A

Project Report

on

AUTHENTICATION USING ENCRYPTED NEGATIVE PASSWORD

submitted in partial fulfillment of the requirements for the award of the

Bachelor of Technology degree

in

COMPUTER SCIENCE AND ENGINEERING

by

K. Sai Vamshi (20EG105356)

Under the guidance of

Dr. T Shyam Prasad

M.Tech., PhD

Assistant Professor, CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Venkatapur (V), Ghatkesar (M), Medchal (D), T.S– 500088

YEAR 2023-24

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the report entitled **"AUTHENTICATION USING ENCRYPTED NEGATIVE PASSWORD"** that is being submitted by K. Sai Vamshi (20EG105356) in partial fulfillment for the award of Bachelor of Technology in Computer Science and Engineering to the Anurag University is a record of bonafide work carried out by him under my guidance and supervision.

The results embodied in this Report have not been submitted to any other University or Institute for the award of any degree or diploma

Signature of Supervisor	**Dean, Department of CSE**
Dr. T Shyam Prasad	**Dr. G Vishnu Murthy**
M.Tech., Ph.D	M.Tech., Ph.D
Assistant Professor, CSE	Professor

External Examiner

DECLARATION

I hereby declare that the report entitled "**AUTHENTICATION USING ENCRYPTED NEGATIVE PASSWORD**" submitted for the award of Bachelor of technology Degree is my original work and the report has not formed the basis for the award of any degree, diploma, associateship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

Place: Anurag University, Hyderabad

K. Sai Vamshi (20EG105356)

Date: 20/04/2024

ACKNOWLEDGEMENT

I would like to express my sincere thanks and deep sense of gratitude to project supervisor **Dr. T Shyam Prasad** for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved my grasp of the subject and steered to the fruitful completion of the work. His patience, guidance and encouragement made this project possible.

I would like to express my special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for his encouragement and timely support in my B.Tech program.

I would like to acknowledge my sincere gratitude for the support extended by **Dr. G. Vishnu Murthy** , Dean, Dept. of CSE ,Anurag University. I also express my deep sense of gratitude to **Dr. V V S S S Balaram** , Academic coordinator. and **Dr. T Shyam Prasad,** Project Coordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated me during the crucial stage of my project work.

ABSTARCT

In the current landscape of escalating cyber threats, the security of user authentication mechanisms stands as a critical concern. Despite the introduction of alternative methods such as biometrics and multi-factor authentication, challenges persist regarding privacy, usability, and scalability. Most of the users prefer regular Text based passwords over every other type due to its simplicity, which remain vulnerable to a myriad of attacks, including brute force, dictionary attacks, and phishing schemes.

This abstract introduces Encrypted Negative Password Authentication (ENPA) as an innovative approach to address these challenges. ENPA enhances security by minimizing the exposure of sensitive information. Even in the event of a data breach, adversaries gain no advantage as the negative passwords are never stored in a recoverable format. Additionally, incorporating encryption and hashing techniques mitigates risks associated with common attacks such as rainbow table attacks.

ENPA offers users a streamlined authentication experience where the inconvenience of complex password management is shifted to the backend. ENPA prioritizes user convenience and security, allowing users to focus on accessing their accounts with ease.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF GRAPHS

1.INTRODUCTION

Password security always attracts great interest from academia and industry. Despite great research achievements on password security, passwords are still cracked since users' careless behaviors. For instance, many users often select weak passwords; they tend to reuse same passwords in different systems; they usually set their passwords using familiar vocabulary for its convenience to remember. In addition, system problems may cause password compromises. It is very difficult to obtain passwords from high security systems. On the one hand, stealing authentication data tables (containing usernames and passwords) in high security systems is difficult. On the other hand, when carrying out an online guessing attack, there is usually a limit to the number of login attempts. However, passwords may be leaked from weak systems. Vulnerabilities are constantly being discovered, and not all systems could be timely patched to resist attacks, which gives adversaries an opportunity to illegally access weak systems. Finally, since passwords are often reused, adversaries may log into high security systems through cracked passwords from systems of low security.

1.1 Background

Password storage schemes are fundamental components of authentication systems, responsible for securely storing and managing user passwords. The evolution of password storage schemes has been driven by the need to address security vulnerabilities and adapt to emerging threats in the ever-changing landscape of cybersecurity.

Different Types of Password Storage Methods:

- Plaintext Storage: This is the most basic and least secure method where passwords are stored as-is, without any encryption or hashing. In this method, passwords are easily readable if attackers gain unauthorized access to the storage system.

- Cryptographic Hashing: Passwords are hashed using cryptographic

algorithms such as MD5, SHA-1, SHA-256, etc. The hash function converts passwords into fixed-length hash values, making it computationally infeasible to reverse-engineer the original password from the hash.

- Salted Hashing: In this method, a unique random value (salt) is added to each password before hashing. Salting prevents identical passwords from producing the same hash, making it more difficult for attackers to use precomputed tables (rainbow tables) for password cracking.

- Key Stretching: Key stretching techniques, such as PBKDF2 (Password-Based Key Derivation Function 2), bcrypt, and scrypt, introduce computational overhead by repeatedly applying a hashing function to the password. This slows down the password hashing process, making it more time-consuming for attackers to crack passwords.

- ENP: ENP (Encrypted Negative Password) is a pioneering approach to authentication that prioritizes user privacy and security. Users create passwords that are then converted into negative passwords. The system encrypts and securely stores hashed versions of these passwords on the server, enhancing resilience against common attacks. With ENP, users can enjoy a streamlined and secure authentication experience while minimizing the risk of unauthorized access to their accounts.

SHA-256: SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that produces a 256-bit (32-byte) hash value. Widely used in various security applications, including password hashing and digital signatures, SHA-256 is known for its strong collision resistance and computational efficiency. Its widespread adoption in

secure protocols and applications underscores its reliability in ensuring data integrity and confidentiality in digital communications.

