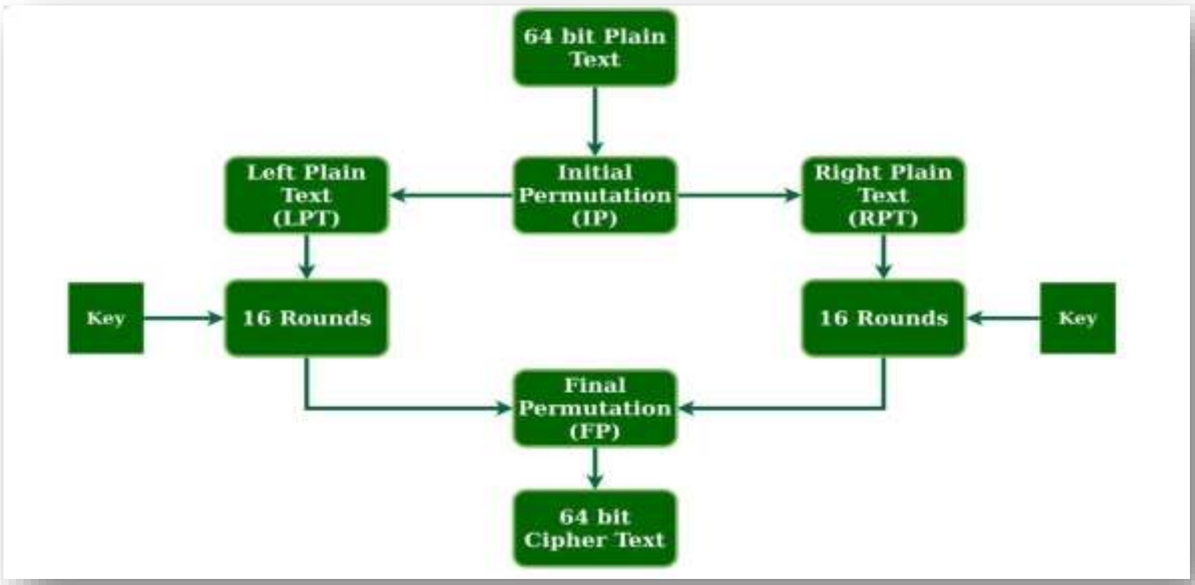


*Figure 8DES algorithm*

## DES algorithm steps

- The 64-bit plain text block is passed to an initial permutation (IP) function to begin the process.
- The plain text is subsequently subjected to the initial permutation (IP).
- Following that, the initial permutation (IP) divides the permuted block into two halves, known as Left Plain Text (LPT) and Right Plain Text (RPT) .
- The encryption operation is repeated 16 times for each LPT and RPT.
- Ultimately, the LPT and RPT are reunited, and the newly combined block is subjected to a Final Permutation (FP).
- As a result of this procedure, the desired 64-bit ciphertext is generated.

We use the same algorithm for decryption, but the order of the 16 round keys is reversed.

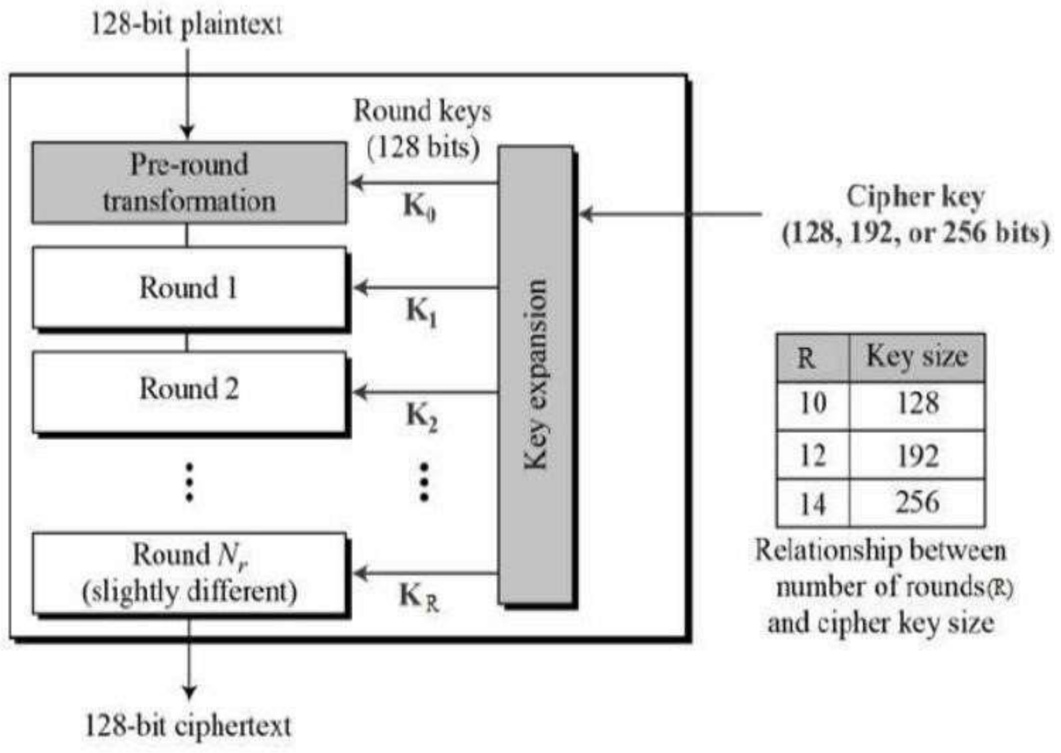
A security provider is required for DES deployment. However, there are numerous providers to pick from, and deciding which one to use is the first and most important stage in the deployment process. Your choice may be influenced by the programming language you're using, such as Java, Python, C, or MATLAB.

## Advanced Encryption Standard(AES) Algorithm

The AES (Rijndael algorithm) Encryption algorithm that uses bytes rather than bits to conduct operations. That is to say, it accepts 128 bits as input and outputs 128 bits of encrypted cipher text. It uses keys of 128, 192, and 256 bits to convert these individual blocks. It merges these blocks together to generate the cipher text after encrypting them.

It's built around a substitution-permutation network, or SP network. It consists of a sequence of linked operations, including substitutions (replacing inputs with certain outputs) and bit shuffling (permutations).

To generate ciphertext, the AES algorithm employs a substitution-permutation (SP) network with many rounds. The number of rounds is determined on the key size. Ten rounds are dictated by a 128-bit key size, 12 rounds by a 192bit key size, and 14 rounds by a 256-bit key size. Each of these rounds requires a round key, but because the algorithm only accepts one key, it must be enlarged to obtain keys for each round, including round 0.



| R | Key size |
|---|---|
| 10 | 128 |
| 12 | 192 |
| 14 | 256 |

*Figure 9AES Algorithm*

## AES algorithm steps

There are four steps in each iteration of the algorithm.

- <u>Substitution of the bytes:</u> The bytes of the block text are first substituted according to rules imposed by predetermined substitution boxes.
- <u>Shifting the rows:</u> Each of the matrix's four rows is shifted to the left. Any 'falling off' entries are re-inserted on the right side of the row.
- <u>Mixing the columns:</u> A particular mathematical function is now used to alter each column of four bytes. This function takes four bytes from one column as input and returns four entirely new bytes that replace the original column. As a result, a new matrix with 16 additional bytes is created. It's worth noting that this stage is skipped in the final round.
- <u>Adding the round key:</u> The matrix's 16 bytes are now treated as 128 bits, and they are XORed with the round key's 128 bits. The ciphertext is the output if it is the final round. Otherwise, the 128 bits are interpreted as 16 bytes, and the process repeats again.

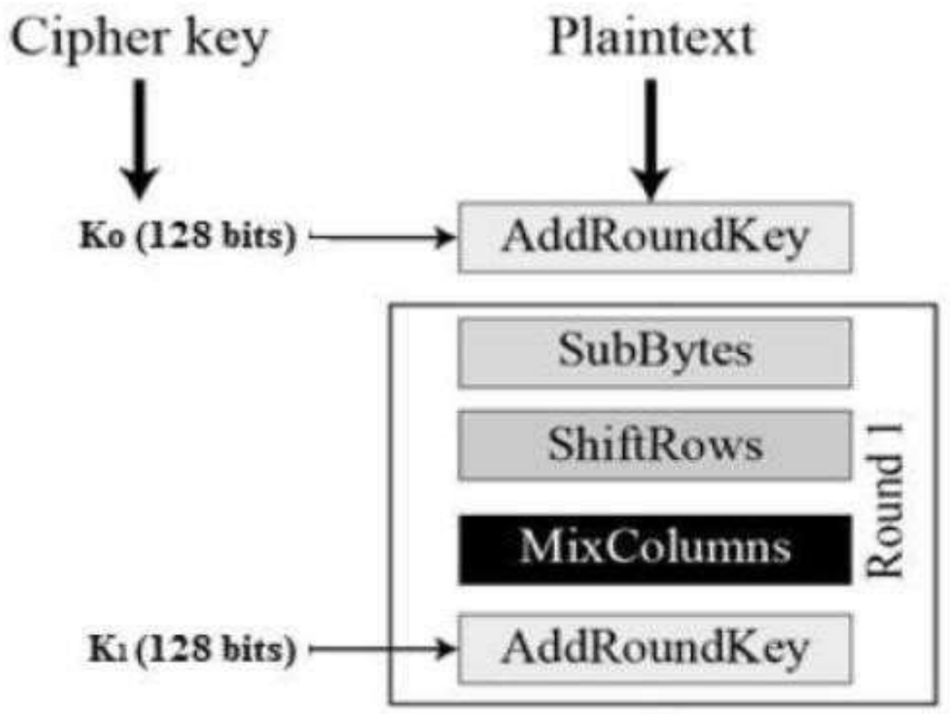


*Figure 10Adding the round key*

In decryption, all these four processes are conducted in reverse order.

## RSA(Rivest, Shamir, Adleman) Algorithm

The most widely used cryptography algorithm is the RSA algorithm. The RSA algorithm was created by three men named Ron Rivest, Adi Shamir, and Len Adlemen in 1977. In 1978, it was initially published.

It's a public-key encryption technique that is used to send safe data over the internet. Because of the technology's typical encryption mechanism, sending confidential and sensitive data over the internet is safe. For security purposes, a code is added to the standard message in this algorithm. The algorithm is based on huge number factorization. Because large numbers are difficult to factor, breaking into the message for intruders is challenging.

The difficulty of factoring huge integers that are the product of two large prime numbers is the source of RSA's security. Multiplying these two integers is simple, but factoring — or obtaining the original prime numbers from the total — is regarded impractical due to the time involved.

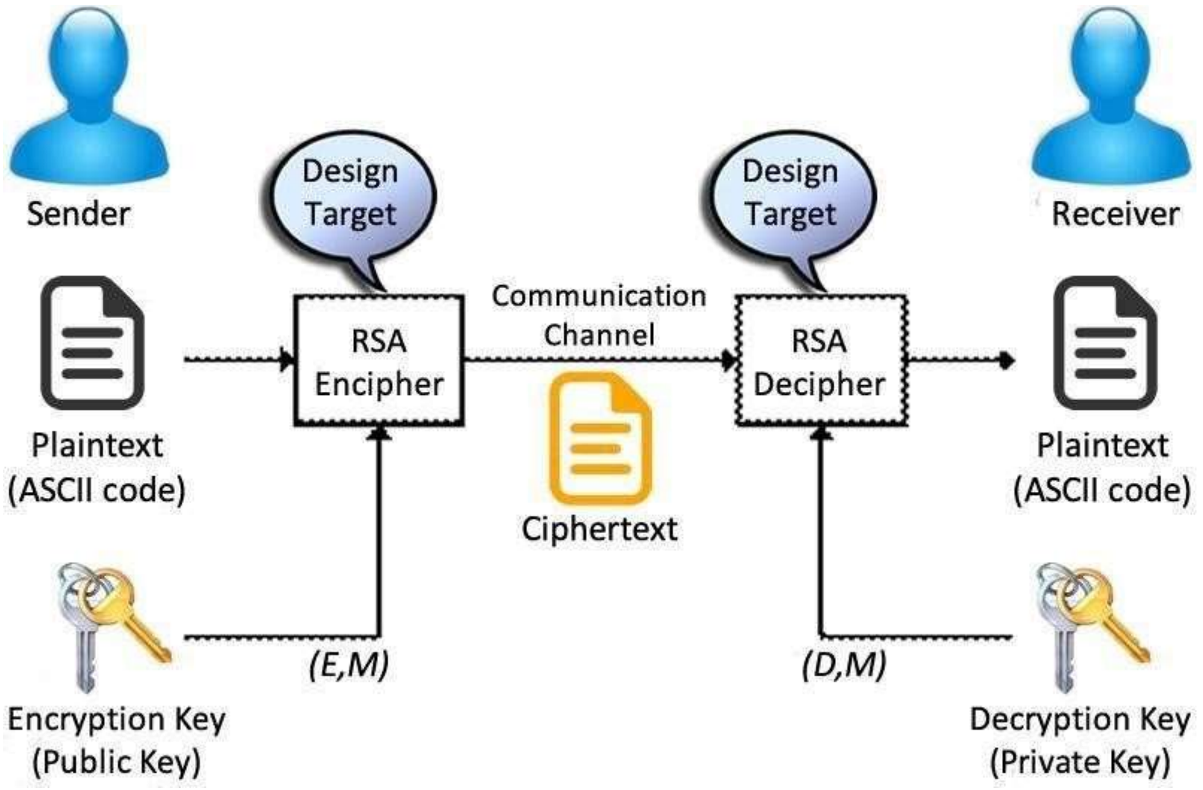


*Figure 11RSA algorithm*

## RSA algorithm steps

- Select two large prime numbers, a and b.
- Multiply these numbers to find s = a x b, where s is called the modulus for encryption and decryption.
- Choose a number e less than s, such that s is relatively prime to (a - 1) x (b -1). It means that e and (a - 1) x (b - 1) have no common factor except 1. Choose "e" such that 1<e < ϕ (s), e is prime to ϕ (s),
- gcd (e,d(s)) =1
- If s = a x b, then the public key is <e, s>. A plaintext message r is encrypted using public key <e, s>. To find ciphertext from the plain text following formula is used to get ciphertext C.

