

PROFESSIONAL TRAINING REPORT

Sathyabama Institute of Science and Technology

(Deemed to be University)

Submitted in partial fulfilment of the requirements for the award
of Bachelor of Technology Degree in Information Technology

By

**Pavithraa S
(40120065)**



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI, TAMILNADU – 600119

April 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

(www.sathyabama.com)

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of
PAVITHRAA S who carried out the project as a "Providing data
security to cinder block volumes using encryption" under our
supervision from January 2023 to April 2024.

Internal Guide

Ms. J Merlin Mary Jenitha M. Tech

Head of the Department

Dr. R Subhashini, M.E., Ph.D.

Internal Examiner

External Examiner

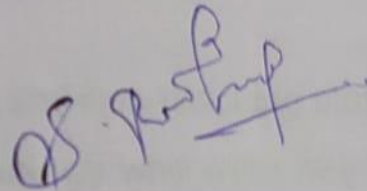


REDMI NOTE 8
AI QUAD CAMERA

DECLARATION

I, **Pavithraa S (40120065)** hereby declare that the Project Report entitled "**Providing data security to cinder block volumes using encryption**" is done by me under the guidance of **Ms . J Merlin Mary Jenitha M. Tech** and is submitted in partial fulfilment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 28/4/23



PLACE: Chennai

SIGNATURE OF THE CANDIDATE



ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T Sasikala, M.E., Ph.D.**, Dean, School of Computing **Dr. R Subhashini, M.E., Ph.D.**, Head of the Department of Information Technology for providing me with necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms. J Merlin Mary Jenitha M. Tech** for her valuable guidance, suggestions and constant encouragement that paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ABSTRACT

Efficient multi-authority attribute-based signcryption (EMABSC) is a cryptographic technique that allows multiple authorities to collaborate in encrypting and signing messages based on certain attributes. In this paper, we propose an improved EMABSC scheme that enhances the security and efficiency of the existing approaches. Our scheme employs a hybrid encryption and signature mechanism to provide confidentiality, integrity, and authenticity of the signed and encrypted messages. It also allows multiple authorities to independently generate partial signatures and encrypt parts of the message based on specific attributes, without revealing any information to each other. To improve the efficiency of the proposed scheme, we utilize a symmetric encryption algorithm in the encryption process, which reduces the computational cost compared to existing asymmetric encryption-based schemes. We also introduce a threshold mechanism that allows for flexible control of the number of authorities needed to decrypt and verify the signature of a message. We conducted a security analysis of the proposed scheme and proved its correctness and security against various attacks. Our performance evaluation results showed that our scheme outperforms existing EMABSC schemes in terms of computational efficiency and communication overhead, making it suitable for use in various applications such as ecommerce, e-government, and cloud computing.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO
	ABSTRACT	v
	LIST OF ABBREVIATIONS	viii
	LIST OF FIGURES	
1.	INTRODUCTION	1
	1.1 Introduction	
	1.2 Securing data with encryption	1
	1.3 Digital Signing and Encryption	2
	1.4 Encryption and digital signing	3
	1.5 Security challenges in cloud computing	3
	1.6 Conceptual differences	4
	1.7 Efficient Scheme	4
2.	LITERATURE SURVEY	6
3.	METHODOLOGY	13
	3.1 Module 1 - Initializing Backend Using Django	13
	3.2.1 Creating Algorithm	13
	3.2 Module 2 - Model Implementation	13
	3.3 Module 3 - Creating Frontend	14
4 .	RESULTS AND DISCUSSION	21
	4.1 Results	21
	4.2 Discussion	21

5.	CONCLUSION	23
	REFERENCES	42
	APPENDIX	
A.	SOURCE CODE	24
B.	OUTPUT SCREENSHOTS	41

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
3.1	SYSTEM ARCHITECTURE	15
3.2	DATA FLOW DIAGRAM 0	16
3.3	DATA FLOW DIAGRAM 1	16
3.4	USE-CASE DIAGRAM	17
3.5	ACTIVITY DIAGRAM	18
3.6	SEQUENCE DIAGRAM	19
3.7	CLASS DIAGRAM	19
3.8	ER DIAGRAM	20
4.1	OUTPUT SCREENSHOT 1	40
4.2	OUTPUT SCREENSHOT 2	40
4.3	OUTPUT SCREENSHOT 3	41
4.4	OUTPUT SCREENSHOT 4	41

LIST OF ACRONYMS AND ABBREVIATIONS

ABBREVIATION	EXPANSION
ABE	Attribute Based Encryption
ABS	Attribute Based Signature
IoT	Internet of things
RBAC	Role-based access control
AMI	Advanced Metering Infrastructure
VCC	Vehicular Cloud Computing
IoV	Internet of Vehicles

CHAPTER – 1

INTRODUCTION

1.1 INTRODUCTION

The encryption technique is a unique process of securing the personal data and information. It is used by social media giants like WhatsApp. The Multi Authority based on signcryption is a huge benefit to the people and also to the industry. Recently, efficient fine-grained access mechanism has been studied as a main concern in cloud storage area for several years. Attribute-based signcryption (ABSC) which is logical combination of attribute-based encryption (ABE) and attribute-based signature (ABS), can provide confidentiality, authenticity for sensitive data and anonymous authentication. At the same time, it is more efficient than previous “encrypt-then-sign” and “sign-then-encrypt” patterns. However, most of the existing ABSC schemes fail to serve for real scenario of multiple authorities and have heavy communication overhead and computing overhead. Hence, we construct a novel ABSC scheme realizing multi-authority access control and constant-size ciphertext that does not depend on the number of attributes or authorities.

Furthermore, our scheme provides public verifiability of the ciphertext and privacy protection for the signcryptor. Specially, it is proven to be secure in the standard model, including ciphertext indistinguishability under adaptive chosen ciphertext attacks and existential unforgeability under adaptive chosen message attack. The propose for a multi-authority attribute-based signcryption scheme with efficient revocation for smart grid downlink communications. In the proposed scheme, grid operators and electricity vendors can send multicast messages securely to different groups of consumers which is required in different applications such as firmware update distribution and sending direct load control messages. Our scheme can ensure the confidentiality and the integrity of the multicasted messages, allows consumers to authenticate the source of the multicasted messages, achieves and non-repudiation property, and allows prompt revocation, simultaneously which are required for the smart grid downlink communications.

1.2 Securing data with encryption

Our security analysis demonstrates that the proposed scheme can thwart various security threats to the smart grid. Our experiments conducted on an Advanced Metering Infrastructure (AMI) testbed confirm that the proposed scheme has low computational

overhead. Threats loom ever present in our digital world, which is why methods of securing data are constantly advancing. Tactics like encryption are used every day on sensitive data-at- rest or in-motion. Encryption is the process of putting data in the form of plaintext into an encryption algorithm, and producing a ciphertext. Ciphertext is a form of data where all the patterns of letters that create words in the plaintext are scrambled into a new text that cannot be read without decrypting the data. Encryption uses a key to ensure the ciphertext cannot be deciphered by anyone but the authorized recipient.

1.3 DIGITAL SIGNING AND ENCRYPTION

Signing of data works to authenticate the sender of the data and tends to implement a form of encryption in its process. The process of signing emails, sensitive data, and other information has become necessary, as it verifies the identity of the sender and ensures the data has not been altered in transit. If a Man in the Middle attack occurred and the data was altered or compromised by the attacker, the recipient of the information would know that this has occurred. The attacker could alter the data, but as they do not have the key used by the sender to sign the data, the recipient of the data will know not to trust the sent data when analyzing the key and data.

The process of digital signing works similarly to encryption. Encryption comes in two types, asymmetric and symmetric encryption. The process of asymmetric encryption works by creating a key pair with a public and private key. The private key is kept secret from everyone but the creator of the key, while the public key is available to everyone. The data is encrypted with the private key, and decrypted when needed with the public key. Symmetric encryption only uses one key for both encryption and decryption. As asymmetric encryption is more secure than symmetric encryption, it tends to be used more often. When sending data to a recipient, the correct method of encryption is to encrypt the data with the recipient's public key, as this means only the owner of the key pair can decrypt that data. Digital signing works oppositely. The data is signed by hashing the message with a hashing algorithm and the sender's private key. This produces a hash digest, which can only be recreated through use of one of the keys in the key pair created by the sender. The recipient then receives the message, the hash digest, and the public key, if they did not already have it. The recipient then uses the sender's public key to hash the message they have received. If the resulting hash digest matches the hash digest that has been sent along with the message, then the identity of the sender has been

confirmed. This also confirms that the data has not been changed in transit. However, signing alone does not ensure the data has not been intercepted and read.

1.4 ENCRYPTION AND DIGITAL SIGNING

To protect data from compromise and authenticate the sender at the same time, encryption and digital signing are used together. They are also both used in tandem to fulfill compliance standards for companies. Standards, like the Federal Information Processing Standards (FIPS) or the General Data Protection Regulation (GDPR), require companies to protect data as securely as possible along with authenticating data received from others. Encryption and digital signing ensure these standards are reached, and that users can be secure in the knowledge that data that is sent to and from them will not be compromised. Confidential or sensitive data should always be encrypted and signed for its own safety. The use of encryption and signing together ensures that the main goals of cryptography, Confidentiality, Integrity, Authenticity, and Non-Repudiation are all met. Confidentiality and integrity are reached when data is encrypted asymmetrically, as only the intended recipient can decrypt the message. Non-repudiation and authenticity occur due to digital signing. Non-repudiation means that using the technique of digital signing, the sender of any information cannot, in the future, say they did not send the data, as the use of their private key confirms that they sent the data.