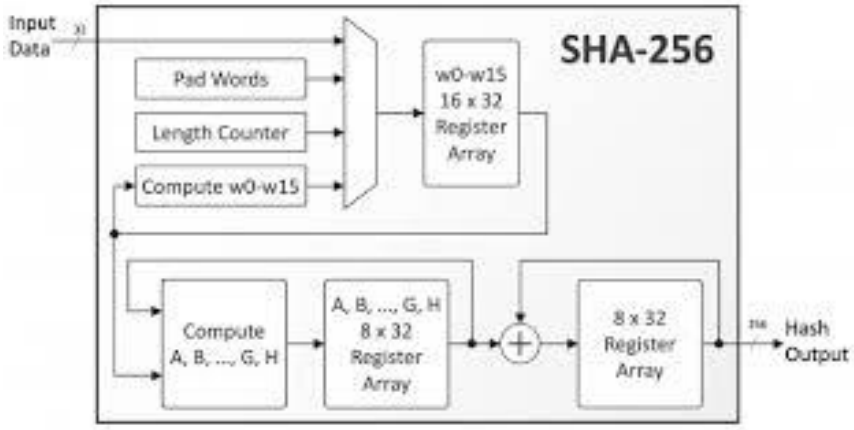
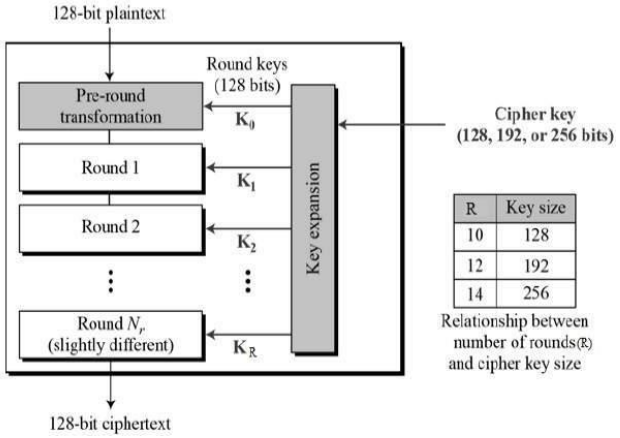


Figure 1.1

Advanced Encryption Standard Algorithm (AES):

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm widely adopted for securing sensitive data. AES operates on fixed-length blocks of data, encrypting and decrypting them using a secret key. Renowned for its efficiency and robustness, AES offers varying key sizes (128, 192, or 256 bits) and operates through a series of substitution, permutation, and mixing operations.



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Figure 1.2 : AES Algorithm

In a typical round of AES encryption, there are four main sub-processes. First, Byte Substitution (SubBytes) replaces each of the 16 input bytes using a fixed lookup table (S-box), resulting in a 4x4 matrix. Next, ShiftRows shifts each row of the matrix

to the left, with different shift amounts for each row. Then, MixColumns transforms each column using a mathematical function, except in the last round. Finally, AddRoundKey XORs the matrix bytes with the round key. If it's the last round, the output is the ciphertext; otherwise, the process repeats for another round. In the decryption process, these sub-processes are applied in reverse order for each round. Unlike Feistel ciphers, encryption and decryption in AES are implemented separately due to this reverse process, though they are closely related.

Padding : Padding PKCS7 is a method used in cryptography to pad plaintext messages before encryption, ensuring that the length of the plaintext is a multiple of the block size required by the encryption algorithm. In PKCS7 padding, bytes are added to the plaintext, with each byte containing the number of padding bytes added. For example, if two bytes are required to pad the plaintext, both bytes would have the value 0x02. This padding scheme simplifies padding removal during decryption and is widely used in encryption standards such as AES.

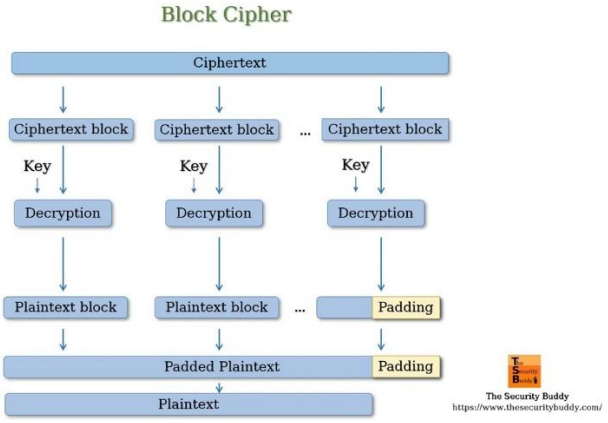


Figure 1.3 : PKCS7

2.LITERATURE SURVEY

2.1 Concerns and Security for Hashing Passwords

Authors, Year : Jonathan Herrera; Md Liakat Ali, 2019.

The aim of this paper is to discuss current concerns with hashing and propose a new system level approach that will add an extra layer of security for storing passwords. The greatest security threat is bad security practices. One sees with some of the greatest hacks, resulting in some of the greatest losses, it was because of bad security practices. It is important to have security for passwords that are designed to last many years once implemented.

2.2 Salty Secret: Let us secretly salt the secret

Authors, Year : E M Wasifur Rahman Chowdhury , M Saifur Rahman, A. B. M. Alim Al Islam, Mohammad Sohel Rahman , 2017.

Using password is, perhaps, still the most versatile method of securing secret and confidential information, even though several recent studies have pointed out possibility of breaching it. A general trend of having different passwords for several user accounts of the same user (such as multiple email accounts, multiple social networking accounts, etc.) can barely overcome the possibility as users mostly prefer retaining similarity among own passwords, which results in the possibility of breaching almost all passwords once only one password gets breached. Consequently, several research studies attempted to strengthen passwords. However, none of the studies is yet to get wide popularity for not being able to achieve a delicate balance between strength of password and user friendliness. To achieve this goal, we present a new password based authentication system in this paper. The proposed system is based on intermixing between a fixed text (conventional part of a password) and a free random text (newly added) at different pre-defined indices having different per-defined lengths. The addition of the free random text adds an additional level of difficulty in breaching the password. We present different variants of our proposed

system along with their possible attack models. We demonstrate strength of our proposed system through rigorous analytical formulation and numerical simulation. Besides, we confirm achieving a delicate balance between strength of the password and user friendliness through performing real user evaluation.

2.3 On the Economics of Offline Password Cracking

Authors, Year: Jeremiah Blocki, Ben Harsha, Samson Zhou, 2020.

We develop an economic model of an offline password cracker which allows us to make quantitative predictions about the fraction of accounts that a rational password attacker would crack in the event of an authentication server breach. We apply our economic model to analyze recent massive password breaches at Yahoo!, Dropbox, LastPass and AshleyMadison. All four organizations were using key-stretching to protect user passwords. In fact, LastPass' use of PBKDF2-SHA256 with 105 hash iterations exceeds 2017 NIST minimum recommendation by an order of magnitude. Nevertheless, our analysis paints a bleak picture: the adopted key-stretching levels provide insufficient protection for user passwords.

2.4 Implementation of Password-based Key Derivation Function for Authentication Scheme in Patrolling System

Authors, Year :, Laurentius Kuncoro Probo Saputra; Willy Sudiarto Raharjo 2019.