C = re mod s

Here, r must be less than s. A larger message (>s) is treated as a concatenation of messages, each of which is encrypted separately.

- To determine the private key, we use the following formula to calculate the d such that:

De mod {(a - 1) x (b - 1)} = 1

Or

De mod ϕ (s) = 1

- The private key is <d, s>. A ciphertext message c is decrypted using private key <d, s>. To calculate plain text r from the ciphertext c following formula is used to get plain text r.

- r = cd mod s

## Blowfish Algorithm

Blowfish is a symmetric-key block cipher created by Bruce Schneier in 1993 and found in a number of cipher suites and encryption products. Blowfish is another algorithm that aims to create a secure cloud environment. By splitting messages down into 64-bit blocks, this symmetric utility decrypts them. Blowfish are known for their speed, adaptability, and unbreak ability. It's also free, thanks to the fact that it's in the public domain, which adds to its appeal. E-commerce platforms, payment security, and password management software all use Blowfish. Blowfish algorithm is one of the fastest block cipher algorithms available for public use. Blowfish is not subject to any patents and is freely available for any one to use.

- In comparison to other encryption techniques, the Blowfish algorithm requires less operations to finish.
- Blowfish's key schedule is lengthy, yet this can be useful because brute force attacks are more difficult.

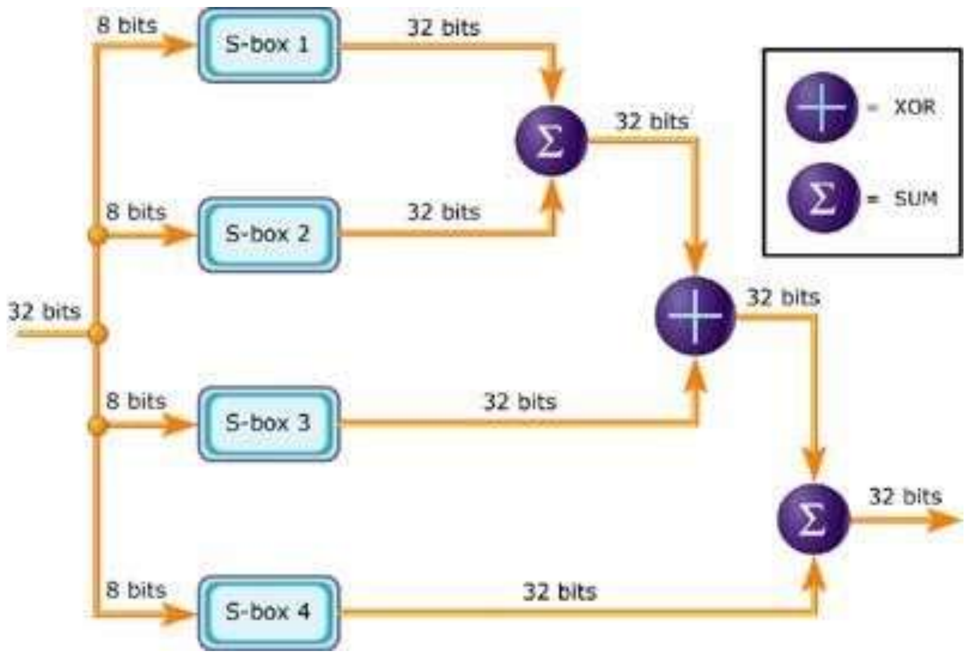
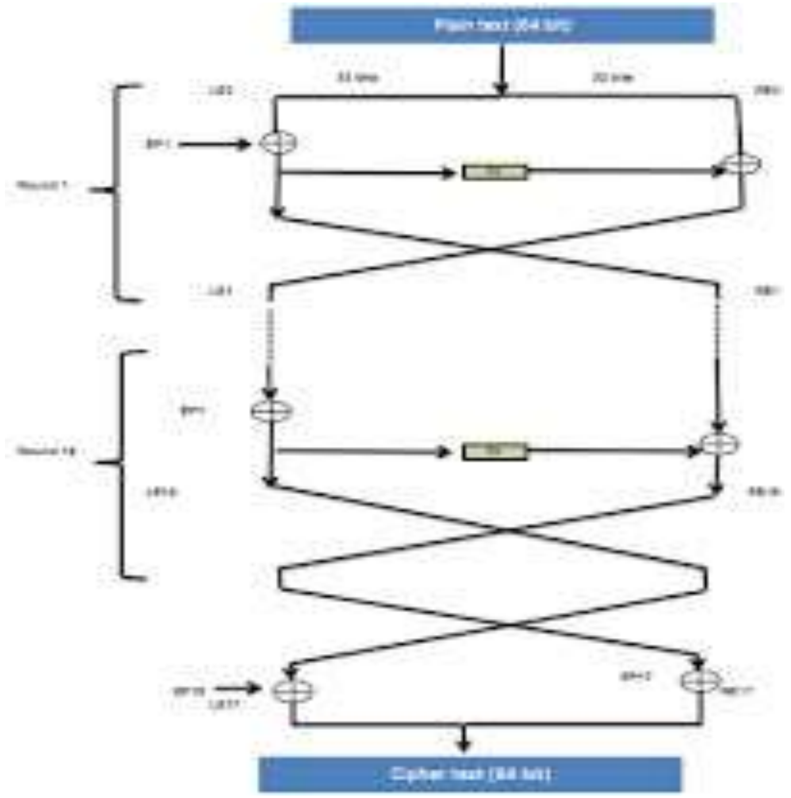


*Figure 12Blowfish Algorithm*

## Blowfish Algorithm steps

- The Blowfish algorithm uses a 64-bit block size, and the resulting key is between 32 and 448 bits long. The algorithm is divided into two sections. The first is for key expansion, while the second is for data encryption.
- The key expansion transforms the 448 bits of a key into subkeys once it receives the request, making the array 4168 bytes long.
- The algorithm now employs a 16-round Feistel cypher as well as big key-dependent S-boxes for data encryption. S-boxes are necessary components of symmetric key algorithms that use the substitution approach.
- Each cycle of substitution in the S-boxes has a different permutation key.



## LSB Stenography

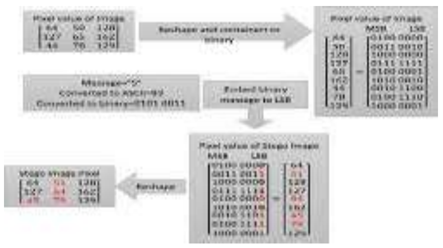
LSB steganography (Least Significant Bit steganography) is a method of hiding information within a digital image by replacing the least significant bit of each pixel with a bit of the hidden message. Since the least significant bit has minimal impact on the visual appearance of an image, the change is imperceptible to the human eye.

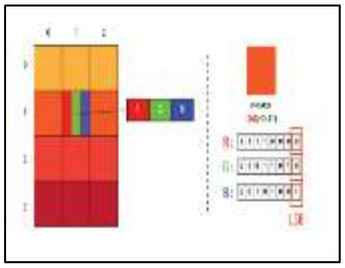
To perform LSB steganography, the following steps can be followed:

1. Convert the secret message to binary.
2. Choose a cover image.
3. Convert the cover image into binary pixels.
4. Replace the least significant bit of each pixel in the cover image with one bit from the secret message.
5. Create a stego-image by converting the modified binary pixels back to an image format.

To retrieve the hidden message from the stego-image, the reverse process is used. The least significant bit of each pixel in the stego-image is extracted and concatenated to form the hidden message in binary. This message can then be converted back to its original form.

LSB steganography is a popular and relatively simple method of hiding information within images. However, it has some limitations, including its susceptibility to attacks `` can extract the hidden information and its tendency to increase the size of the cover image.





# Chapter-3 Research Work

## Parameters of Evaluation

Each encryption algorithm presents strengths and weaknesses in terms of their parameters. Some parameters that determine encryption performance are described as follows:

1) Encryption time: Measured in milliseconds, depend on the data block length and key length. It directly influences the performance of the encryption algorithm. The performance of an algorithm is regarded as advanced when the encryption time is rapid.

2) Decryption time: The time period to regain the original text from ciphertext; it is also measured in milliseconds. The performance of an algorithm is regarded as superior when the decryption time is rapid.

3) Memory used: A low memory usage is desirable because it affects system cost.

4) Throughput: Is computed through way of dividing the whole encoded block size on the entire encode time. The power consumption of the algorithm will decrease, if the throughput cost increases

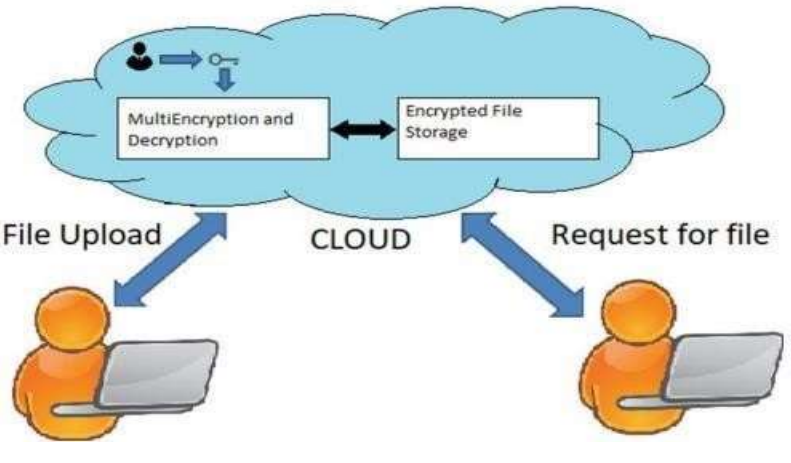


*Figure 13Cloud Model*

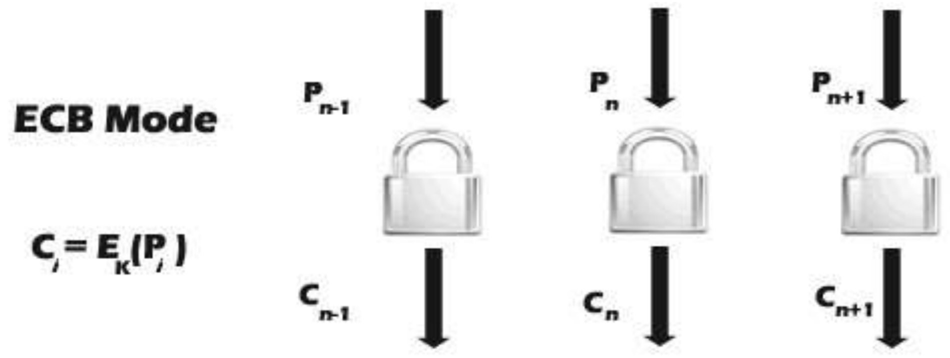
One of the main categorization methods for encryption techniques commonly used is based on the form of the input data they operate on. The two types are Block Cipher and Stream Cipher. This section discusses the main features in the two types, operation mode, and compares between them in terms of security and performance.

**Block Cipher**

Before starting to describe the key characteristics of block cipher, the definition of cipher word must be presented. "A cipher is an algorithm for performing encryption (reverse is decryption) ".

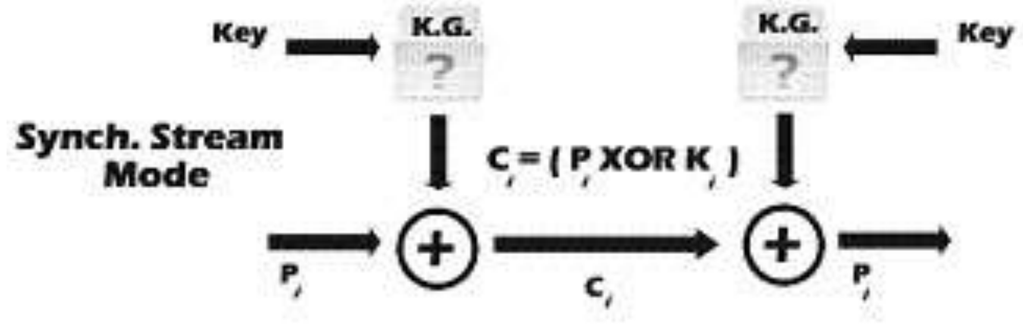
In this method data is encrypted and decrypted if data is in from of blocks. In its simplest mode, you divide the plain text into blocks which are then fed into the cipher system to produce blocks of cipher text.