1.5 SECURITY CHALLENGES IN CLOUD COMPUTING

Cloud computing represents today's most exciting computing paradigm shift in information technology. However, security and privacy are perceived as primary obstacles to its wide adoption. Here, the authors outline several critical security challenges and motivate further investigation of security solutions for a trustworthy public cloud environment.

An Internet of Vehicles (IoV) allows forming a self-organized network and broadcasting messages for the vehicles on roads. However, as the data are transmitted in an insecure network, it is essential to use an authentication mechanism to protect the privacy of vehicle users. Recently, Ying et al. proposed an authentication protocol for IoV and claimed that the protocol could resist various attacks. Unfortunately, we discovered that their protocol suffered from an offline identity guessing attack, location spoofing attack, and replay attack, and consumed a considerable amount of time for authentication.

To resolve these shortcomings, we propose an improved protocol. In addition, we provide a formal proof to the proposed protocol to demonstrate that our protocol is indeed secure. Compared with previous methods, the proposed protocol performs better in terms of security and performance.

In several distributed systems a user should only be able to access data if a user possesses a certain set of credentials or attributes. Currently, the only method for enforcing such policies is to employ a trusted server to store the data and mediate access control. However, if any server storing the data is compromised, then the confidentiality of the data will be compromised. In this paper we present a system for realizing complex access control on encrypted data that we call ciphertext-policy attribute-based encryption. By using our techniques encrypted data can be kept confidential even if the storage server is untrusted; moreover, our methods are secure against collusion attacks.

1.6 CONCEPTUAL DIFFERENCES

Previous attribute-based encryption systems used attributes to describe the encrypted data and built policies into user's keys; while in our system attributes are used to describe a user's credentials, and a party encrypting data determines a policy for who can decrypt. Thus, our methods are conceptually closer to traditional access control methods such as role-based access control (RBAC). In addition, we provide an implementation of our system and give performance measurements. The right to privacy is a very essential service should be provided to all the users across the spectrum of internet. Safety and Security is at forefront of delivering a good product and attract the users towards the product. Signature with encryption technique is mostly been used the smart service providers, which in turn has created a robust structure for the system.

1.7 EFFICIENT SCHEME

Our Results We resolve this problem in the affirmative. We give an efficient scheme for multiauthority attribute-based encryption. We allow the sender to specify for each authority k a set of attributes monitored by that authority and a number d_k so that the message can be decrypted only by a user who has at least d_k of the given attributes from every authority. We allow any number of attribute authorities to be corrupted, and guarantee the security of encryption as long as the required attributes cannot be obtained exclusively from those authorities and the trusted authority remains honest.

We also provide several extensions to our basic multiauthority scheme. We describe techniques to allow the encryptor to determine for each ciphertext how many attributes to require from each authority. We also describe a variant of our scheme in which the encryptor can specify a number D such that a user can decrypt if he has sufficient numbers of the given attributes from at least D authorities. It is this variant that would be used to implement the RI example above. In this example, we have 3 authorities, and the ciphertext will include 1 attribute from each. However, we only want to require that a user must have satisfactory attributes from 2 out of the 3 authorities in order to decrypt.

CHAPTER – 2

LITERATURE SURVEY

- [1] L. Wang, Z. Guan, Z. Chen and M. Hu, "Multi-receiver signcryption scheme with multiple key generation centers through public channel in edge computing," in China Communications, vol. 19, no. 4, pp. 177-198, April 2022**

The Internet of Things has been playing an important role in our daily life. A multi- receiver signcryption scheme is promising to protect the transmission data when an edge device broadcasts its sensing data to many different end devices at once. The system private key of those devices is currently kept secret by a single point of failure, which may compromise the whole system. For this, we propose a system private key generation center that allows KGC to send secrets through the public channel without the secure channel.

The emerging edge computing technology for the Internet of Things has been playing an important role in our daily life. It is promising to utilize a multi-receiver signcryption scheme to protect the transmission data when an edge device broadcasts its sensing data to many different end devices at a time. There are several things to consider when we design a signcryption scheme. First existing schemes need to maintain a secure channel to generate the user private key, which may increase economic costs. Second the system private key of those schemes is kept secret by a single key generation center (KGC), and the single point of failure of KGC may compromise the whole system. For this, we propose a multi-receiver multi- message signcryption scheme without the secure channel. Firstly the scheme allows KGC to send secrets through the public channel, which reduces maintenance costs. Secondly, to eliminate the single point of failure, the scheme utilizes multiple KGCs to manage the system private key, and updates the secret of each KGC periodically to resist advanced persistent threat attacks. We demonstrate that the proposed scheme can achieve expected security properties. Performance analysis shows that it is with shorter ciphertext length and higher efficiency.

- [2] Y. Zhao, A. Ruan, G. Dan, J. Huang and Y. Ding, "Efficient Multi-Authority Attribute-based Signcryption with Constant-Size Ciphertext," 2021 IEEE Conference on Dependable and Secure Computing (DSC), 2021**

An efficient fine-grained access mechanism has been studied as a main concern in cloud storage area for several years. Attribute-based signcryption (ABSC) which is logical combination of attribute-based encryption (ABE) and attribute-based signature (ABS), can provide confidentiality, authenticity for sensitive data and anonymous authentication. Specially, it is proven to be secure in the standard model, including ciphertext indistinguishability under adaptive chosen ciphertext attacks and existential unforgeability under adaptive chosen message attack.

Attribute-based signature is a very attractive cryptographic primitive [1]. ABS is divided into key-policy ABS (KP-ABS) and signature-policy ABS (SP-ABS). The former KP-ABS uses the access policy (structure) to generate the signing key, and the message can be signed only when its attribute set satisfies the access policy. The latter SP-ABS is the opposite. The signer possesses the signing key corresponding to his attributes and the message with an access policy. In the signature generation stage, a valid signature can be generated if and only if the attributes of the signer satisfy the access policy. In the signature verification phase, the verifier can only ensure that the signer's attributes satisfy the message's access policy but cannot distinguish the signer's identity. These are the unforgeability and privacy (anonymity) of ABS. Since ABS has fine-grained access control, anonymous authentication, privacy protection, and other good properties, it finds broad applications in many fields, such as private access control and anonymous credential.

[3] L. Shuquan and Z. Huiqi, "Online/Offline Attribute-Based Encryption with Multi-Authority Access Control," 2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2021

Fine-grained access control may be implemented successfully using the CP-ABE access control technique. It has been shown that the technique, particularly for users of mobile devices, may significantly cut processing time. The majority of the intricate exponential computations will be completed by the cloud service provider, allowing mobile devices to decode the ciphertext. The plan also enables multi-attribute authority to effectively counter the conspiracy assaults. When the device is not in use, the offline encryption will be completed, making encryption quicker and more effective. The solution is suggested in this study as a decentralised CP-ABE access control mechanism with offline/online encryption and outsourced decryption. The offline encryption will be accomplished when the device is

not in use, which makes encryption faster and more efficient. In the decryption stage, the user will generate the transformed key and hand it over to the honest but curious cloud service provider for decryption preprocessing.

The cloud service provider will complete most of the complex exponential calculations, so that the ciphertext can be decrypted quickly and safely by mobile devices. In addition, the scheme supports multi-attribute authority so that the conspiracy attacks can be effectively resisted. It is shown that the scheme can greatly reduce the computing time especially for mobile device users.

- [4] **Y. Ming, B. He and C. Wang, "Efficient Revocable Multi-Authority Attribute-Based Encryption for Cloud Storage," in IEEE Access, vol. 9, pp. 42593-42603, 2021**

Multi-authority attribute-based encryption (MA-ABE) is usually adopted for cloud storage. It provides fine-grained access control and confidentiality of data. But only one attribute authority can assign the users' attributes, enabling the data to be shared only within the management domain of the attribute authority. The proposed scheme gets its advantages in that it is more economical in computation and storage. It can not only satisfy the need for the fine-grained access control and confidentiality of data, but also make the data shared among different multiple attribute authorities.

These techniques are combined to form Attribute Based Signcryption (ABSC). However, most existing solutions are impractical for real-world scenarios due to the involvement of multiple third-party services and high computation and communication overhead. Therefore, we propose a novel ABSC technique that provides privacy protection and public verifiability for the signcryptor and ciphertext, respectively. Our approach aims to enhance the security of cloud storage systems by addressing the limitations of existing solutions.

- [5] **Y. Li et al., "SDABS: A Flexible and Efficient Multi-Authority Hybrid Attribute-Based Signature Scheme in Edge Environment," in IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 3, pp. 1892-1906, March 2021**

The explosive growth of the Internet of Things and modern networking technologies lay the foundation for the development of intelligent transportation systems and smart cities. To

analyze massive data under the required time for transportation issues, the edge computing paradigm is applied. However, data reliability and security are still facing many challenges in the edge environment. In this article, we propose a multi-authority hybrid attribute-based signature scheme (SDABS).

The multi-layer policy tree is applied to support flexible and various access policies, which also naturally form user groups and help data searching.

Besides, the multi-authority structure is more suitable for the distributed edge environment. We evaluate SDABS from both theoretical analysis and practical analysis. Compared with two classical signature schemes (MABS and ODMA-ABS), experimental results demonstrate that the proposed SDABS can achieve better performance at an acceptable cost in the terms of attributes and attribute authorities.