Outsourced security guards are often used by building owners or companies to help them maintain security. Problems arise from complaints about how companies can monitor surveillance activities in real-time and reliably. Supervisors must be able to know with certainty who is the security guard on duty. Therefore, an authentication process to oversee the work of security officers is needed in the patrolling system. The authentication system required the security guards to come to the location to read the QR code on the display device. The QR code was dynamically and only a valid

account in an android application could verify the codes. QR code information was generated using key derivation function based on shift schedule information and security guard account number.

2.5 Authentication by Encrypted Negative Password

Authors, Year :, Wenjian Luo; Yamin Hu; Hao Jiang; Junteng Wang, 2019.

Secure password storage is a vital aspect in systems based on password authentication, which is still the most widely used authentication technique, despite some security flaws. In this paper, we propose a password authentication framework that is designed for secure password storage and could be easily integrated into existing authentication systems. In our framework, first, the received plain password from a client is hashed through a cryptographic hash function (e.g., SHA-256). Then, the hashed password is converted into a negative password. Finally, the negative password is encrypted into an encrypted negative password (ENP) using a symmetric-key algorithm (e.g., AES), and multi-iteration encryption could be employed to further improve security. The cryptographic hash function and symmetric encryption make it difficult to crack passwords from ENPs. Moreover, there are lots of corresponding ENPs for a given plain password, which makes precomputation attacks (e.g., lookup table attack and rainbow table attack) infeasible. The algorithm complexity analyses and comparisons show that the ENP could resist lookup table attack and provide stronger password protection under dictionary attack. It is worth mentioning that the ENP does not introduce extra elements (e.g., salt); besides this, the ENP could still resist precomputation attacks. Most importantly, the ENP is the first password protection scheme that combines the cryptographic hash function, the negative password, and the symmetric-key algorithm, without the need for additional information except the plain password.

Table 1 : Comparison between methods

Sl.No	Author	Strategies	Advantages	Disadvantages
1.	Jonathan Herrera; Md Liakat Ali	Hashed-Password	Easy to use and implement. One-way transformation, Consistency.	Brute-force attacks, dependency on hash algorithm entirely, collision vulnerabilities, Dictionary attacks.
2.	E M Wasifur Rahman Chowdhury; M Saifur Rahman; A. B. M. Alim Al Islam; M Sohel Rahman	Salted-Password	Unique hash values, Randomization, Scalability.	Implementation complexity, Storage overhead, Incompatibility, Salt visibility, Increased Computation time.
3.	Jeremiah Blocki; Benjamin Harsha; Samson Zhou	Key-Streching	Resistant to brute-force attacks, Customizable iterations.	Increased computational overhead, Resource intensiveness, Potential DOS attacks, bad user experience.
4.	Laurentius Kuncoro Probo Saputra; Willy Sudiarto Raharjo	Password-Based Key Derivation Function (PBKDF2)	Customizable iterations, Supported by wide range of programming languages.	Susceptible to brute-force attacks using GPU acceleration due to its simple structure.
5.	Wenjian Luo, Yamin Hu, Hao Jiang, and Junteng Wang	Encrypted Negative Password (ENP)	Resistance to Lookup table attack, resistance to dictionary attack, User friendly experience with secure password storage.	Complexity compared to other methods.

3. SYSTEM SPECIFICATIONS

3.1 System Requirements

- **Hardware Requirements:**

System : Intel i3 Processor

Ram : 4 GB or more

- **Software Requirements:**

Operating system : Windows 10.

Coding Language : Python

IDE : Visual Studio Code

Frontend: Django, HTML, CSS

Database: MySQL

4. SYSTEM METHODOLOGY

4.1 System Analysis

4.1.1 Existing System

Before the advent of Encrypted Negative Password Authentication (ENPA), traditional authentication systems predominantly relied on password-based methods. These systems typically involved users creating passwords, which were stored on servers in hashed or encrypted form for subsequent verification during login attempts. Some common existing authentication systems before ENPA include:

Plaintext Storage: In this basic method, passwords were stored in plaintext format on servers. While simple to implement, this approach posed significant security risks as passwords could be easily compromised if attackers gained unauthorized access to the server.

Cryptographic Hashing: Many authentication systems employed cryptographic hashing algorithms such as MD5 or SHA-1 to convert passwords into fixed-length hash values.

These hash values were stored on servers instead of plaintext passwords, reducing the risk of password exposure in the event of a security breach.

Salted Hashing: To enhance security, some systems utilized salted hashing, where a unique random value (salt) was added to each password before hashing. This helped mitigate risks associated with rainbow table attacks by ensuring that identical passwords resulted in different hash values.

Key Stretching: Some authentication systems incorporated key stretching techniques like PBKDF2 or bcrypt to increase the computational cost of password hashing. This made it more difficult for attackers to crack passwords through brute force or dictionary attacks.

Multi-factor Authentication (MFA): While not strictly password-based, MFA systems added an extra layer of security by requiring users to provide additional verification factors such as SMS codes, tokens, or biometric data in addition to passwords.

Overall, these existing authentication systems provided varying levels of security but were still vulnerable to common attacks such as brute force, dictionary attacks, and password reuse. ENPA introduced a novel approach to authentication, addressing some of the limitations and vulnerabilities of traditional password-based systems while prioritizing user privacy and security.

4.1.2 Proposed System

Encrypted Negative Password is the password storage scheme that we are focusing on in

this project. Like key stretching, it gives high security against lookup table attacks and dictionary attacks. The Encrypted Negative Password (ENP) storage mechanism is a novel approach to securely store user passwords while prioritizing privacy and security.

Negative Password: It is the transformed password after it undergoes several iterations using an algorithm called Negative password generation algorithm. It takes hashed password as input and gives negative password as output which is a collection of

characters such as ‘1’,’0’,’*’.

- The system consists of two main modules: Registration and Login/Authentication. In the Registration module, users are required to input their Email ID and a chosen password. The system then proceeds to hash the received plain password using a cryptographic hash function and converts it into a negative password through a Negative Database (NDB) generation algorithm. The negative password is encrypted using the AES algorithm, and the user's email, hashed password, negative password, and encrypted password are securely stored in the database.
- For the Login/Authentication module, users need to provide their Email ID and password. Upon entering, the system checks the existence of the username in the authentication data table. If found, it retrieves the corresponding Encrypted Negative Password (ENP) and decrypts it using the AES algorithm. The negative password is then obtained through decryption, completing the authentication process.

In this project, HTML, CSS, and JavaScript are used on the front end, and Python is used on the back end. Here, we have used the Django framework and MySQL Database.