ECB (Electronic Codebook Mode) is the basic form of clock cipher where data blocks are encrypted directly to generate its correspondent ciphered blocks. More discussion about modes of operations will be discussed later.



**Stream Cipher**

Stream cipher functions on a stream of data by operating on it bit by bit. Stream cipher consists of two major components: a key stream generator, and a mixing function. Mixing function is usually just an XOR function, while key stream generator is the main unit in stream cipher encryption technique. For example, if the key stream generator produces a series of zeros, the outputted ciphered stream will be identical to the original plain text. Figure below shows the operation of the simple mode in stream cipher.



## Mode of Operations

This section explains the two most common modes of operations in Block Cipher encryption-ECB and CBC- with a quick visit to other modes.

There are many variances of block cipher, where different techniques are used to strengthen the security of the system. The most common methods are: ECB (Electronic Codebook Mode), CBC (Chain Block Chaining Mode), and OFB (Output Feedback Mode). ECB mode is the CBC mode uses the cipher block from the previous step of encryption in the current one, which forms a chain-like encryption process. OFB operates on plain text in away similar to stream cipher that will be described below, where the encryption key used in every step depends on the encryption key from the previous step.

There are many other modes like CTR (counter), CFB (Cipher Feedback), or 3DES specific modes that are not discussed in this paper due to the fact that in this paper the main concentration will be on ECB and CBC modes.

## Work Results

To give more prospective about the performance of the compared algorithms, this section discusses the results obtained from other resources.

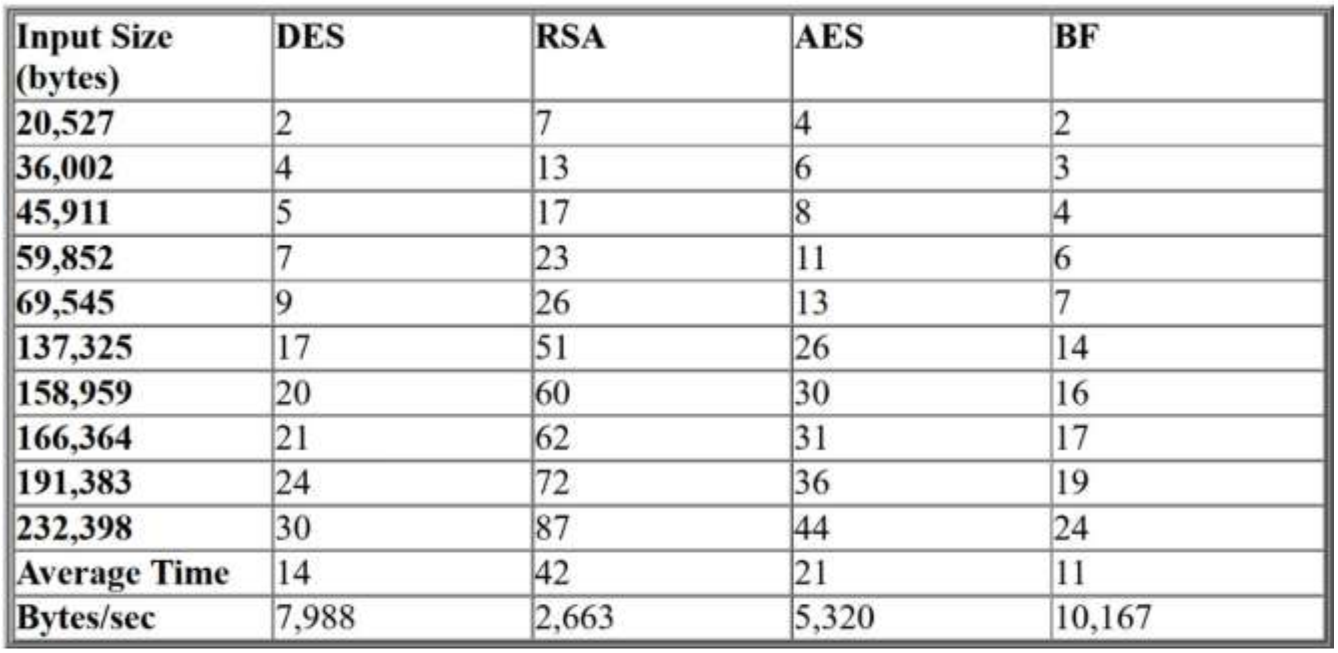
One of the known cryptography libraries is Crypto++. Crypto++ Library is a free C++ class library of cryptographic schemes. Currently the library consists of the following, some of which are other people's code, repackaged into classes.

The speed benchmarks for some of the most commonly used cryptographic algorithms were tested. All were coded in C++, compiled with Microsoft Visual C++ .NET 2003 (whole program optimization, optimize for speed, P4 code generation), and ran on a Pentium 4 2.1 GHz processor under Windows XP SP 1. 386 assembly routines were used for multipleprecision addition and subtraction. SSE2 intrinsics were used for multiple-precision multiplication.

It can be noticed from the table that not all the modes have been tried for all the algorithms. Nonetheless, these results are good to have an indication about what the presented comparison results should look like.

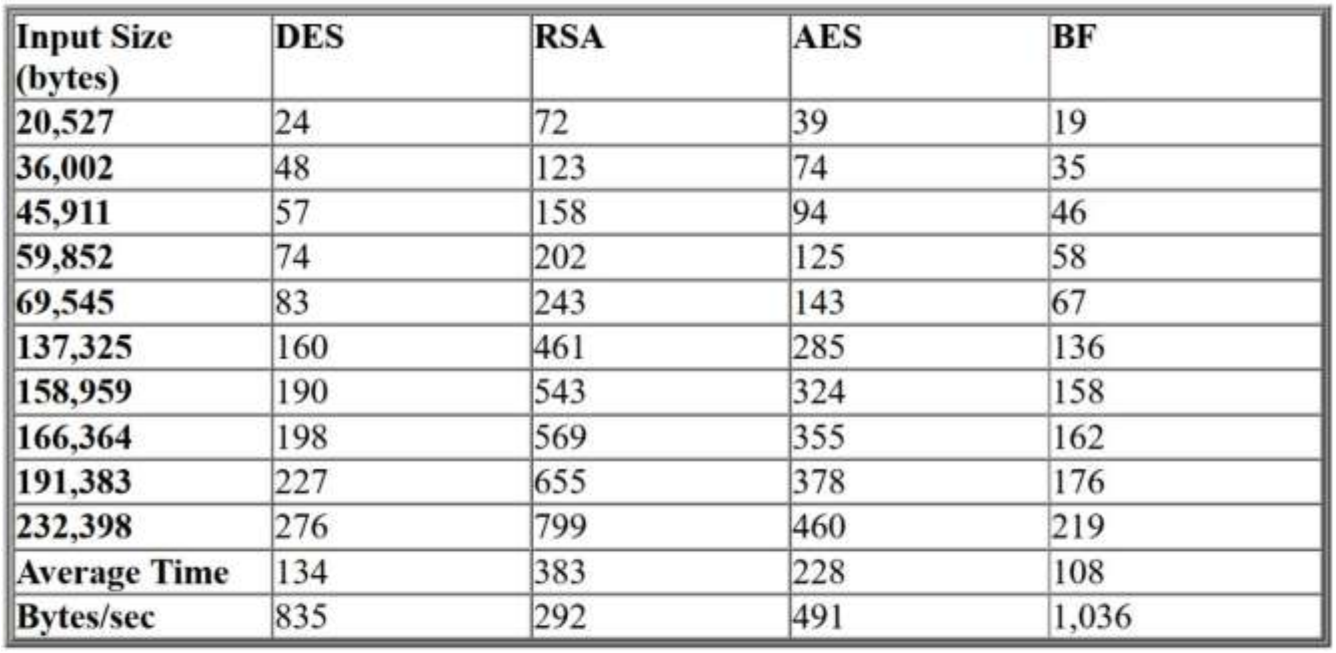
Also it is shown that Blowfish and AES have the best performance among others. And both of them are known to have better encryption (i.e. stronger against data attacks) than the other two.

In conclusion, secure file storage in the cloud using hybrid cryptography is essential for protecting sensitive data and ensuring that it remains secure, even in a cloud computing environment. The proposed system offers a more robust and secure file storage solution by combining three encryption techniques and hashing techniques for integrity. The system also ensures confidentiality by encrypting data in transit and at rest.
Tables 1 and 2 show the results of their experiments, where they have conducted it on two different machines: P-II 266 MHz and P-4 2.4 GHz.



| Input Size (bytes) | DES | RSA | AES | BF |
|---|---|---|---|---|
| 20,527 | 2 | 7 | 4 | 2 |
| 36,002 | 4 | 13 | 6 | 3 |
| 45,911 | 5 | 17 | 8 | 4 |
| 59,852 | 7 | 23 | 11 | 6 |
| 69,545 | 9 | 26 | 13 | 7 |
| 137,325 | 17 | 51 | 26 | 14 |
| 158,959 | 20 | 60 | 30 | 16 |
| 166,364 | 21 | 62 | 31 | 17 |
| 191,383 | 24 | 72 | 36 | 19 |
| 232,398 | 30 | 87 | 44 | 24 |
| Average Time | 14 | 42 | 21 | 11 |
| Bytes/sec | 7,988 | 2,663 | 5,320 | 10,167 |

Comparative execution times (in seconds) of encryption algorithms in ECB mode on a P-4
2.4 GHz machine



| Input Size (bytes) | DES | RSA | AES | BF |
|---|---|---|---|---|
| 20,527 | 24 | 72 | 39 | 19 |
| 36,002 | 48 | 123 | 74 | 35 |
| 45,911 | 57 | 158 | 94 | 46 |
| 59,852 | 74 | 202 | 125 | 58 |
| 69,545 | 83 | 243 | 143 | 67 |
| 137,325 | 160 | 461 | 285 | 136 |
| 158,959 | 190 | 543 | 324 | 158 |
| 166,364 | 198 | 569 | 355 | 162 |
| 191,383 | 227 | 655 | 378 | 176 |
| 232,398 | 276 | 799 | 460 | 219 |
| Average Time | 134 | 383 | 228 | 108 |
| Bytes/sec | 835 | 292 | 491 | 1,036 |

Comparative execution times (in seconds) of encryption algorithms in ECB mode on a P-II
266 MHz machine

From the results it is easy to observe that Blowfish has an advantage over other algorithms in terms of throughput. It has also conducted comparison between the algorithms in stream mode using CBC, but since this paper is more focused on block cipher the results were omitted.

The results showed that Blowfish has a very good performance compared to other algorithms. Also it showed that AES has a better performance than 3DES and DES. Amazingly it shows also that 3DES has almost 1/3 throughput of DES, or in other words it needs 3 times than DES to process the same amount of data.

The comparison was performed on the following algorithms: DES, Triple DES (3DES), RC2 and AES (Rijndael). The results shows that AES outperformed other algorithms in both the number of requests processes per second in different user loads, and in the response time in different user-load situations.

This section gave an overview of comparison results achieved by other people in the field.

## Simulation Procedure

By considering different sizes of data blocks (0.5MB to 20MB) the algorithms were evaluated in terms of the time required to encrypt and decrypt the data block. All the implementations were exact to make sure that the results will be relatively fair and accurate.

The Simulation program accepts three inputs: Algorithm, Cipher Mode and data block size.