- [6] **K. Huang, "Revocable Large Universe Decentralized Multi-Authority Attribute-Based Encryption Without Key Abuse for Cloud-Aided IoT," in IEEE Access, 2021.**

MA-ABE is considered a promising technique to protect data confidentiality and achieve finegrained access control for large-scale cross-domain applications. Data confidentiality and access control are the key technologies of secure Internet of things (IoT). Much research is devoted to using audit technologies and trace technologies to determine who should be responsible for the misused key. But they can't solve the key abuse problem, since users still can leak the key and the leaked keys can still decrypt the ciphertext correctly. In this article, the author proposes the first revocable large universe multi-authority attribute-based encryption without key abuse. The data user is unable to generate the available key different from her/his legal key by using her/his legal key. Neither CSP nor authority can generate the available decryption key or decrypt the ciphertext (even if the access policy is satisfied by the attributes it controls) using the keys it controls. An efficient user-attribute revocation mechanism is given and only a few operations are needed when decryption in the proposed scheme. The performance analysis results indicate that the proposed scheme is more efficient and suitable for the IoT.

- [7] **X. Liu, W. Chen, Y. Xia and R. Shen, "TRAMS: A Secure Vehicular Crowdsensing Scheme Based on Multi-Authority Attribute-Based Signature," in IEEE Transactions on Intelligent Transportation Systems 2020.**

How to achieve on-demand message authentication and provide privacy protection of sensing vehicles are challenging in accurate sensing tasks. We propose a secure vehicular crowdsensing scheme based on multi-authority attribute-based signature (TRAMS), which allows the publisher to flexibly customize a fine-grained policy that the potential participants

Also, we propose a key management scheme, which can improve vehicle-based sensing efficiency in the Internet of Vehicles.

With the growing number of vehicles and popularity of various services in vehicular cloud computing (VCC), message exchanging among vehicles under traffic conditions and in emergency situations is one of the most pressing demands, and has attracted significant attention. However, it is an important challenge to authenticate the legitimate sources of broadcast messages and achieve fine-grained message access control. In this work, we propose SmartVeh, a secure and efficient message access control and authentication scheme in VCC.

- [8] **J. Priyanka, K. R. Rajeshwari and M. Ramakrishnan, "Operative Access Regulator for Attribute Based Generalized Signcryption Using Rough Set Theory," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2020.**

The personal health record has been shared and preserved easily with cloud core storage. The PHR has been stored under the access priorities of Signature only, encryption only and signcryption only mode. The proposed ABGS performance fulfills the secrecy, authentication and also other security principles. The Generalized Signcryption defined priority wise access to diminish the cost and increase the effectiveness of the proposed model.

As a result, in IoT contexts, information between IoT devices and end-users must be communicated safely; thus, security is critical. The Internet of Things (IoT) requires a secure access-control and key agreement mechanism. The two points of "node-authentication" and

"key agreement" are used to facilitate safe communication between nearby detecting devices... As a result, Internet of Things solutions must safeguard two critical aspects of their operations: personal data and identity management. The processing power, memory, and energy consumption of Internet of Things devices, on the other hand, are constrained.

[9] X. Yang, R. Liu, G. Chen, M. Wang and C. Wang, "Security analysis of a certificateless signcryption mechanism without bilinear mapping," 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2020.

In addition to provide security features like message integrity, non-repudiation, and secrecy, certificateless signcryption may also address the issues of public key certificate management and key escrow.

A certificateless signcryption process without bilinear mapping was presented by Zhou et al., and its security was shown using the random oracle model's computational Diffie Hellman problem and discrete logarithm issue.

Analysis, however, reveals security weaknesses in this technique. In other words, attackers are able to fake authentic signatures on any communication. We then provide the precise assault procedure.

However, by giving concrete attacks about confidentiality and unforgeability, this study finds that the certificateless signcryption scheme proposed is not secure. Thus, based on Computational Diffie-Hellman (CDH) and Discrete Logarithm (DL), an efficient and security CLS scheme without pairings is proposed in this paper. It is provably secure in the ROM under the CDH and DL assumption. Moreover, compared with other existing CLS schemes in the computational complexity, the scheme without using pairings operation has more efficiency and security

[10] Q. Xu, C. Tan, Z. Fan, W. Zhu, Y. Xiao and F. Cheng, "Secure Multi-Authority Data Access Control Scheme in Cloud Storage System Based on Attribute-Based Signcryption," in IEEE Access, 2018

ABSC is suitable for fine-grained access control in a semi-trusted cloud environment. The attributes of both the signcryptor and the designcryptor can be protected to be known by the authorities and cloud server. The proposed scheme is proven to be secure in the standard model. It has the ability to provide confidentiality, unforgeability, anonymous authentication, and public verifiability. This system could be used in a multi-authority cloud storage system.

Cloud file storage systems are the current trend of enterprises and also of individual users. Due to the malicious or unauthorized users, file sharing among the cloud users has become the biggest challenge in the recent times. Attribute-based signcryption (ABSC) is known as the versatile cryptographic primitive which achieves the fine-grained access control over robust cloud storage. ABSC combines attribute-based encryption (ABE) and attribute-based signatures to achieve privacy-oriented confidentiality along with the authenticity. Unfortunately, most of the present ABE and ABSC schemes leverage heavy computational overheads against the key length, the ciphertext size or the expressive access structures used. In this paper.

CHAPTER – 3

METHODOLOGIES

3.1 MODULE 1: INITIALIZING BACKEND USING DJANGO

Following the completion of the project's basic setup and the addition of the first app, you may use the name utility to create a new directory or add a new python package to your app. Create a duplicate of the init.py file and transfer it to the utilities folder. Create a new text document in the utilities folder and name it encryption util.py. Build a new feature that encrypts the supplied data. Protect the ENCRYPT KEY. Keep it in the production.py file of settings, and don't add it to git. Due to the possibility of needing to send encoded data via a URL, we are additionally converting the encoded text to a url-safe base64 format. Return null if an error occurred, and log the error message. We can decrypt any encrypted text by just doing the opposite of what was done to encrypt it.

3.2 MODULE 2: MODEL IMPLEMENTATION

3.2.1 CREATING ALGORITHM

The fernet recipe is part of the cryptography package, and it's a method for encrypting and authenticating data that you can use. It is not too difficult to implement, and it removes a significant amount of the element of uncertainty that is inherent in cryptography. A message that has been encrypted and authenticated with fernet cannot be read by an adversary nor can it be tampered with when it has been correctly implemented. In addition to an initialization vector that is selected at random, fernet requires three critical inputs, which are as follows:

- A message in plaintext that was entered by the user. This is the information that the user wishes to have encrypted, and it is presented in the form of a byte string that can be chosen at random.
- A key that was generated by the user and has 256 bits of length.
- The time at the moment.

After being processed by the fernet formula, these inputs will result in the production of a token. This has both the message in an encrypted form as well as an HMAC that authenticates the message (we will explain what an HMAC is further down the page).

Together, they make it impossible to view the message or make any changes to it if the key is not there. We will use a twin fernet layer for this encryption decryption thing. So we will create two fernet cipher pipeline to pass our file from that pipeline and create ciphered file.

Developing a system for file chunking and dechunking: In this step, we will develop a system that, given a given file, will either chunk it or dechunk it before saving it on the server. During the process of decryption, we will combine all of the files and then send the merged set to the pipeline for decryption.

Introducing a Second Element Auth: After generating an authentication token, we will send it to the user's email address using Amazon Simple Email Service (SES). The secondary Django server will have API endpoints to store user data together with the authentication token.