Modules and their Description

The system comprises 1 major module with their sub-modules as follows:

Module:

- Registration:
- The user will have to enter their Email ID.
- They can enter a password.
- Hash the received plain password using a selected cryptographic hash function.
- Convert the hashed password into a negative password using an NDB (Negative Database) generation algorithm.
- Encrypt the negative password using an AES algorithm.
- Store the user's email, hashed password, negative password, and encrypted password in the database.

- **Login/Authentication:**
- To log in, the user will have to enter their Email ID and password.
- If the username exists, search the authentication data table for the Encrypted Negative Password (ENP) corresponding to the received username.
- Decrypt the ENP using the AES algorithm.
- Obtain the negative password by decrypting the ENP.

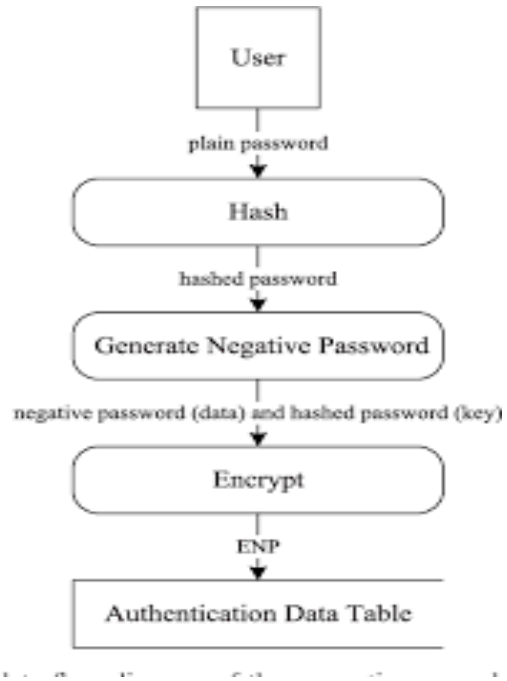


Figure 4.1 : Client side of ENP

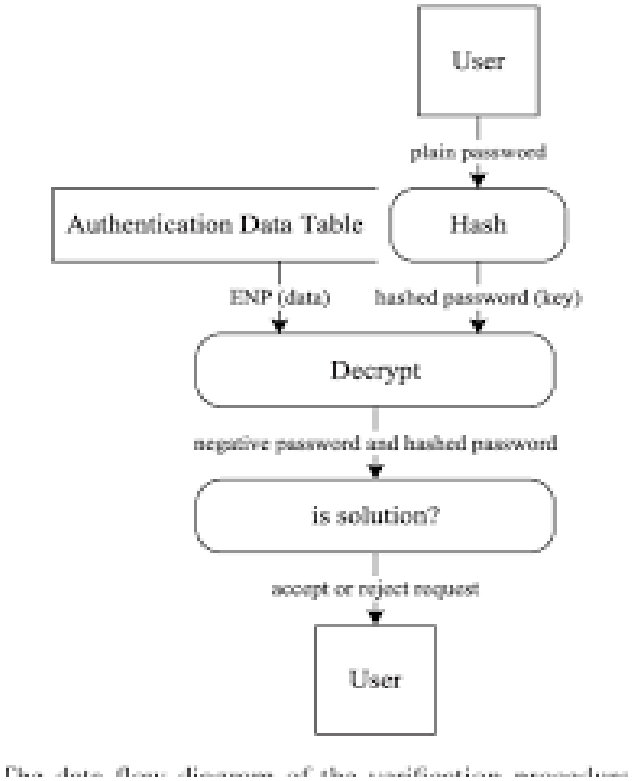


Figure 4.2: Server side of ENP

4.1.3 Proposed Method illustration: For example, A person wants to access a particular site and this process is divided into 2 phases.

A. Registration Phase The registration phase is divided into six steps.

(1) On the client side, a user enters his/her username and password. Then, the username and plain password are transmitted to the server through a secure channel.

(2) If the received username exists in the authentication data table, “The username already exists!” is returned, which means that the server has rejected the registration request, and the registration phase is terminated; otherwise, go to Step (3).

(3) The received password is hashed using the selected cryptographic hash function.

(4) The hashed password is converted into a negative password using an NDB generation algorithm.

(5) The negative password is encrypted to an ENP using the selected symmetric-key algorithm, where the key is the hash value of the plain password. Here, as an additional option, multi-iteration encryption could be used to further enhance passwords.

(6) The username and the resulting ENP are stored in the authentication data table and "Registration success" is returned, which means that the server has accepted the registration request.

B. Authentication Phase The authentication phase is divided into five steps.

(1) On the client side, a user enters his/her username and password. Then, the username and plain password are transmitted to the server through a secure channel.

(2) If the received username does not exist in the authentication data table, then "Incorrect username or password!" is returned, which means that the server has rejected the authentication request, and the authentication phase is terminated; otherwise, go to Step (3).

(3) Search the authentication data table for the ENP corresponding to the received username;

(4) The ENP is decrypted (one or more times according to the encryption setting in the registration phase) using the selected symmetric-key algorithm, where the key is the hash value of the plain password; thus, the negative password is obtained.

(5) If the hash value of the received password is not the solution of the negative password, then "Incorrect username or password!" is returned, which means that the server has rejected the authentication request, and the authentication phase is terminated; otherwise, "Authentication success" is returned, which means that the server has accepted the authentication request.

4.1.4 Registration Algorithm

As with any authentication scheme, ENP has two algorithms: the registration algorithm and the verification algorithm. The registration algorithm takes as input a new user's desired username and password, storing the username and some data based on the password (for example, its hash value, in the simplest case) in the internal authentication data table. The verification algorithm takes as input a username and password, checks whether the username is in the table, and checks whether the password matches the corresponding entry.

The registration algorithm of ENP has three main steps, as follows.

1. Hash the password

2. Convert to a negative password

3. Encrypt the negative password

The first step is simple: the password is hashed using the chosen cryptographic hash (SHA-256) function, so that the output has a constant number of bits, say m .In the second step, we convert the hashed password into negative password. This step is crucial because the algorithms for generating a negative password from a hashed password introduces randomness. However, importantly, any possible negative password can be solved to yield the original hashed password by simply checking which bit string is not in the database. In particular, We do not need to know the randomness that was used for the reverse direction, which removes the need to store additional data in the table. However, a negative password on its own is not more secure than just a hashed password, since as noted, any negative password can be solved to obtain the original hashed password. Therefore, an encryption step is necessary. In this step, we calculate Enc(H(pass), NP), where H(pass) is the hashed password used as the key, and NP is the negative password calculated in the second step. Note that it is important that the chosen encryption algorithm takes m-bit keys, since that is the length of H(pass).