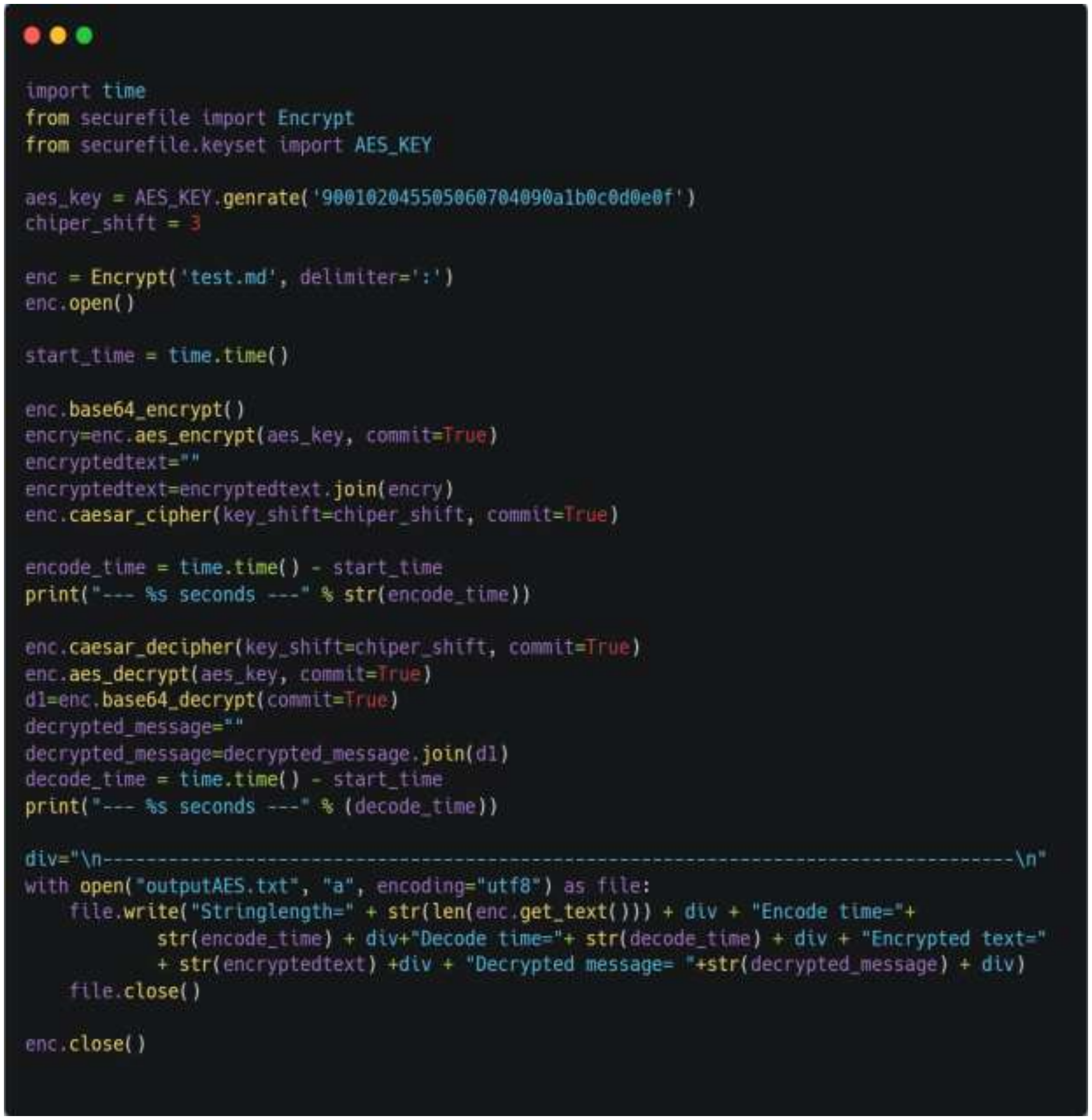
After a successful execution, the data generated, encrypted, and decrypted are shown. Notice that most of the characters cannot appear since they do not have character representation.

Another comparison is made after the successful encryption/decryption process to make sure that all the data are processed in the right way by comparing the generated data (the original data blocks) and the decrypted data block generated from the process.

# Chapter-4 Implementation of Algortihms

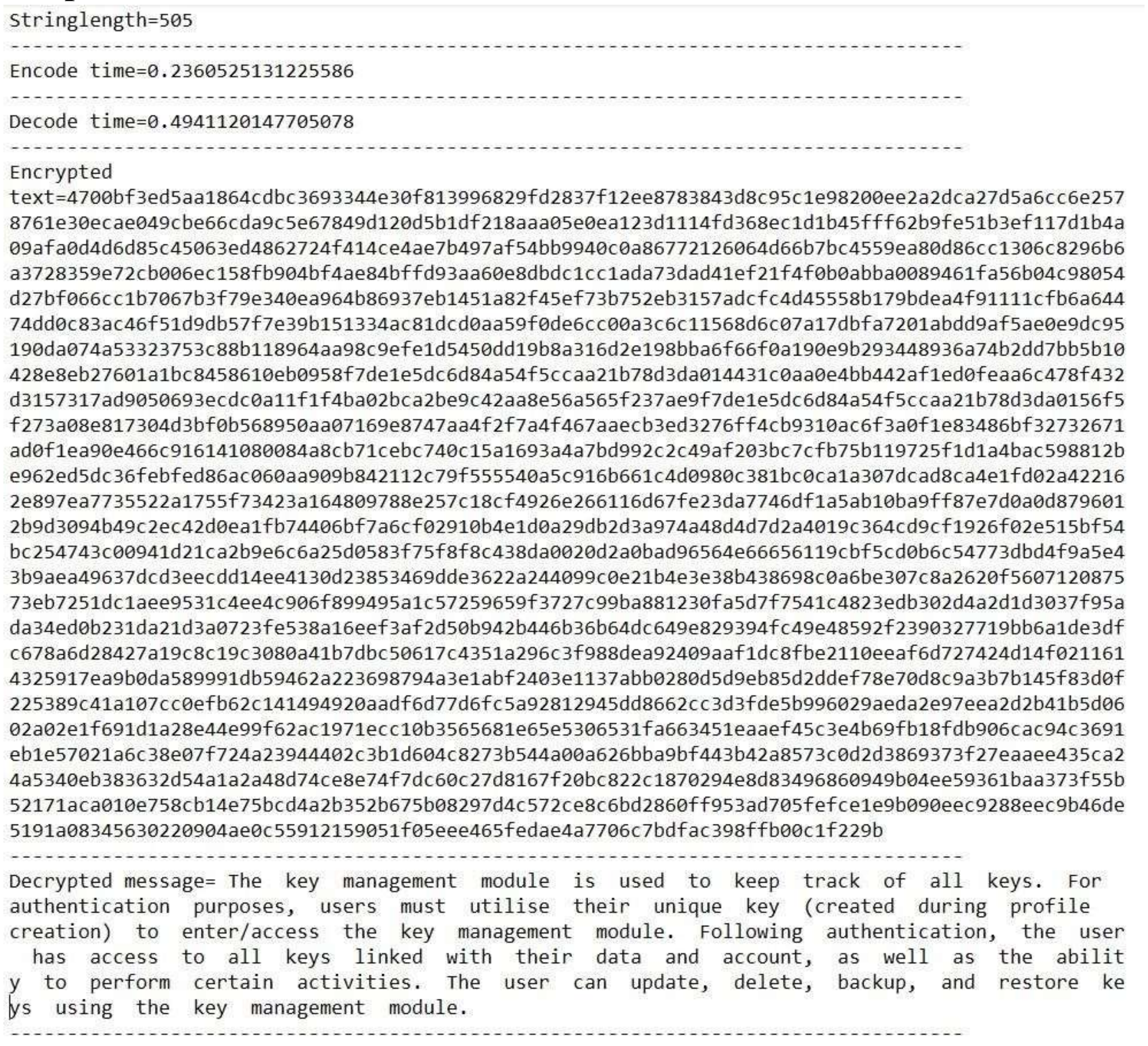
## Advanced Encryption Standard (AES) Algorithm

**Program:**



```
import time
from securefile import Encrypt
from securefile.keyset import AES_KEY

aes_key = AES_KEY.genrate('900102045505060704090a1b0c0d0e0f')
chiper_shift = 3

enc = Encrypt('test.md', delimiter=':')
enc.open()

start_time = time.time()

enc.base64_encrypt()
encry=enc.aes_encrypt(aes_key, commit=True)
encryptedtext=""
encryptedtext=encryptedtext.join(encry)
enc.caesar_cipher(key_shift=chiper_shift, commit=True)

encode_time = time.time() - start_time
print("--- %s seconds ---" % str(encode_time))

enc.caesar_decipher(key_shift=chiper_shift, commit=True)
enc.aes_decrypt(aes_key, commit=True)
d1=enc.base64_decrypt(commit=True)
decrypted_message=""
decrypted_message=decrypted_message.join(d1)
decode_time = time.time() - start_time
print("--- %s seconds ---" % (decode_time))

div="\n---------------------------------------------------------------------------------\n"
with open("outputAES.txt", "a", encoding="utf8") as file:
    file.write("Stringlength=" + str(len(enc.get_text())) + div + "Encode time="+
            str(encode_time) + div+"Decode time="+ str(decode_time) + div + "Encrypted text="
            + str(encryptedtext) +div + "Decrypted message= "+str(decrypted_message) + div)
    file.close()

enc.close()
```

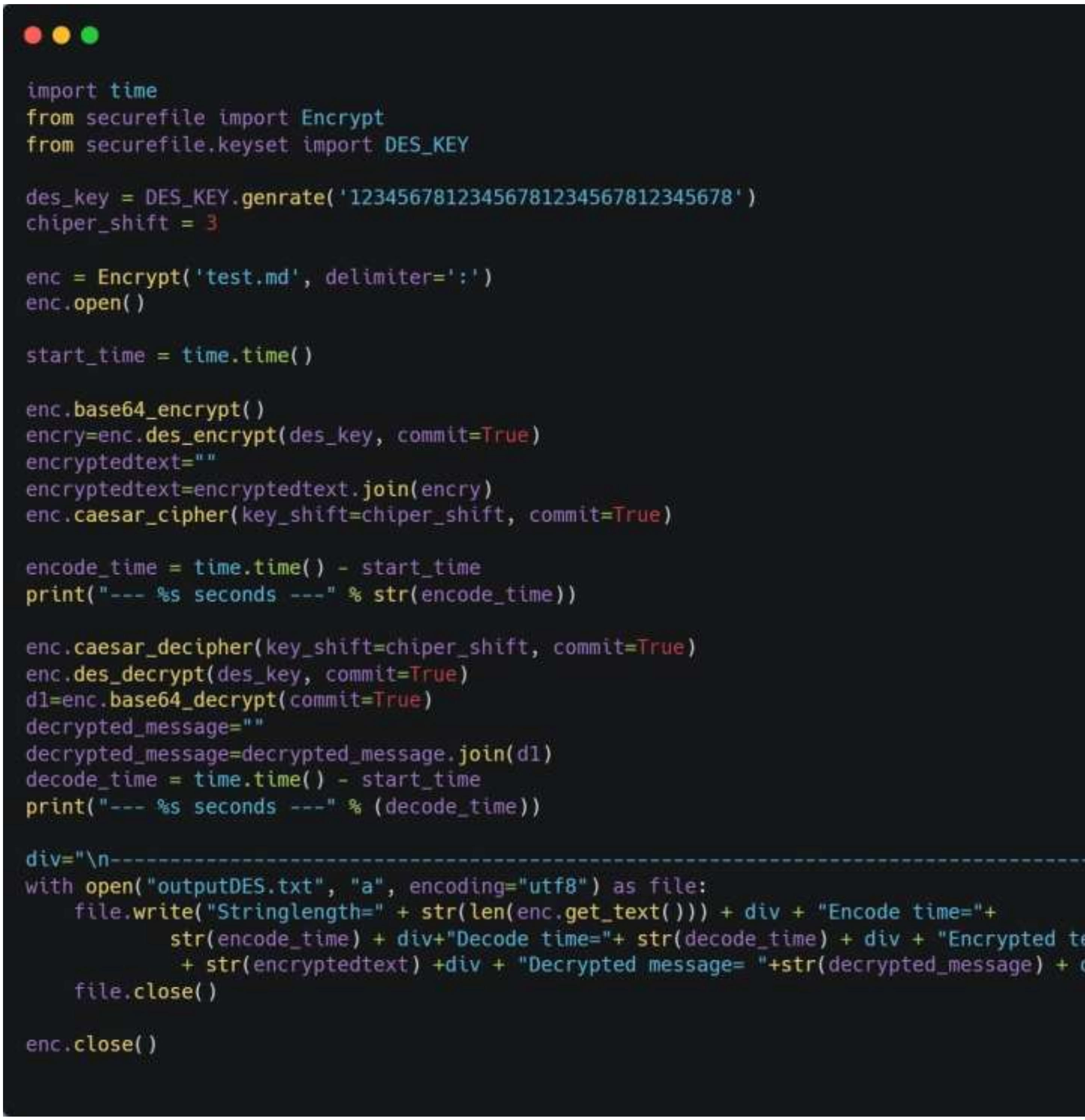
## Output:



```
Stringlength=505
--------------------------------------------------------------------------------
Encode time=0.2360525131225586
--------------------------------------------------------------------------------
Decode time=0.4941120147705078
--------------------------------------------------------------------------------
Encrypted
text=4700bf3ed5aa1864cdbc3693344e30f813996829fd2837f12ee8783843d8c95c1e98200ee2a2dca27d5a6cc6e257
8761e30ecae049cbe66cda9c5e67849d120d5b1df218aaa05e0ea123d1114fd368ec1d1b45fff62b9fe51b3ef117d1b4a
09afa0d4d6d85c45063ed4862724f414ce4ae7b497af54bb9940c0a86772126064d66b7bc4559ea80d86cc1306c8296b6
a3728359e72cb006ec158fb904bf4ae84bffd93aa60e8dbdc1cc1ada73dad41ef21f4f0b0abba0089461fa56b04c98054
d27bf066cc1b7067b3f79e340ea964b86937eb1451a82f45ef73b752eb3157adcfc4d45558b179bdea4f91111cfb6a644
74dd0c83ac46f51d9db57f7e39b151334ac81dcd0aa59f0de6cc00a3c6c11568d6c07a17dbfa7201abdd9af5ae0e9dc95
190da074a53323753c88b118964aa98c9efe1d5450dd19b8a316d2e198bba6f66f0a190e9b293448936a74b2dd7bb5b10
428e8eb27601a1bc8458610eb0958f7de1e5dc6d84a54f5ccaa21b78d3da014431c0aa0e4bb442af1ed0feaa6c478f432
d3157317ad9050693ecdc0a11f1f4ba02bca2be9c42aa8e56a565f237ae9f7de1e5dc6d84a54f5ccaa21b78d3da0156f5
f273a08e817304d3bf0b568950aa07169e8747aa4f2f7a4f467aaecb3ed3276ff4cb9310ac6f3a0f1e83486bf32732671
ad0f1ea90e466c916141080084a8cb71cebc740c15a1693a4a7bd992c2c49af203bc7cfb75b119725f1d1a4bac598812b
e962ed5dc36febfed86ac060aa909b842112c79f555540a5c916b661c4d0980c381bc0ca1a307dcad8ca4e1fd02a42216
2e897ea7735522a1755f73423a164809788e257c18cf4926e266116d67fe23da7746df1a5ab10ba9ff87e7d0a0d879601
2b9d3094b49c2ec42d0ea1fb74406bf7a6cf02910b4e1d0a29db2d3a974a48d4d7d2a4019c364cd9cf1926f02e515bf54
bc254743c00941d21ca2b9e6c6a25d0583f75f8f8c438da0020d2a0bad96564e66656119cbf5cd0b6c54773dbd4f9a5e4
3b9aea49637dcd3eecdd14ee4130d23853469dde3622a244099c0e21b4e3e38b438698c0a6be307c8a2620f5607120875
73eb7251dc1aee9531c4ee4c906f899495a1c57259659f3727c99ba881230fa5d7f7541c4823edb302d4a2d1d3037f95a
da34ed0b231da21d3a0723fe538a16eef3af2d50b942b446b36b64dc649e829394fc49e48592f2390327719bb6a1de3df
c678a6d28427a19c8c19c3080a41b7dbc50617c4351a296c3f988dea92409aaf1dc8fbe2110eeaf6d727424d14f021161
4325917ea9b0da589991db59462a223698794a3e1abf2403e1137abb0280d5d9eb85d2ddef78e70d8c9a3b7b145f83d0f
225389c41a107cc0efb62c141494920aadf6d77d6fc5a92812945dd8662cc3d3fde5b996029aeda2e97eea2d2b41b5d06
02a02e1f691d1a28e44e99f62ac1971ecc10b3565681e65e5306531fa663451eaaef45c3e4b69fb18fdb906cac94c3691
eb1e57021a6c38e07f724a23944402c3b1d604c8273b544a00a626bba9bf443b42a8573c0d2d3869373f27eaaee435ca2
4a5340eb383632d54a1a2a48d74ce8e74f7dc60c27d8167f20bc822c1870294e8d83496860949b04ee59361baa373f55b
52171aca010e758cb14e75bcd4a2b352b675b08297d4c572ce8c6bd2860ff953ad705fefce1e9b090eec9288eec9b46de
5191a08345630220904ae0c55912159051f05eee465fedae4a7706c7bdfac398ffb00c1f229b
--------------------------------------------------------------------------------
Decrypted message= The  key  management  module  is  used  to  keep  track  of  all  keys.  For
authentication  purposes,  users  must  utilise  their  unique  key  (created  during  profile
creation)  to  enter/access  the  key  management  module.  Following  authentication,  the  user
  has  access  to  all  keys  linked  with  their  data  and  account,  as  well  as  the  abilit
y  to  perform  certain  activities.  The  user  can  update,  delete,  backup,  and  restore  ke
ys  using  the  key  management  module.
--------------------------------------------------------------------------------
```

