ATTRIBUTE-BASED PRIVACY-PRESERVING DATA SHARING FOR DYNAMIC GROUPS IN CLOUD COMPUTING

Main Project Report

Submitted for the Partial Fulfilment of the Requirements

for the Award of the Degree of

BSc. CYBER FORENSICS

By

Student Name: SUJITH SAJEEV

Reg no: 200021045229

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF TECHNOLOGY & APPLIED SCIENCES

CENTRE FOR PROFESSIONAL AND ADVANCED STUDIES

KOTTAYAM, KERALA

2020-2023

SCHOOL OF TECHNOLOGY & APPLIED SCIENCES

CENTRE FOR PROFESSIONAL AND ADVANCED STUDIES

KOTTAYAM, KERALA

CERTIFICATE

Certified that this is a Bonafide Main Project report by SUJITH SAJEEV reg no. 200021045229 for the partial fulfillment of the requirement for the award of B.Sc. Cyber Forensic at School of Technology and Applied Sciences, Centre for Professional and Advanced Studies, Pullarikunnu, Mahatma Gandhi University Kottayam during 2020- 2023.

Mrs. Jisha Mary George
(Head of the Department)

Mrs. Aryadevi K
(Internal Guide)

Internal Examiner

External Examiner

Place: Kottayam

Date:

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would not be complete without mentioning the people who have helped us to make it possible. First and foremost, we thank God Almighty for establishing us to complete this project.

We express our gratitude to *Dr. Bindu M.S*, the Principal of School of Technology and Applied Sciences, Pullarikkunnu, Kottayam and *Mrs. Jisha Mary George*, the Head of Department of Computer Science, for providing with the best facilities and atmosphere for the completion of this project.

We are grateful to *Mrs.Aryadevi k,* Lecturer and Project Guide, for her supervision. We also take this opportunity to thank all other staff members of Computer Science Department for their help and support.

I'm especially grateful to my classmates for their assistance, criticisms, and useful insights. Finally, this project would have not been possible without the confidence, endurance, and support of my family. My family have always been a source of inspiration and acknowledgement.

DECLARATION

We, hereby declare that the main project report "Attribute-Based Privacy-Preserving Data Sharing For Dynamic Groups In Cloud Computing" is submitted in partial fulfillment of the requirements for the sixth semester of Bachelor of Computer Science and it is a report of the original work done by myself in the department of Cyber Forensics of Mg University, School of Technology and Applied Sciences, Kottayam.

Place: Kottayam

Date:

SUJITH SAJEEV

ABSTRACT

The sharing of personal data with multiple users from different domains has benefited considerably from the rapid advances of cloud computing, and it is highly desirable to ensure the sharing file is not exposed to unauthorised users or cloud providers. Unfortunately, issues such as achieving the flexible access control of the sharing file, preserving the privacy of the receivers, forming the receiver groups dynamically, and achieving high efficiency in encryption and decryption still remain challenging. To deal with these challenges, we provide a novel anonymous attribute-based broadcast encryption (A2B2E) that features the property of hidden access policy and enables the data owner to share his or her data with multiple participants who are inside a predefined receiver set and fulfil the access policy. We first suggest a concrete A^2B^2E scheme together with the rigorous and formal security proof without the support of the random oracle model. Then, we design an efficient and secure data sharing system by incorporating the A^2B^2E scheme, a verifiable outsourcing decryption technique for attribute-based encryption, and the idea of online and offline attribute-based encryption. Extensive security analysis and performance evaluation demonstrate that our data sharing system is secure and practical. In this way, only the authorised user can read the encrypted data stored on the cloud server.

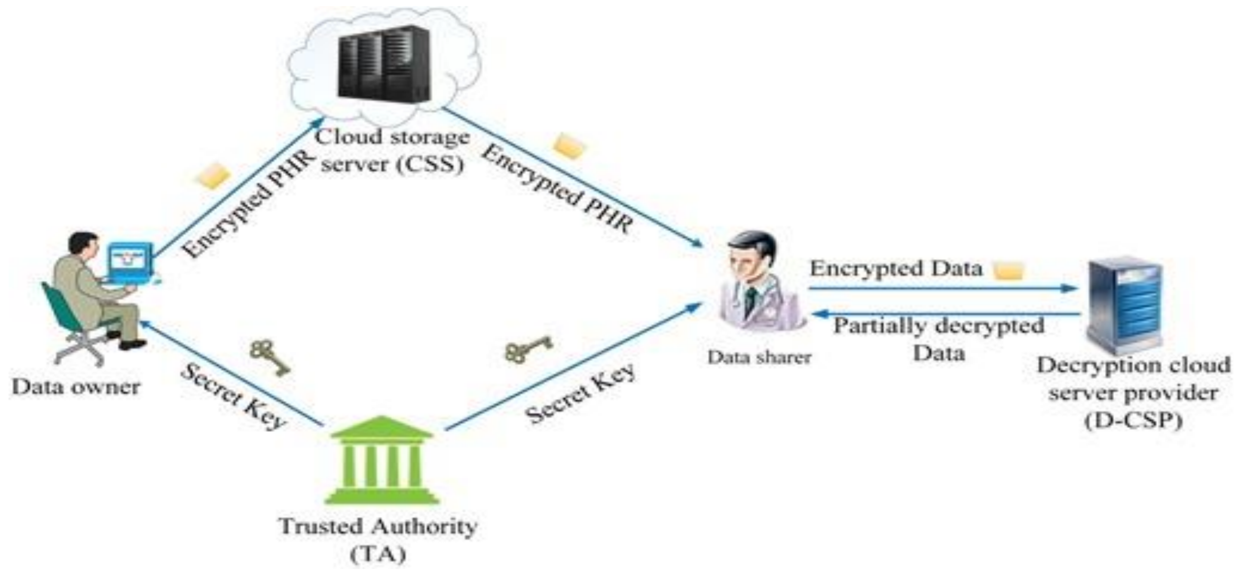
CONTENTS

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

Cloud computing is recognized as an alternative to traditional information technology due to its intrinsic resource sharing and low maintenance characteristics. Cloud computing provides an economical and efficient solution for sharing group resources among cloud users. The widespread popularity of cloud computing has enabled users to conveniently share their own data with a wide range of other users from different domains. Not only can an individual user obtain more interesting and useful data, but sharing that data with others can benefit society's insights as well. In spite of the potential profits of cloud computing, users' concern about the privacy of their sensitive data has become the main barrier preventing cloud computing from wide deployment and adoption. One of the essential security requirements of data sharing in the cloud is to ensure that the users can fully control the access to their personal data since the unauthorised disclosure of these data to unscrupulous organisations or individuals could damage the interests of the users. Moreover, different users may be granted diverse access privileges according to real demand. Therefore, it is strongly desirable to design secure data sharing mechanisms to realise flexible data access control mechanisms such that the shared data could only be read by authorised entities. Unfortunately, when sharing the data in a group while preserving the data, identity privacy is still a challenging issue due to frequent changes in membership. To overcome this problem, a secure data sharing scheme for dynamic groups is proposed so that any user within a group can share the data in a secure manner by leveraging both the group signature and dynamic broadcast encryption techniques. It should enable any cloud user to anonymously share data with others within the group and support efficient member revocation. The storage overhead and encryption computation cost are dependent on the number of revoked users.

Attribute-based encryption (ABE) is a talented cryptographic technique that achieves fine-grained data access control. It provides a way to set major access policies based on different attributes of the requester, environment, or data object. Attribute-based encryption (ABE) seems to be a promising solution to provide fine-grained and versatile access control over the encrypted sharing data due to its expressiveness. Concretely, the ABE scheme offers one- many encryption by utilising attributes and access policies embedded into the private key and ciphertext, respectively. A particular ciphertext can only be decrypted by a specified private key if the attributes involved match the policy. Thus, each user with a different set of attributes is allowed to decrypt different pieces of data per the security policy. This successfully eliminates the need to rely on the data storage server to prevent unauthorised data access, which is the traditional access control approach of such a reference monitor. To achieve secure cloud-assisted data sharing, the data owner can define the access policy, deciding which set of users to grant access to his or her data, and encrypt these data using the ABE mechanism according to the corresponding access policy before uploading them to the cloud server. In this way, only the authorised user can read the encrypted data stored on the cloud server.



Cloud computing is an economical and efficient solution for sharing group resources among cloud users, but there are security concerns. To address this, a secure data sharing scheme for dynamic groups is proposed using attribute-based encryption (ABE). ABE provides one-to-many encryption by utilising attributes and access policies embedded into the private key and ciphertext, respectively. To achieve secure cloud-assisted data sharing, the data owner can define the access policy, decide which set of users to grant access to his or her data, and encrypt these data using the ABE mechanism according to the corresponding access policy before uploading them to the cloud server.

CHAPTER 2: PROJECT OVERVIEW

2.1 AIM

The aim of attribute-based privacy-preserving group data sharing for dynamic groups in cloud computing is to enable users to share data with a selected group of individuals or organisations while preserving the privacy of the data. This is achieved by using the attributes or characteristics of the data and the users to determine who has access to the data.

The goal is to provide a mechanism for data sharing that is both secure and efficient, allowing users to share data without compromising the confidentiality, integrity, or availability of the data. By using attribute-based access control, users can define policies that specify which attributes are required to access certain data, and only users who have the necessary attributes will be able to access the data.

Privacy is preserved by encrypting the data and ensuring that only authorized users with the correct attributes can decrypt the data. This approach allows users to share data with a group of individuals or organizations while minimizing the risk of unauthorized access and data breaches.

2.2 OBJECTIVES

The objectives of attribute-based privacy-preserving group data sharing in cloud computing are to provide a secure, efficient, and flexible mechanism for sharing data while preserving the privacy and confidentiality of the data. This approach is becoming increasingly important in today's data-driven world, where data breaches and unauthorised access to sensitive information are significant concerns for individuals, organisations, and governments. The main objectives of attribute-based privacy-preserving group data sharing for dynamic groups in cloud computing are:

- Privacy Preservation: To provide a mechanism for sharing data securely with a selected group of individuals or organizations while preserving the privacy and confidentiality of the data. This is achieved by using the attributes or characteristics of the data and the users to determine who has access to the data.

- Access Control: To enable users to control who has access to their data based on specific attributes or characteristics, such as user roles, location, or organization affiliation. This ensures that only authorised users with the required attributes can access the data.

- Scalability: To leverage the scalability and cost-effectiveness of cloud computing services for data storage, processing, and sharing. This allows users to store and share large amounts of data without the need for expensive on-premise infrastructure.

- Efficiency: To provide an efficient mechanism for sharing data that minimizes the risk of unauthorized access and data breaches. This is achieved by using attribute-based access control, which enables users to define policies that specify which attributes are required to access certain data.

- Flexibility: To provide a flexible mechanism for sharing data that can be customized to meet the needs of different users and organizations. This is achieved by allowing users to define their own attributes and policies for data access.

2.3 SCOPE

The scope of attribute-based, privacy-preserving group data sharing in cloud computing is quite broad and can be applied in various contexts. The scope of attribute-based privacy-preserving group data sharing for dynamic groups in cloud computing is to provide a secure and efficient mechanism for sharing data while preserving the privacy and confidentiality of the data. This approach is becoming increasingly important in today's data-driven world, where data breaches and unauthorised access to sensitive information are significant concerns for individuals, organisations, and governments. Some of the key aspects that fall within the scope of this approach include:

- Access Control: The ability to control who has access to data based on specific attributes or characteristics, such as user roles, location, or organization affiliation.
- Data privacy is the ability to protect sensitive or confidential data from unauthorised access or disclosure while still allowing authorised users to access the data.
- Cloud Computing: The ability to leverage the scalability, flexibility, and cost-effectiveness of cloud computing services for data storage, processing, and sharing
- Group Data Sharing: The ability to share data securely with a selected group of individuals or organisations while preserving the privacy and confidentiality of the data
- Attribute-Based Encryption: The ability to encrypt data using attributes or characteristics of the data and the users to determine who has access to the data.
- Data Aggregation: The ability to aggregate data from multiple sources while preserving the privacy and confidentiality of the data

2.4 Motivation

Unfortunately, several challenges should be handled before applying the ABE mechanism to secure data sharing in dynamic groups. Let us take the example of an IT organisation where the staff is assigned to various groups and they intend to share certain files and data with other members of the group. Also, one staff member will be in multiple groups, and the CEO is a member of every group. Suppose a group member, Bob, needs to share a file regarding the upcoming project with Group 1, where he is a member of both Group 1 and Group 2. Here, Bob needs to encrypt the files separately such that the files can only be accessed by members of GROUP 1. While another member who is also a member of both Group 1 and Group 2 needs to send a file to Group 2, she needs to encrypt the file so that only Group 2 members can access it. Since, both Alice and Bob encrypt the file using their private keys, and thus it can be decrypted using their public keys. In this scheme, any staff member that knows the public key of Alice and Bob can access that file easily, and it does not provide any security. To overcome this challenge, we can use various attributes of the group and the group members to encrypt the data or the file being shared. This solution seems inefficient since the ciphertext size and the computational cost of encryption grow with the number of attributes associated with the identities of the involved groups in a linear way. Thus, it is desirable that the group identities involved in the encryption can be determined dynamically by the data owner according to his or her practical needs. Second, in most existing ABE schemes, the personal information of the ciphertext receiver, which is defined by the attribute set, can be trivially obtained from the ciphertext since the access policy is usually stored in plaintext form. Obviously, it is a serious violation of the PHR data sharer's privacy, which also severely disgraces the privacy of the data owner. Ideally, an ABE scheme should preserve the receiver's privacy by ensuring that the ciphertext does not disclose any information about the access policy. Last but not least, a major efficiency drawback of ABE mechanisms is that the computational cost during encryption and decryption is usually considered to be costly and may grow linearly with the number of attributes involved in the access policy. This drawback may essentially limit the application of ABE on a mobile device with constrained resources.

2.5 OUR CONTRIBUTION

In this paper, the group-oriented ABE scheme with privacy preservation is introduced to simplify the access policy by containing two kinds of "identities": one user's group identity and a set of identities that represent the attributes the user owns. By integrating the idea of broadcast encryption (BE), our group-oriented ABE scheme enforces flexible access control for users in multiple groups by efficiently using the user's group identity. If our group-oriented ABE scheme is adopted in the above scenario, the access policy can be simplified such that the granted groups can be designated by the data owner in an ad hoc manner. Moreover, our ABE scheme can preserve the privacy of the ciphertext receiver by providing the property of hidden access. The contributions of our scheme are threefold.

1) Anonymous attribute-based broadcast encryption (A^2B^2E) is proposed to simultaneously feature the property of hidden access policy and enable a sender to share ciphertexts of data with users who are inside a predefined receiver group and fulfil the access policy. Besides the concrete construction, we formalise the security model for A^2B^2E and present rigorous security analysis in the standard model. This is the first effort, as far as we know, considering A2B2E, and we believe the A^2B^2E scheme is of independent interest.

2) The proposed data sharing protocol is constructed based on the A^2B^2E scheme and is especially suitable for large-scale users, such that the participants in the system are partitioned into distinct groups and each user belongs to a specific group. By exploiting our proposed A^2B^2E scheme, the data owner can ensure that only the users whose group identities are inside a dynamically specified group set and attributes fulfil the predefined access policy can simultaneously access the shared data. Meanwhile, the proposed data sharing protocol is also appropriate for lightweight devices because most of the computation costs on both the data owner and data sharer sides are carried out in the offline phase or outsourced to the cloud server in a verifiable manner.

3) We have performed extensive simulations to evaluate the performance of our data sharing protocol and related work. The experimental result demonstrates that our data sharing protocol is highly practical.

2.6 RELATED WORK

To design flexible data sharing with the support of the ABE technique, Li et al. put forward a versatile and patient-centric access control mechanism for data sharing with multiple data owners by leveraging multiauthority ABE (MA-ABE). Moreover, the user or attribute revocation function has also been considered to revoke access privileges in cases of dispute. By considering the issues of preserving the data owner's identity privacy and tracing the malicious data sharer, Zhou et al. proposed a traceable and revocable MA-ABE scheme by incorporating the idea of attribute revocation and revocable storage and constructed an access control mechanism with multilevel privacy preserving for data storing in the cloud. To enable legal data sharers to access data anonymously and guarantee the authenticity of the shared data, Liu et al. presented an attribute-based signcryption to offer signcryption and access control simultaneously. Without giving the concrete construction, Narayan et al. suggested a multifunctional security framework to feature fine-grained access control, keyword search, and direct revocation based on the ABBE. However, these schemes only concentrated on the data's confidentiality or the data owner's identity privacy, leaving the data sharers' privacy issues unsolved. What makes matters worse is that the existing schemes are not suitable for the large-scale e-users who are within different collaborative organisations.

To protect the privacy of the recipient, Nishide et al. put forward an ABE scheme equipped with the property of partially hidden access structures. In such a scheme, only partial attribute values are hidden, whereas the attribute names can be disclosed to anyone. To conceal the access policy entirely, Zhou et al. suggested a novel approach to hiding the access policy while preserving privacy efficiently by employing the conjunctive access policy technique. Despite Zhou et al.'s claim that the privacy of access policies was achieved in their scheme, wildcard attributes in access structures are not hidden in their scheme, which undoubtedly leaks the privacy of the receivers. Recently, Phuong et al. proposed a hidden policy CP-ABE scheme by utilising two different vectors to represent the access policy and user attributes, respectively, and then applying the inner product encryption technique to hide the access policy.

However, both the ciphertext size and computational overhead increase linearly with the number of attributes, which makes it unaffordable for lightweight devices. In addition, the construction of the PHR data sharing system is in conflict with its scalability for dynamic groups.

Geographically scattered receivers were specified among a set of legitimate users through an insecure channel. Similar to ABE, BE can also provide one-to-many encryption. In BE, a group of receivers is explicitly specified by the sender during encryption, and only receivers in this group can perform the decryption. Obviously, receivers in the BE need to be represented individually, and the sender has to explicitly maintain a group of potential receivers. In some scenarios with large scales and dynamic users, it is difficult, if not impossible, to obtain exact knowledge of the group of prospective receivers. ABBE, the combination of ABE and BE, has been introduced to achieve direct revocation in ABE, where the sender can exclude revoked users from the receiver group directly. Obviously, the sender in ABBE still needs to possess the enumerative list of receivers in the system to achieve direct revocation. It is fair to say that the role of ABBE is somewhat confusing. If the sender knows the exact details of each member in the system, the ABE does not make sense since the sender can achieve one-to-many encryption with BE directly. That is to say, the ABBE has been utilised in the wrong manner until now. A ciphertext policy, ABBE (CP-ABBE), is proposed by Lubicz et al. to allow a broadcaster or encryptor to dynamically select a subgroup of privileged users who are identified with their corresponding attributes. Attrapadung et al. suggested another approach by skillfully combining the existing BE and ABE schemes to construct ABBE in both ciphertext-policy and key-policy environments. Junod and Karlov describe a secure ABBE scheme to support complex AND, OR, and NOT gates as the access policy. However, the ciphertext size of the above schemes increases linearly with the complexity of the access formula. Zhou et al. proposed a new ABBE that achieves a constant ciphertext size and obtains a cheaper cost of storing the predefined attributes. Phuong et al. proposed two novel ABBE schemes with a constant ciphertext size, namely, the KP-ABBE and CP-ABBE schemes. So far, the privacy-preserving property has not been investigated in the ABBE paradigm, which limits its application where the privacy of receivers matters.