The resulting output is the encrypted negative password, which gives the scheme its name, and is stored in the authentication data table.

4.1.5 Verification Algorithm

Given a username and password, the process of verifying the password consists of the following four steps.

1. Retrieve the ENP corresponding to the given username

2. Hash the password

3. Decrypt the ENP to get a negative password

4. Check that the hashed password is the solution to the negative password

4.1.6 Advantages of Proposed System :

- The system provides an added layer of security through the use of encryption and negative password generation techniques.
- It mitigates common vulnerabilities associated with traditional password systems.
- The system reduces the risk of password breaches and unauthorized access, safeguarding user credentials.
- The system discourages users from adopting insecure password practices.
-
- The major advantages of the proposed system are as mentioned below:

- Enhanced Security:

Utilizes cryptographic hash functions to securely store and protect user passwords in the authentication database. Incorporates negative authentication criteria to detect and prevent unauthorized access attempts, enhancing security against various attacks such as brute force and dictionary attacks. Employs symmetric-

key encryption to ensure the confidentiality and integrity of authentication data during transmission between the client and server, mitigating the risk of eavesdropping attacks.

Given below are the complexities of the lookup table attacks and dictionary attacks

Schemes	Lookup table attack		Dictionary attack	
	Time complexity.	Space complexity	Time complexity	Space complexity
Hashed password	$O(N_d * N_p * T_{m\_hash})$	$O(N_d)$	$O(N_d * N_p * (T_h + T_{m\_hash}))$	$O(1)$
Salted password	$O(N_d * 2^l * N_p * T_{m\_hash})$	$O(N_d * 2^l)$	$O(N_d * N_p * (T_h + T_{m\_hash}))$	$O(l)$
Key stretching	$O(N_d * 2^l * N_p * T_{m\_ks})$	$O(N_d * 2^l)$	$O(N_d * N_p * (T_{ks} + T_{m\_ks}))$	$O(l)$
ENPI	$O(N_d * m! * N_p * T_{m\_NP})$	$O(N_d * m!)$	$O(N_d * N_p * (T_h + [n*]T_d + T_{m\_NP}))$	$O(m^2)$

Table 2 : Complexities of attacks

- Usability:

  Maintains a user-friendly interface, allowing users to securely authenticate with their passwords without significant additional complexity. Provides a seamless registration and login process, minimizing friction for users while ensuring robust security measures are in place. Offers flexibility in handling authentication attempts, allowing users to authenticate securely from various devices and locations.

- Scalability:

  Designed to scale effectively to accommodate a growing user base and increasing authentication demands. Utilizes efficient cryptographic techniques and authentication mechanisms to ensure optimal performance even under high load conditions.

- Flexibility:

Adaptable to different authentication requirements and security policies, allowing organizations to customize authentication parameters based on their specific needs. Integrates seamlessly with existing authentication systems and infrastructures, enabling organizations to leverage their current investments while enhancing security measures.

- Resilience Against Attacks:

  Provides resilience against common password-based attacks such as brute force, dictionary, and rainbow table attacks through the use of cryptographic hash functions and negative authentication criteria. Enables proactive detection and mitigation of suspicious activities through continuous monitoring and evaluation of negative authentication criteria.

- Compliance and Regulatory Alignment:

  Helps organizations align with regulatory requirements and industry best practices related to password security and data protection. Facilitates compliance with standards such as GDPR (General Data Protection Regulation), HIPAA (Health Insurance Portability and Accountability Act), and PCI DSS (Payment Card Industry Data Security Standard).

- User Trust and Confidence:

  Enhances user trust and confidence in the authentication system by implementing robust security measures and safeguarding user credentials against unauthorized access. Demonstrates a commitment to protecting user privacy and confidentiality, fostering positive user experiences and long-term user relationships.

- Future-Proofing:

Positions organizations for future advancements in authentication technology and evolving cybersecurity threats by adopting a forward-thinking approach to password security. Provides a foundation for integrating additional authentication factors and advanced security features to address emerging threats and enhance overall security posture.

4.2 System Study

4.2.1 Feasibility Study

A Feasibility Study is a high-level capsule version of the entire process intended to answer several questions: What is the problem? Is there any feasible solution to the given problem? Is the problem even worth solving? A feasibility study is conducted once the problem is clearly understood

- Technical Feasibility

  In this step, we verify whether the proposed systems are technically feasible or not. i.e., all the technologies required to develop the system are available readily or not. Technical Feasibility determines whether the organization has the technology and skills necessary to carry out the project and how this should be obtained. The system can be feasible because of the following grounds:

  - All necessary technology exists to develop the system.
  - This system is too flexible and it can be expanded further.
  - This system can give guarantees of accuracy, ease of use, reliability and data security.
  - This system can give instant responses to inquiries.

    Our project is technically feasible because all the technology needed for our project is readily available.

Operating System: Windows 7 or higher

Languages: python

Database System : My SQL 5.6

Documentation Tool: MS – Word

- Economic Feasibility

Economically, this project is completely feasible because it requires no extra financial investment and concerning time, it is completely possible to complete this project in 6 months.

In this step, we verify which proposal is more economical. We compare the financial benefits of the new system with the investment. The new system is economically feasible only when the financial benefits are more than the investments and expenditures. Economic Feasibility determines whether the project goal can be within the resource limits allocated to it or not. It must determine whether it is worthwhile to process the entire project or whether the benefits obtained from the new system are not worth the costs. Financial benefits must be equal to or exceed the costs. In this issue, we should consider:

- The cost to conduct a full system investigation.
- The cost of h/w and s/w for the class of application being considered.
- The development tools.
- The cost of maintenance etc...

Our project is economically feasible because the cost of development is very minimal when compared to the financial benefits of the application.

- Operational Feasibility

In this step, we verify different operational factors of the proposed systems like manpower, time etc., and whichever solution uses less operational resources, is the best operationally feasible solution. The solution should also be operationally possible to implement. Operational Feasibility determines if the proposed system satisfies user objectives and could be fitted into the current system operation.

- The methods of processing and presentation are completely accepted by the clients since they can meet all user requirements.
- The clients have been involved in the planning and development of the system.
- The proposed system will not cause any problems under any circumstances.

Our project is operationally feasible because the time requirements and personnel requirements are satisfied. We are a team of four members and we worked on this project for three working months.

4.3 Security Analysis

In this section we provide an analysis of the ENP scheme compared to several other common password storage schemes introduced in section 2. We consider both theoretical and practical security, performance, and scalability.