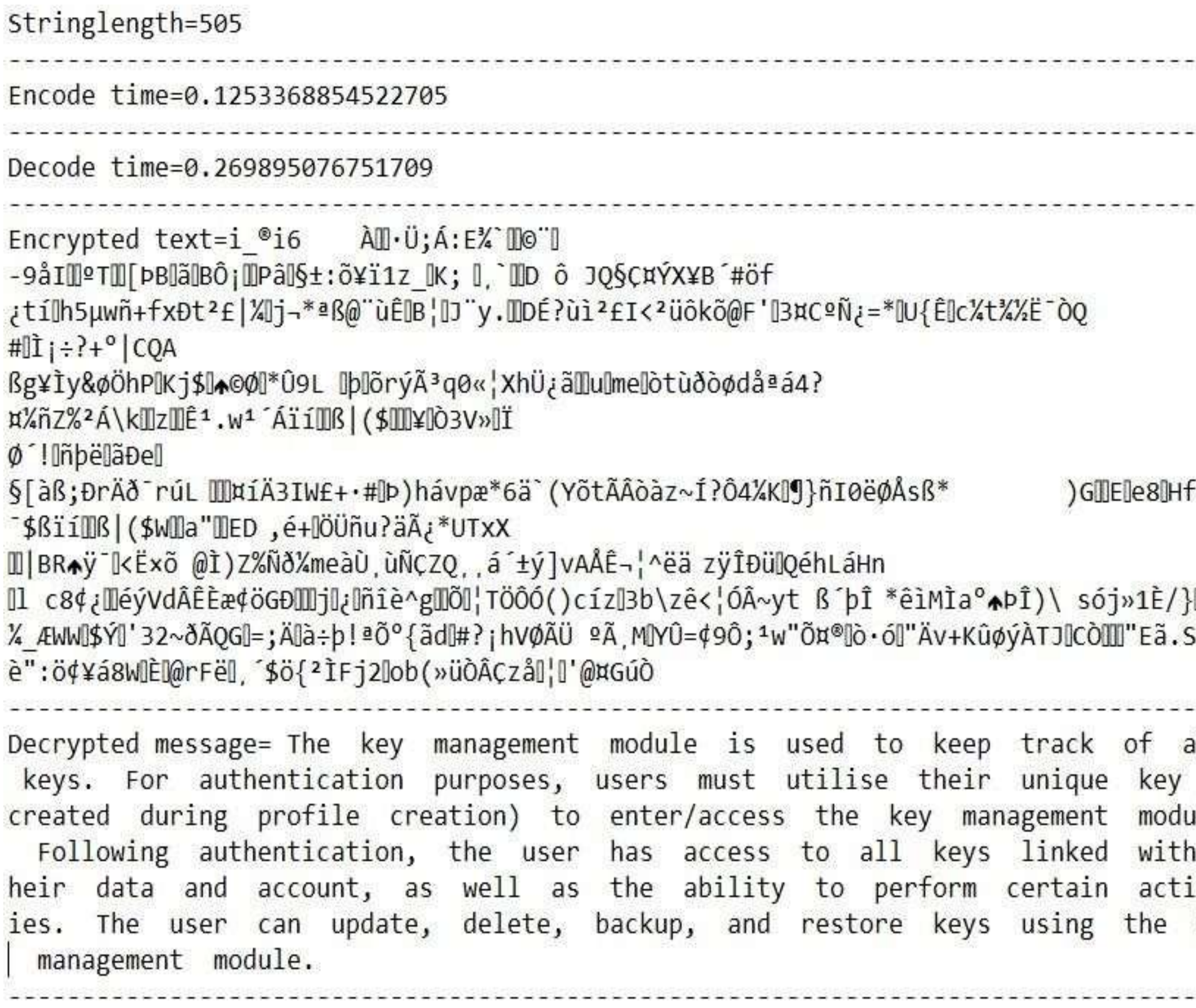
## Data Encryption Standard (DES) Algorithm

**Program:**



```python
import time
from securefile import Encrypt
from securefile.keyset import DES_KEY

des_key = DES_KEY.genrate('12345678123456781234567812345678')
chiper_shift = 3

enc = Encrypt('test.md', delimiter=':')
enc.open()

start_time = time.time()

enc.base64_encrypt()
encry=enc.des_encrypt(des_key, commit=True)
encryptedtext=""
encryptedtext=encryptedtext.join(encry)
enc.caesar_cipher(key_shift=chiper_shift, commit=True)

encode_time = time.time() - start_time
print("--- %s seconds ---" % str(encode_time))

enc.caesar_decipher(key_shift=chiper_shift, commit=True)
enc.des_decrypt(des_key, commit=True)
d1=enc.base64_decrypt(commit=True)
decrypted_message=""
decrypted_message=decrypted_message.join(d1)
decode_time = time.time() - start_time
print("--- %s seconds ---" % (decode_time))

div="\n-------------------------------------------------------------------------------------
with open("outputDES.txt", "a", encoding="utf8") as file:
    file.write("Stringlength=" + str(len(enc.get_text())) + div + "Encode time="+
            str(encode_time) + div+"Decode time="+ str(decode_time) + div + "Encrypted te
             + str(encryptedtext) +div + "Decrypted message= "+str(decrypted_message) + c
    file.close()

enc.close()
```

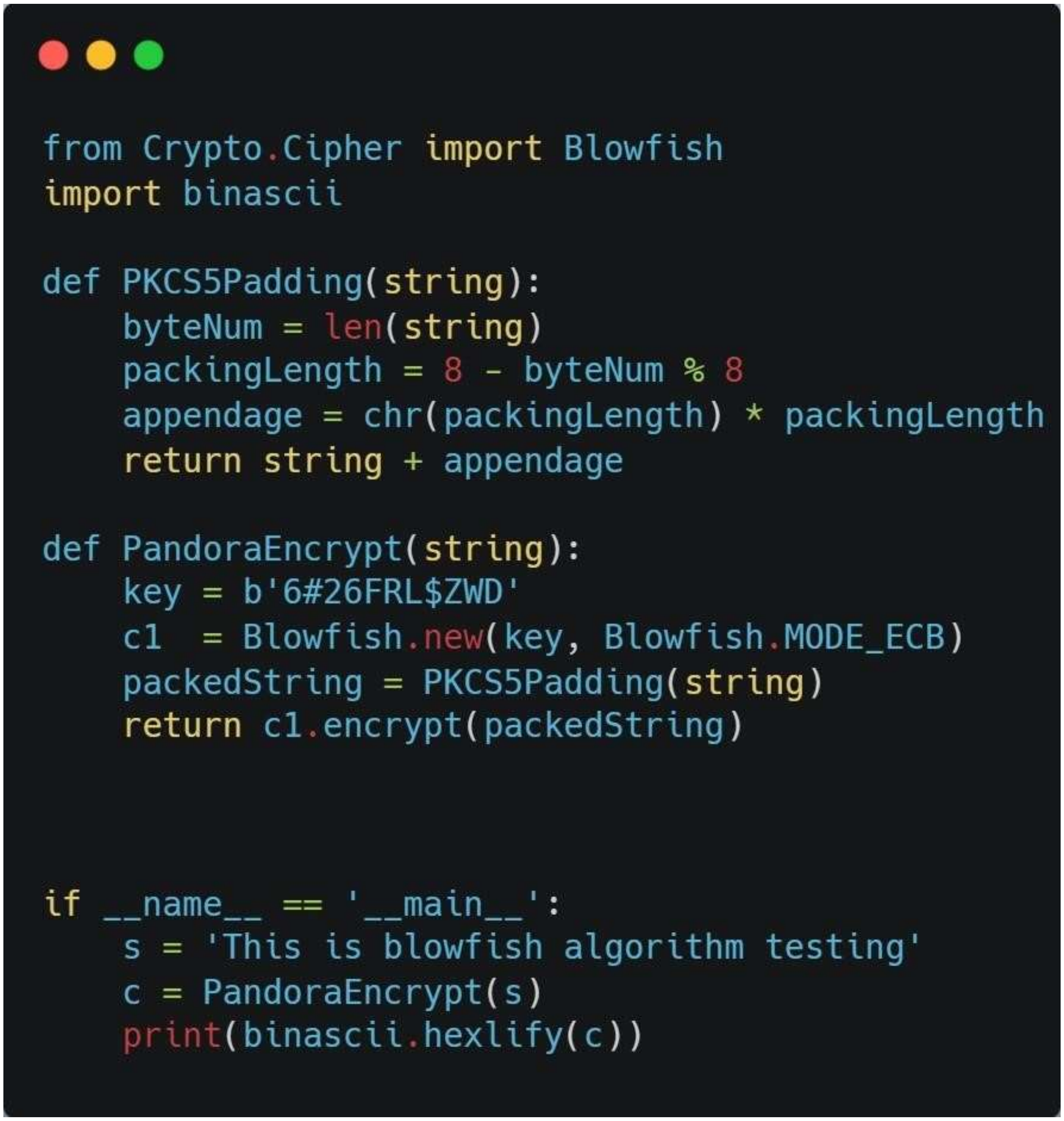
**Output:**



```
Stringlength=505
------------------------------------------------------------------------
Encode time=0.1253368854522705
------------------------------------------------------------------------
Decode time=0.269895076751709
------------------------------------------------------------------------
Encrypted text=i_®i6    À·Ü;Á:E¾`©¨
-9åIºT[ÞBãBÔ¡Pâ§±:õ¥ï1z_K; ¸`D ô JQ§Ç¤ÝX¥B´#öf
¿tíh5µwñ+fxÐt²£|¼j¬*ªß@¨ùÊB¦J¨y.DÉ?ùì²£I<²üôkõ@F'3¤CºÑ¿=*U{Êc¼t¾½Ë¯ÒQ
#Ì¡÷?+°|CQA
ßg¥Ìy&øÖhPKj$©Ø*Û9L þõrýÃ³q0«¦XhÜ¿ãumeòtùðòødåªá4?
¤¼ñZ%²Á\kzÊ¹.w¹´Áïíß|($¥Ò3V»Ï
Ø´!ñþëãÐe
§[àß;ÐrÄð¯rúL ¤íÄ3IW£+·#Þ)hávpæ*6ä`(YõtÃÂòàz~Í?Ô4¼K¶}ñI0ëØÅsß*       )GEe8Hf
¯$ßïíß|($Wa"ED ,é+ÖÜñu?äÃ¿*UTxX
|BRÿ¯<Ë×õ @Ì)Z%Ñð¼meàÙ¸ùÑÇZQ¸¸á´±ý]vAÅÊ¬¦^ëä zÿÎÐüQéhLáHn
l c8¢¿éýVdÂÊÈæ¢öGÐj¿ñîè^gÕ¦TÖÔÓ()cíz3b\zê<¦ÓÂ~yt ß´þÎ *êìMÌa°ÞÎ)\ sój»1È/}
¼_ÆWW$Ý'32~ðÃQG=;Äà÷þ!ªÕ°{ãd#?¡hVØÃÜ ºÃ¸MYÛ=¢9Ô;¹w"Õ¤®ò·ó"Äv+KûøýÀTJCÒ"Eã.S
è":ö¢¥á8WÈ@rFë¸´$ö{²ÌFj2ob(»üÒÂÇzå¦'@¤GúÒ
------------------------------------------------------------------------
Decrypted message= The  key  management  module  is  used  to  keep  track  of  a
 keys.  For  authentication  purposes,  users  must  utilise  their  unique  key
created  during  profile  creation)  to  enter/access  the  key  management  modu
  Following  authentication,  the  user  has  access  to  all  keys  linked  with
heir  data  and  account,  as  well  as  the  ability  to  perform  certain  acti
ies.  The  user  can  update,  delete,  backup,  and  restore  keys  using  the
| management  module.
------------------------------------------------------------------------
```