In, Kallahalla et al. proposed a cryptographic storage system that enables secure file sharing on untrusted servers, named Plutus. By dividing files into file groups and encrypting each file group with a unique file-block key, the data owner can share the filegroups with others by delivering the corresponding lockbox key, where the lockbox key is used to encrypt the file-block keys. However, it brings about a heavy key-sharing overhead for large-scale file sharing. Additionally, the file-block key needs to be updated and distributed again for a user revocation. Files stored on the untrusted server include two parts: file metadata and file data. The file metadata implies access control information, including a seri es of encrypted key blocks, each of which is encrypted under the public key of authorised users. Thus, the size of the file metadata is proportional to the number of authorised users. The user revocation in the scheme is an intractable issue, especially for large-scale sharing since the file metadata needs to be updated. In their addition version, the NNL construction is used for efficient key revocation. However, when a new user joins the group, the private key of each user in an NNL system needs to be recomputed, which may limit the application for dynamic groups. Another concern is that the computation overhead of encryption linearly increases with the sharing scale. Ateniese et al. leveraged proxy-based encryptions to secure distributed storage. Specifically, the data owner encrypts blocks of content with unique and symmetric content keys, which are further encrypted under a master public key. For access control, the server uses proxy cryptography to directly reencrypt the appropriate content key(s) from the master public key to a granted user's public key. Unfortunately, a collusion attack between the untrusted server and any revoked malicious user can be launched, which enables them to learn the decryption keys of all the encrypted blocks. In, Yu et al. presented a scalable and fine-grained data access control scheme in cloud computing based on the KPABE technique. The data owner uses a random key to encrypt a file, where the random key is further encrypted with a set of attributes using KPABE. Then, the group manager assigns an access structure and the corresponding secret key to authorised users, such that a user can only decrypt a ciphertext if and only if the data file attributes satisfy the access structure. To achieve user revocation, the manager delegated the tasks of data file reencryption and user secret key update to cloud servers. However, the single-owner manner may hinder the implementation of applications in the scenario where any member of a group should be allowed to store and share data fil es with others. Lu et al. proposed a secure provenance scheme, which is built upon group signatures and ciphertext-policy attribute-based encryption techniques. Particularly, the system in their scheme is set with a single attribute. Each user obtains two keys after the registration: a group signature key and an attribute key. Thus, any user is able to encrypt a data file using attribute-based encryption, and others in the group can decrypt the encrypted data using their attribute keys. Meanwhile, the user signs encrypted data with her group signature key for privacy preservation and traceability. However, user revocation is not supported in their scheme. From the above analysis, we can view that how to securely share data files in a multiple-owner manner for dynamic groups while preserving identity privacy from an untrusted cloud remains to be a challenging issue. In this paper, we propose a novel Mona protocol for secure data sharing in cloud computing. Compared with accessible works, Mona offers unique features as follows:

Any user in the group can store and share data files with others via the cloud.

The encryption complication and size of cipher texts are independent with the number of revoked users in the system.

User revocation can be achieved without updating the private keys of the remaining users.

A new user can directly decrypt the files stored in the cloud before participating.

2.7 PROJECT RISK

While attribute-based encryption for group data sharing in cloud computing can offer many benefits, there are also several risks that should be considered, including:

- Data Breaches: A data breach can occur if an attacker gains unauthorized access to the encrypted data or the encryption keys. This can result in the compromise of sensitive or confidential data.

- Misuse of Data: If the attributes used for access control are not well defined, there is a risk that authorised users may misuse the data they have access to.

- Over-Reliance on Encryption: While encryption can provide an effective mechanism for protecting data, it should not be the only security control used. Organizations need to implement additional security controls, such as access controls, monitoring, and auditing, to ensure the security of their data.

- Complexity: Attribute-based encryption can be complex to implement and manage, which can increase the risk of errors or misconfigurations.

- Key Management: The management of encryption keys can be challenging, particularly when multiple users or organizations are involved. The loss or theft of encryption keys can result in the compromise of sensitive data.

- Compliance: Compliance with regulations and standards such as GDPR, HIPAA, or PCI-DSS can be challenging when implementing attribute-based encryption for group data sharing. Organisations need to ensure that they are meeting the necessary regulatory requirements.

2.8 MANAGING PROJECT RISKS

To mitigate the risks, organizations should carefully plan and implement attribute-based encryption for group data sharing in cloud computing. They should also implement appropriate security controls and regularly review and update their security policies and procedures. Additionally, they should provide training to their employees to ensure they understand how to use the attribute-based encryption system effectively and securely. To overcome the risks involved in attribute-based encryption for group data sharing in cloud computing, organizations can take several measures, including:

- Implement Strong Access Controls: Implementing strong access controls is essential to ensuring that only authorised users with the necessary attributes can access the encrypted data. This can include implementing multi-factor authentication, role-based access control, and least privilege access.

- Use Strong Encryption: Implementing strong encryption algorithms and using long and complex encryption keys can help protect the data from attackers.

- Ensure Proper Key Management: Implementing proper key management processes can help ensure the integrity and confidentiality of the encryption keys. This can include using secure key storage and distribution mechanisms and regularly rotating encryption keys.

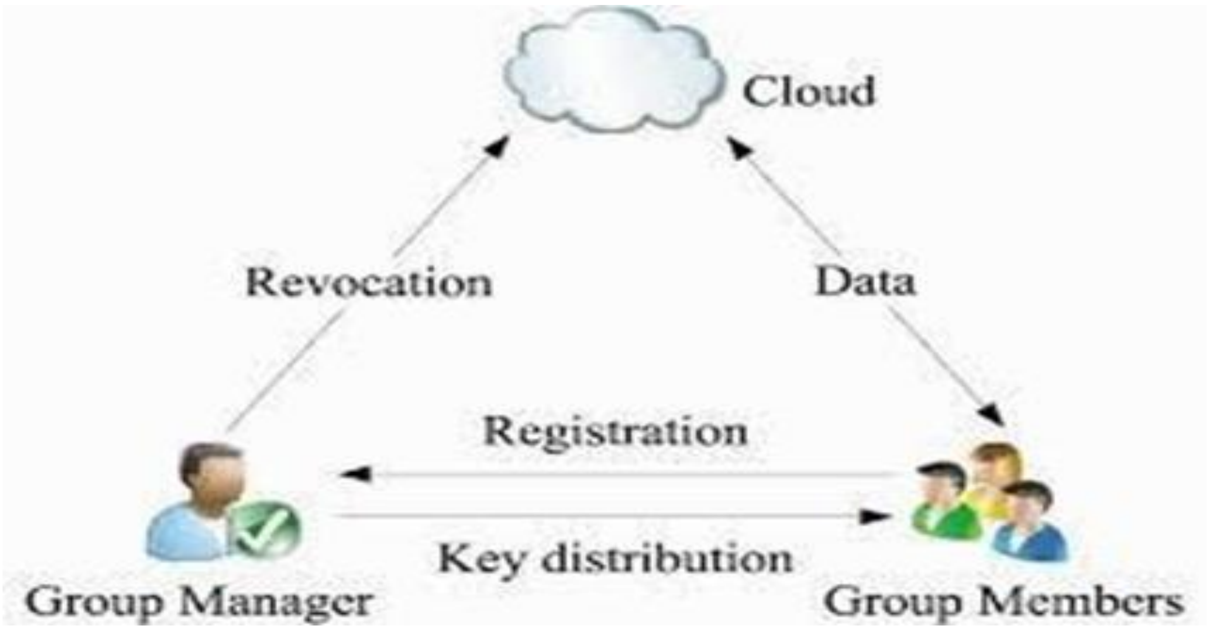
- Regularly Monitor and Audit: Regularly monitoring and auditing access to the encrypted data can help detect unauthorised access and potential data breaches. This can include implementing intrusion detection and prevention systems and reviewing audit logs regularly.

- Train Employees: Training employees on how to use the attribute-based encryption system effectively and securely can help mitigate the risk of errors or misconfigurations.

- Comply with regulations and standards: organisations should ensure that they comply with relevant regulations and standards, such as GDPR, HIPAA, or PCI-DSS. This can include implementing appropriate security controls and regularly reviewing and updating their security policies and procedures

2.9 PROJECT EXPLANATION

The project is being applied to an IT firm where the staff at various designations are assigned to one or more groups based on the requirements. The CEO (admin), who is also a staff member, is responsible for adding members to various groups; admin is a member of every group. When any of the group members needs to share any data or file with a specified group, they use attribute-based encryption to encrypt the same, and then they upload that file into the specified group. Here, various attributes such as the name of the staff, staff id, login id, and group id are used to create keys for each member of a specified group while adding that specified member to that group. These keys are stored in the database for future access and use. A member of a specified group can only download the file that was uploaded. If the user who uploaded the file accidentally uploaded the file to another group where he is a member, then that member will not be able to download that file since the attributes and hence the keys don't match. The encrypted file is also uploaded to the cloud server and the users need to retrieve the file from the cloud.



CHAPTER 3: ENCRYPTION OVERVIEW

3.1 ENCRYPTION

Encryption is a form of data security in which information is converted to ciphertext. Only authorised people who have the key can decipher the code and access the original plaintext information. In even simpler terms, encryption is a way to render data unreadable to an unauthorised party. This serves to thwart cybercriminals, who may have used quite sophisticated means to gain access to a corporate network—only to find out that the data is unreadable and therefore useless. Encryption not only ensures the confidentiality of data or messages, but it also provides authentication and integrity, proving that the underlying data or messages have not been altered in any way from their original state. Data can be encrypted "at rest," when it is stored, or "in transit," while it is being transmitted somewhere else.

In computing, unencrypted data is also known as plaintext, and encrypted data is called *ciphertext*. The formulas used to encode and decode messages are called *encryption algorithms*, or cyphers. To be effective, a cypher includes a variable as part of the algorithm. The variable, which is called a *key*, is what makes a cipher's output unique. When an encrypted message is intercepted by an unauthorised entity, the intruder has to guess which cypher the sender used to encrypt the message, as well as what keys were used as variables. The time and difficulty of guessing this information are what make encryption such a valuable security tool. Encryption has been a longstanding way for sensitive information to be protected. Historically, it was used by militaries and governments. In modern times, encryption is used to protect data stored on computers and storage devices, as well as data in transit over networks.

Encryption is commonly used to protect data in transit and at rest. Every time someone uses an ATM or buys something online with a smartphone, encryption is used to protect the information being relayed. Businesses are increasingly relying on encryption to protect applications and sensitive information from reputational damage when there is a data breach.

There are three major components to any encryption system:

- data
- encryption engine
- key management.

In laptop encryption, all three components are running or stored in the same place: on the laptop. In application architectures, however, the three components usually run or are stored in separate places to reduce the chance that compromise of any single component could result in compromise of the entire system.

3.2 WORKING OF AN ENCRYPTION

Encryption is the process of converting plain text or data into an unreadable form, called cipher text, using an algorithm or a mathematical function. This is done to protect the confidentiality and integrity of the information being transmitted or stored. Only authorized parties who possess the decryption key can decipher the message and retrieve the original content.

Encryption is commonly used in various applications, such as online transactions, email communication, data storage, and digital signatures. It is also an important tool for maintaining security and privacy in sensitive areas such as military and government communications. The main goal of encryption is to protect the confidentiality, integrity, and authenticity of the data being transmitted or stored.

Various steps involved in the working of an encryption are:

- Data Preparation: The first step is to prepare the data that needs to be encrypted. This can be any type of data, such as text, files, images, videos, etc.

- Encryption Algorithm: An encryption algorithm is used to transform the data into an unreadable format. This can be done using different techniques such as substitution, permutation, or a combination of both. The encryption algorithm takes in the data and a secret key to produce the cipher text.

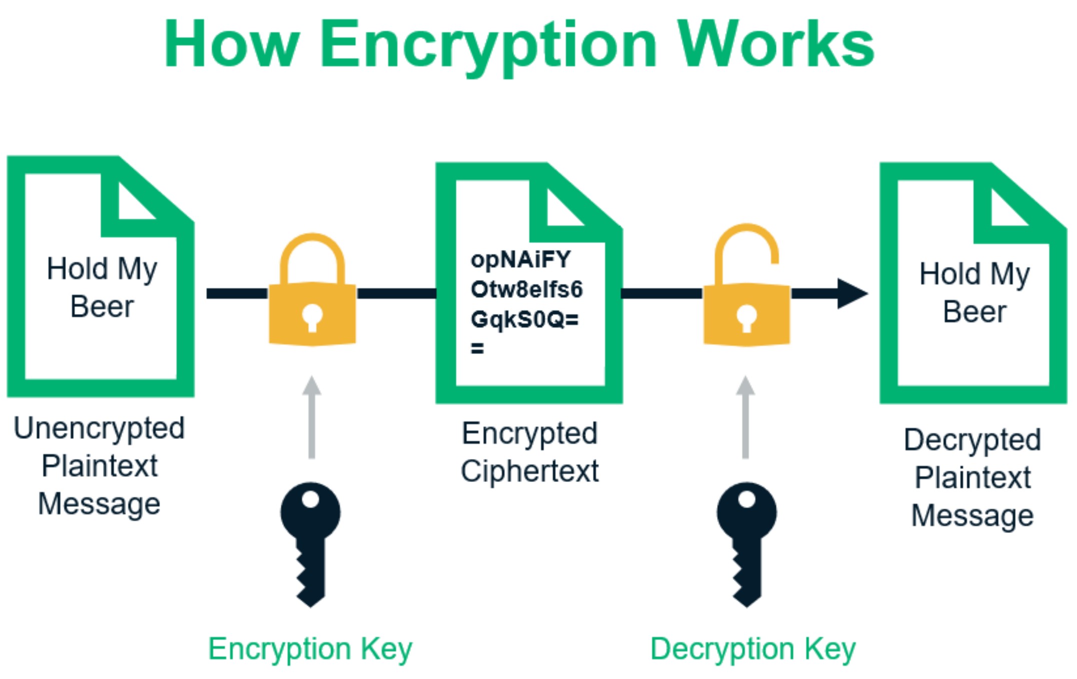
- Secret Key: A secret key is a unique value used by the encryption algorithm to transform the data into cypher text. Only authorised parties who possess the secret key can decrypt the cypher text and retrieve the original data.

- Transmission: The cipher text is transmitted to the receiver through a secure channel such as a virtual private network (VPN), secure sockets layer (SSL), or transport layer security (TLS).

- Decryption Algorithm: The receiver uses a decryption algorithm to transform the cipher text back into the original data. The decryption algorithm takes in the cipher text and the secret key to produce the original data.

- Secret Key Exchange: The secret key is exchanged between the sender and receiver through a secure key exchange protocol such as the Diffie-Hellman key exchange or the RSA key exchange.

- Data Integrity: In addition to confidentiality, encryption can also be used to ensure data integrity. This involves using a message authentication code (MAC) or a digital signature to verify the authenticity and integrity of the data.



3.3 WHY ENCRYPTION IS IMPORTANT

Encryption plays an important role in securing many different types of information technology (IT) assets. Encryption is a critical tool for protecting sensitive information and ensuring data security. Encryption is of great importance in today's digital age for the following reasons:

- Confidentiality: Encryption helps protect the confidentiality of sensitive information such as personal data, financial information, and passwords. It ensures that only authorised parties can access the information and prevents unauthorised access by hackers or cybercriminals.

- Data Integrity: Encryption helps to ensure the integrity of data by detecting any unauthorised modification, corruption, or tampering with it. It enables organisations to verify that the data has not been altered during transmission or storage.

- Compliance: Encryption is a requirement for compliance with many regulatory frameworks, such as the General Data Protection Regulation (GDPR), the Health Insurance Portability and Accountability Act (HIPAA), and the Payment Card Industry Data Security Standard (PCI-DSS).

- Business Protection: Encryption helps protect the business from data breaches and cyberattacks. It safeguards the company's reputation and prevents legal liabilities and financial losses.

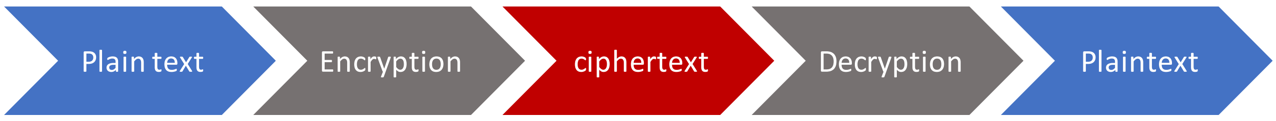
- National Security: Encryption is used by governments and military organizations to protect sensitive information and communications. It helps safeguard national security and protect against espionage and cyberattacks.

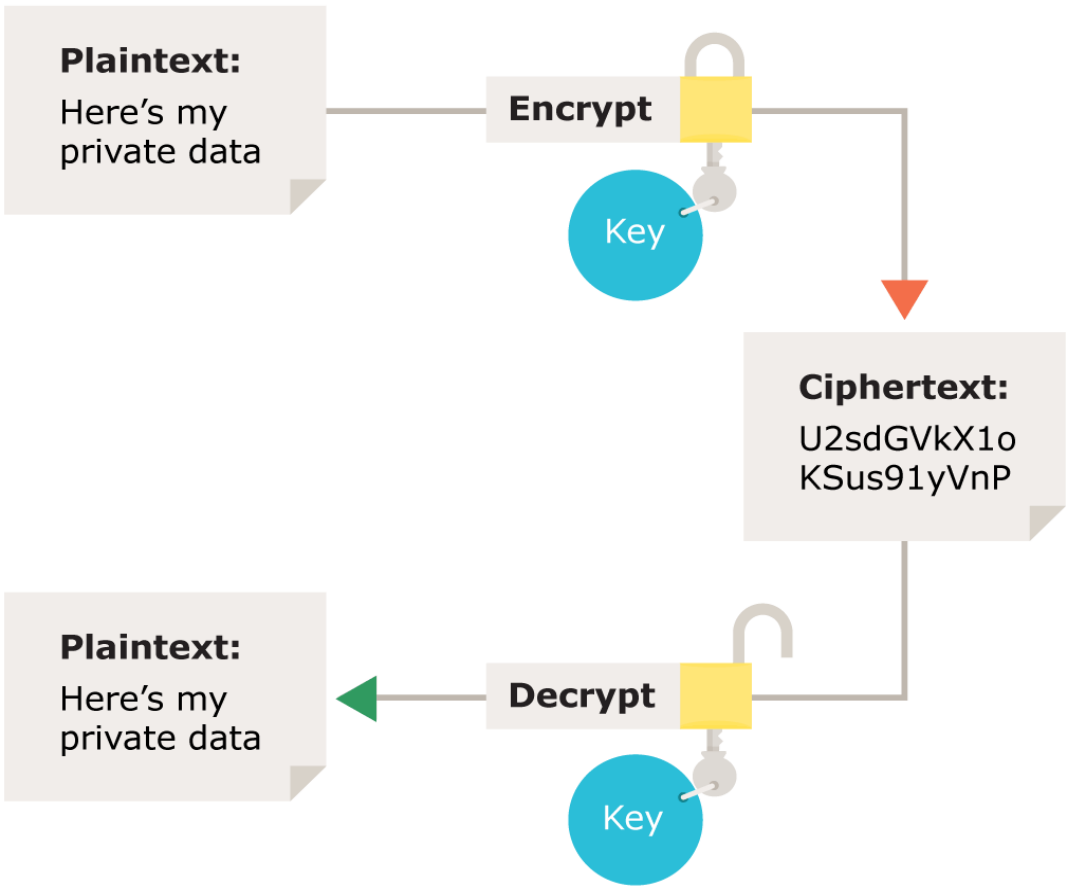
3.4 DECRYPTION

Decryption is the process of converting encrypted data or cipher text back into its original, readable form using a decryption algorithm and a secret key. The decryption algorithm is designed to reverse the encryption algorithm used to encrypt the data, and the secret key is used to unlock the encryption and retrieve the original data. Decryption is an essential process for ensuring the confidentiality of sensitive information. Without decryption, the encrypted data would be unreadable, and the information would be useless to the authorized recipient. The use of strong encryption algorithms and secure key exchange protocols can help ensure the confidentiality and integrity of the data being transmitted or stored.

The decryption process involves the following steps:

- Cipher text: The cipher text, which is the encrypted data, is received by the receiver.
- Secret Key: The receiver uses the secret key to decrypt the cipher text.
- Decryption Algorithm: The receiver uses a decryption algorithm to decrypt the cipher text. The decryption algorithm is designed to reverse the encryption algorithm used to encrypt data.
- Original Data: Once the decryption is complete, the original, readable data is obtained.