Theoretical Security

We can evaluate the theoretical security of the scheme under lookup and dictionary attacks, and compares the security to what is achieved by hashed password, salted password, and key stretching schemes. Here, we provide a summary of the conclusions reached in the paper and add in a comparison to the peppered password scheme, as well as some of our own takeaways.

- Lookup Table Attacks

  In a lookup table attack, an adversary has access to the password authentication database and attempts to precompute a table mapping plaintext passwords to their corresponding hash values. The adversary performs a search and compare between their precomputed table and the authentication database to see if they can determine any username, password pairs. To perform a lookup table attack on an ENP scheme using m-bit password hashes, an adversary would need to compute all possible ENPs for every password it wishes to check for a given username (usually a list of most common passwords). According to the paper's findings that ENP provides more security against lookup table attacks than existing schemes. We note, however, that since an exponential space growth factor is sufficient to render precomputing a lookup table on modern machines, the factorial growth does not seem to add much practical security, at least in a modern context.

- Dictionary Attacks

  In a dictionary attack, an adversary tries to brute-force guess passwords corresponding to hashes in the authentication database. These attacks are impossible to prevent altogether, but can be made so time consuming that they become impractical to undertake. To perform a dictionary attack against the ENP scheme, an attacker would hack into the authentication database and, for each ENP in the database and for each of the Np

passwords that the attacker wishes to check: Obtain a hash of the plaintext password (in time Th), Decrypt the ENP to a negative password, using the password hash as a key (in time Td) , Check if the password hash is the solution of the decrypted negative password (in time Ts).The time complexity of this process depends entirely on how long each of these steps take. Since the adversary repeats these steps for each of the Np passwords in their password list to test against and for each of the Nd ENPs in the authentication table, the total time complexity of conducting a dictionary attack is $O(Nd \cdot Np \cdot (Th + Td + Ts))$

Practical Security

Next, we assess what the security of ENP might look like in practice. Since there is no need for salt, there is no possibility for administrative error such as salt re-use or making salts too short. There is also no risk of publicly exposing a global secret that could compromise the entire system, as is a risk in peppered password schemes. Peppered password schemes have the important benefit of forcing the attacker to brute-force the entire database for every possible pepper even after a database breach. If the attacker knows access to the salt and plaintext password of some user (including themselves), however, it is possible that they could brute-force the pepper. Once the pepper is obtained, the attacker would have significantly reduced the overall security of the system. It is important for this scenario that the passwords are salted, as otherwise the security is reduced to that of a hashed password scheme, which is not much. Still, after a breach, it is non-trivial to change the pepper. Given the difficulty of key rotation and breach recovery, it is important to store a separate pepper for each application managed by an entity.

In an ENP scheme, the attacker does not learn anything about the rest of the passwords in the database by obtaining a single username-password pair. This is because negative

passwords are generated from an (independent) random permutation for each password. Even if there is a database breach, one could simply recompute the negative passwords with fresh randomness for each compromised password. This seems to make the scheme much more practical to maintain than peppering.

The paper also highlights the benefits of not needing to store a salt for each password. The absence of a salt, however, does not seem to matter much for practical

security, as the main benefit of storing a unique salt for each password is that compromising a username-password pair does not leak any information about other passwords

in the system. This feature is already achieved by ENP, as noted above.

- Performance

  Like key stretching, as we noted earlier, the security of ENP against dictionary attacks depends on the computational price of authentication. Thus, a downside of ENP is that there exists a fundamental tradeoff between performance and security. Instead, peppered password schemes do not rely on the time to authenticate for security against dictionary attacks, introducing a significant benefit over ENP.

- Scalability

  A notable advantage of ENP is that not needing to store a salt with each password improves the scalability when compared to salting and key stretching. Thus, the scheme has the same space complexity as a simply hashed password table, but with significantly more security. Peppered schemes have similar space complexity, only requiring a single secret value to be stored. Since peppering is often combined with salting, however, in practicality ENP still exhibits more space benefits and thus scalability. In practicality, the added space needed for

salts likely is not a problem for modern machines. Therefore, even though ENP is theoretically more scalable than the other considered schemes, this is not necessarily a significant consideration.

5. IMPLEMENTATION

Implementation is the realization of an application, or execution of a plan, idea, model, design, specification, standard, algorithm, or policy. In other words, an implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through programming and deployment. Implementation is one of the most important phases of the Software Development Life Cycle (SDLC). It encompasses all the processes involved in getting new software or hardware operating properly in its environment, including installation, configuration, running, testing, and making necessary changes. Specifically, it involves coding the system using a particular programming language and transferring the design into an actual working system. This phase of the system is conducted with the idea that whatever is designed should be implemented; keeping in mind that it fulfills user requirements, objective, and scope of the system. The implementation phase produces the solution to the user problem.

5.1 Software requirements

Python

Python is a versatile programming language known for its readability and extensive libraries, often used for web development, data analysis, artificial intelligence, and more.

MySQL

MySQL is a popular relational database management system, widely used in conjunction with web applications to store and manage data efficiently.

Django

Django is a high-level Python web framework that simplifies the development of web applications by providing built-in tools for handling tasks such as URL routing, database interaction, and user authentication.

HTML and CSS

HTML and CSS are fundamental languages for building websites: HTML provides the structure and content, while CSS controls the presentation and styling, enabling the creation of visually appealing and responsive web pages.

XAMPP Server

XAMPP is a free and open-source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages.

Visual Studio Code

VSCode is a popular source-code editor developed by Microsoft for Windows, Linux, and macOS, which supports various programming languages and features such as debugging, syntax highlighting, and code completion, making it a preferred choice for many developers working with Python, HTML, CSS, and other languages.

5.2 Modules and Libraries:

1. hashlib

This module in Python provides various hashing algorithms. It allows you to generate secure hash digests of data, such as passwords or other sensitive information.

2. Cryptography

This is a library for secure communications and cryptographic protocols. The line “from cryptography.fernet” import Fernet specifically imports the Fernet symmetric encryption algorithm, which is often used for encrypting and decrypting data securely.

3. Django

This is a high-level Python web framework that simplifies the development of web applications. The line from django.shortcuts import render is a typical import statement used in Django to import the render function, which is commonly used to render HTML templates with context data and return an HTTP response.

This module offers functions for padding and unpadding data according to various padding schemes, ensuring proper data alignment before encryption and after decryption.

4. padding Module

In Python, a padding module typically refers to a collection of functions or classes designed to add padding to data, particularly in the context of cryptography or data processing. Padding is often used to ensure that data fits into fixed-size blocks, which is common in encryption algorithms and network protocols.