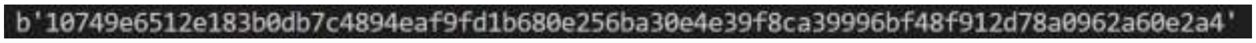
## Blowfish algorithm

**Program:**



```
from Crypto.Cipher import Blowfish
import binascii

def PKCS5Padding(string):
    byteNum = len(string)
    packingLength = 8 - byteNum % 8
    appendage = chr(packingLength) * packingLength
    return string + appendage

def PandoraEncrypt(string):
    key = b'6#26FRL$ZWD'
    c1  = Blowfish.new(key, Blowfish.MODE_ECB)
    packedString = PKCS5Padding(string)
    return c1.encrypt(packedString)



if __name__ == '__main__':
    s = 'This is blowfish algorithm testing'
    c = PandoraEncrypt(s)
    print(binascii.hexlify(c))
```

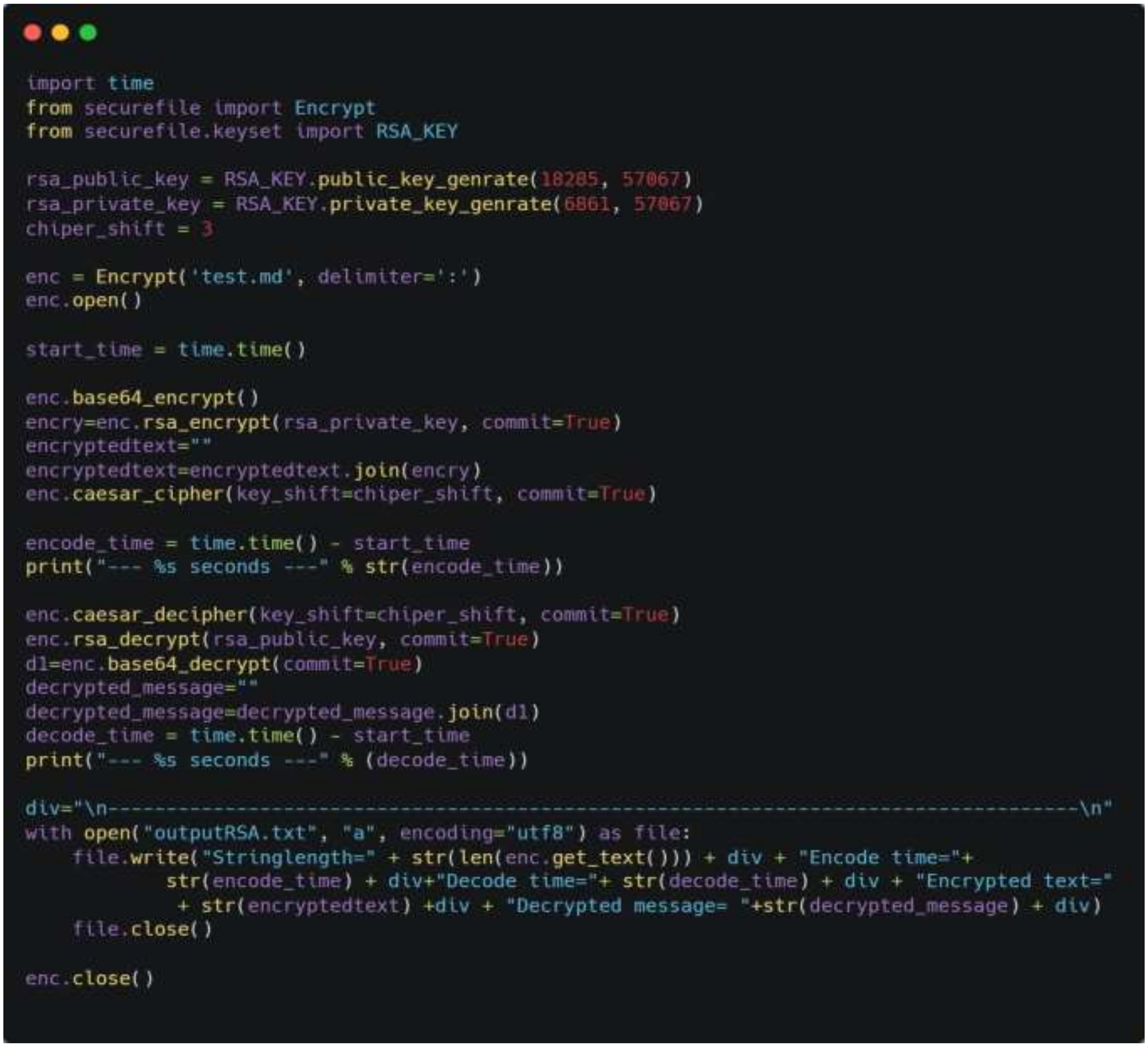
**Output:**



```
b'10749e6512e183b0db7c4894eaf9fd1b680e256ba30e4e39f8ca39996bf48f912d78a0962a60e2a4'
```

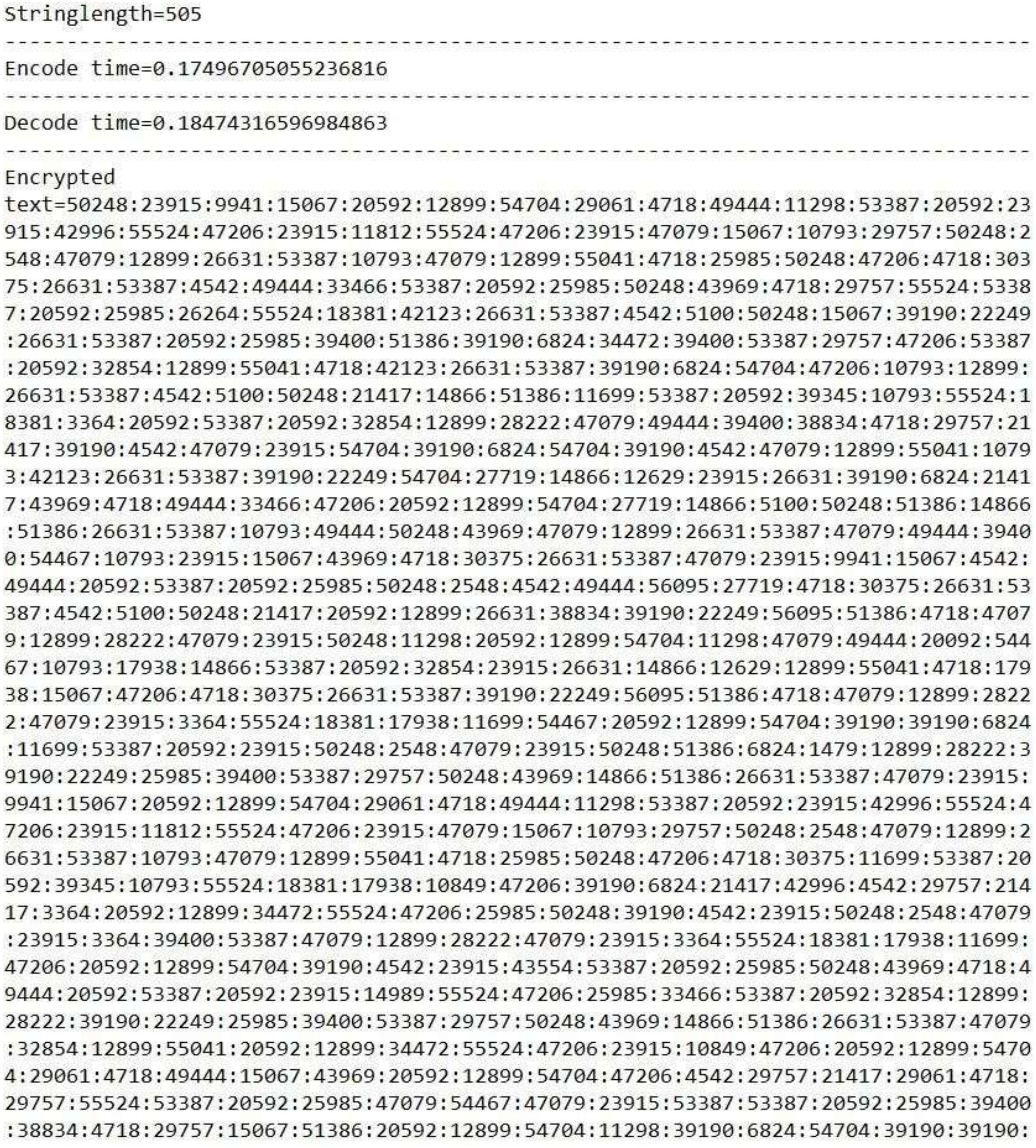
## RSA(Rivest,  Shamir,  Adleman) Algorithm

### Program:

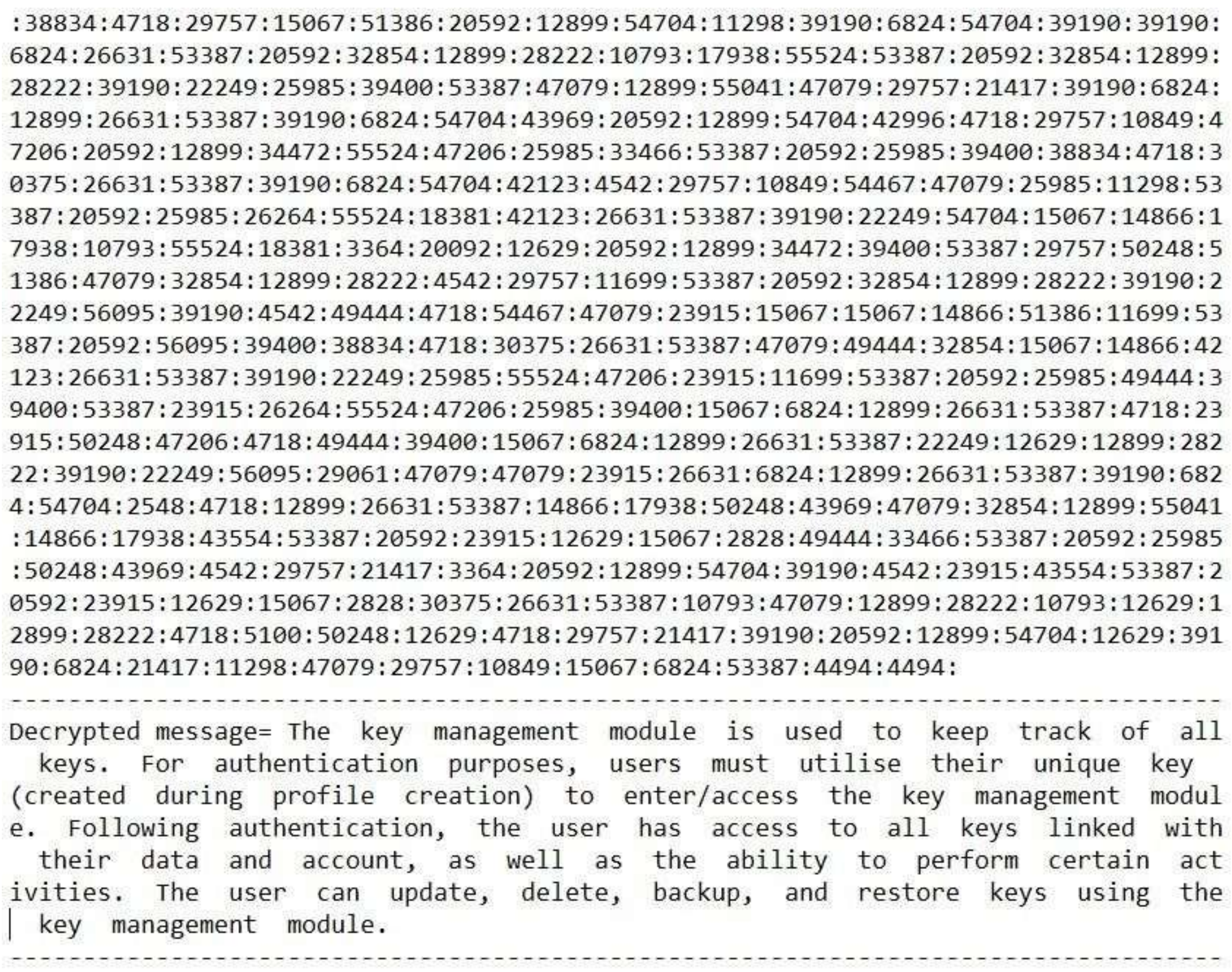


```python
import time
from securefile import Encrypt
from securefile.keyset import RSA_KEY

rsa_public_key = RSA_KEY.public_key_genrate(18285, 57067)
rsa_private_key = RSA_KEY.private_key_genrate(6861, 57067)
chiper_shift = 3

enc = Encrypt('test.md', delimiter=':')
enc.open()

start_time = time.time()

enc.base64_encrypt()
encry=enc.rsa_encrypt(rsa_private_key, commit=True)
encryptedtext=""
encryptedtext=encryptedtext.join(encry)
enc.caesar_cipher(key_shift=chiper_shift, commit=True)

encode_time = time.time() - start_time
print("--- %s seconds ---" % str(encode_time))

enc.caesar_decipher(key_shift=chiper_shift, commit=True)
enc.rsa_decrypt(rsa_public_key, commit=True)
d1=enc.base64_decrypt(commit=True)
decrypted_message=""
decrypted_message=decrypted_message.join(d1)
decode_time = time.time() - start_time
print("--- %s seconds ---" % (decode_time))

div="\n-------------------------------------------------------------------------------------\n"
with open("outputRSA.txt", "a", encoding="utf8") as file:
    file.write("Stringlength=" + str(len(enc.get_text())) + div + "Encode time="+
            str(encode_time) + div+"Decode time="+ str(decode_time) + div + "Encrypted text="
             + str(encryptedtext) +div + "Decrypted message= "+str(decrypted_message) + div)
    file.close()

enc.close()
```