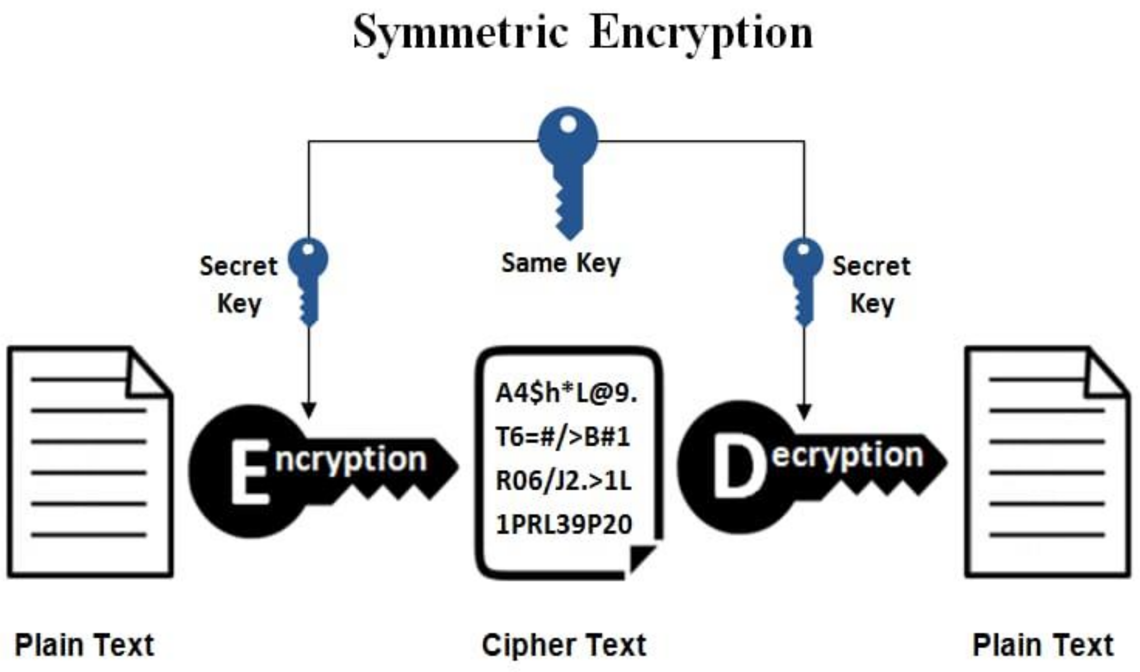

Plaintext:
Here's my private data
Encrypt
Key
Ciphertext:
U2sdGVkX1o KSus91yVnP
Plaintext:
Here's my private data
Decrypt
Key

TYPES OF ENCRYPTION TECHNIQUES

- Encryption plays a vital role in ensuring the security and confidentiality of data in various applications such as e-commerce, banking, communication, and military operations.

  There are two main types of encryption techniques: symmetric-key encryption and public-key encryption.

  1. **Symmetric-Key Encryption**: Symmetric-key encryption, also known as secret-key encryption, uses the same key to encrypt and decrypt data. The sender and receiver both possess the same secret key, and they use it to encrypt and decrypt messages. Some examples of symmetric-key encryption algorithms include AES (Advanced Encryption Standard), DES (Data Encryption Standard), and 3DES (Triple DES).



Advantages of Symmetric-Key Encryption

Symmetric encryption is a reliable and efficient technique for securing data in many applications.

Symmetric encryption has several advantages that make it a popular choice for securing data:

- Efficiency: Symmetric encryption is typically faster and more efficient than other encryption techniques, such as public-key encryption. This is because symmetric encryption algorithms require less computational overhead and can encrypt and decrypt data at a much faster rate.

- Simplicity: Symmetric encryption is a simple technique that requires only one key for encryption and decryption. This makes it easy to implement and manage, especially in situations where there are only a few users or devices involved.

- Security: When implemented correctly, symmetric encryption can provide strong security for sensitive data. The use of a secret key to encrypt and decrypt data ensures that only authorised parties can access and read the data.

- Agility: Symmetric encryption allows for key agility, which means that the keys can be changed regularly to provide additional security. This can be especially useful in situations where the key might be compromised or when a new user needs to access the encrypted data.

- Compatibility: Symmetric encryption is widely supported by many different operating systems, platforms, and devices. This makes it easy to use and integrate into existing systems and applications.

Disadvantages of Symmetric-Key Encryption

Symmetric encryption has some limitations that can affect its use in certain situations:

- Key Management: In symmetric encryption, the same key is used for both encryption and decryption. This means that the key must be shared between the sender and receiver, and the security of the encryption depends on the secrecy of the key. Key management can be challenging when dealing with a large number of users or devices, and any compromise of the key can compromise the security of the entire system.

- Lack of Forward Secrecy: In symmetric encryption, if the key is compromised, all past and future encrypted messages can be decrypted. This means that there is no forward secrecy, and a compromise of the key can lead to a complete loss of confidentiality.

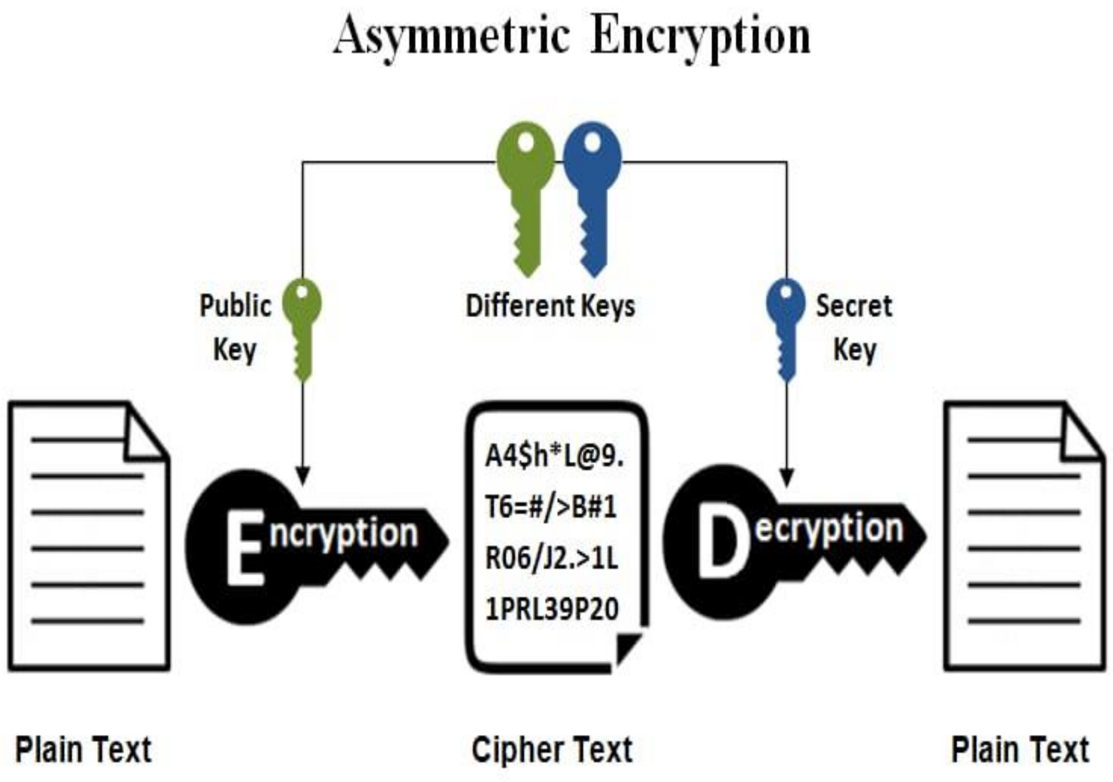
- Scalability: Symmetric encryption can be difficult to scale to large networks or distributed systems. Each pair of communicating parties must share a unique key, which can be difficult to manage when the number of users or devices increases.

- Key Distribution: In order to use symmetric encryption, the sender and receiver must have access to the same key. This can be difficult to achieve in some situations, such as when the communicating parties are not physically present in the same location.

- Lack of Non-Repudiation: Symmetric encryption does not provide non-repudiation, meaning that a sender can deny sending a message or claim that the message was tampered with after it was sent.

Despite these limitations, symmetric encryption remains a widely used and effective technique for securing data in many applications. However, in situations where the limitations of symmetric encryption are a concern, other encryption techniques, such as public-key encryption, may be more appropriate.

2. **Public-Key Encryption**: Public-key encryption, also known as asymmetric encryption, uses a pair of keys for encryption and decryption. One key is public and can be distributed to anyone, while the other key is private and must be kept secret. The sender uses the recipient's public key to encrypt the message, and the recipient uses their private key to decrypt the message. RSA and Elliptic Curve Cryptography (ECC) are some examples of public-key encryption algorithms.



Advantages of Public-Key Encryption

Public-key encryption is a powerful technique for securing data in many applications. While it may be slower and less efficient than symmetric encryption, the advantages of public-key encryption make it an important tool for protecting sensitive information in many different contexts.

Public-key encryption, also known as asymmetric encryption, has several advantages over symmetric encryption:

- Key Distribution: Public-key encryption eliminates the need for secure key distribution, which is a significant challenge in symmetric encryption. In public-key encryption, each user has a unique public and private key pair, and the public key can be freely distributed. This makes it easy to add new users to the system without having to distribute new keys.

- Non-Repudiation: Public-key encryption provides non-repudiation, meaning that the sender cannot deny sending a message or claim that the message was tampered with after it was sent. This is because the sender's private key is required to sign the message, and the signature can be verified using the sender's public key.

- Scalability: Public-key encryption is scalable to large networks or distributed systems, as each user has a unique key pair. This makes it easy to add new users or devices to the system without affecting the security of existing users.

- Forward Secrecy: Public-key encryption provides forward secrecy, meaning that if a private key is compromised, only the messages that were encrypted with that key are compromised. This is because each message is encrypted with a unique session key that is generated for that message.

- Security: Public-key encryption provides strong security for sensitive data. The use of two keys, one public and one private, ensures that only the intended recipient can decrypt the message and that the message has not been tampered with during transmission.

Disadvantages of Public-Key Encryption

Public-key encryption, like any encryption technique, has some disadvantages that can impact its use in certain situations.

- Computational Overhead: Public-key encryption algorithms are typically slower and require more computational resources than symmetric encryption algorithms. This can be a concern in situations where performance is critical, such as in high-speed data networks or real-time systems.

- Key Management: Public-key encryption requires the management of two keys for each user, which can be challenging when dealing with a large number of users or devices. Additionally, the security of the system depends on the secrecy of the private key, which must be protected and managed carefully.

- Compatibility: Public-key encryption is not as widely supported as symmetric encryption, which can make it difficult to use and integrate into existing systems and applications. Additionally, some devices or platforms may not have the processing power or memory required to handle public-key encryption.

- Key Distribution: While public keys can be freely distributed, private keys must be kept secret. This can be challenging in situations where the private key must be shared between multiple users or devices, or when a user loses their private key and needs to generate a new one.

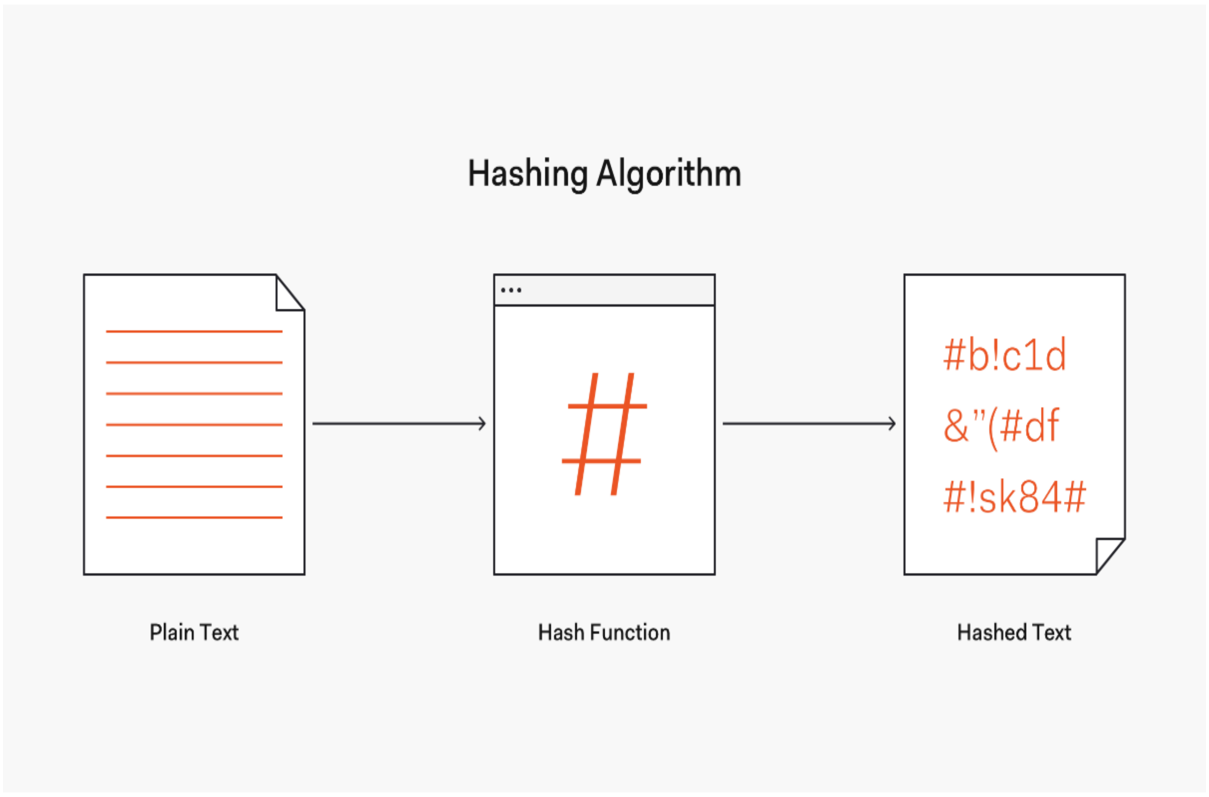
- Security: While public-key encryption is generally considered to be secure, there is always the possibility of attacks or vulnerabilities being discovered. Additionally, some attacks, such as side-channel attacks, can be more effective against public-key encryption than symmetric encryption.

Public-key encryption is a powerful technique for securing data in many applications, but it has some limitations that must be considered when designing and implementing a system. In some situations, a combination of public-key and symmetric encryption may provide the best balance of security and efficiency.

Apart from these two main types, there are also other types of encryption techniques, such as:

i. **Hashing**:

Hashing is a technique used to convert data into a fixed-size, unique digital fingerprint called a hash value. It is used to verify the integrity of data by comparing the hash value of the original data with the hash value of the received data.



Advantages of Hashing

Hashing is a useful technique for securing and processing data. It provides data integrity, data privacy, and fast processing speed, among other advantages. It is important to use a good hash function and implement hashing correctly to ensure that the data is secure. Hashing is a one-way function that takes an input (a message or some data) and produces a fixed-size output (the hash value or digest). Here are some advantages of hashing:

- Data Integrity: Hashing can be used to verify the integrity of data. By comparing the hash values of two sets of data, you can determine if they are identical or if any changes have been made to the original data.

- Password Storage: Hashing can be used to store passwords securely. Instead of storing the actual password, a hash value of the password can be stored. When a user enters their password, the hash value of the entered password can be compared with the stored hash value to verify the password.

- Data Privacy: Hashing can be used to protect sensitive data. By hashing the data before transmitting or storing it, you can ensure that the data cannot be read or accessed by unauthorised parties.

- Fast and Efficient: Hashing is a fast and efficient way of processing data. The hash function can process large amounts of data quickly and produce a fixed-size output.

- Collision Resistance: A good hash function has a low probability of producing the same hash value for different input data. This means that it is very difficult for an attacker to find two different input values that produce the same hash value.

- Data anonymization: hashing can be used to anonymize data. By hashing certain data fields in a database, you can prevent the identification of individuals while still allowing for the analysis of the data.

Disadvantages of Hashing

- One-way Function: Hashing is a one-way function, meaning that it cannot be reversed to recover the original data. This makes it difficult to retrieve the original data if it is lost or corrupted.

- Collision Attacks: While a good hash function has a low probability of producing the same hash value for different input data, it is still possible for two different input values to produce the same hash value. This is known as a collision, and collision attacks can be used by attackers to manipulate the data.

- Rainbow Table Attacks: Rainbow Table Attacks are a type of pre-computed attack that can be used to crack hashed passwords. Attackers can pre-compute a table of hash values for commonly used passwords and then compare the hash values of stolen password hashes with the pre-computed table to find the original passwords.

- Weak hash functions: If a weak hash function is used, it can be vulnerable to brute-force attacks, where an attacker tries all possible input values until the correct hash value is found.

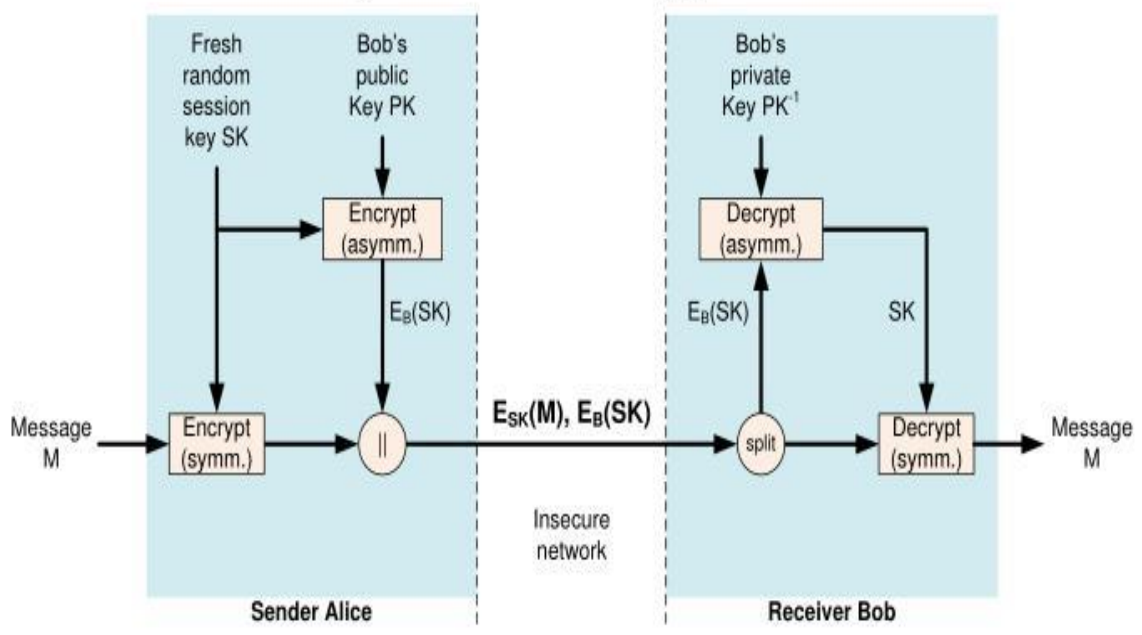
- Salt Management: When hashing passwords, it is important to use a salt value to prevent attackers from using pre-computed tables to crack the passwords. However, managing salt values can be challenging in large systems or databases.

- Hash Length: The length of the hash value can impact its security. If the hash value is too short, it can be vulnerable to collision attacks, while if the hash value is too long, it can impact system performance.

ii. **Hybrid Encryption**:

Hybrid encryption is a combination of symmetric and public-key encryption. In this technique, the message is encrypted using a symmetric-key algorithm, and then the symmetric key is encrypted using the recipient's public key. The recipient can then use their private key to decrypt the symmetric key, and then use the symmetric key to decrypt the message.

Hybrid encryption



- Symmetric encryption is fast; asymmetric is convenient
- Hybrid encryption = symmetric encryption with random session key + asymmetric encryption of the session key

Advantages of hybrid encryption

Advantages include:

- Secure Key Exchange: Hybrid encryption allows for a secure key exchange between the sender and recipient using asymmetric encryption. Asymmetric encryption is used to exchange a symmetric key that is then used for symmetric encryption, providing a secure method of key exchange.

- Efficient Encryption: Hybrid encryption is efficient because symmetric encryption is used for the bulk of the data, which is faster and more efficient than asymmetric encryption.

- Strong Security: Hybrid encryption provides strong security because it combines the advantages of both symmetric and asymmetric encryption. The symmetric key used for encryption is only known by the sender and recipient, while the public key used for key exchange is known by anyone. This makes it difficult for attackers to intercept and decrypt the data.

- Flexibility: Hybrid encryption is flexible and can be used in a variety of scenarios. For example, it can be used for secure file transfers, email encryption, or secure communication between two parties.

- Scalability: Hybrid encryption is scalable, making it suitable for large-scale deployments. It can be used in complex systems and be easily integrated with other security mechanisms.

Disadvantages of Hybrid Encryption

Disadvantages include:

- Complexity: Hybrid encryption can be more complex than using either symmetric or asymmetric encryption alone. It requires the implementation and management of both symmetric and asymmetric encryption systems.

- Key Management: Hybrid encryption requires managing both symmetric and asymmetric keys. This can be challenging in large-scale deployments or systems with many users and can lead to potential security risks if keys are not managed properly.

- Compatibility: Hybrid encryption may not be compatible with all systems or software. If the recipient does not have compatible encryption software, the encrypted data may not be accessible.

- Overhead: Hybrid encryption can create additional overhead in the encryption and decryption processes, especially when compared to symmetric encryption alone. This can impact system performance and speed.