5. cipher Module:

.In Python, a cipher module typically refers to a collection of functions or classes designed to perform encryption and decryption operations using cryptographic algorithms. Ciphers are essential components of cryptography, providing mechanisms for secure communication by transforming plaintext data into ciphertext and vice versa.

6. algorithms Module:

This module an refers to a collection of functions or classes that implement various algorithms for solving specific problems. These algorithms can range from simple tasks like sorting and searching to more complex computations like cryptographic functions or mathematical optimizations.

7. mode Module:

The mode is the value that appears most frequently in a dataset. A dataset can have one mode, multiple modes, or no mode at all. In Python, you can calculate the mode of a list of numbers using the mode() function from the statistics module

8. PBKDF2HMAC Module:

The PBKDF2 calculation function takes several input parameters: hash function for the HMAC, the password (bytes sequence), the salt (bytes sequence), iterations count and the output key length (number of bytes for the derived key).

9. HttpResponse Module:

The request module in Python is used to make HTTP requests to web pages. A web server usually sends an HTTP response back when it receives a request. The HTTP response comprises of status code, content, and encodings.

10. get_object_or_404:

The get_object_or_404 is a convenient shortcut function provided by Django, which is used for getting an object from a database using a model's manager and raising an Http404 exception if the object is not found.

6.3 Implementation steps

- Open command prompt and perform the given commands as mentioned in the Figure below

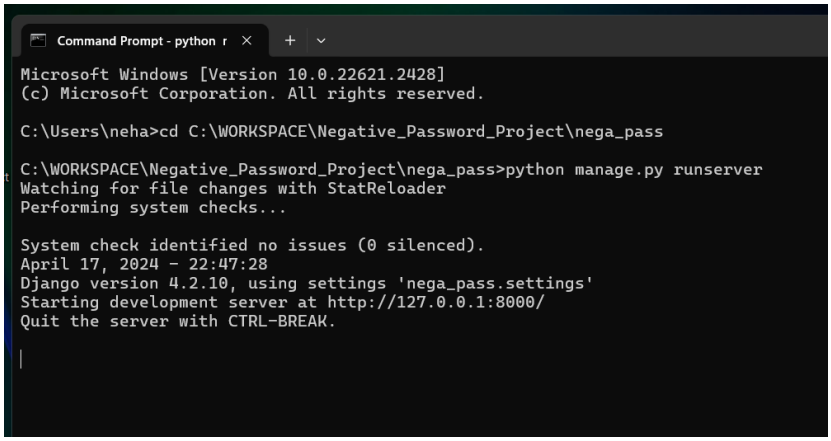


Figure 5.1 : Commands

- Open XAMPP server and start Apache and MySQL. To view the database, click on MySQL admin.

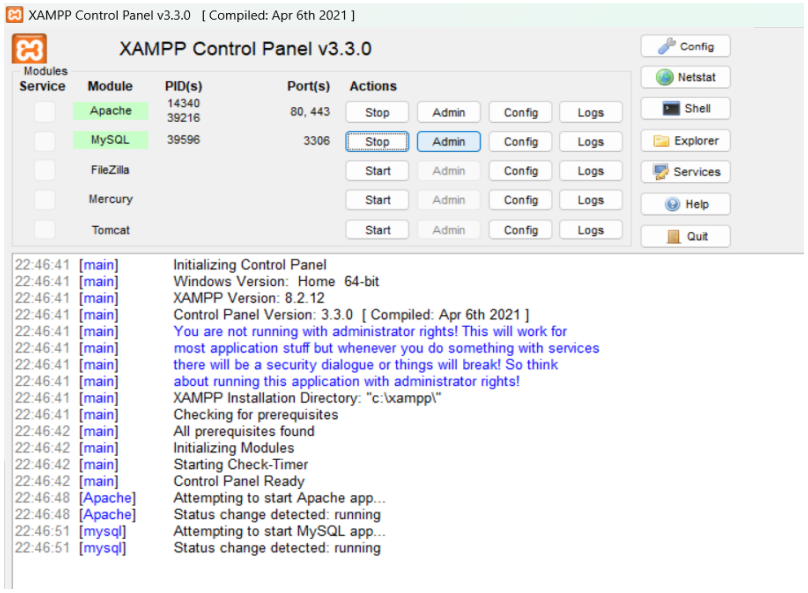


Figure 5.2 : XAMPP Server

- After going to the respective directory run the command “python manage.py runserver”.
- Open google chrome and type “localhost:8000”.

5.4 Coding

Main.py

```
import hashlib

import os

from cryptography.hazmat.primitives import padding,hashes

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.backends import default_backend

from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC



def generate_key(password, salt=b'salt'):

    kdf = PBKDF2HMAC(

        algorithm=hashes.SHA256(),

        length=32,

        salt=salt,

        iterations=100000,

        backend=default_backend()

    )

    key = kdf.derive(password.encode())
```

```
        print(type(key))
        return key
def getpasswords(password):
        salt = os.urandom(16)    # Generate a random salt
        key = generate_key(password, salt)
        hashed_password = hash_password(password)
        #negative_password = generate_negative_password(password)
        negative_password = generate_binary_negative_password(password)
        encrypted_password = encrypt_password(negative_password, key)
        decrypted_password = decrypt_password(encrypted_password, key)
      print("Hashed:", hashed_password.hex())
        print("Negative:", negative_password)
        print("Encrypted:", encrypted_password.hex())
        print("Decrypted:", decrypted_password.decode())    # Assuming the password is
in string format
        print("Key:", key)
return hashed_password, negative_password, encrypted_password
def hash_password(password):
        digest = hashlib.sha256()
```

```
        digest.update(password.encode())
        hashed_password = digest.digest()
        print(type(hashed_password))
        return hashed_password

def generate_negative_password(password):
        negative_password = ''.join(str(-ord(char)) for char in password)
        print(type(negative_password))
        return negative_password

def generate_binary_negative_password(password):
        negative_password = ''.join(str(-ord(char)) for char in password)
        binary_password = '*'.join(format(ord(char), '08b') for char in negative_password)
        return binary_password

def encrypt_password(password, key):
        # Initialize AES cipher in ECB mode
        cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
```

```
    encryptor = cipher.encryptor()
    padder = padding.PKCS7(128).padder()
    padded_password = padder.update(password.encode()) + padder.finalize()
    encrypted_password = encryptor.update(padded_password) + encryptor.finalize()
    print(type(encrypted_password))
    return encrypted_password


def decrypt_password(encrypted_password, key):
    # Initialize AES cipher in ECB mode
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_password = decryptor.update(encrypted_password) + decryptor.finalize()
    unpadder = padding.PKCS7(128).unpadder()
    unpadded_password = unpadder.update(decrypted_password) + unpadder.finalize()
    return unpadded_password
getpasswords("demo1234")
```