**Output:**



```
Stringlength=505
--------------------------------------------------------------------------------
Encode time=0.17496705055236816
--------------------------------------------------------------------------------
Decode time=0.18474316596984863
--------------------------------------------------------------------------------
Encrypted
text=50248:23915:9941:15067:20592:12899:54704:29061:4718:49444:11298:53387:20592:23
915:42996:55524:47206:23915:11812:55524:47206:23915:47079:15067:10793:29757:50248:2
548:47079:12899:26631:53387:10793:47079:12899:55041:4718:25985:50248:47206:4718:303
75:26631:53387:4542:49444:33466:53387:20592:25985:50248:43969:4718:29757:55524:5338
7:20592:25985:26264:55524:18381:42123:26631:53387:4542:5100:50248:15067:39190:22249
:26631:53387:20592:25985:39400:51386:39190:6824:34472:39400:53387:29757:47206:53387
:20592:32854:12899:55041:4718:42123:26631:53387:39190:6824:54704:47206:10793:12899:
26631:53387:4542:5100:50248:21417:14866:51386:11699:53387:20592:39345:10793:55524:1
8381:3364:20592:53387:20592:32854:12899:28222:47079:49444:39400:38834:4718:29757:21
417:39190:4542:47079:23915:54704:39190:6824:54704:39190:4542:47079:12899:55041:1079
3:42123:26631:53387:39190:22249:54704:27719:14866:12629:23915:26631:39190:6824:2141
7:43969:4718:49444:33466:47206:20592:12899:54704:27719:14866:5100:50248:51386:14866
:51386:26631:53387:10793:49444:50248:43969:47079:12899:26631:53387:47079:49444:3940
0:54467:10793:23915:15067:43969:4718:30375:26631:53387:47079:23915:9941:15067:4542:
49444:20592:53387:20592:25985:50248:2548:4542:49444:56095:27719:4718:30375:26631:53
387:4542:5100:50248:21417:20592:12899:26631:38834:39190:22249:56095:51386:4718:4707
9:12899:28222:47079:23915:50248:11298:20592:12899:54704:11298:47079:49444:20092:544
67:10793:17938:14866:53387:20592:32854:23915:26631:14866:12629:12899:55041:4718:179
38:15067:47206:4718:30375:26631:53387:39190:22249:56095:51386:4718:47079:12899:2822
2:47079:23915:3364:55524:18381:17938:11699:54467:20592:12899:54704:39190:39190:6824
:11699:53387:20592:23915:50248:2548:47079:23915:50248:51386:6824:1479:12899:28222:3
9190:22249:25985:39400:53387:29757:50248:43969:14866:51386:26631:53387:47079:23915:
9941:15067:20592:12899:54704:29061:4718:49444:11298:53387:20592:23915:42996:55524:4
7206:23915:11812:55524:47206:23915:47079:15067:10793:29757:50248:2548:47079:12899:2
6631:53387:10793:47079:12899:55041:4718:25985:50248:47206:4718:30375:11699:53387:20
592:39345:10793:55524:18381:17938:10849:47206:39190:6824:21417:42996:4542:29757:214
17:3364:20592:12899:34472:55524:47206:25985:50248:39190:4542:23915:50248:2548:47079
:23915:3364:39400:53387:47079:12899:28222:47079:23915:3364:55524:18381:17938:11699:
47206:20592:12899:54704:39190:4542:23915:43554:53387:20592:25985:50248:43969:4718:4
9444:20592:53387:20592:23915:14989:55524:47206:25985:33466:53387:20592:32854:12899:
28222:39190:22249:25985:39400:53387:29757:50248:43969:14866:51386:26631:53387:47079
:32854:12899:55041:20592:12899:34472:55524:47206:23915:10849:47206:20592:12899:5470
4:29061:4718:49444:15067:43969:20592:12899:54704:47206:4542:29757:21417:29061:4718:
29757:55524:53387:20592:25985:47079:54467:47079:23915:53387:53387:20592:25985:39400
:38834:4718:29757:15067:51386:20592:12899:54704:11298:39190:6824:54704:39190:39190:
```



```
:38834:4718:29757:15067:51386:20592:12899:54704:11298:39190:6824:54704:39190:39190:
6824:26631:53387:20592:32854:12899:28222:10793:17938:55524:53387:20592:32854:12899:
28222:39190:22249:25985:39400:53387:47079:12899:55041:47079:29757:21417:39190:6824:
12899:26631:53387:39190:6824:54704:43969:20592:12899:54704:42996:4718:29757:10849:4
7206:20592:12899:34472:55524:47206:25985:33466:53387:20592:25985:39400:38834:4718:3
0375:26631:53387:39190:6824:54704:42123:4542:29757:10849:54467:47079:25985:11298:53
387:20592:25985:26264:55524:18381:42123:26631:53387:39190:22249:54704:15067:14866:1
7938:10793:55524:18381:3364:20092:12629:20592:12899:34472:39400:53387:29757:50248:5
1386:47079:32854:12899:28222:4542:29757:11699:53387:20592:32854:12899:28222:39190:2
2249:56095:39190:4542:49444:4718:54467:47079:23915:15067:15067:14866:51386:11699:53
387:20592:56095:39400:38834:4718:30375:26631:53387:47079:49444:32854:15067:14866:42
123:26631:53387:39190:22249:25985:55524:47206:23915:11699:53387:20592:25985:49444:3
9400:53387:23915:26264:55524:47206:25985:39400:15067:6824:12899:26631:53387:4718:23
915:50248:47206:4718:49444:39400:15067:6824:12899:26631:53387:22249:12629:12899:282
22:39190:22249:56095:29061:47079:47079:23915:26631:6824:12899:26631:53387:39190:682
4:54704:2548:4718:12899:26631:53387:14866:17938:50248:43969:47079:32854:12899:55041
:14866:17938:43554:53387:20592:23915:12629:15067:2828:49444:33466:53387:20592:25985
:50248:43969:4542:29757:21417:3364:20592:12899:54704:39190:4542:23915:43554:53387:2
0592:23915:12629:15067:2828:30375:26631:53387:10793:47079:12899:28222:10793:12629:1
2899:28222:4718:5100:50248:12629:4718:29757:21417:39190:20592:12899:54704:12629:391
90:6824:21417:11298:47079:29757:10849:15067:6824:53387:4494:4494:
----------------------------------------------------------------------------------
Decrypted message= The  key  management  module  is  used  to  keep  track  of  all
  keys.  For  authentication  purposes,  users  must  utilise  their  unique  key
(created  during  profile  creation)  to  enter/access  the  key  management  modul
e.  Following  authentication,  the  user  has  access  to  all  keys  linked  with
  their  data  and  account,  as  well  as  the  ability  to  perform  certain  act
ivities.  The  user  can  update,  delete,  backup,  and  restore  keys  using  the
  key  management  module.
----------------------------------------------------------------------------------
```

# Conclusion

We examined numerous cryptographic methods and their various components in this research work in order to safeguard cloud infrastructure. Cloud computing is an emerging, next-generation technology that is rapidly gaining traction around the world. It offers various benefits, but there are still some security issues with this technology. There are numerous challenges in implementing various techniques, yet each technique overcomes threats in some capacity. We also explored numerous domains and sub-techniques of cryptography algorithms in this study. Cryptography works on the principle of data confidentiality, integrity and authentication. One of the main goals of this study was to gather possible solutions for securing our data on the cloud. We've discussed and analyzed the performance of Blowfish Encryption Algorithm, Advanced Encryption Standard (AES), Data Encryption Standard (DES), RivestShamir-dleman encryption (RSA), and homomorphic algorithm, among other cryptographic algorithms.
Due to cost efficiency and less hands on management, data owners are very much interested in outsourcing their data in cloud which can access to data as a service. In this paper, we reviewed the schemes and methodologies using attribute based encryption in cloud computing and a novel proposed Attribute based Encryption system for executing data sharing and retrieval simultaneously in cloud. Further work includes the evaluation of our proposal for effective data sharing and retrieval in public cloud. We've also discussed the concept of our proposed cloud analyst, which allows us to analyze the efficiency of cryptographic algorithms on cloud so that the users can upload data, which is then encrypted and stored on a cloud server, and then decrypted and viewed by the user flow works effectively and much efficiently. Because the data is secure, users can share it with others without worry of data theft or criminal attacks. The cryptographic algorithms vary in terms of parameters which includes encipherment and decipherment time, memory, throughput and CPU utilization. This research analyzes the want to improve a combine encipherment algorithm that mixes various encipherment algorithms on the basis of all appropriate factors that are used to increase the overall safety and security of encipherment methods. The AES algorithm is the fastest and most recent of all the algorithms that we have researched. It is one of the safest algorithms available. AES gives you the option of using a 128-bit, 192-bit, or 256-bit key, making it much more secure than DES's 56-bit key.

## References

1. Shraddha Soni, Himani Agrawal , Dr. (Mrs.) Monisha Sharma, “Analysis and Comparison between AES and DES Cryptographic Algorithm”, International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, Issue 6, December 2012

2. Shashi Mehrotra Seth, Rajan Mishra, “Comparative Analysis Of Encryption Algorithms For Data Communication”, IJCST Vol. 2, Issue 2, June 2011

3. Aman Kumar, Dr. Sudesh Jakhar, Mr. Sunil Makkar, “Comparative Analysis between DES and RSA Algorithm’s”, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 7, July 2012 ISSN: 2277 128X

4. Gurpreet Singh, Supriya, “A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security”, International Journal of Computer Applications (0975–8887) Volume 67–No.19, April 2013

5. Dr. Prerna Mahajan & Abhishek Sachdeva, “A Study of Encryption Algorithms AES, DES and RSA for Security”,Global Journal of Computer Science and Technology Network, Web & Security Volume 13 Issue 15 Version 1.0 Year 2013

6. Liam Morris, “Analysis of Partially and Fully Homomorphic Encryption”,ochester Institute of Technology, Rochester, New York

7. Iram Ahmad, Archana Khandekar, “Homomorphic Encryption Method Applied to Cloud Computing”, International Journal of Information & Computation Technology. ISSN 0974-2239 Volume 4, Number 15 (2014), pp. 1519-1530