- Cost: Implementing a hybrid encryption system can be costly due to the need for both symmetric and asymmetric encryption mechanisms. This can make it less practical for smaller organisations or individuals.

CHAPTER 4: ATTRIBUTE BASED ENCRYPTION

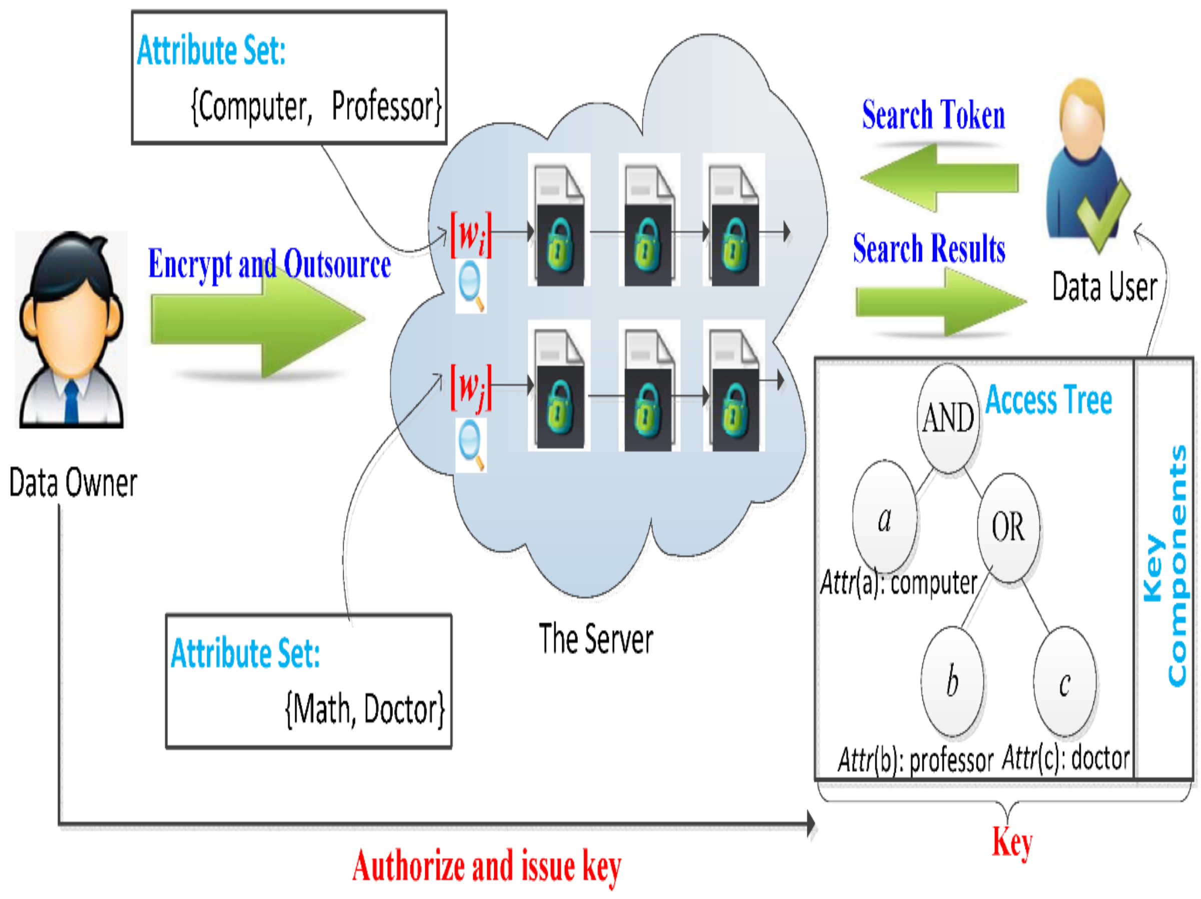
Attribute-based encryption (ABE) is a relatively recent approach that reconsiders the concept of public-key cryptography. In traditional public-key cryptography, a message is encrypted for a specific receiver using the receiver's public key. Identity-based cryptography, and in particular identity-based encryption (IBE), changed the traditional understanding of public-key cryptography by allowing the public key to be an arbitrary string, e.g., the email address of the receiver. ABE goes one step further and defines the identity not as atomic but as a set of attributes, e.g., roles, and messages can be encrypted with respect to subsets of attributes (key-policy ABE, KP-ABE) or policies defined over a set of attributes (ciphertext-policy ABE, CP-ABE). The key issue is that someone should only be able to decrypt a ciphertext if the person holds a key for "matching attributes," where user keys are always issued by some trusted party.

Attribute-based encryption (ABE) is a type of encryption scheme that allows data to be encrypted and decrypted based on specific attributes rather than specific identities. This type of encryption is often used in applications where data needs to be shared securely between multiple users with varying levels of access. In ABE, data is encrypted using a set of attributes that define the access control policies for the data. For example, a file containing confidential financial information might be encrypted with the attribute "Financial Department" so that only members of that department can access the file. When a user attempts to access the encrypted data, their credentials are compared to the attributes associated with the data. If the user's credentials match the access control policies associated with the data, the user is granted access, and the data is decrypted. Otherwise, the user is denied access, and the data remains encrypted.

There are two main types of ABE schemes:

i. attribute-based encryption with public key cryptography (ABE-PKC)
ii. attribute-based encryption with symmetric key cryptography (ABE-SKC).

In ABE-PKC, the encryption and decryption keys are based on public key cryptography, while in ABE-SKC, symmetric key cryptography is used. ABE has many applications in areas such as healthcare, finance, and government, where sensitive data needs to be securely shared among multiple parties with different access levels. However, ABE is also more complex than traditional encryption schemes and requires careful design to ensure security and performance.



4.1 WORKING OF ATTRIBUTE BASED ENCRYPTION

The workings of attribute-based encryption (ABE) can be explained in the following steps:

- Attribute-based encryption setup: In this step, a trusted authority generates the public and private key pair for the encryption scheme. The authority also defines a set of attributes that can be used to encrypt and decrypt the data.

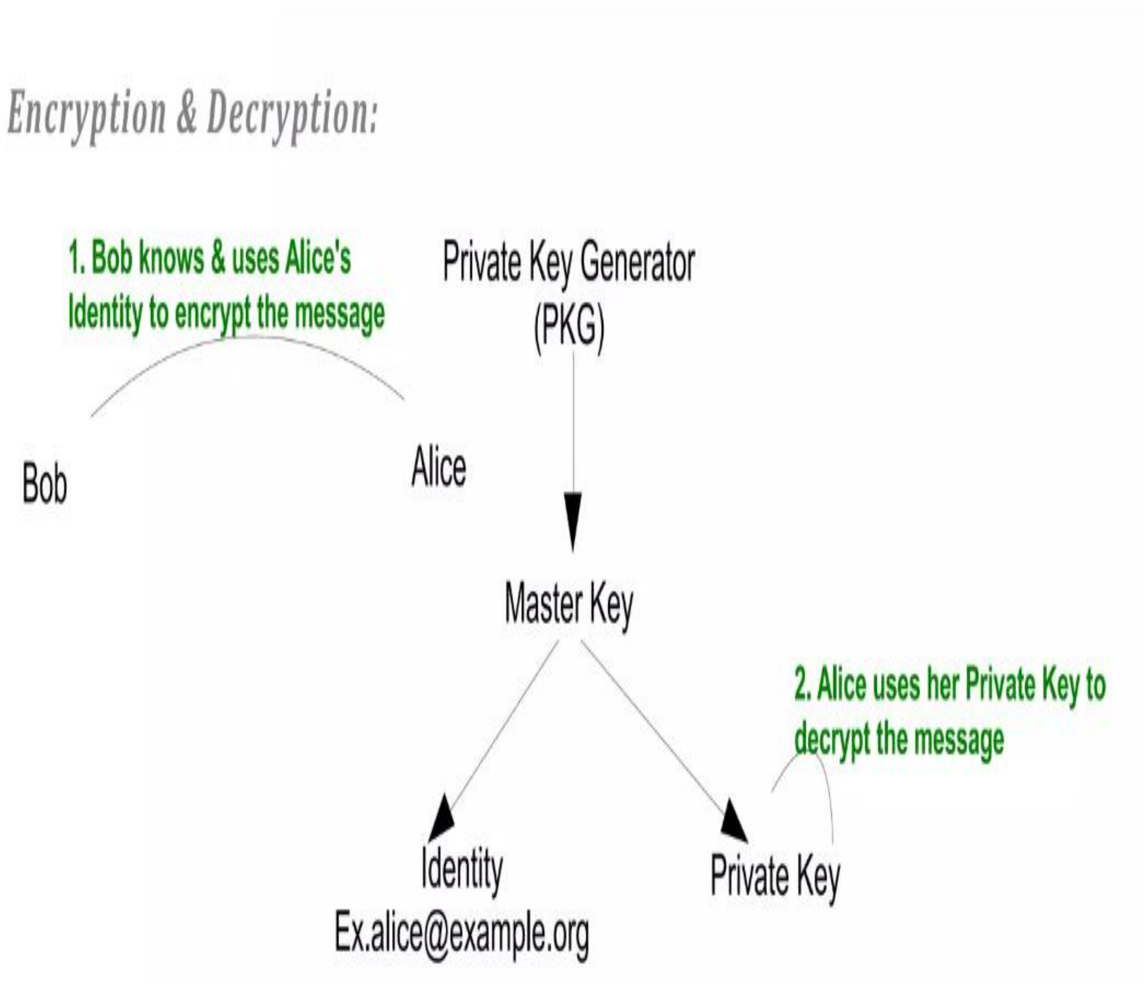
- Encryption: To encrypt the data, the user defines an access policy based on attributes. The user selects a set of attributes that are required to access the data and encrypts the data using those attributes.

- Key generation: When a user wants to access the encrypted data, the user requests a decryption key from the trusted authority. The user provides the access policy in the form of attributes to the authority.

- Decryption: The authority generates a decryption key based on the access policy and provides it to the user. The user then uses the decryption key to decrypt the data.

The decryption process works as follows:

i. The user presents the decryption key to the encrypted data.
ii. The decryption key is compared to the access policy defined during the encryption process.
iii. If the decryption key satisfies the access policy, the data is decrypted and presented to the user. If the decryption key does not satisfy the access policy, the data remains encrypted and cannot be accessed by the user.


Encryption & Decryption:
1. Bob knows & uses Alice's Identity to encrypt the message
Private Key Generator (PKG)
Bob
Alice
Master Key
2. Alice uses her Private Key to decrypt the message
Identity Ex.alice@example.org
Private Key

4.2 ADVANTAGES OF ATTRIBUTE BASED ENCRYPTION

ABE provides a flexible and secure approach to access control that can be applied to a wide range of applications, making it a valuable tool for organizations that need to share sensitive data among multiple parties with different access levels.

- Fine-grained access control: ABE allows for fine-grained access control to the data as it is based on attributes rather than specific identities. This makes it useful in situations where multiple parties need to access the data with different levels of access.

- Scalability: ABE can scale to a large number of users and data attributes without increasing the complexity of the encryption and decryption processes. This is because access policies can be defined based on a set of attributes rather than specific identities.

- Data sharing: ABE allows for secure data sharing between multiple parties with different access levels without the need for a centralised authority to manage the access control.

- Security: ABE provides a high level of security by ensuring that only users with the appropriate attributes can access the data. It also protects against attacks such as data leakage and theft.

- Privacy: ABE provides a level of privacy by allowing data to be encrypted based on attributes rather than specific identities. This makes it difficult for unauthorised parties to access the data.

- Compliance: ABE can help organisations comply with regulations such as GDPR and HIPAA by allowing them to control access to sensitive data and ensuring that it is only accessed by authorised parties.

4.3 DISADVANTAGES OF ATTRIBUTE BASED ENCRYPTION

- Complexity: ABE is more complex than traditional encryption schemes, which can make it more difficult to implement and manage. This can lead to higher costs and an increased risk of errors or security vulnerabilities.

- Key management: ABE requires careful key management to ensure that the access policies are enforced correctly. This can be challenging when dealing with a large number of users and attributes.

- Performance: ABE can be more computationally expensive than traditional encryption schemes, which can impact performance and scalability.

- Compatibility: ABE may not be compatible with all existing systems and applications, which can make it difficult to integrate into an organization's existing infrastructure.

- Lack of standardization: ABE is a relatively new encryption scheme, and there is currently no widely accepted standard for its implementation. This can make it more difficult to ensure interoperability and security across different systems.

- Risk of attribute inference: ABE relies on the use of attributes to define access policies, which can potentially reveal sensitive information about users or data. For example, if an attribute is used to identify a particular group or demographic, this could reveal personal information about users or data.

CHAPTER 5: ALGORITHM USED

In the proposed project, attribute-based encryption is used as the main encryption mechanism. That means various attributes of the staff are considered to generate various keys that will be used for encryption, sign-in key generation, verification key generation, hashing, and decryption. Here, various algorithms are used for digital signatures, hash generation, and encryption.

For digital signatures, the Elliptic Curve Digital Signature Algorithm (ECDSA) is used. For encryption and decryption, the AES algorithm is used. For hash generation, SHA is used.

5.1 ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM

The Elliptic Curve Algorithm (ECA) is a public-key cryptography algorithm based on the mathematical concept of elliptic curves. The algorithm uses the properties of these curves to generate a pair of keys (public and private) that can be used for encryption and decryption. The elliptic curve algorithm offers several advantages over other public-key cryptography algorithms, such as RSA. It offers higher security with shorter key lengths, which means that it requires less processing power and memory to perform the same cryptographic operations. Additionally, it offers better resistance against attacks such as brute-force attacks and quantum computing attacks. The elliptic curve algorithm is a popular and widely used public-key cryptography algorithm, particularly in applications that require high security with limited resources.

ECDSA (Elliptic Curve Digital Signature Algorithm) is a digital signature algorithm based on the Elliptic Curve Cryptography (ECC) system. It is used to create digital signatures for verifying the authenticity of messages and documents. ECDSA is a variant of the Digital

Signature Algorithm (DSA) that uses elliptic curve cryptography instead of the traditional integer factorization techniques used by DSA.

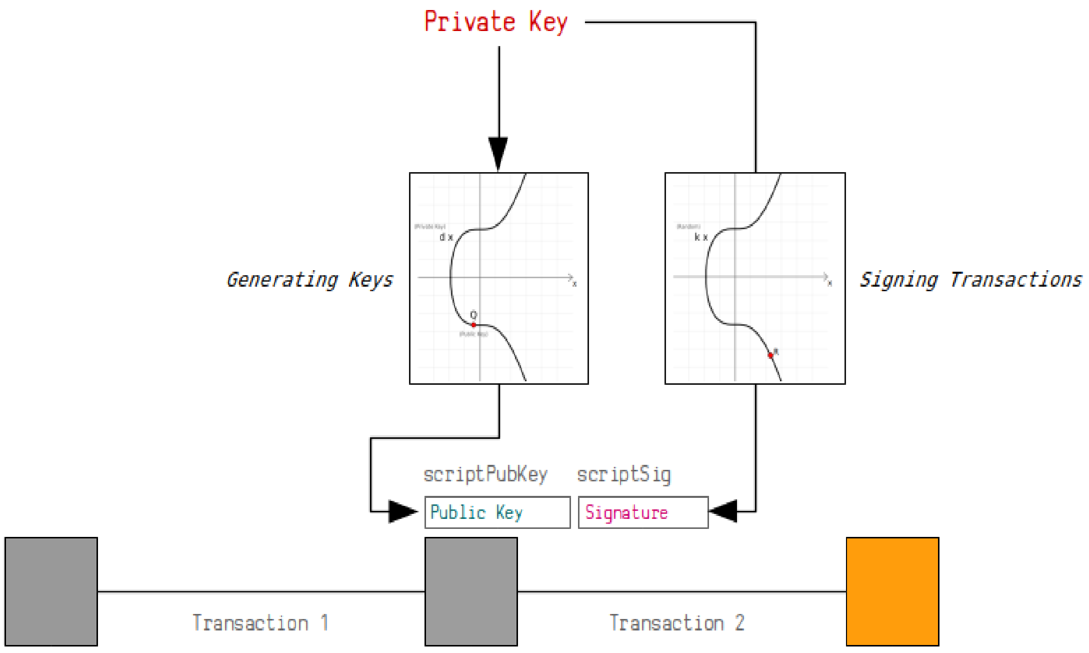
The ECDSA algorithm consists of the following steps:

- Key Generation: The first step in using ECDSA is to generate a pair of keys: a private key and a public key. The private key is a random number, while the public key is a point on the elliptic curve derived from the private key.

- Signing: To sign a message using ECDSA, the signer first generates a random number called a nonce. They then use the nonce and their private key to compute a signature value. The signature value is a pair of numbers (r, s) that represent the signature.

- Verification: To verify the signature, the verifier needs the signer's public key, the message, and the signature value. The verifier uses these inputs to compute a value called the "hash" of the message, which is a fixed-size value computed using a hash function. The verifier then uses the signer's public key, the hash value, and the signature value to compute a new point on the elliptic curve. If this point matches the original point used to create the signature, then the signature is valid.

The security of ECDSA is based on the intractability of the elliptic curve discrete logarithm problem, which involves finding the private key from the public key. This problem is believed to be computationally difficult, which makes ECDSA a secure digital signature algorithm.



ECDSA has several advantages over other digital signature algorithms. It provides strong security with smaller key sizes, making it more efficient and suitable for use in resource-constrained environments such as mobile devices and embedded systems. Additionally, ECDSA can be used with various elliptic curves, which allows for customization based on specific security requirements. ECDSA is a versatile and efficient digital signature algorithm that provides strong security and is suitable for a wide range of applications. Its smaller key sizes and faster computations make it particularly well-suited for use in resource-constrained environments, while its customizable security and compatibility with existing infrastructure make it an attractive option for organisations of all sizes.

ADVANTAGES OF ECDSA

- Smaller key sizes: ECDSA uses smaller key sizes compared to other digital signature algorithms, such as RSA. This results in faster computations and smaller signatures, which are particularly important in resource-constrained environments, such as mobile devices and embedded systems.

- Faster computations: ECDSA has faster computations compared to other digital signature algorithms due to its use of elliptic curve cryptography. This makes ECDSA suitable for real-time applications that require fast signature verification.

- Customizable security: ECDSA can be used with various elliptic curves, allowing for customization based on specific security requirements. This flexibility makes it an attractive option for organizations that require different levels of security for different applications.

- Strong security: ECDSA provides strong security due to the intractability of the elliptic curve discrete logarithm problem, which involves finding the private key from the public

-

  key. This problem is believed to be computationally difficult, which makes ECDSA a secure digital signature algorithm.

- Compatibility with existing infrastructure: ECDSA is compatible with existing infrastructure, including digital certificates and public key infrastructures (PKIs). This makes it easy to integrate into existing systems and networks.

DISADVANTAGES OF ECDSA

Disadvantages include:

- Limited standardisation: While ECDSA is widely used, it has less standardisation than other digital signature algorithms, such as RSA. This can make it difficult for organisations to ensure interoperability and compatibility across different systems and networks.

- Key management: Like all digital signature algorithms, ECDSA requires proper key management to ensure the security and integrity of signed messages. This can be challenging, particularly in large-scale deployments, and requires careful planning and implementation.

- Complexity: ECDSA can be more complex to implement and use compared to other digital signature algorithms, such as RSA. This can make it more difficult for non-experts to understand and use correctly.

- Vulnerabilities to side-channel attacks: ECDSA, like all cryptographic systems, is vulnerable to side-channel attacks, which involve exploiting weaknesses in the physical implementation of the algorithm. For example, attackers may use timing or power analysis to extract sensitive information from the algorithm. While there are countermeasures available to mitigate these attacks, they add complexity and cost to the implementation of ECDSA.

-

- Patent issues: Some elliptic curve parameters used in ECDSA may be subject to patents or other intellectual property rights. This can limit the use of certain elliptic curves and add complexity to legal compliance for organizations using ECDSA.

ECDSA does not just need to be used in the signing of certificates; it can be used anywhere RSA has been with the same effect in the end. Public-key cryptography methods are found in everything from TLS/SSL to code signing. The government uses ECDSA to protect internal communications, while Tor uses it to maintain anonymity for their users. These are just a few of the uses ECDSA can be put to, but all cryptosystems face an issue with the emergence of quantum computing. Quantum computing threatens to make all classic cryptosystems, from AES to RSA to ECDSA, obsolete. The methods used in quantum computing mean previously strong methods like ECDSA will need an update to use quantum cryptography or become obsolete.