3.3 MODULE 3: CREATING FRONTEND

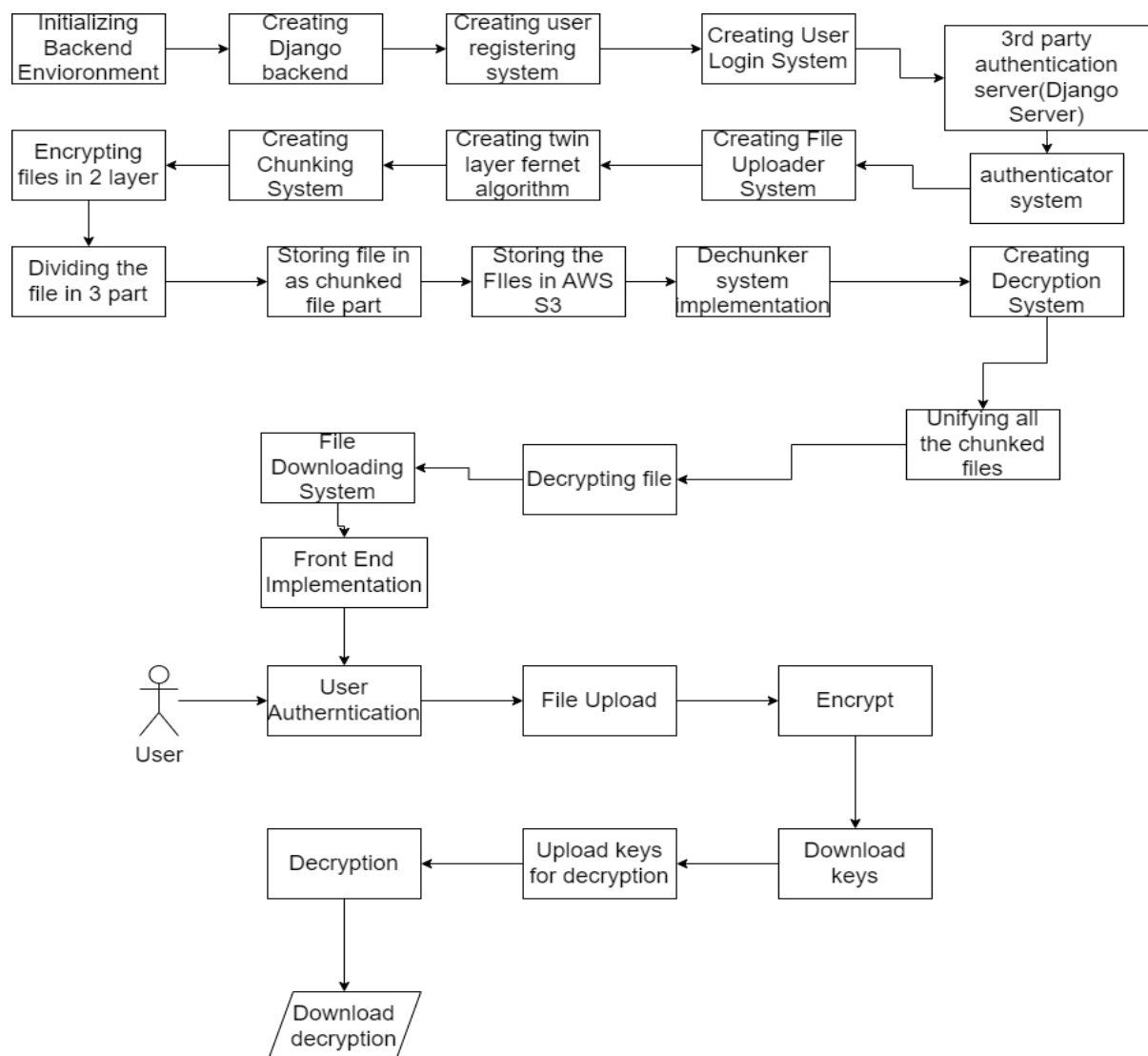
We have designed a web-based application in order to provide the authorised users of the programme with the ability to communicate with one another and share data with one another. The user is required to give their information in a form for Owner Registration, and then as a verification step, he is required to provide an image as a captcha.

The form may be found on the user's dashboard. On the internet, you may find this particular form. After that, the administrator examines the user's profile and, once they are satisfied with the information, provides the user with their credentials. The next thing that needs to be done in the procedure is for the user to upload the file into the system while concurrently providing a title and description that are acceptable.

After that step, the twin layer Fernet method that was utilised in the model is applied to the text files and processed using those files. A response is given to the

acceptance of the user- key request. After a protracted period of time, the user finally acquires the key that is required to download encrypted information in the form of information that is valuable.

A front-end developer creates websites and applications using web languages such as HTML, CSS, and JavaScript that allow users to access and interact with the site or app. When you visit a website, the design elements you see were created by a front-end developer.