8. https://docs.python.org/3/

## Appendix

### Code for AES Algorithm:

### Encryption :

```cpp
/* encrypt.cpp
*      Performs encryption using AES 128-bit
*      @author Cecelia Wisniewska
 */

#include <iostream>
#include <cstring>
#include <fstream>
#include <sstream>
#include "structures.h"
 using namespace std;

/* Serves as the initial round during encryption
*      AddRoundKey is simply an XOR of a 128-bit block with the 128-bit key.
 */ void AddRoundKey(unsigned char * state, unsigned char * roundKey) {     for (int
i = 0; i < 16; i++) {          state[i] ^= roundKey[i];
    }
}

/* Perform substitution to each of the 16 bytes
*      Uses S-box as lookup table
 */
void SubBytes(unsigned char * state) {     for (int i = 0; i < 16; i++) {
state[i] = s[state[i]];
    }
}

// Shift left, adds diffusion void ShiftRows(unsigned char * state) {     unsigned
char tmp[16];

    /* Column 1 */     tmp[0] = state[0];     tmp[1] = state[5];     tmp[2] =
state[10];     tmp[3] = state[15];

    /* Column 2 */
```

```
                                                  tmp[4] = state[4];
```

```
tmp[5] = state[9];     tmp[6] = state[14];     tmp[7] = state[3];
```

```
tmp[5] = state[9];     tmp[6] = state[14];     tmp[7] = state[3];
```

```
    /* Column 3 */     tmp[8] = state[8];     tmp[9] = state[13];     tmp[10] =
state[2];     tmp[11] = state[7];

    /* Column 4 */     tmp[12] = state[12];     tmp[13] = state[1];     tmp[14] =
state[6];     tmp[15] = state[11];
     for (int i = 0; i < 16; i++) {         state[i] = tmp[i];
    }
}

 /* MixColumns uses mul2, mul3 look-up tables
  * Source of diffusion
  */ void MixColumns(unsigned char * state) {     unsigned char tmp[16];
     tmp[0] = (unsigned char) mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3];
tmp[1] = (unsigned char) state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3];
tmp[2] = (unsigned char) state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]];
tmp[3] = (unsigned char) mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]];


      tmp[4] = (unsigned char)mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7];
    tmp[5] = (unsigned char)state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7];
    tmp[6] = (unsigned char)state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]];
    tmp[7] = (unsigned char)mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]];
     tmp[8] = (unsigned char)mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11];
tmp[9] = (unsigned char)state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11];
tmp[10] = (unsigned char)state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]];
tmp[11] = (unsigned char)mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]];
     tmp[12] = (unsigned char)mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15];
    tmp[13] = (unsigned char)state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^
state[15];     tmp[14] = (unsigned char)state[12] ^ state[13] ^ mul2[state[14]] ^
```

```
mul3[state[15]]; tmp[15] = (unsigned char)mul3[state[12]] ^ state[13] ^ state[14]
^
```

```
mul2[state[15]];
     for (int i = 0; i < 16; i++) {         state[i] = tmp[i];
    }
}

/* Each round operates on 128 bits at a time
 * The number of rounds is defined in AESEncrypt()
 */
void Round(unsigned char * state, unsigned char * key) {
    SubBytes(state);
    ShiftRows(state);
    MixColumns(state);
    AddRoundKey(state, key);
}

 // Same as Round() except it doesn't mix columns
void FinalRound(unsigned char * state, unsigned char * key) {
    SubBytes(state);
    ShiftRows(state);
    AddRoundKey(state, key);
}

/* The AES encryption function
 * Organizes the confusion and diffusion steps into one function
 */ void AESEncrypt(unsigned char * message, unsigned char * expandedKey, unsigned char
* encryptedMessage) {      unsigned char state[16]; // Stores the first 16 bytes of
original message

    for (int i = 0; i < 16; i++) {         state[i] = message[i];
    }
    int numberOfRounds = 9;

    AddRoundKey(state, expandedKey); // Initial round
     for (int i = 0; i < numberOfRounds; i++) {
        Round(state, expandedKey + (16 * (i+1)));
    }
    FinalRound(state, expandedKey + 160);

    // Copy encrypted state to buffer     for (int i = 0; i < 16; i++) {
        encryptedMessage[i] = state[i];
}
```

```
}  int main() {
     cout << "=============================" << endl;     cout << " 128-bit AES
Encryption Tool   " << endl;     cout << "=============================" << endl;
     char message[1024];
     cout << "Enter the message to encrypt: ";     cin.getline(message,
sizeof(message));     cout << message << endl;

    // Pad message to 16 bytes     int originalLen = strlen((const char *)message);
     int paddedMessageLen = originalLen;
     if ((paddedMessageLen % 16) != 0) {         paddedMessageLen = (paddedMessageLen
/ 16 + 1) * 16;     }     unsigned char * paddedMessage = new unsigned
char[paddedMessageLen];     for (int i = 0; i < paddedMessageLen; i++) {         if
(i >= originalLen) {             paddedMessage[i] = 0;
        }        else {             paddedMessage[i] = message[i];
        }
    }
    unsigned char * encryptedMessage = new unsigned char[paddedMessageLen];

    string str;     ifstream infile;     infile.open("keyfile", ios::in |
ios::binary);

    if (infile.is_open())
    {         getline(infile, str); // The first line of file should be the key
infile.close();
    }
    else cout << "Unable to open file";
    istringstream hex_chars_stream(str);
```

```
unsigned char key[16];
```

```
    int i = 0;      unsigned int c;      while (hex_chars_stream >> hex >> c)
    {          key[i] = c;          i++;
    }      unsigned char expandedKey[176];
    KeyExpansion(key, expandedKey);

    for (int i = 0; i < paddedMessageLen; i += 16) {
        AESEncrypt(paddedMessage+i, expandedKey, encryptedMessage+i);
    }      cout << "Encrypted message in hex:" << endl;      for (int i = 0; i <
paddedMessageLen; i++) {          cout << hex << (int) encryptedMessage[i];
cout << " ";
    }      cout << endl;

    // Write the encrypted string out to file "message.aes"      ofstream outfile;
outfile.open("message.aes", ios::out | ios::binary);      if (outfile.is_open())
    {          outfile << encryptedMessage;          outfile.close();
        cout << "Wrote encrypted message to file message.aes" << endl;      }
else cout << "Unable to open file";

    // Free memory      delete[] paddedMessage;      delete[] encryptedMessage;
    return 0; }
```

**Decryption:**

```
/* decrypt.cpp
*     Performs decryption using AES 128-bit
*     @author Cecelia Wisniewska
 */
#include <iostream>
#include <cstring>
#include <fstream>
#include <sstream>
#include "structures.h"
 using namespace std;

/* Used in Round() and serves as the final round during decryption
*     SubRoundKey is simply an XOR of a 128-bit block with the 128-bit key.
*     So basically does the same as AddRoundKey in the encryption
 */ void SubRoundKey(unsigned char * state, unsigned char * roundKey) {     for
(int i = 0; i < 16; i++) {         state[i] ^= roundKey[i];
    }
}

/* InverseMixColumns uses mul9, mul11, mul13, mul14 look-up tables
*     Unmixes the columns by reversing the effect of MixColumns in encryption
 */ void InverseMixColumns(unsigned char * state) {     unsigned char tmp[16];
     tmp[0] = (unsigned char)mul14[state[0]] ^ mul11[state[1]] ^ mul13[state[2]] ^
mul9[state[3]];       tmp[1]  = (unsigned  char)mul9[state[0]]  ^  mul14[state[1]]  ^
mul11[state[2]] ^ mul13[state[3]];
    tmp[2]  =  (unsigned  char)mul13[state[0]]  ^  mul9[state[1]]  ^  mul14[state[2]]  ^
mul11[state[3]];
    tmp[3]  =  (unsigned  char)mul11[state[0]]  ^  mul13[state[1]]  ^  mul9[state[2]]  ^
mul14[state[3]];
     tmp[4] = (unsigned char)mul14[state[4]] ^ mul11[state[5]] ^ mul13[state[6]] ^
mul9[state[7]];       tmp[5]  =  (unsigned  char)mul9[state[4]]  ^  mul14[state[5]]  ^
mul11[state[6]] ^ mul13[state[7]];      tmp[6]  =  (unsigned  char)mul13[state[4]]  ^
mul9[state[5]] ^ mul14[state[6]] ^ mul11[state[7]];
    tmp[7]  =  (unsigned  char)mul11[state[4]]  ^  mul13[state[5]]  ^  mul9[state[6]]  ^
mul14[state[7]];
```

```
    tmp[8] = (unsigned char)mul14[state[8]] ^ mul11[state[9]] ^ mul13[state[10]] ^
mul9[state[11]];        tmp[9]  =  (unsigned  char)mul9[state[8]]  ^  mul14[state[9]]  ^
mul11[state[10]] ^ mul13[state[11]];      tmp[10]  = (unsigned  char)mul13[state[8]] ^
mul9[state[9]]  ^  mul14[state[10]]  ^  mul11[state[11]];        tmp[11]  =  (unsigned
char)mul11[state[8]] ^ mul13[state[9]] ^ mul9[state[10]] ^ mul14[state[11]];
     tmp[12] = (unsigned char)mul14[state[12]] ^ mul11[state[13]] ^ mul13[state[14]] ^
mul9[state[15]];       tmp[13]  =  (unsigned  char)mul9[state[12]]  ^  mul14[state[13]]  ^
mul11[state[14]] ^ mul13[state[15]];      tmp[14]  = (unsigned  char)mul13[state[12]] ^
mul9[state[13]]  ^  mul14[state[14]]  ^  mul11[state[15]];        tmp[15]  =  (unsigned
char)mul11[state[12]] ^ mul13[state[13]] ^ mul9[state[14]] ^ mul14[state[15]];
     for (int i = 0; i < 16; i++) {         state[i] = tmp[i];
    }
}

// Shifts rows right (rather than left) for decryption void ShiftRows(unsigned char *
state) {     unsigned char tmp[16];

    /* Column 1 */     tmp[0] = state[0];     tmp[1] = state[13];     tmp[2] =
state[10];     tmp[3] = state[7];

    /* Column 2 */     tmp[4] = state[4];     tmp[5] = state[1];     tmp[6] =
state[14];     tmp[7] = state[11];

    /* Column 3 */     tmp[8] = state[8];
    tmp[9] = state[5];     tmp[10] = state[2];     tmp[11] = state[15];

   /* Column 4 */    tmp[12] = state[12];     tmp[13] = state[9]; tmp[14] =
state[6];
```

```
   tmp[15] = state[3];

   for (int i = 0; i < 16; i++) {        state[i] = tmp[i];
   }
}

/* Perform substitution to each of the 16 bytes
```

```
*     Uses inverse S-box as lookup table
 */ void SubBytes(unsigned char * state) {     for (int i = 0; i < 16; i++) { //
Perform substitution to each of the 16 bytes         state[i] = inv_s[state[i]];
    }
}

/* Each round operates on 128 bits at a time
*     The number of rounds is defined in AESDecrypt()
*     Not surprisingly, the steps are the encryption steps but reversed
 */
void Round(unsigned char * state, unsigned char * key) {
    SubRoundKey(state, key);
    InverseMixColumns(state);
    ShiftRows(state);
    SubBytes(state);
}

// Same as Round() but no InverseMixColumns

void InitialRound(unsigned char * state, unsigned char * key) {
    SubRoundKey(state, key);
    ShiftRows(state);
    SubBytes(state);
}

/* The AES decryption function
 * Organizes all the decryption steps into one function
 */  void AESDecrypt(unsigned char * encryptedMessage, unsigned char *
expandedKey, unsigned char * decryptedMessage)
{    unsigned char state[16]; // Stores the first 16 bytes of encrypted message
```

```
   for (int i = 0; i < 16; i++) {        state[i] = encryptedMessage[i];    }

InitialRound(state, expandedKey+160);
   int numberOfRounds = 9;

   for (int i = 8; i >= 0; i--) {
       Round(state, expandedKey + (16 * (i + 1)));
    }

    SubRoundKey(state, expandedKey); // Final round
    // Copy decrypted state to buffer     for (int i = 0; i < 16; i++) {
decryptedMessage[i] = state[i];
    }
}  int main()
{      cout << "=============================" << endl;     cout << " 128-bit AES
Decryption Tool " << endl;     cout << "=============================" << endl;

    // Read in the message from message.aes     string msgstr;     ifstream infile;
infile.open("message.aes", ios::in | ios::binary);
     if (infile.is_open())
    {         getline(infile, msgstr); // The first line of file is the message
cout << "Read in encrypted message from message.aes" << endl;         infile.close();
    }
    else cout << "Unable to open file";

    char * msg = new char[msgstr.size()+1];
     strcpy(msg, msgstr.c_str());

    int n = strlen((const char*)msg);
    unsigned char * encryptedMessage = new unsigned char[n];
```

```
   for (int i = 0; i < n; i++) {
       encryptedMessage[i] = (unsigned char)msg[i];    }

   // Free memory delete[] msg;
   // Read in the key     string keystr;     ifstream keyfile;
keyfile.open("keyfile", ios::in | ios::binary);
     if (keyfile.is_open())     {
        getline(keyfile, keystr);
        // The first line of file should be the key          cout << "Read in the 128-
bit key from keyfile" << endl;         keyfile.close();
    }      else cout << "Unable to open file";
     istringstream hex_chars_stream(keystr);     unsigned char key[16];     int i =
0;     unsigned int c;     while (hex_chars_stream >> hex >> c)
    {         key[i] = c;         i++;
    }      unsigned char expandedKey[176];
    KeyExpansion(key, expandedKey);
         int messageLen = strlen((const char *)encryptedMessage);

    unsigned char * decryptedMessage = new unsigned char[messageLen];

    for (int i = 0; i < messageLen; i += 16) {
        AESDecrypt(encryptedMessage + i, expandedKey, decryptedMessage + i);
    }      cout << "Decrypted message in hex:" << endl;     for (int i = 0; i <
messageLen; i++) {         cout << hex << (int)decryptedMessage[i];         cout << "
";
   }
   cout << endl;    cout << "Decrypted message: ";    for (int i = 0; i < 
messageLen; i++) {        cout << decryptedMessage[i];
   } cout << endl;
```

```
    return 0;
}
```