5.2 AES ALGORITHM

AES (Advanced Encryption Standard) is a symmetric key encryption algorithm that is widely used to protect data in transit and at rest. It was developed by Joan Daemen and Vincent Rijmen and was adopted as a standard by the National Institute of Standards and Technology (NIST) in 2001.

AES works by using a secret key to encrypt and decrypt data. The algorithm operates on fixed-length blocks of data (128 bits), which are transformed through a series of rounds. The number of rounds depends on the key size, with 10 rounds for a 128-bit key, 12 rounds for a 192-bit key, and 14 rounds for a 256-bit key.

The AES algorithm consists of the following steps:

- Key Expansion: The first step is to expand the secret key into a set of round keys that will be used in the encryption and decryption process.
- Initial Round: In the first round, the plaintext is XORed with the first-round key.
- Rounds: In each subsequent round, the plaintext is transformed using a set of operations that includes substitution, permutation, and XOR with a round key.
- Final Round: The final round is similar to the previous rounds, but it does not include the permutation step.
- Output: The resulting ciphertext is the final output of the encryption process.

AES provides strong security and is widely used to protect sensitive data, including financial transactions, government communications, and personal information.

AES is a widely used and trusted encryption algorithm that provides strong security and efficient performance for a wide range of applications. AES has been widely adopted as a standard encryption algorithm and is used in many other applications and systems beyond those listed above. Its strength, speed, and flexibility make it an attractive option for organizations and developers seeking to provide strong encryption and protection of sensitive data. AES is widely used and implemented in a variety of applications and systems to provide secure encryption and protection of sensitive data.

5.2.1 WORKING OF AES

AES (Advanced Encryption Standard) works by using a secret key to transform plaintext into ciphertext, and vice versa. The algorithm operates on fixed-length blocks of data (128 bits), which are transformed through a series of rounds. The number of rounds depends on the key size, with 10 rounds for a 128-bit key, 12 rounds for a 192-bit key, and 14 rounds for a 256-bit key.

Here are the steps involved in the AES algorithm:

- Key Expansion: The first step is to expand the secret key into a set of round keys that will be used in the encryption and decryption process. This is done by applying a set of operations to the original key, which generates a sequence of round keys.

- Initial Round: In the first round, the plaintext is XORed with the first round key.

- Rounds: In each subsequent round, the plaintext undergoes a series of transformations that include substitution, permutation, and XOR with a round key. The transformations performed in each round depend on the key size and the number of rounds.

- Final Round: The final round is similar to the previous rounds, but it does not include the permutation step.

- Output: The resulting ciphertext is the final output of the encryption process.

The transformations performed in each round of the AES algorithm include:

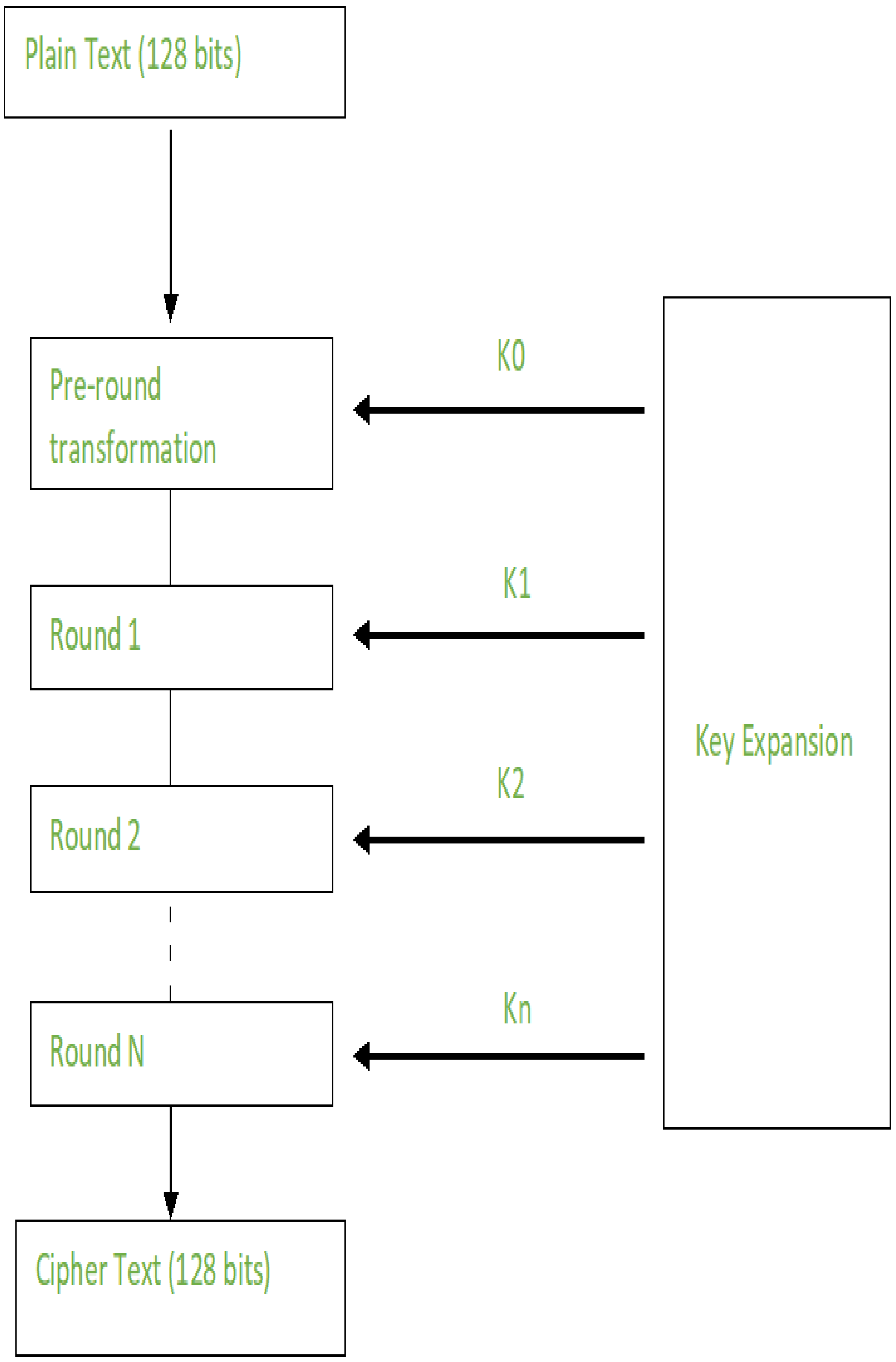
- SubBytes: This step substitutes each byte of the block with a corresponding value from a fixed lookup table, called the S-box.

- ShiftRows: This step shifts the rows of the block to create diffusion and prevent patterns from appearing in the ciphertext.

- MixColumns: This step combines the columns of the block by multiplying them with a fixed matrix to provide additional diffusion and confusion.

- AddRoundKey: This step XORs the block with a round key generated during the key expansion step.

In the final round, the MixColumns step is omitted to simplify the decryption process. To decrypt the ciphertext, the process is reversed by applying the inverse transformations in the reverse order, using the same key.


Plain Text (128 bits)
Pre-round transformation
Round 1
Round 2
Round N
Cipher Text (128 bits)
K0
K1
K2
Kn
Key Expansion

5.2.2 ADVANTAGES OF AES

- Strong Security: AES is considered to be one of the most secure encryption algorithms available today. It uses a symmetric key to encrypt and decrypt data, and it has undergone extensive analysis and testing to ensure its security.

- High Speed: AES encryption and decryption is relatively fast and efficient, making it suitable for use in a wide range of applications, including those that require real-time data processing.

- Flexibility: AES can be used with different key sizes (128, 192, or 256 bits), allowing users to choose the appropriate level of security for their needs.

- Widely Adopted: AES is widely adopted and supported by many platforms and applications, making it a standard encryption algorithm that can be used across different systems.

- Resistance to Attacks: AES is designed to be resistant to known attacks, including brute-force attacks, differential cryptanalysis, and linear cryptanalysis.

- Hardware Support: AES encryption and decryption can be accelerated by dedicated hardware, such as encryption accelerators and CPUs with AES instructions, which can improve performance in high-speed applications.

5.2.3 DISADVANTAGES OF AES

- Key Management: AES requires the use of a secret key to encrypt and decrypt data, and this key must be kept secure to prevent unauthorized access. Key management can be complex and challenging, especially in large organizations or systems where multiple keys are used.

- Vulnerability to Side-Channel Attacks: AES implementations can be vulnerable to side-channel attacks, which are attacks that exploit information leaked by the hardware or software used to implement the algorithm. For example, attackers can measure the power consumption or electromagnetic radiation emitted by a device to extract the secret key used by the AES algorithm.

- Lack of Forward Secrecy: AES does not provide forward secrecy, which means that if an attacker obtains the secret key used to encrypt past communications, they can decrypt those communications even if the key is changed in the future.

- Limited Protection against Malware: AES can protect data only when it is in transit or at rest. It does not protect against malware or other types of attacks that compromise the integrity of the system or device used to encrypt or decrypt the data.

- Standardization Issues: While AES is a widely adopted and standardized encryption algorithm, there have been concerns about the integrity of the standardization process and the potential for backdoors or other vulnerabilities to be introduced into the algorithm.

5.3 SHA-1 HASHING ALGORITHM

Hashing is an important tool for ensuring the integrity, security, and efficient processing of data in a wide range of applications.

SHA-1 is a cryptographic hash function that generates a fixed-length 160-bit hash value. It was developed by the United States National Security Agency (NSA) and was published as a federal standard in 1995.

SHA-1 takes an input message of any length and generates a fixed-size output, which is a 160-bit hash. The output is typically represented as a 40-character hexadecimal string.

The SHA-1 algorithm operates in a similar way to other hash functions, using a series of logical operations and mathematical functions to transform the input message into the fixed-size output hash. The algorithm uses a compression function that takes as input a 512-bit block of the message and the previous output hash, and produces a new 160-bit output hash.

SHA-1 is widely used in various applications, including digital signatures, message authentication codes, and other forms of digital security. However, due to recent advances in cryptanalysis, it is no longer considered secure for use in critical applications. As a result, it has been deprecated in favor of more secure hash functions like SHA-256 and SHA-3.

WORKING OF SHA-1

- Step 1 − Append padding bits:

  The original message is padded and its duration is congruent to 448 modulo 512. Padding is continually inserted although the message already has the desired length. Padding includes a single 1 followed by the essential number of 0 bits.

- Step 2 − Append length:

  A 64-bit block considered as an unsigned 64-bit integer (most essential byte first), and defining the length of the original message (before padding in step 1), is added to the message. The complete message's length is a multiple of 512.

- Step 3 −Initialize the buffer:

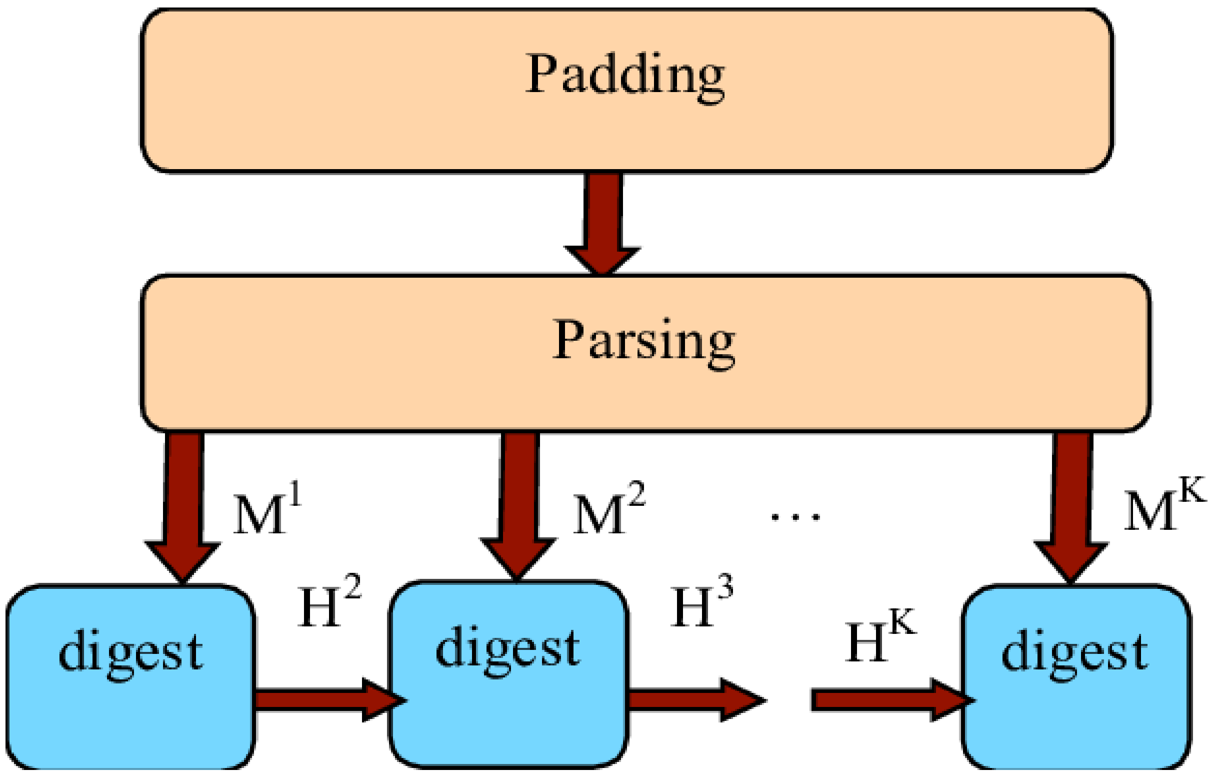
  The buffer includes five (5) registers of 32 bits each indicated by A, B, C, D, and E. This 160-bit buffer can be used to influence temporary and final outcomes of the compression function. These five registers are initialized to the following 32-bit integers (in hexadecimal notation).

- Step 4 − Process message in 512-bit blocks:

  The compression function is divided into 20 sequential steps includes four rounds of processing where each round is made up of 20 steps. The four rounds are structurally same as one another with the only difference that each round need a different Boolean function, which it can define as f1, f2, f3, f4 and one of four multiple additive constants Kt ($0 \leq t \leq 79$) which is based on the step under consideration.

- Step 5 – Output:

  After processing the final 512-bit message block t (considering that the message is divided into t 512-bit blocks), and it can obtain a 160-bit message digest.



ADVANTAGES OF SHA-1

- Efficiency: SHA-1 is a relatively fast algorithm that can generate a hash value for a message of any size in a reasonable amount of time. This makes it well-suited for use in applications that require the efficient computation of hash values, such as digital signatures or message authentication.

- Simplicity: The SHA-1 algorithm is relatively simple to implement and can be easily integrated into a wide range of software applications

- Interoperability: SHA-1 is a widely used hash function that is supported by many software libraries and systems. This makes it easy to share and verify hash values across different platforms and applications.

DISADVANTAGES OF SHA-1

- Security: SHA-1 has been shown to be vulnerable to collision attacks, where two different inputs can produce the same hash value. This means that an attacker could potentially create a fraudulent message with the same hash value as a legitimate one, compromising the integrity of the data.

- Deprecation: Due to the security vulnerabilities in SHA-1, it has been deprecated by most security organizations and software vendors, and is no longer considered a secure hash function for critical applications.

- Limited Hash Size: SHA-1 generates a fixed-size 160-bit hash value, which is relatively small compared to more modern hash functions like SHA-256 or SHA-3. This limits its use in applications that require a larger hash size for stronger security.

- Length Extension Attack: SHA-1 is vulnerable to a length extension attack, where an attacker can append additional data to a valid message without knowing the original contents of the message. This could allow an attacker to manipulate the hash value and compromise the integrity of the data.

-

- Performance: While SHA-1 is relatively fast compared to some other cryptographic hash functions, it is slower than some newer algorithms like SHA-256 and SHA-3. This could be a disadvantage in applications that require high-speed processing of large amounts of data.

5.4 BASE 64 ENCODING

Base64 is a binary-to-text encoding scheme that represents binary data in an ASCII string format. The term "base64" refers to the fact that each 64-character ASCII string represents a 6-bit binary sequence.

In the Base64 encoding scheme, each 6 bits of the binary data is represented by a corresponding ASCII character. The resulting ASCII string is therefore larger than the original binary data, but it can be transmitted as text over a variety of network protocols that only support text-based data.

To encode data in Base64, a software program or library is typically used, which takes the binary data as input and generates a Base64-encoded string as output. The resulting string can then be transmitted over the desired network protocol as text data. Similarly, to decode Base64 data, the Base64-encoded string is input to a software program or library that converts it back into the original binary data.

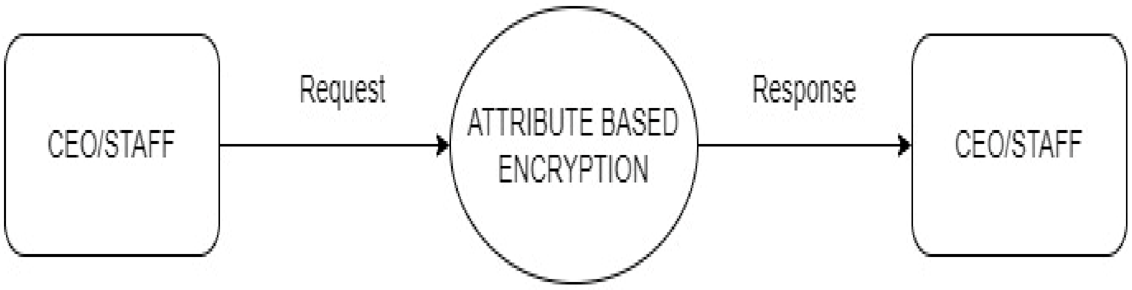
Base64 encoding is used to convert binary data into a text format that can be easily transmitted over various network protocols or mediums that only support text-based data. In Base64 encoding, binary data is divided into groups of 6 bits, and each 6-bit group is mapped to a corresponding ASCII character. This results in a string of ASCII characters that can be easily transmitted over the network. Base64 encoding is widely used in email systems, web browsers, and other applications that need to transmit binary data over a network.

5.5 DATA FLOW DIAGRAM

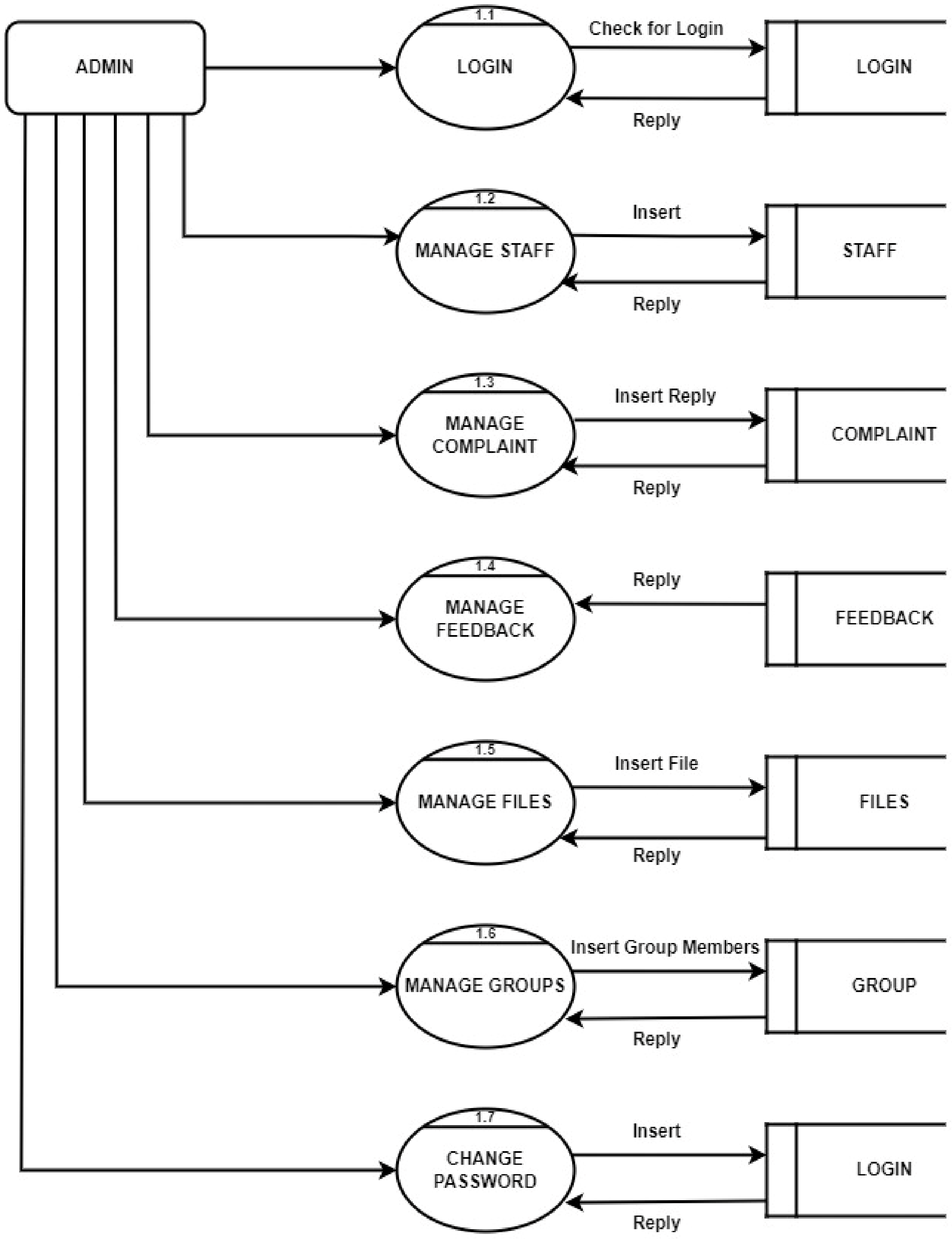
A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually “say” things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO. That’s why DFDs remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems.