Text

Fig 3.1 – Architecture Diagram

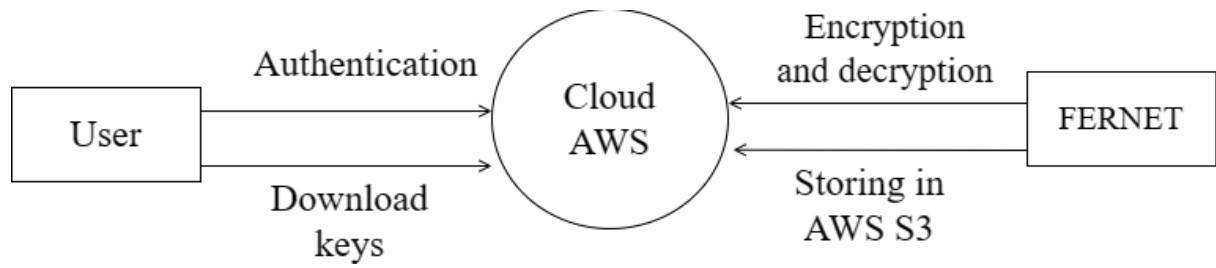


Fig 3.2 – Data Flow Diagram Level 0

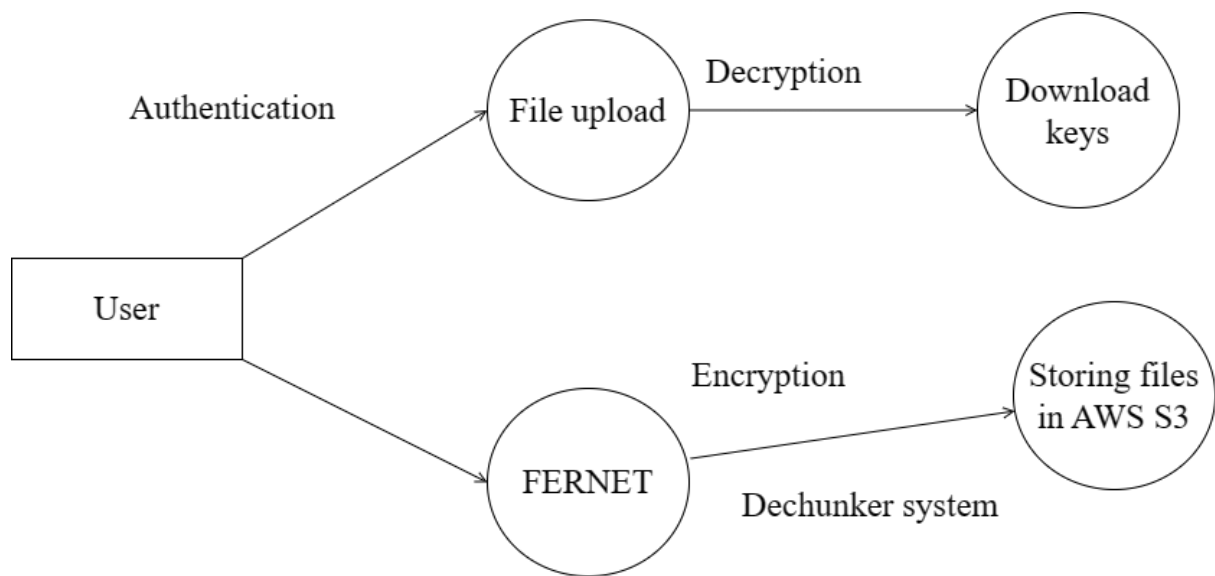


Fig 3.3– Data Flow Diagram Level 1

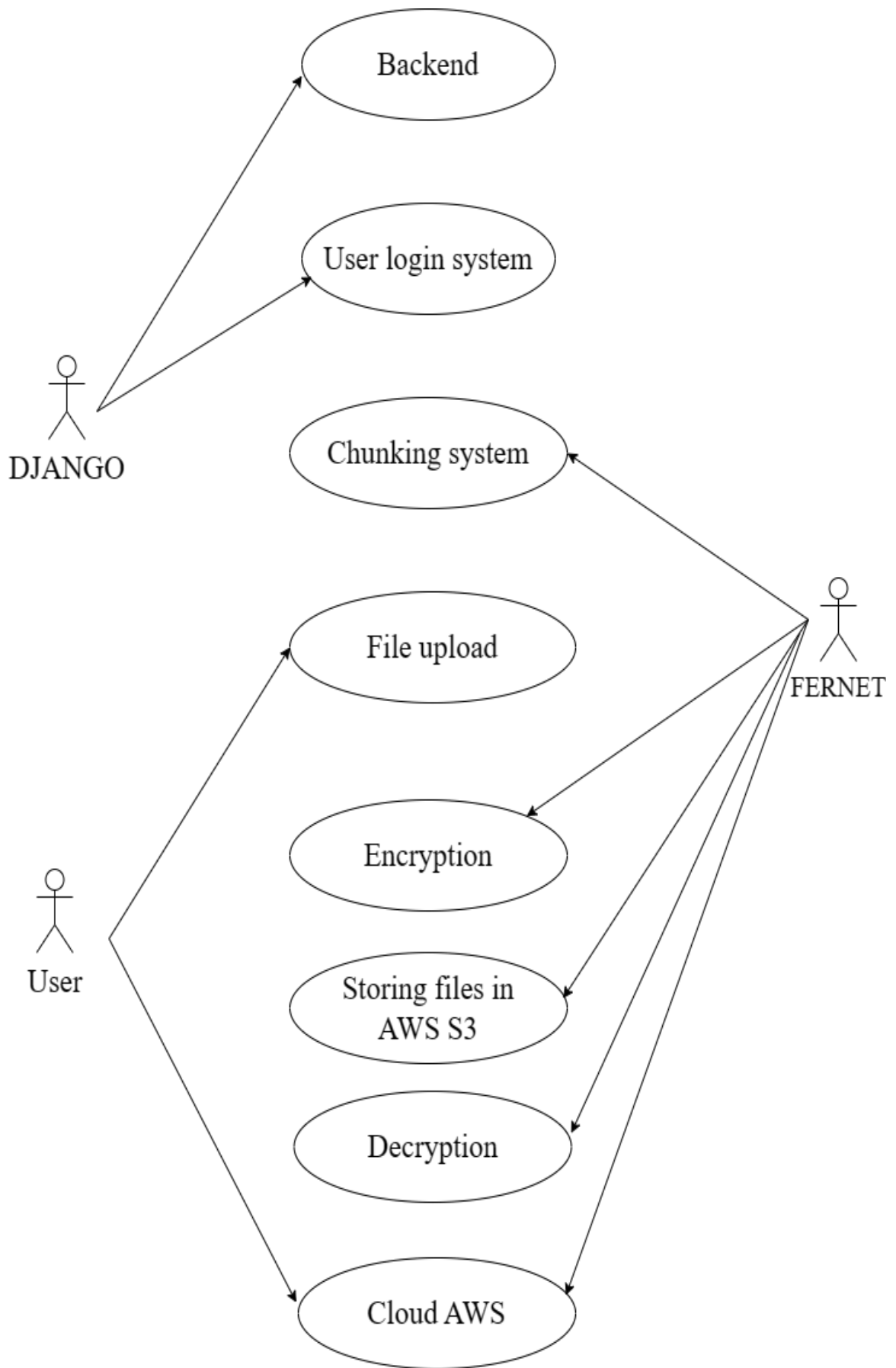


Fig 3.4 – Use-Case Diagram

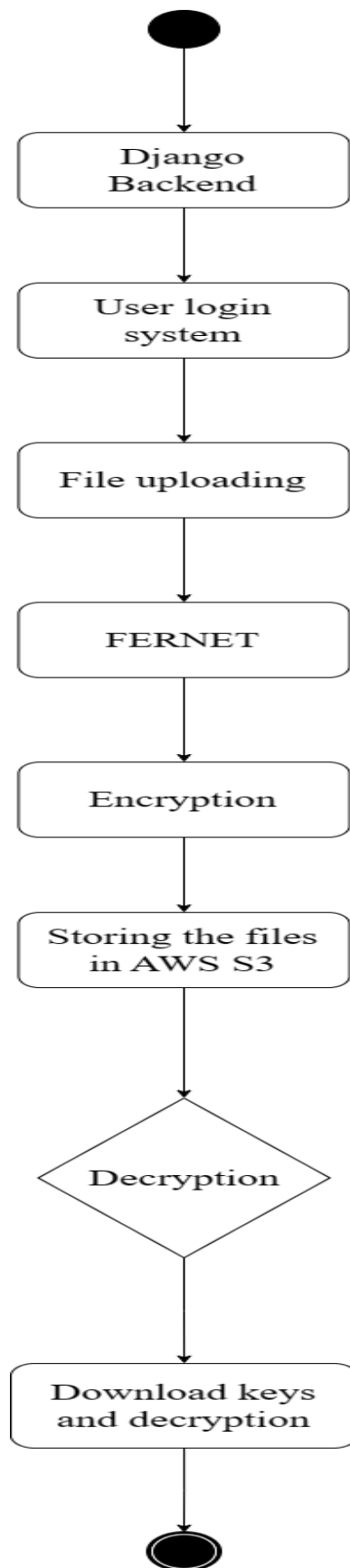


Fig 3.5 – Activity Diagram

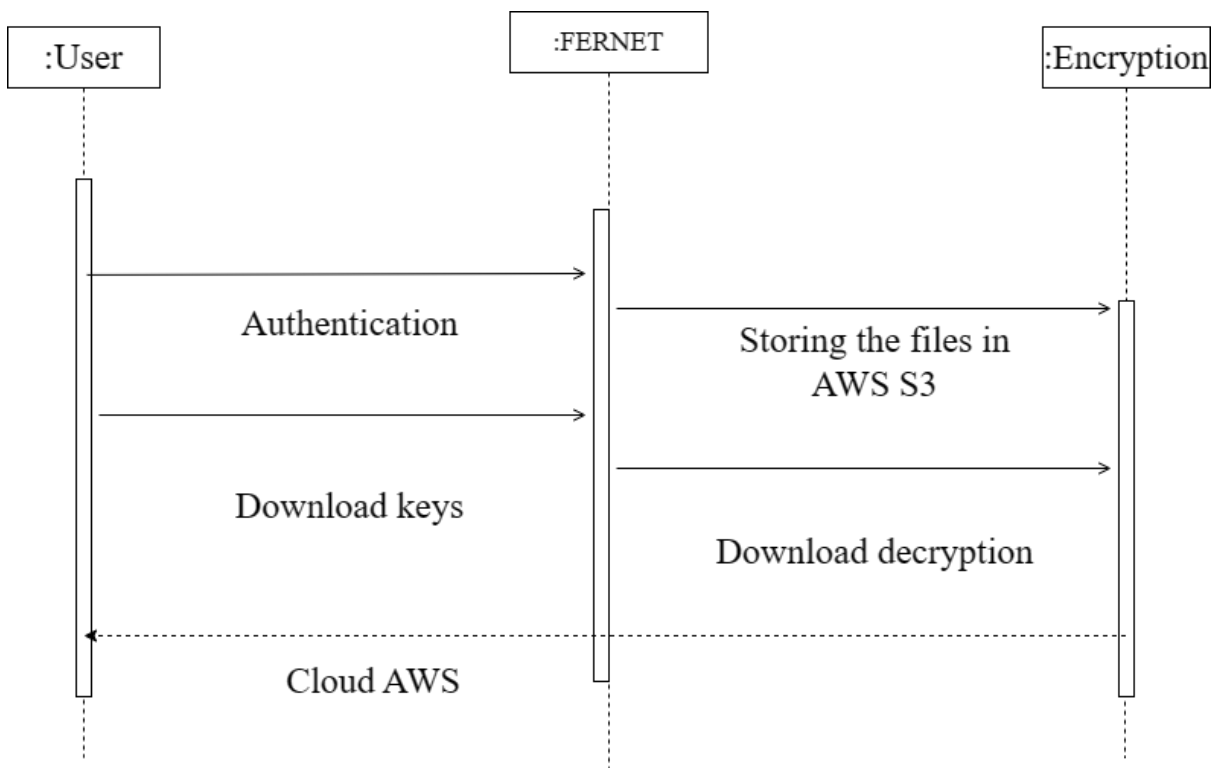


Fig 3.6 – Sequence Diagram

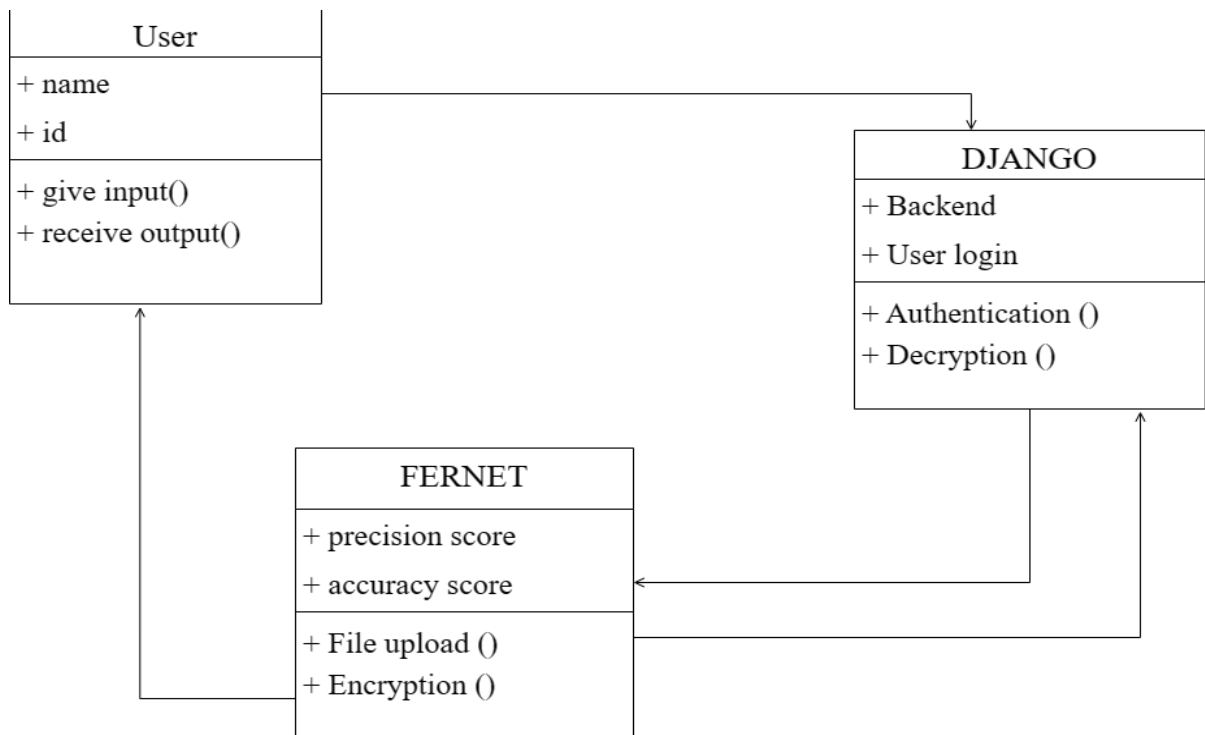


Fig 3.7– Class Diagram

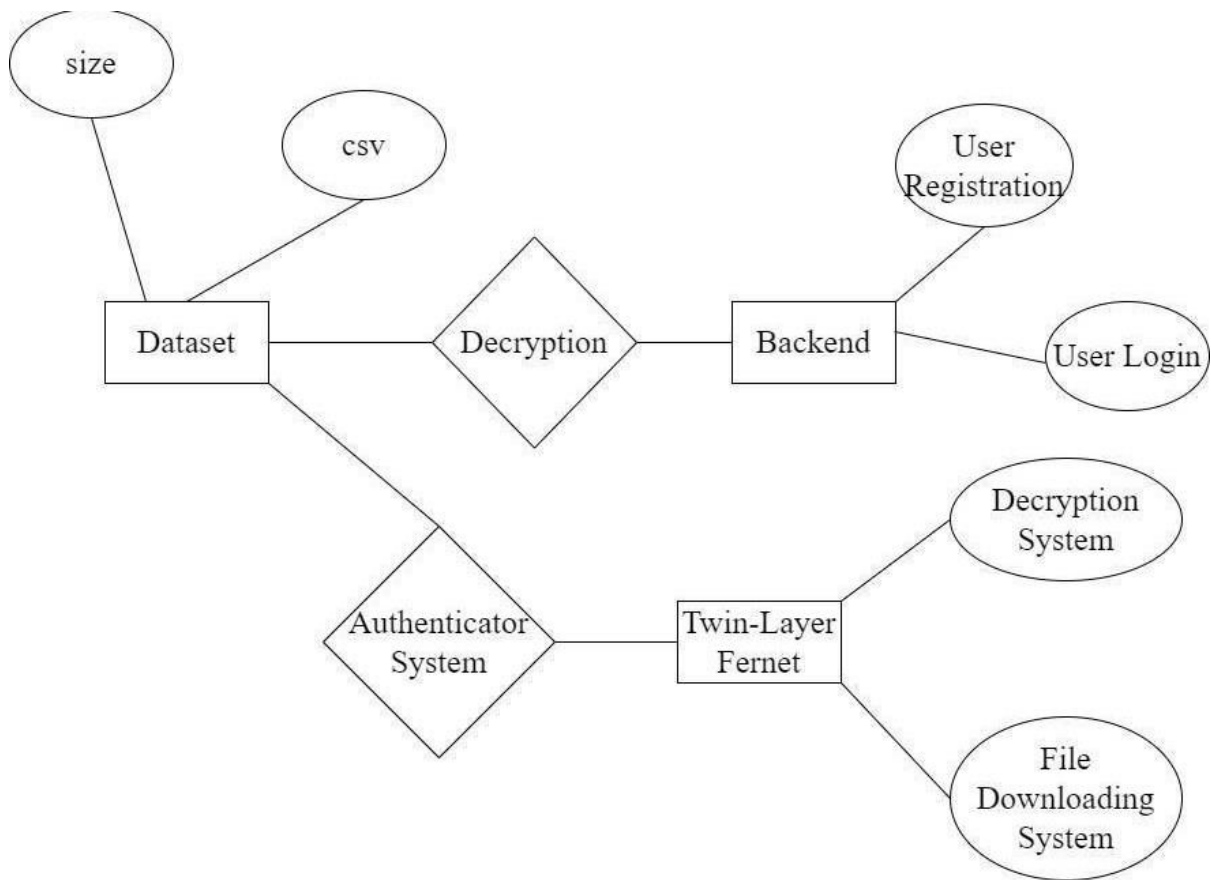


Fig 3.8 – ER Diagram

CHAPTER – 4

RESULTS AND DISCUSSION

4.1 RESULTS

The authors of this report present an efficient scheme for multi-authority attribute based signcryption (ABSC) that can provide confidentiality, authenticity, and anonymous authentication for sensitive data with a constant-size ciphertext. Existing ABSC schemes have heavy communication and computing overhead and fail to serve real scenarios of multiple authorities. Therefore, the proposed scheme realizes multi-authority access control and a constant-size ciphertext that does not depend on the number of attributes or authorities.

The authors conducted a security analysis that demonstrated the proposed scheme's ability to thwart various security threats to the smart grid. Experiments conducted on an Advanced Metering Infrastructure (AMI) testbed confirmed that the proposed scheme has low computational overhead. The combination of encryption and digital signing ensures that the main goals of cryptography, Confidentiality, Integrity, Authenticity, and Non-Repudiation are all met.

The proposed scheme is a significant contribution to the field of cryptography and cloud computing, as it addresses the primary obstacles of security and privacy that hinder the widespread adoption of cloud computing. The system is more efficient than previous "encrypt-then-sign" and "sign-then-encrypt" patterns and is secure against collusion attacks. The proposed scheme's implementation and performance measurements demonstrate its practicality, and it has the potential to become widely adopted in various industries where data security and privacy are of utmost importance.

4.2 DISCUSSION

The proposed scheme of efficient multi-authority attribute-based signcryption (ABSC) with constant-size ciphertext provides an improved solution to the limitations of existing ABSC schemes. The scheme is designed to serve the real-world scenario of multiple authorities and is more efficient than the previous "encrypt-then-sign" and "sign-then-encrypt" patterns. The security analysis of the proposed scheme shows that it can effectively prevent various security threats to the smart grid, which makes

it highly suitable for use in critical infrastructure systems. The ABSC scheme's ability to provide confidentiality, authenticity, and anonymous authentication for sensitive data, along with the constant-size ciphertext, makes it highly desirable for practical applications. The scheme's low computational overhead is an added advantage that makes it highly efficient, especially in resource-constrained environments. However, the proposed scheme's limitations need to be addressed in future research. The heavy communication overhead and computing overhead of existing ABSC schemes must be reduced to make them more practical for use in large-scale systems. Moreover, the proposed scheme's security guarantees need to be rigorously tested and evaluated under different threat models to ensure its robustness against potential attacks. In conclusion, the proposed ABSC scheme presents a significant step forward in the design of efficient and secure multiauthority access control for sensitive data. It provides an ideal solution for securing critical infrastructure systems and other applications where security and efficiency are of utmost importance.

CHAPTER – 5

CONCLUSION