Data flow diagram for proposed project is:

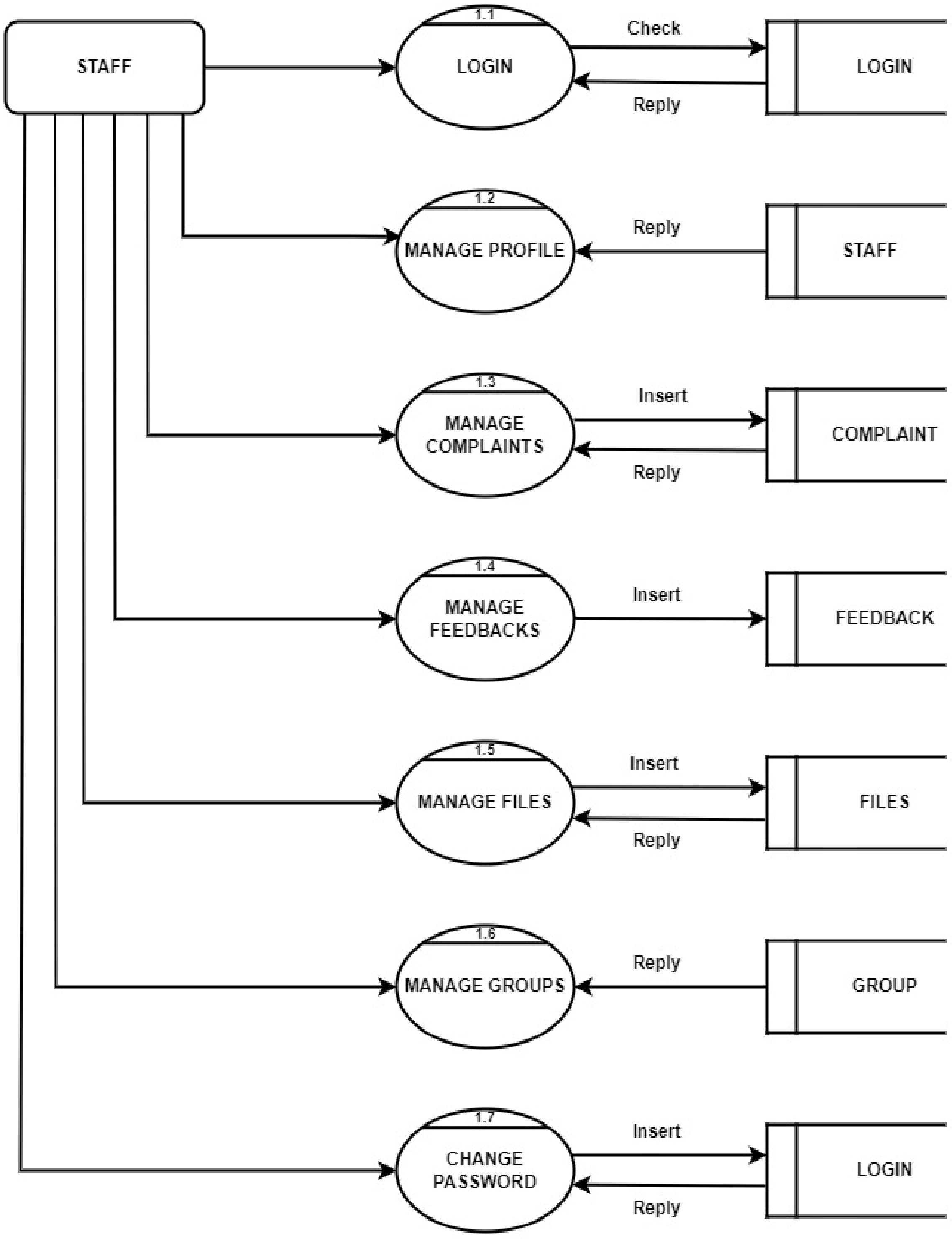
Level 0



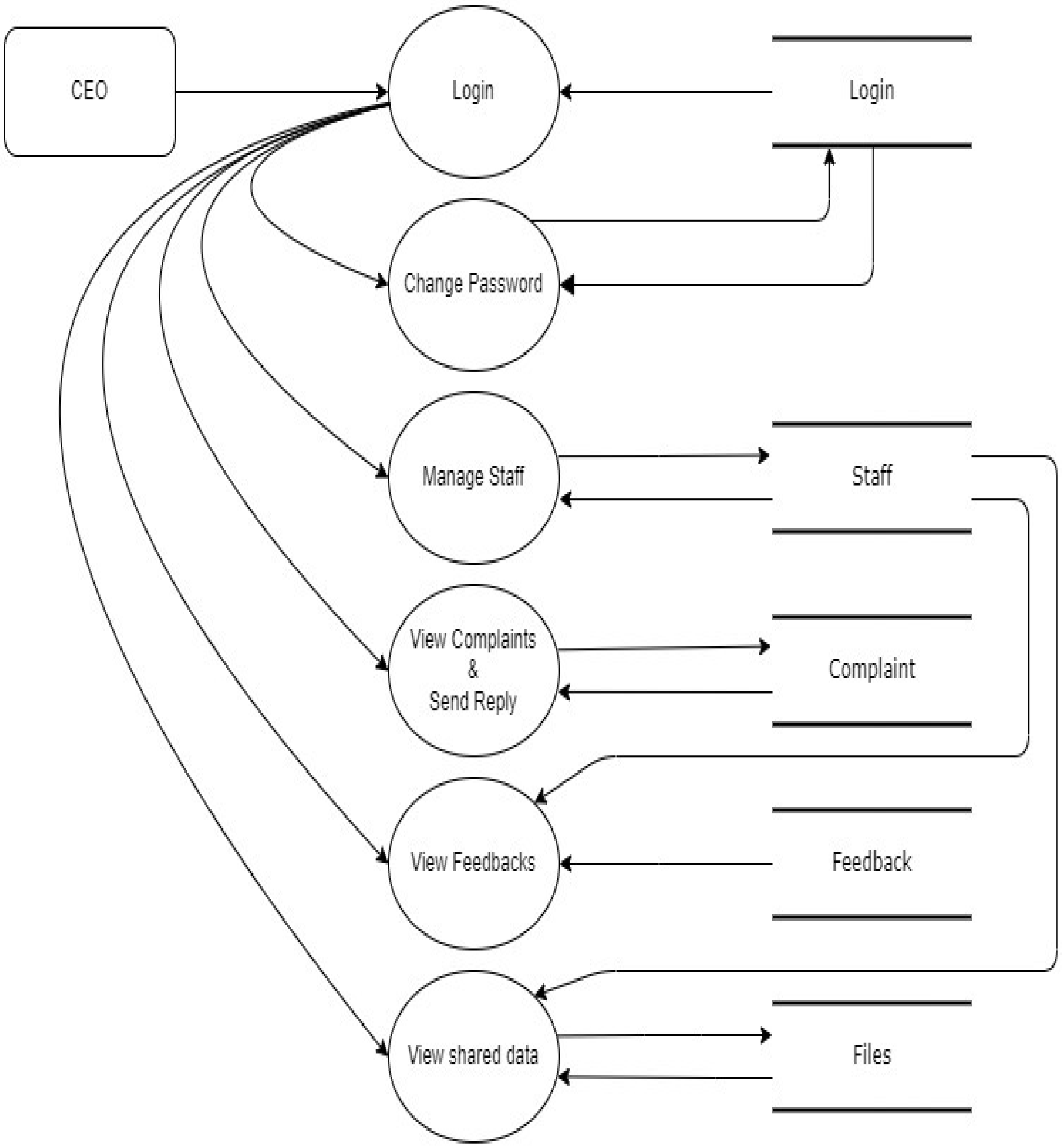
LEVEL 1 ADMIN SIDE



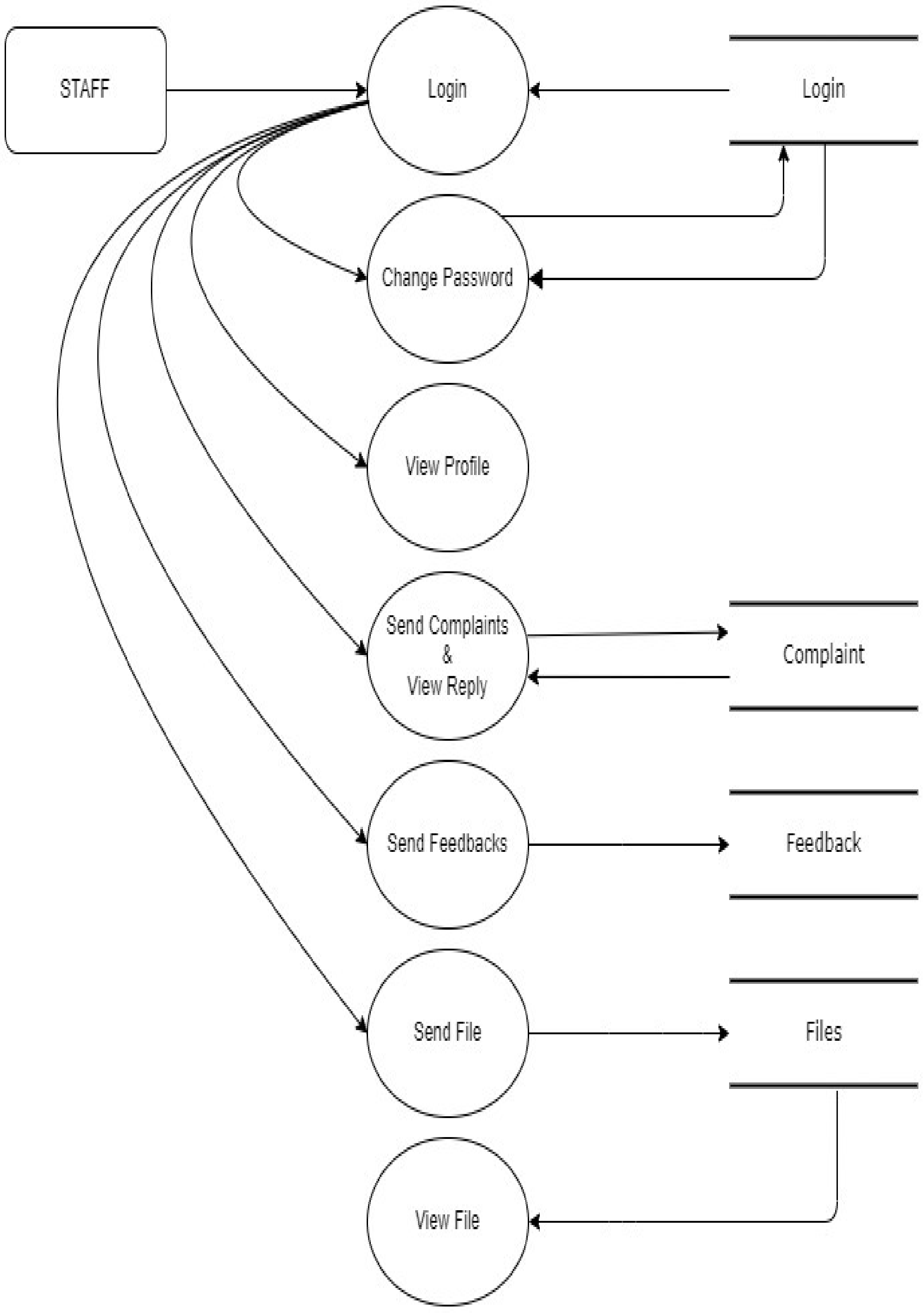
LEVEL 1 STAFF SIDE



LEVEL 2 ADMIN SIDE



LEVEL 2 STAFF SIDE



CHAPTER 6: SOURCE CODE AND OUTPUT

6.1 WORKING ENVIRONMENT

6.1.1 LANGUAGE SPECIFICATION

- Programming Language used in this project is Python.
- Python is a high-level, general-purpose and a very popular programming language.
- Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting-edge technology in Software Industry.
- Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.
- Python allows programming in Object-Oriented and Procedural paradigms.
- Hyper Text Markup Language is used for web page creation.
- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.
- SQL is used for database

6.1.2 HARWARE SPECIFICATION

- Processor: Intel 5 (minimum) [Ryzen 5 used]
- RAM: 4 GB (minimum) [8 GB used]
- Hard Disk: Intel 5 (minimum) [Ryzen 5 used]
- Display: 14" monitor (for more comfort: minimum) [15.6’’ used]

6.1.3 SOFTWARE SPECIFICATION

- Windows 11 is used
- Python 3.6 is used
- Django framework of PyCharm is used
- Dreamweaver is used for page designing
- SQLyog Community is used for storing data in the database.
- AmazonS3 is used as the cloud storage

6.2 SOURCE CODE

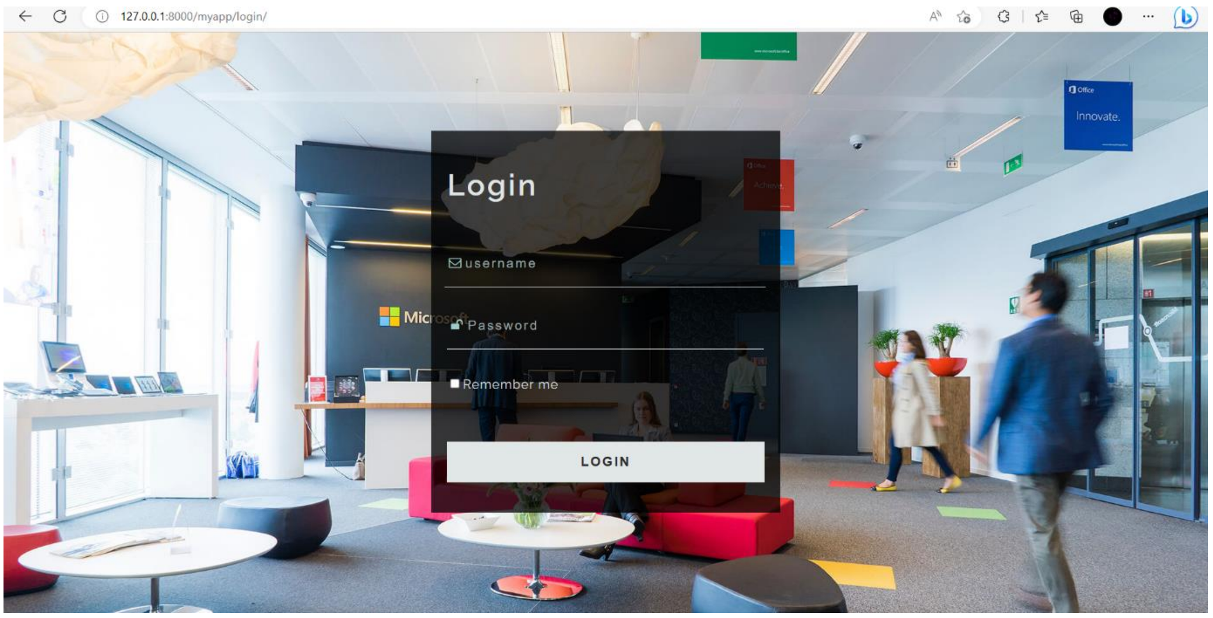
```
import base64
import datetime
import hashlib
from hashlib import sha256
from struct import pack
from itertools import count, chain
from ecdsa.ellipticcurve import Point
import urllib
from ecdsa.util import string_to_number, number_to_string, randrange
from ecdsa.curves import SECP256k1
from django.core.files.storage import FileSystemStorage
from django.db.models import Exists, OuterRef
from django.http import HttpResponse, request
from django.shortcuts import render, redirect
import random



def download_file_staff(request,fid):
    res=file_tb.objects.get(id=fid)
    groupid=res.groupmember_id.group_id_id

    groupmember_tbobj=
groupmember_tb.objects.get(group_id_id=groupid,staff_id__login_id=request.session['lid'])



    gmb= groupmember_tbobj

    print(gmb.verification_key,"lk")

    print(gmb.signin_key,"kks")


    import base64

    import pyAesCrypt


pyAesCrypt.decryptFile("C:\\Users\\User\\PycharmProjects\\attribute_based_encryption
\\media\\" + res.file+".aes",

"C:\\Users\\User\\PycharmProjects\\attribute_based_encryption\\media\\aa" +res.file ,
                        passw="44555555555555555555555555555555")
```

```
   with
open("C:\\Users\\User\\PycharmProjects\\attribute_based_encryption\\media\\"+res.file
+".aes", "rb") as image_file:
        a=image_file.read()

        from ecdsa import SigningKey, NIST192p

        base64_string = base64.b64decode(gmb.signin_key.encode("ascii"))

        sk2 = SigningKey.from_string(base64_string, curve=NIST192p)

        m = sk2.verifying_key

        import boto3

        # Creating Session With Boto3.
        session = boto3.Session(aws_access_key_id="AKIAT3Y2D6A25LHJXEGF",

aws_secret_access_key="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")

        s3 = session.resource('s3')

        s3 = boto3.client('s3',aws_access_key_id="AKIAT3Y2D6A25LHJXEGF",

aws_secret_access_key="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")


hs=user_file_hash.objects.get(file_id_id=res.id,staff_id__login_id=request.session['lid']).file
_hash

        firsthash=base64.b64decode(hs.encode('ascii'))

        print("okkkk")

        kk = m.verify(firsthash, a)

        print(kk," check abcd")
        if kk == True:

file_path="C:\\Users\\User\\PycharmProjects\\attribute_based_encryption\\media\\aa"
+ res.file
            with open(file_path, 'rb') as fh:
                response = HttpResponse(fh.read(), content_type="application/vnd.ms-excel")
                response['Content-Disposition'] = 'inline; filename=' + file_path
                return response


            return HttpResponse('failed')
        else:
            pass


            return HttpResponse('failed')


# ----------------------------algo----------------------
def group_members(request):
    return render(request,'CEO/GROUP_MEMBERS.html')
def random_scalar():
    return randrange(SECP256k1.order)



def group_members_post(request):
    cc=request.POST.getlist("checkbox")
    print(cc)
    for i in cc:
        gmob=groupmember_tb()
        gmob.staff_id = staff_tb.objects.get(id=i)
        gmob.group_id = group_tb.objects.get(id=request.session['gid'])


        from ecdsa import SigningKey, NIST192p

        sk = SigningKey.generate(curve=NIST192p)  # uses NIST192p

        sk = sk.to_string()

        print(type(sk))

        import base64
        base64_bytes = base64.b64encode(sk)
        base64_string = base64_bytes.decode("ascii")

        print(base64_string, type(base64_string))

        gmob.signin_key= base64_string
        gmob.signin_key= base64_string
        gmob.save()

    return group(request)

def group_members_postold(request):
    cc=request.POST.getlist("checkbox")
    print(cc)
    for i in cc:
        gmob=groupmember_tb()
        gmob.staff_id = staff_tb.objects.get(id=i)
        gmob.group_id = group_tb.objects.get(id=request.session['gid'])
        gmob.save()

    mycount=groupmember_tb.objects.filter(group_id_id=request.session['gid']).count()
    print(mycount)

    secrets = [random_scalar() for i in range(mycount)]
    print("secrt")
    print(secrets)
    print(len(secrets))

    pubkeys = [G * s for s in secrets]
    print("pubkeys")
    print(pubkeys)
    print(pubkeys[1])

    rings = [pubkeys[:mycount]]
    print(random.choice(secrets[:mycount]))
    print("show scrt key")
    secret_keys = [random.choice(secrets[:mycount])]
    print(secret_keys)
    print(len(secret_keys))

    allmembers=groupmember_tb.objects.filter(group_id_id=request.session['gid'])
    for i in range(mycount):
        uskeobjs=user_group_key()
        uskeobjs.staff_id=allmembers[i].staff_id
        print(allmembers[i].staff_id)
        uskeobjs.groupmem_id_id=allmembers[i].id
        uskeobjs.KEY=pubkeys[i]
        uskeobjs.SC_KEY=secrets[i]
        uskeobjs.save()


    return group(request)


def upload_file_post(request):
    file = request.FILES['fileField']
    filename=request.POST['textfield']
    from datetime import datetime
    s = datetime.now().strftime("%Y%m%d%H%M%S") + file.name
    fs = FileSystemStorage()
    fn = fs.save(s, file)
    path = s
    a=None


    import pyAesCrypt
```