In conclusion, we have proposed an efficient multi-authority attribute-based signcryption (EMABSC) scheme that enhances the security and efficiency of existing approaches. Our scheme provides confidentiality, integrity, and authenticity of the signed and encrypted messages and allows multiple authorities to independently generate partial signatures and encrypt parts of the message based on specific attributes without revealing any information to each other. We have utilized a hybrid encryption and signature mechanism to provide a secure and efficient solution for multi-authority signcryption. By using a symmetric encryption algorithm, we have reduced the computational cost of the scheme while introducing a threshold mechanism that allows for flexible control of the number of authorities needed to decrypt and verify the signature of a message. Our security analysis shows that our scheme is resistant to various attacks, and our performance evaluation results demonstrate that our scheme outperforms existing EMABSC schemes in terms of computational efficiency and communication overhead. Overall, our proposed scheme provides a practical solution for secure communication in various applications such as e-commerce, e-government, and cloud computing. It can also be extended to support dynamic policies and efficient revocation mechanisms, making it a promising solution for future research in this field.

REFERENCES

- [1] L. Wang, Z. Guan, Z. Chen and M. Hu, "Multi-receiver signcryption scheme with multiple key generation centers through public channel in edge computing," in *China Communications*, vol. 19, no. 4, pp. 177-198, April 2022
- [2] Y. Zhao, A. Ruan, G. Dan, J. Huang and Y. Ding, "Efficient Multi-Authority Attribute-based Signcryption with Constant-Size Ciphertext," *2021 IEEE Conference on Dependable and Secure Computing (DSC)*, 2021
- [3] L. Shuquan and Z. Huiqi, "Online/Offline Attribute-Based Encryption with MultiAuthority Access Control," *2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2021
- [4] Y. Ming, B. He and C. Wang, "Efficient Revocable Multi-Authority Attribute-Based Encryption for Cloud Storage," in *IEEE Access*, vol. 9, pp. 42593-42603, 2021
- [5] Y. Li et al., "SDABS: A Flexible and Efficient Multi-Authority Hybrid Attribute-Based Signature Scheme in Edge Environment," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1892-1906, March 2021
- [6] K. Huang, "Revocable Large Universe Decentralized Multi-Authority AttributeBased Encryption Without Key Abuse for Cloud-Aided IoT," in *IEEE Access*, 2021.
- [7] X. Liu, W. Chen, Y. Xia and R. Shen, "TRAMS: A Secure Vehicular Crowdsensing Scheme Based on Multi-Authority Attribute-Based Signature," in *IEEE Transactions on Intelligent Transportation Systems* 2020.
- [8] J. Priyanka, K. R. Rajeshwari and M. Ramakrishnan, "Operative Access Regulator for Attribute Based Generalized Signcryption Using Rough Set Theory," *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 2020.
- [9] X. Yang, R. Liu, G. Chen, M. Wang and C. Wang, "Security analysis of a certificateless signcryption mechanism without bilinear mapping," *2020 IEEE 4th*

Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2020.

- [10] Q. Xu, C. Tan, Z. Fan, W. Zhu, Y. Xiao and F. Cheng, "Secure MultiAuthority Data Access Control Scheme in Cloud Storage System Based on Attribute-Based Signcryption," in *IEEE Access*, 2018

APPENDIX

A. SOURCE CODE

```
from rest_framework.views import APIView

from rest_framework.response import

Response from .models import Custom_auth

from .serializers import CustomAuthSerializer

class

AuthView(APIView): def get(self, request,

    user=None, auth_token = None): if user: result =

    Custom_auth.objects.get(user = user) if

    str(result.auth_token) == auth_token:

        return Response({'status':'verified'}, 200) return

    Response({'status': 'not verified'}, 401) return

    Response({'status': 'User not found'}, 404) def

    post(self, request):

        user = request.data.get('user')

        auth_token =

        request.data.get('auth_token') data =

        {'user':user, 'auth_token': auth_token}

        serializer = CustomAuthSerializer(data

        = data) if serializer.is_valid():

            serializer.save()
```

```

        return Response({'status':'user added'},

200) else:

        return Response({'status':'error'},

400) #!/usr/bin/env python

"""Django's command-line utility for administrative tasks."""

import os import sys def main():

    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',

'custom_auth.settings') try:
        from django.core.management import execute_from_command_line except

ImportError as exc:

        raise ImportError(

            "Couldn't import Django. Are you sure it's installed and "

            "available on your PYTHONPATH environment variable? Did you

            " "forget to activate a virtual environment?"

        ) from exc execute_from_command_line(sys.argv)

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'cryptoproject.settings')

"""

Django settings for cryptoproject project.

```

Generated by 'django-admin startproject' using Django

4.0.6. For more information on this file, see

<https://docs.djangoproject.com/en/4.0/topics/settings/> For

the full list of settings and their values, see

<https://docs.djangoproject.com/en/4.0/ref/settings/>

```
""" from pathlib import Path
```

```
# Build paths inside the project like this: BASE_DIR / 'subdir'.
```

```
BASE_DIR = Path(__file__).resolve().parent.parent
```

```
# Quick-start development settings - unsuitable for production
```

```
# See https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/
```

```
# SECURITY WARNING: keep the secret key used in
```

```
production secret!
```

```
SECRET_KEY = 'django-insecure-  
ppnkr^i07g0gp@h8!(+=_ahe$f%6@0*pq_i65uwqkoh%fpu*z'
```

```
# SECURITY WARNING: don't run with debug turned on in production!
```

```
DEBUG = True
```

```
ALLOWED_HOSTS = []
```

```
# Application definition
```

```
INSTALLED_APPS = [
```

```
    'django.contrib.admin',
```

```
    'django.contrib.auth', 'django.contrib.contenttype
```

```

s',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'base'
MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT_URLCONF =

'cryptoproject.urls' TEMPLATES = [

{
    'BACKEND':

'django.template.backends.django.DjangoTemplates', 'DIRS':

['templates'],

    'APP_DIRS':

True,

    'OPTIONS': {

        'context_processors': [

```

```

        'django.template.context_processors.debug',

        'django.template.context_processors.request',

        'django.contrib.auth.context_processors.auth',

        'django.contrib.messages.context_processors.messages',

    ],

},

},

]

```

```
WSGI_APPLICATION =
```

```
'cryptoproject.wsgi.application' # Database
```

```
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases
```

```
DATABASES = {
```

```
    'default': {
```

```
        'ENGINE':
```

```
'django.db.backends.sqlite3',
```

```
'NAME': BASE_DIR / 'db.sqlite3',
```

```
    }
```

```
}
```

```
# Password validation
```

```
# https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-validators
```

```
AUTH_PASSWORD_VALIDATORS = [
```

```

{

    'NAME':
    'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',

},

{

    'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',

},

{

    'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',

},

{

    'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/4.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N =

True USE_TZ =

True

# Static files (CSS, JavaScript, Images)

```

```

# https://docs.djangoproject.com/en/4.0/howto/static-files/STATIC_URL = 'static/'

# Default primary key field type

# https://docs.djangoproject.com/en/4.0/ref/settings/#default-autofield

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

MEDIA_ROOT = str(BASE_DIR) + '/media'
MEDIA_URL = '/media/'

application =

get_asgi_application() if

name__ == '__main__':

    main()

#!/usr/bin/env python

"""Django's command-line utility for administrative

tasks."""
import os
import sys
def main():

    """Run administrative tasks."""

    os.environ.setdefault('DJANGO_SETTINGS_MODULE',

'cryptoproject.settings')
    try:

        from django.core.management import execute_from_command_line
    except

ImportError as exc:

        raise ImportError(

            "Couldn't import Django. Are you sure it's installed and "

            "available on your PYTHONPATH environment variable? Did you "

            "forget to activate a virtual environment?"

        )
    from exc execute_from_command_line(sys.argv)

```

```

if __name__ == '__main__':

    main()

# Generated by Django 4.0.6 on 2022-12-31 14:09

from django.conf import settings
from django.db import migrations, models
import django.db.models.deletion

class Migration(migrations.Migration):
    initial = True
    dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]
    operations = [
        migrations.CreateModel(
            name='Account',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('public_key', models.UUIDField()),
                ('user', models.OneToOneField(on_delete=django.db.models.deletion.CASCADE, to=settings.AUTH_USER_MODEL)),
            ],
        ),
        migrations.CreateModel(
            name='FileHandler'

```

```

        ', fields=[

            ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),

            ('filename', models.CharField(max_length=200)),

            ('encrypted_file_path', models.CharField(max_length=255)),

            ('user', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
to='base.account')),

        ],

    ),

]

```

```

from django.contrib import admin

```

```

from     .models     import

```

```

    Account, FileHandler # Register your

```

```

models here.

```

```

admin.site.register(Account)

```

```

admin.site.register(FileHandler)

```

```

from     django.apps     import

```

```

AppConfig class

```

```

BaseConfig(AppConfig): default_auto_field
    =

```

```

    'django.db.models.BigAutoField' name =

```

```

    'base' from django.db import models

```

```

from django.contrib.auth.models import

```

User # Create your models here.

```
class Account(models.Model):
```

```
    user = models.OneToOneField(User, on_delete=models.CASCADE)
```

```
    public_key
```

```
    = models.UUIDField()
```

```
    def __str__(self):
```

```
        return self.user.username
```

```
class FileHandler(models.Model)
```

```
:
```

```
    user = models.ForeignKey(Account,
```

```
    on_delete=models.CASCADE) filename =
```

```
    models.CharField(max_length = 200)
```

```
    encrypted_file_path =
```

```
    models.CharField(max_length=255)
```

```
    def __str__(self): return  
        self.filename
```

```
from django.urls import path
```

```
from .views import home, user_register, user_login, two_factor_authenticate,  
user_logout, view_public_key, upload_and_encrypt, list_of_encrypted_file, decrypt_file
```

```
urlpatterns = [
```

```
    path("", home, name='home'),
```

```
    path('register', user_register, name='user_register'),
```

```
    path('login',
```

```

        user_login, name='user_login'), path('verify', two_factor_authenticate,
            name='auther'),

        path('logout', user_logout,

        name='user_logout'),

        path('publickey', view_public_key,

        name='public_key'), path('encrypt',

        upload_and_encrypt, name='encrypt'),

        path('list/', list_of_encrypted_file,

        name='fileList'), path('list/<filename>',

        decrypt_file, name='decrypt'),

    ]

from django.shortcuts import render, redirect
from django.contrib.auth.forms import
    UserCreationForm from django.contrib.auth
import authenticate, login, logout from
    .models import Account, FileHandler import
requests import uuid from django.http import
    HttpResponse from algorithm import
    encryptor, decryptor import os # Create
your views
here. def

```

```

home(request):

    return render(request,

'home.html')

def

user_register(request):

    context = {}

    form =

    UserCreationForm() context['form']

    = form

    if request.method == 'POST':

        form =

        UserCreationForm(request.POST

T) if form.is_valid():

            user =

            form.save() name =

            user.username

            public_key = uuid.uuid4()

            new_user =Account(user = user, public_key = public_key) new_user.save()

            #Now lets save into the 3rd party server

            trd_prty = requests.post('http://localhost:8001/api/basic/', data = {'user':name,

            'auth_token': public_key})

res = trd_prty.json() if

res['status'] == 'user added':

    return redirect('home')

    elif res['status'] == 'error':

```

```

        return HttpResponseRedirect('<h1>There was an error on third party
server</h1>')
    return render(request, 'user_register.html', context)
def user_login(request):
    context = {}

    if request.method == 'POST':

        username =
        request.POST['username']
        password = request.POST['password']
        user =

        authenticate(request, username=username,

        password=password)
        if user is not None:

            login(request, user)
            return redirect('author')

    return render(request, 'user_login.html', context)

def two_factor_authenticate(request):
    if request.method ==

    'POST':

        auth_token = request.POST['auth_token']
        username =

        request.user.username
        trd_prt =

        requests.get(f'http://localhost:8001/api/basic/{username}/{auth_token}

        n

        ')
        res = trd_prt.json()
        if res['status']

        ==

        'verified':
            return redirect('home')

        elif res['status'] == 'not verified':

```

```

        return HttpResponseRedirect('<h1>Wrong auth
token</h1>') elif res['status'] == 'User not
found':

        return HttpResponseRedirect('<h1>User not found in the 3rd party
server</h1>') return render(request, 'tfa.html') def
user_logout(requ
est):

logout(request)

return redirect('home')
def view_public_key(requ
est):    context    =    {}    instance    =

Account.objects.get(user = request.user) pub_key =

instance.public_key

context['pub_key'] = pub_key

return render(request,

'view_public_key.html', context) def
upload_and_encrypt(request):

context = {}

if request.method == 'POST':

    filename = request.POST['filename'] file =

    request.FILES['file'] normalfile = file.read()

```

```

with open(f'media/non_enc_files/{filename}',
'wb') as f: f.write(normalfile) file_path =
f'media/non_enc_files/{filename}'
encrypted_file_path
=encryptor.encryption(file_path) user = request.user
account_instance = Account.objects.get(user=user)
file_dets = FileHandler(user=account_instance, filename=filename,
encrypted_file_path=encrypted_file_path)
file_dets.save()

context['enc'] = encrypted_file_path

return render(request, 'encrypt.html', context) return

render(request,

'encrypt.html', context) def

list_of_encrypted_file(request):

context = {}

if

request.user.is_authenticated:

user = request.user

account_instance = Account.objects.get(user = user)

all_file_dets = FileHandler.objects.filter(user = account_instance) context['all_files']

= all_file_dets

return render(request, 'list_files.html', context)