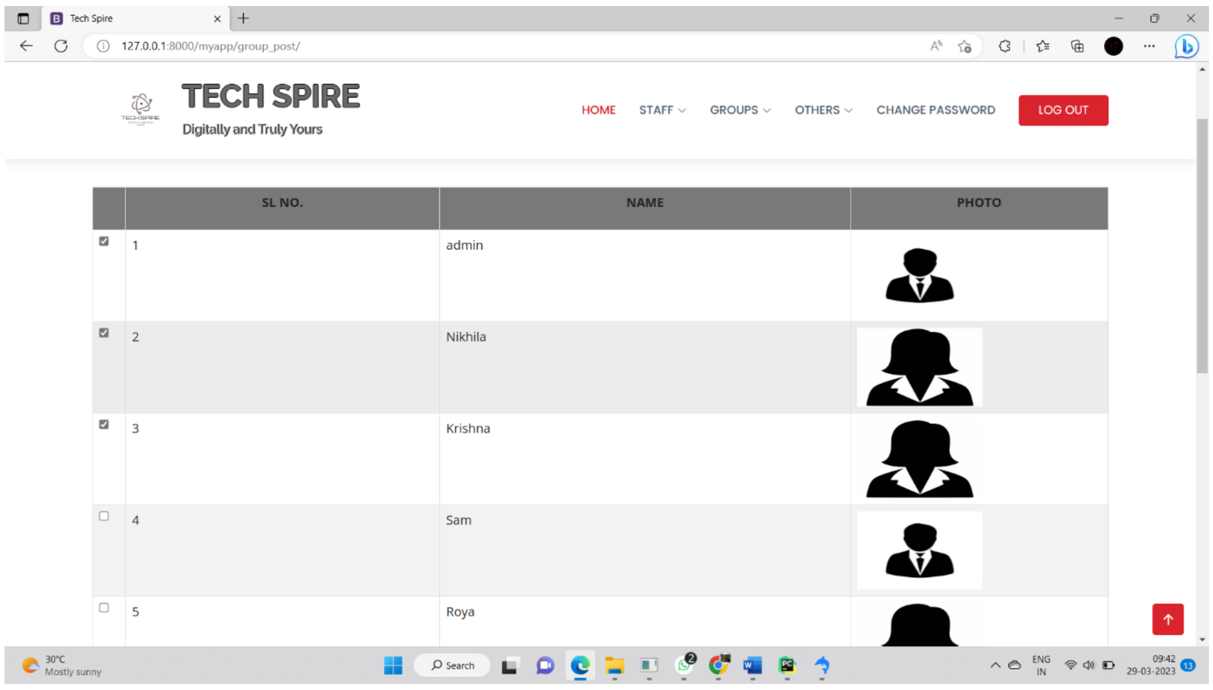
```
pyAesCrypt.encryptFile("C:\\Users\\User\\PycharmProjects\\attribute_based_encryption
\\media\\"+s,"C:\\Users\\User\\PycharmProjects\\attribute_based_encryption\\media\\"
+s+".aes",passw="xxxxxxxxxxxxxxxxxxxxxxxxx")

   with
open("C:\\Users\\User\\PycharmProjects\\attribute_based_encryption\\media\\"+s+".ae
s", "rb") as image_file:

       a=image_file.read()
   encoded_string= base64.b64encode(a)
   print("en st",len(encoded_string))
   print(encoded_string)

   m = hashlib.sha1()
   m.update(encoded_string)



   fob=file_tb()
   fob.file=path
   fob.groupmember_id_id=request.session["gmid"]
   fob.date=datetime.now()
   fob.file_name=s
   fob.save()

   yyy = []
   members=groupmember_tb.objects.filter(group_id_id=request.session["gid"])
   for i in members:
       from ecdsa import SigningKey, NIST192p

       base64_string =  base64.b64decode(i.signin_key.encode("ascii"))

       sk2 = SigningKey.from_string(base64_string, curve=NIST192p)


       m=sk2.sign(a)
       print(m)

       print(type(m))


       user_file_hashpbj= user_file_hash()
       user_file_hashpbj.file_id_id=fob.id
       user_file_hashpbj.staff_id_id=i.staff_id_id
       user_file_hashpbj.file_hash= base64.b64encode(m).decode('ascii')
       user_file_hashpbj.save()

       import boto3
```

```
        #Creating Session With Boto3.
        session =
boto3.Session(aws_access_key_id="AKIAT3Y2D6A25LHJXEGF",aws_secret_access_ke
y="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")
        #Creating S3 Resource From the Session.
        s3 = session.resource('s3')

s3.meta.client.upload_file("C:\\Users\\User\\PycharmProjects\\attribute_based_encryptio
n\\media\\"+s+".aes", 'abesharecloud', s )

    return HttpResponse('''<script>alert('FILE
UPLOADED');window.location='/myapp/view_group_staff'</script>''')



def upload_file_postold(request):
    file = request.FILES['fileField']
    filename=request.POST['textfield']
    from datetime import datetime
    s = datetime.now().strftime("%Y%m%d%H%M%S") + file.name
    fs = FileSystemStorage()
    fn = fs.save(s, file)
    path = fs.url(s)

    with
open("C:\\Users\\User\\PycharmProjects\\attribute_based_encryption\\media\\"+s,
"rb") as image_file:

        a=image_file.read()
    encoded_string= base64.b64encode(a)
    print("en st",len(encoded_string))
    print(encoded_string)

    m = hashlib.sha1()
    m.update(encoded_string)

    print(m.hexdigest())
    a_msg = m.hexdigest()

    yyy = []
    members=groupmember_tb.objects.filter(group_id_id=request.session["gid"])
    for i in members:

        sckey=i.signin_key
        sckey=sckey.replace("(","").replace(")","")
        yyy.append(int(sckey))
    print(yyy)
    pubkeys = [G * s for s in yyy]
    rings = [pubkeys[:2]]

    print("pupu")
    print(pubkeys)
    sig1, sign2 = sign(a_msg, rings, yyy)
    print(sign2)

    fob=file_tb()
    fob.file=path
    fob.groupmember_id_id=request.session["gmid"]
    fob.date=datetime.now()
    fob.file_name=filename
    fob.save()

    return
HttpResponse('''<script>alert('FILEUPLOADED');window.location="/myapp/view_gro
up_st   aff"</script>''')
def H(m):
    print("m")
    print (m)
    return sha256(m).digest()


def string_to_scalar(s):
    n = string_to_number(s)
    assert 0 <= n < SECP256k1.order
    return n

def bor_H(m, r, i, j):
    print("bor_h 4 values")
    print(m)
    print(r)
    print(i)
    print(j)


    r = serialize_point(r) if isinstance(r, Point) else r
    return string_to_scalar(H(m + r + pack('!ii', i, j)))

def serialize_point(p): # SEC compressed format
    return chr((p.y() & 1) + 2) + number_to_string(p.x(), SECP256k1.order)
def sign(message, rings, secret_keys):


    secret_keys = {G * secret : secret for secret in secret_keys}
    print ("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$")

    print (secret_keys)
    print ("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$")
    print (secret_keys)
    print ("hellllo")

    known_pubkeys = secret_keys.keys()

    print (known_pubkeys)
    known_keys_by_ring = [set(known_pubkeys) & set(ring) for ring in rings]



    for ring in rings:
        print ("******************************")
        print (known_pubkeys)
        print (ring)
        print ("================================")



    # check we know a secret key in each ring
    assert all(known_keys_by_ring)
    known_key_indexes = [ring.index(known.pop()) for ring, known in zip(rings,
known_keys_by_ring)]
    M = H(message + ''.join(map(serialize_point, chain(*rings))))
    s = [[random_scalar() for _ in ring] for ring in rings]
    k = [random_scalar() for _ in ring]
    e0_hash = sha256()
    for ring, known_key_index, i in zip(rings, known_key_indexes, count()):
        r_i_j = k[i] * G
        for j in range(known_key_index + 1, len(ring)):
            e_i_j = bor_H(M, r_i_j, i, j)
            r_i_j = s[i][j] * G + e_i_j * ring[j]
        e0_hash.update(serialize_point(r_i_j))
    e0_hash.update(M)  # this step not in paper?
    e0 = e0_hash.digest()
    for ring, known_key_index, i in zip(rings, known_key_indexes, count()):
        e_i_j = bor_H(M, e0, i, 0)
        for j in range(0, known_key_index):
            r_i_j = s[i][j] * G + e_i_j * ring[j]
            e_i_j = bor_H(M, r_i_j, i, j + 1)
        secret = secret_keys[ring[known_key_index]]
        s[i][known_key_index] = (k[i] - e_i_j * secret) % SECP256k1.order
    return e0, s
```

6.3 OUTPUT



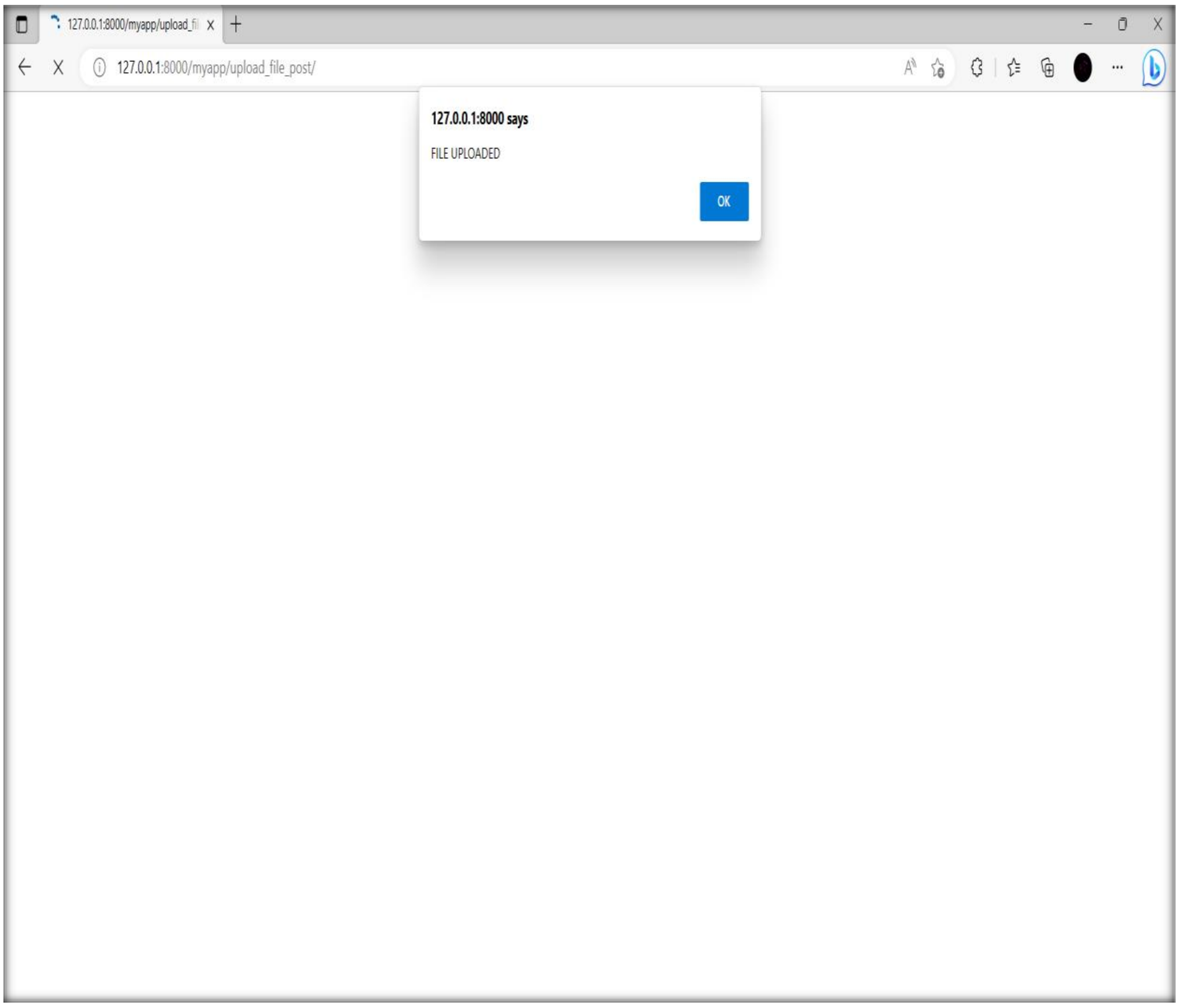
Login form



	SL NO.	NAME	PHOTO
☑	1	admin	
☑	2	Nikhila	
☑	3	Krishna	
☐	4	Sam	
☐	5	Roya	

Group is created by the CEO. CEO can select the members using check boxes.

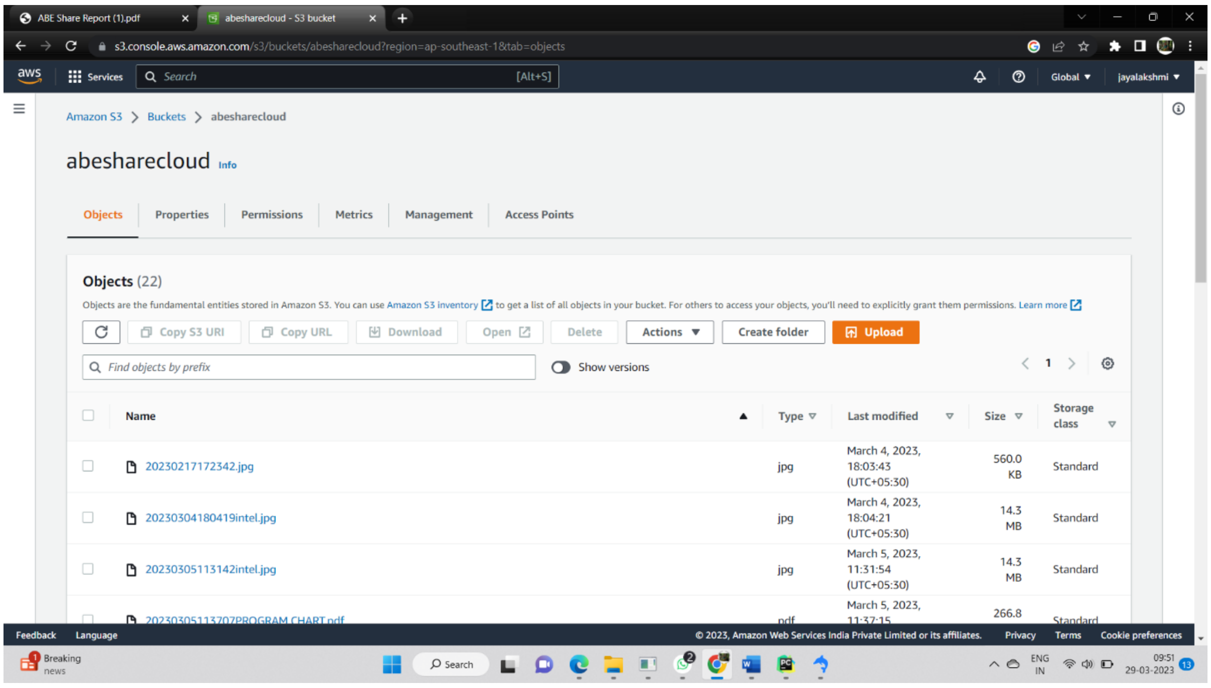
After the group is created signing key and verification keys are generated based on the group details.


127.0.0.1:8000/myapp/upload_file_post/

127.0.0.1:8000 says

FILE UPLOADED

A user uploads a file to their group . While uploading the file is being encrypted using the keys generated using the attributes.


ABE Share Report (1).pdf
abesharecloud - S3 bucket
s3.console.aws.amazon.com/s3/buckets/abesharecloud?region=ap-southeast-1&tab=objects
Services
Search [Alt+S]
Global
jayalakshmi
Amazon S3 > Buckets > abesharecloud

abesharecloud Info

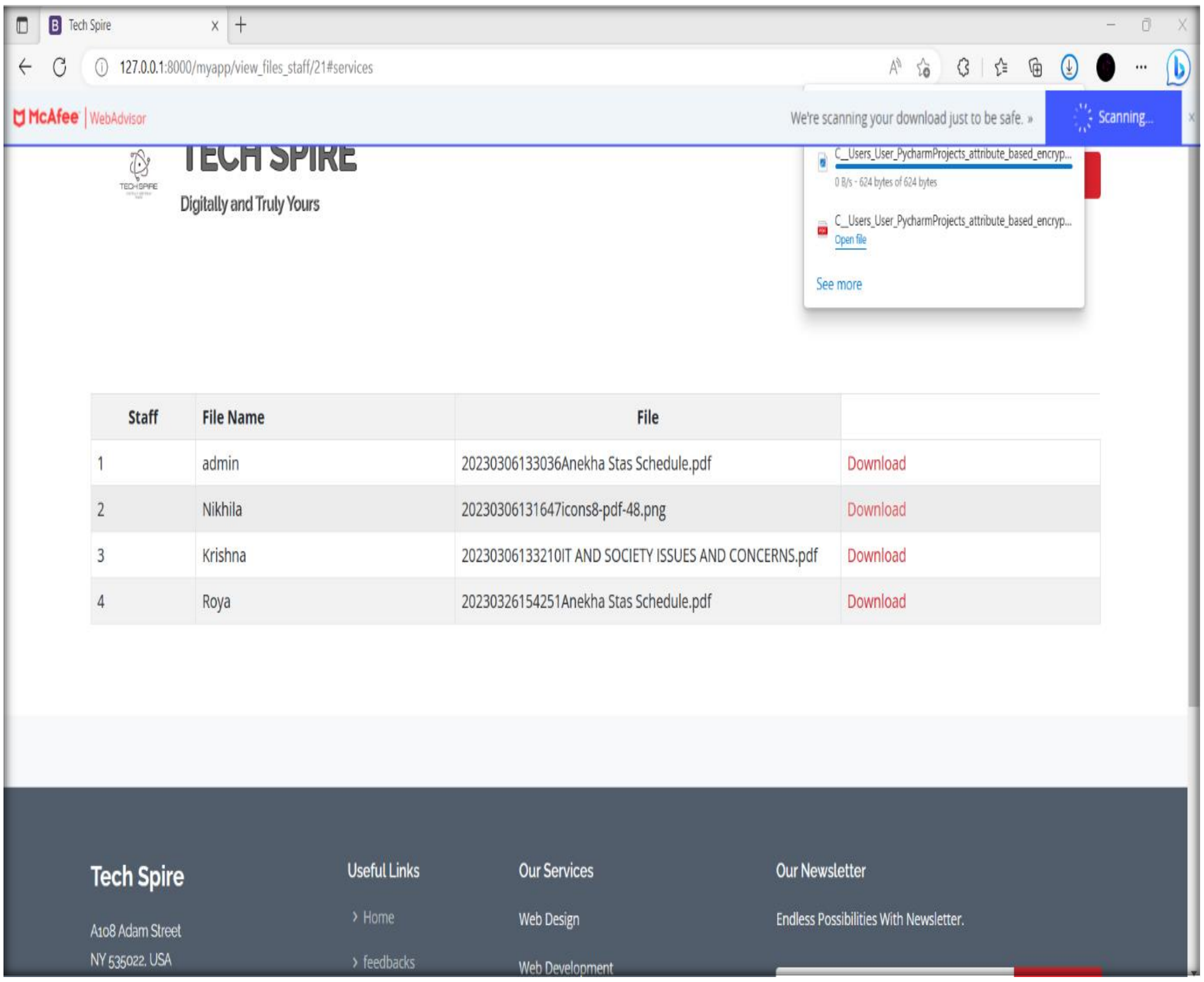
Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (22)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
20230217172342.jpg	jpg	March 4, 2023, 18:03:43 (UTC+05:30)	560.0 KB	Standard
20230304180419intel.jpg	jpg	March 4, 2023, 18:04:21 (UTC+05:30)	14.3 MB	Standard
20230305113142intel.jpg	jpg	March 5, 2023, 11:31:54 (UTC+05:30)	14.3 MB	Standard
20230305113707PROGRAM CHART.pdf	pdf	March 5, 2023, 11:37:15	266.8	Standard

The uploaded file is stored in the cloud AmazonS3.


Tech Spire
127.0.0.1:8000/myapp/view_files_staff/21#services
McAfee | WebAdvisor
We're scanning your download just to be safe. »
Scanning...
C_Users_User_PycharmProjects_attribute_based_encryp...
0 B/s - 624 bytes of 624 bytes
C_Users_User_PycharmProjects_attribute_based_encryp...
Open file
See more
TECH SPIRE
Digitally and Truly Yours
Tech Spire
A108 Adam Street
NY 535022, USA
Useful Links
Home
feedbacks
Our Services
Web Design
Web Development
Our Newsletter
Endless Possibilities With Newsletter.

Staff	File Name	File	
1	admin	20230306133036Anekha Stas Schedule.pdf	Download
2	Nikhila	20230306131647icons8-pdf-48.png	Download
3	Krishna	20230306133210IT AND SOCIETY ISSUES AND CONCERNS.pdf	Download
4	Roya	20230326154251Anekha Stas Schedule.pdf	Download

The file uploaded by one staff is being downloaded by another staff in the same group .

6.4 PACKAGES USED

PyAes

A pure-Python implementation of the AES (FIPS-197) block-cipher algorithm and common modes of operation (CBC, CFB, CTR, ECB, OFB) with no dependencies beyond standard Python libraries.

Pycryptodome

PyCryptodome is a self-contained Python package of low-level cryptographic primitive, It supports Python 2.7, Python 3.5 and newer. The installation procedure depends on the package you want the library to be in. All modules are installed under the Crypto package. PyCryptodome is not a wrapper to a separate C library like OpenSSL. To the largest possible extent, algorithms are implemented in pure Python. Only the pieces that are extremely critical to performance (e.g. block ciphers) are implemented as C extensions.

Ecdsa

This is an easy-to-use implementation of ECC (Elliptic Curve Cryptography) with support for ECDSA (Elliptic Curve Digital Signature Algorithm), EdDSA (Edwards-curve Digital Signature Algorithm) and ECDH (Elliptic Curve Diffie-Hellman), implemented purely in Python, released under the MIT license. With this library, you can quickly create key pairs (signing key and verifying key), sign messages, and verify the signatures. You can also agree on a shared secret key based on exchanged public keys. The keys and signatures are very short, making them easy to handle and incorporate into other protocols.

Boto3

Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python, which allows Python developers to write software that makes use of services like Amazon S3 and Amazon EC2. You can find the latest, most up to date, documentation at our doc site,

including a list of services that are supported. Boto3 is maintained and published by Amazon Web Services. It allows you to directly create, update, and delete AWS resources from your Python scripts.

Base 64

The base64 module have functions which helps to encode the text or binary data into base64 format and decode the base64 data into text or binary data. The base64 module is used to encode and decode the data

6.5 CODE EXPLANATION

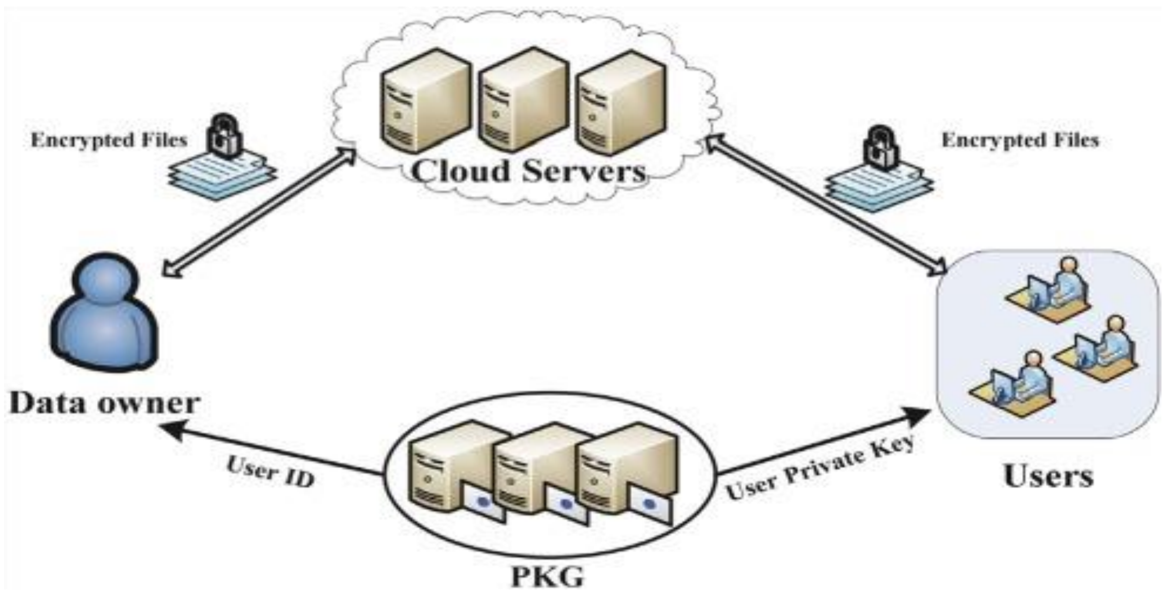
At the time of group creation a signing key and a verification key(it is generated from the signing key) is generated for each using using the attributes of the both user and the group using **ecdsa.** After the generation of both keys only the signing key is stored in the database and the verification key is not stored to improve the security and to prevent the stealing of the verification key by an intruder or outsider.

At the time of file upload by any user to their respected group the file being encrypted using **pyAes** . At the same time , a hash is generated using the ecdsa signing key by making use of **SHA-1** and the generated hash is stored in the database for further use. The uploaded file encrypted using the verification key .Since , the hash cannot be reversed the security concerns are also covered. Also, the whole encrypted file is pushed to the cloud **amazon S3** using **boto3.**

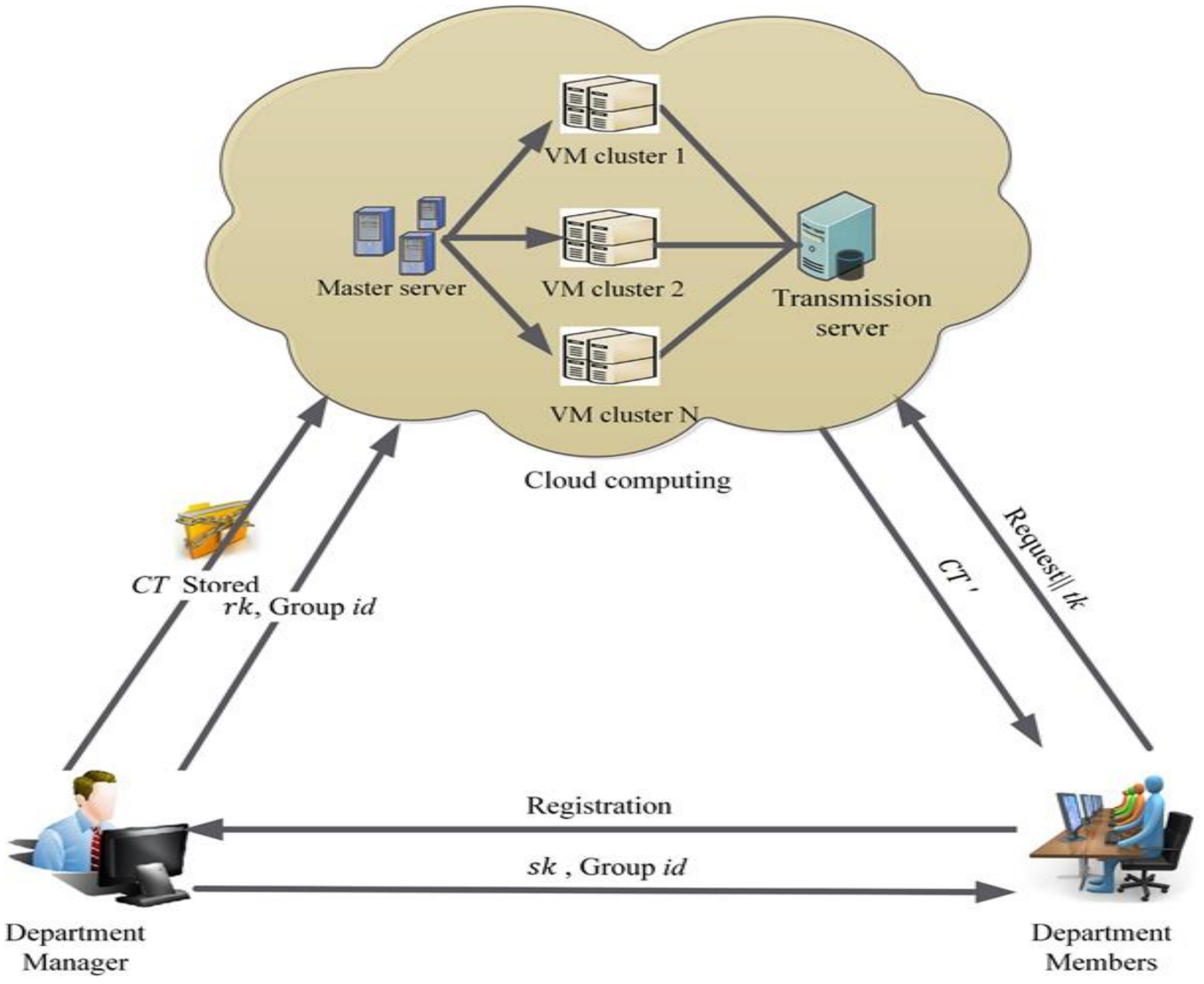
During the file download time, the verification key is generated form the signing key and the hash for the respective signing key is also generated . If the currently generated hash is same as the stored hash value the whole encrypted file is retrived from the cloud using **boto3**. The file is only retrieved if the hashes are matched otherwise it is not. Then the retrieved file is decrypted using the verification key that is generated using the signing key.

AmazonS3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements.



SYSTEM MODEL


VM cluster 1
Master server
VM cluster 2
Transmission server
VM cluster N
Cloud computing
CT Stored
rk, Group id
CT '
Request|| tk
Registration
sk , Group id
Department Manager
Department Members

CHAPTER 7: SUMMARY AND CONCLUSIONS

7.1 COMPARISON BETWEEN ABE AND OTHER COMMONLY USED ENCRYPTION MECHANISM

FACTORS	ABE	OTHER ENCRYPTION TECHNIQUES
Access Control	ABE allows for more fine-grained access control In ABE, access to encrypted data is determined by attributes, such as age, job title, or nationality	Traditional encryption, on the other hand, typically requires a single key or password to decrypt the data.
Complexity	ABE is more complex and it requires additional steps to encrypt and decrypt data. Also, key management is complex.	Not complex and simple key management and encryption and decryption.
Flexibility	ABE is more flexible allows for more flexible access control policies.	It is not flexible.

Performance	ABE is typically slower due to its complexity. Requires more processing power and energy.	It is basically faster and requires minimal amount of processing time and memory
Applications	ABE is well-suited for applications that require fine-grained access control, such as healthcare, finance, and government.	Traditional encryption techniques are more commonly used in applications where access control is less of a concern, such as file sharing and email.

7.2 APPLICATIONS

Attribute-based encryption (ABE) has numerous applications in a variety of industries and domains. ABE can be used in healthcare to protect patient data by allowing only authorized healthcare providers with the necessary attributes to access patient records. In cloud computing, ABE can be used to provide secure access to cloud storage by allowing users with specific attributes, such as job roles or departments, to access specific files and data. In the context of the Internet of Things (IoT), ABE can be used to secure communication between IoT devices by allowing only devices with specific attributes to communicate with each other. In financial services, ABE can be used to protect financial data by allowing only authorized personnel with the necessary attributes to access sensitive data. Similarly, government agencies can use ABE to control access to sensitive data by allowing only personnel with specific security clearances to access it. Overall, ABE can be a valuable tool in any scenario where fine-grained access control based on user attributes is needed, rather than simply granting access based on a user's identity.

Attribute-based encryption (ABE) is a type of encryption that allows access control to be enforced based on attributes or properties of users, rather than on specific identities or keys. The following are some of the applications of attribute-based encryption:

Healthcare: Attribute-based encryption can be used to protect patient data by allowing only authorized healthcare professionals with the required attributes to access the data.

Cloud computing: ABE can be used to provide secure access to cloud storage by allowing users with specific attributes, such as job roles or departments, to access specific files and data.

IoT (Internet of Things): ABE can be used to secure communication between IoT devices by allowing only devices with specific attributes to communicate with each other.

Financial services: ABE can be used to secure financial data by allowing only authorized personnel with the required attributes to access sensitive data.

Government agencies: ABE can be used by government agencies to control access to sensitive data by allowing only personnel with specific security clearances to access it.

Overall, attribute-based encryption can be useful in any scenario where there is a need for fine-grained access control based on user attributes, rather than simply granting access based on a user's identity.

7.3 FUTURE SCOPE

Attribute-based encryption (ABE) is an emerging field of research that holds significant promise for the future of data security and privacy. As data continues to grow in volume and complexity, there is an increasing need for fine-grained access control and data confidentiality. ABE offers a powerful solution to this problem, allowing data to be encrypted and decrypted based on user attributes, such as roles or permissions, rather than on individual identities. This approach offers several advantages over traditional encryption schemes, including improved scalability, fine-grained access control, and enhanced privacy.

The future scope of ABE is broad and includes a wide range of potential applications in various domains. For example, ABE can be used to secure communication and data exchange in cloud computing environments, IoT networks, and healthcare systems. It can also be used to provide fine-grained access control to data used for data analytics and machine learning, ensuring that only authorized personnel can access sensitive information. Additionally, ABE has the potential to play a critical role in preserving privacy by allowing users to share data with only those who have the necessary attributes or permissions, rather than sharing data with all users.

As the field of ABE continues to evolve, there are several key areas of future development that are likely to have a significant impact. These include improving key management and scalability, developing more efficient and secure ABE schemes, and exploring the potential applications of ABE in emerging technologies, such as quantum computing. With these developments, ABE is likely to play an increasingly important role in securing data and communication in a wide range of domains in the coming years.

7.4 CHALLENGES OF ATTRIBUTE BASED ENCRYPTION

Attribute-based encryption (ABE) is a type of encryption scheme that uses user attributes, such as roles or permissions, as a basis for encrypting and decrypting data. While ABE has several advantages, including fine-grained access control and data confidentiality, it also faces several challenges, including:

Key Management: ABE requires the use of a master key to encrypt data, and this key must be kept secure to prevent unauthorized access. Key management can be complex and challenging, especially in large organizations or systems where multiple keys are used.

Scalability: ABE can become complex and inefficient when used with large numbers of users or attributes, leading to scalability issues that can impact system performance.

Revocation: ABE does not provide an easy mechanism for revoking access to data once it has been shared. This can be a problem if users with access to sensitive data leave the organization or if their attributes change.

Security: ABE implementations can be vulnerable to attacks, such as key recovery attacks or collusion attacks, which can compromise the security of the encryption scheme.

Interoperability: ABE schemes can be difficult to implement across different platforms or systems, leading to interoperability issues that can limit their effectiveness.

7.5 CONCLUSION

A fresh ABE cryptosystem named is proposed to provide the property of hidden access policy and the corresponding ciphertext can simply be accessed by users whose identities are listed in a specified receiver set and simultaneously can correctly fulfill the predefined access policy. With our proposed system, each data sharer is featured with a group identity and a group of attributes, and the shared data are encrypted according to the specified access policy and a set of group identities. In this way, only the users belonging to the specified group and matching the desired access policies can decipher the shared data. Moreover, access policy with the ciphertext will not be disclosed by anyone, not even the legitimate data receivers. Finally, the rigorous security proof in the standard model and extensive simulations demonstrate that our data sharing system is selective IND-CPA secure and practical. There are several interesting open research problems. In our construction, the size of the secret key and ciphertext increases linearly with the number of wildcards in the associated access structure. One natural open problem is to design this scheme with constant-size secret key and ciphertext size. By considering that the attributes and group identities are shared by multiple users, a malicious user may intentionally disclose his/her decryption privilege to gain financial profit or other reasons without being detected. Therefore, it is also interesting to identify/trace the malicious user efficiently. In addition, we proved only selective security of our construction and thus achieving adaptive-security in this construction is another direct open problem. allows a user to share data with others within the group without revealing data and identity privacy to the cloud. Additionally, it supports efficient user revocation and new user joining. More specifically, efficient user revocation can be achieved through a public revocati on list without updating the private keys of remaining users and new users can directly decrypt files from the cloud before their participation.

REFERENCES

[1]. P. Junod and A. Karlov, "An efficient publickey attribute-based broadcast encryption scheme allowing arbitrary access policies in Tenth annual ACM workshop on digital rights management. ACM, 2010, pp. 13–24.

[2]. Lam, S.S-zebeni, and L.Buttyan, "Invitationoriented: Key management for Dynamic groups in an asynchronous communication model," Submitted to 4th International Workshop on Security in CloudComputing, 2012.

[3]. N. Cao, S. Yu, Z. Yang, W. Lou, and Y. T. Hou, "LT Codes-based Secure and Reliable Cloud Storage Service," in the Proceedings of IEEE INFOCOM 2012, 2012, pp. 693–701.

[4]. B. Wang, B. Li, and H. Li, "Knox: PrivacyPreserving Auditing for Shared Data with large Groups in the Cloud," in the Proceedings of ACNS 2012, June 2012,

[5]. B. Libert and D. Vergnaud, "Unidirectional chosen-ciphertext secure proxy re-encryption," Information Theory, IEEE Transactions on, vol. 57, no. 3, pp. 1786–1802, march 2011.

[6]. S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving Secure, Scalable, and Fine-Grained Data Access Control in Cloud Computing," Proc. IEEE INFOCOM, pp. 534-542, 2010.

[7]. Mahesh, Bhasutkar, Maninti Venkateswarlu, and M.Raghavendra. "End-to-end congestion control techniques for Router." 2011 International Conference on Communication Systems and Network Technologies. IEEE, 2011

[8]. Mahesh, B., and K. Shyam Sunder Reddy. "Router Aided Congestion ControlTechniques." Second International Conference on Information Systems and Technology.

[9]. Mahesh, B. "Dynamic Update and Public Auditing with Dispute Arbitration for Cloud Data." Journal of Advanced Database Management & Systems 4.3 (2017): 14-19.

[10]. Mahesh, B., et al. "A Review on Data Deduplication Techniques in Cloud." Embedded Systems and Artificial Intelligence. Springer, Singapore, 2020. 825-833.Chin-Su Ko, K.Kim,

R.Hwang, Y. Kim and S.Rhee, "Robust Audio Watermarking in wavelet domain using PN sequences", Proc. of ICIS2005 published by IEEE. 120

[11]. Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Dynamic Audit Services for Integrity Verification of Outsourced Storage in Clouds," in the Proceedings of ACM SAC 2011, 2011, pp. 1550–1557.

[12]. H. Abu-Libdeh, L. Prince-house and H. Weather-spoon, RACS: a case for cloud storage diversity, ACM, 2010, pp. 229-240.

[13]. Taka-bi , H.; Joshi, J.B.D.; Ahn, G.; , "Security and Privacy Challenges in Cloud Computing," Security& Privacy, IEEE, vol.8, no.6, pp.24-31, Nov-Dec.2010. doi:10.1109/MSP.2010.186.

[14]. Kamara, Seny and Lauter, Kristin, Cryptographic cloud storage, FC'10 Proceedings of the 14th international conference on Financialcryptograpy and data security, pp.136-149, 2010