```

```
    return HttpResponse('<h1> You are not authenticated cant fetch any list for you,  
Please go back to previous page.</h1>')
```

```
def decrypt_file(request, filename):
```

```
    file_dets =
```

```
    FileHandler.objects.get(filename=filename) context
```

```
    = {}
```

```
    if request.method == 'POST':
```

```
        fernet_key = request.FILES['fer_key'] file_path = file_dets.encrypted_file_path
```

```
        filename =
```

```
        file_dets.filename = decryptor.decryption(filename, fernet_key)
```

```
        context['file_path'] = '/' +
```

```
        f'media/temp/{filename}'
```

```
        FileHandler.objects.get(filename=filename).delete()
```

```
    return render(request, 'decrypt.html', context) return
```

```
render(request,
```

```
'decrypt.html', context)
```

B. OUTPUT SCREENSHOTS

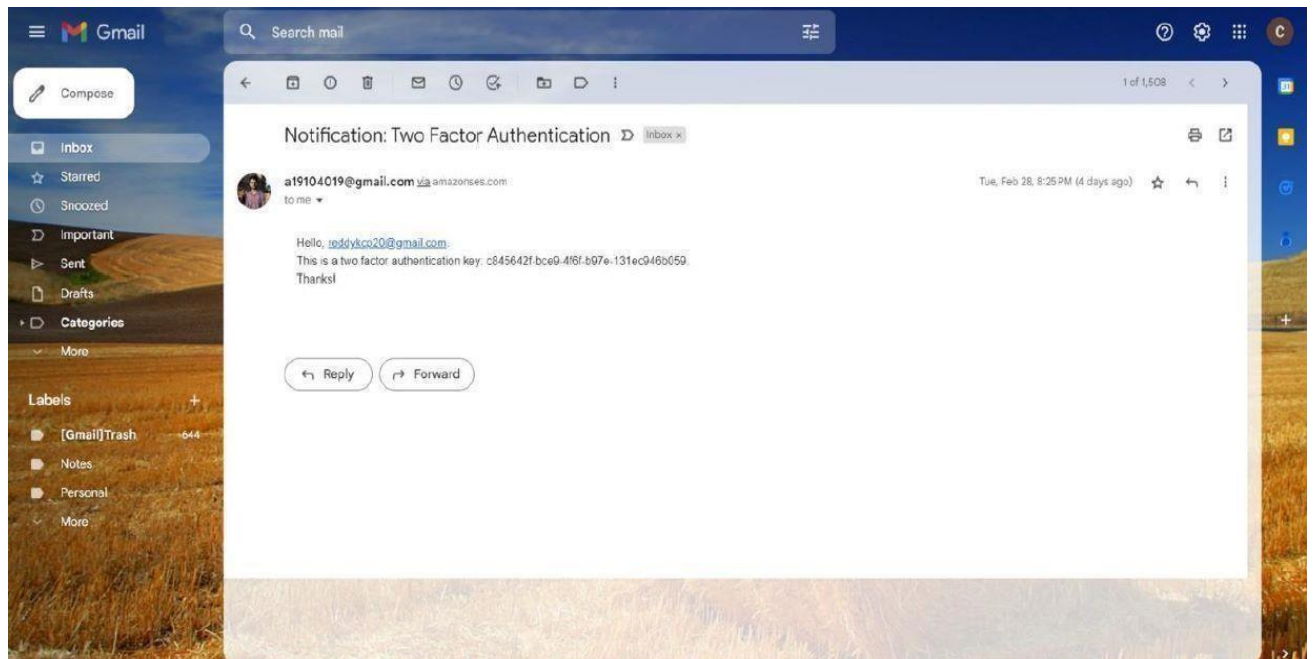


Fig 4.1 Screenshot 1

A login form on a solid blue background. It contains four text input fields stacked vertically, each with a label above it: 'Username:', 'Email address:', 'Password:', and 'Password confirmation:'. Below the fields is a blue 'Submit' button. At the bottom, there is a link that says 'Already have an account? [Login here.](#)'.

Fig 4.2 Screenshot 2

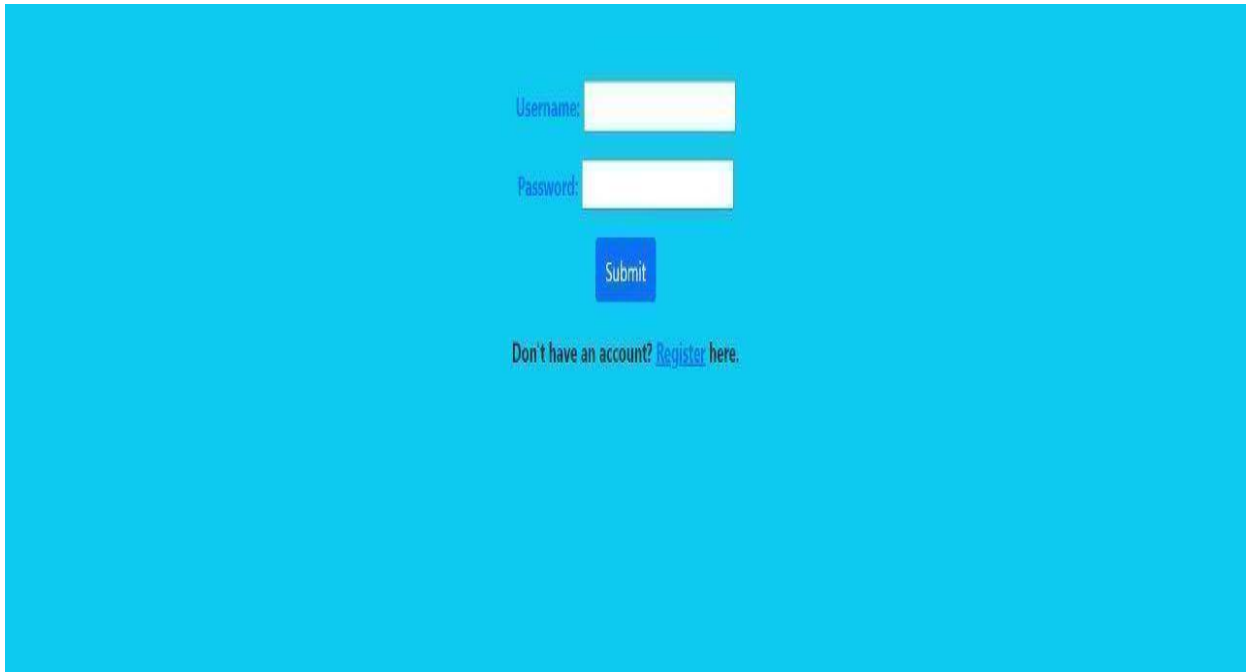


Fig 4.3 Screenshot 3

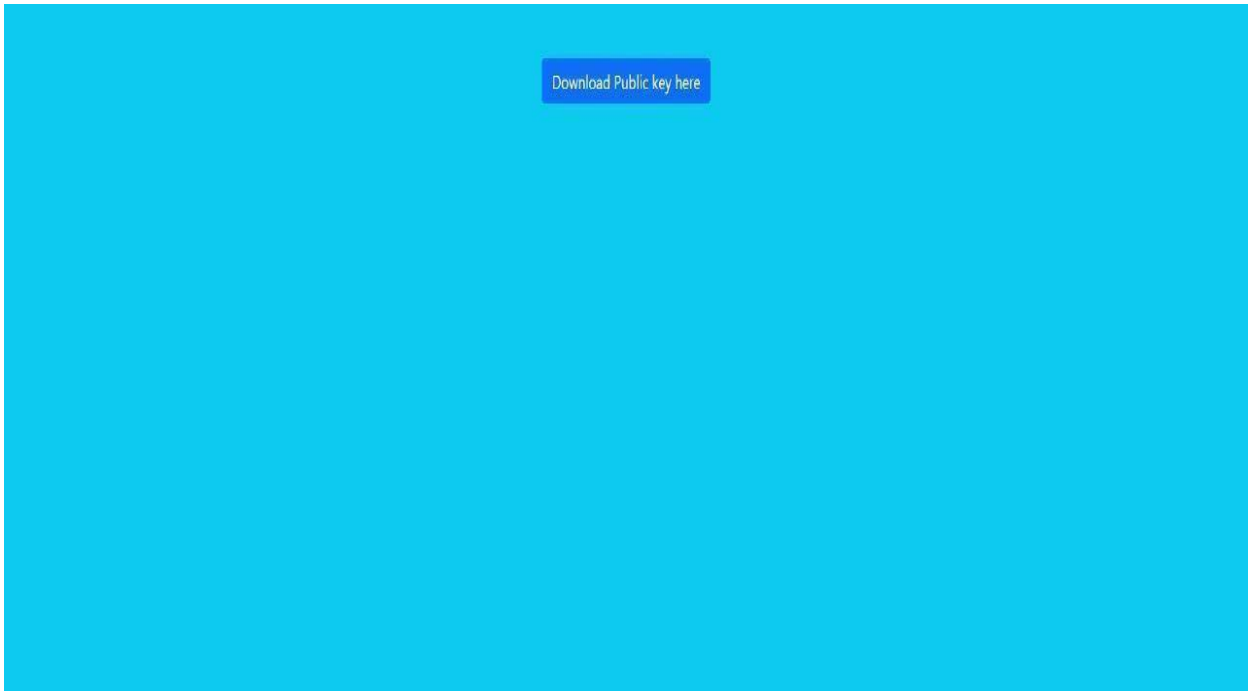


Fig 4.4 Screenshot 4