views.py

```
from django.shortcuts import render,HttpResponse,redirect,get_object_or_404
from home.models import UserDetails
from django.contrib import messages
import hashlib
import os
from cryptography.hazmat.primitives import padding,hashes
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC


# Create your views here.
def index(request):
    return render(request,'index.html')

def login(request):
    return render(request,'login.html')

def create(request):
    return render(request,'createaccount.html')

def logout(request):
    if request.method == "POST":
        email = request.POST.get('email')
        password = request.POST.get('password')

        try:
            user = UserDetails.objects.get(email = email)
```

```
        except UserDetails.DoesNotExist:
            user = None

        if user is not None:
            stored_encrypted_password = user.encrypted_password    # Convert to bytes
            stored_key = b'\xaa\x89t\xda\xc5\xdd#\xb3\x82\x80\xec\xff\xb70\x00`\xc8\xb0\xfa\xc8iU\x13\xd4\xa1\xcf\xac\xdc\xd3PF\xdc'
            decrypted_password = decrypt_password(stored_encrypted_password, stored_key)
            print(decrypted_password.decode())

            negative_password = generate_binary_negative_password(password)
            hashed_password = hash_password(password)
            print("Negative pass",user.negative_password)
            print("")

            if(hashed_password == user.hashed_password and decrypted_password == negative_password.encode()):
                messages.success(request, 'You have successfully logged in.')
                request.session['user_id'] = user.id
                request.session['email'] = user.email
                return redirect("/")
            else:
                messages.error(request, 'Wrong password')
                return redirect("/login")

        else:
```

```
                messages.error(request, 'Account not found')
                return redirect("/login")

        return render(request, 'logout.html')

def failed(request):
        if request.method == "POST":
                email = request.POST.get('email')
                password = request.POST.get('password')
                confirmPassword = request.POST.get('confirmPassword')
                hashed_password,negative_password,encrypted_password =
getpasswords(password)

                if UserDetails.objects.filter(email=email).exists():
                        messages.error(request, 'Email already exists. Please Log In')
                        return redirect("/create")

                if password!=confirmPassword:
                        return render(request, 'failed.html')
                else:
                        user = UserDetails( email = email,
                                                hashed_password = hashed_password,
                                                negative_password = negative_password,
                                                encrypted_password = encrypted_password)
                        user.save()
                        messages.success(request, 'Account created successfully!')
                        return redirect("/")
```

```
    return render(request, 'failed.html')


def logout_2(request):
    request.session.clear()
    return redirect('/')


# def generate_key(password, salt=b'salt'):
#     # Derive a 32-byte key using PBKDF2 with SHA-256
#     kdf = PBKDF2HMAC(
#         algorithm=hashes.SHA256(),
#         length=32,
#         salt=salt,
#         iterations=100000,
#         backend=default_backend()
#     )
#     key = kdf.derive(password.encode())
#     return key


def getpasswords(password):
    # salt = os.urandom(16)
    # key = generate_key(password, salt)
    key = b'\xaa\x89t\xda\xc5\xdd#\xb3\x82\x80\xec\xff\xb70\x00`\xc8\xb0\xfa\xc8iU\x13\xd4\xa1\xcf\xac\xdc\xd3PF\xdc'
    hashed_password = hash_password(password)
    negative_password = generate_binary_negative_password(password)
    encrypted_password = encrypt_password(negative_password, key)
    decrypted_password = decrypt_password(encrypted_password, key)
```

```
    return hashed_password, negative_password, encrypted_password

def hash_password(password):
    digest = hashlib.sha256()
    digest.update(password.encode())
    hashed_password = digest.digest()
    return hashed_password

def generate_negative_password(password):
    negative_password = ''.join(str(-ord(char)) for char in password)
    return negative_password

def generate_binary_negative_password(password):
    negative_password = ''.join(str(-ord(char)) for char in password)
    binary_password = '*'.join(format(ord(char), '08b') for char in negative_password)
    return binary_password


def encrypt_password(password, key):
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
    encryptor = cipher.encryptor()

    padder = padding.PKCS7(128).padder()
    padded_password = padder.update(password.encode()) + padder.finalize()

    encrypted_password = encryptor.update(padded_password) + encryptor.finalize()
    return encrypted_password
```

```
def decrypt_password(encrypted_password, key):
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_password = decryptor.update(encrypted_password) + decryptor.finalize()
    unpadder = padding.PKCS7(128).unpadder()
    unpadded_password = unpadder.update(decrypted_password) + unpadder.finalize()
    return unpadded_password
```

6. RESULTS

Given below are the screenshots of the project.

- **Register and login**

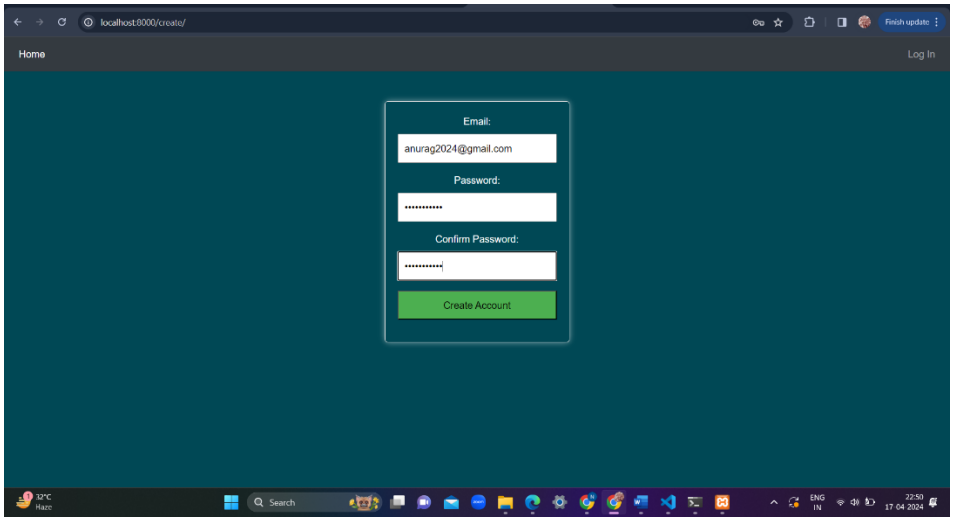


Figure 6.1 : Register Page

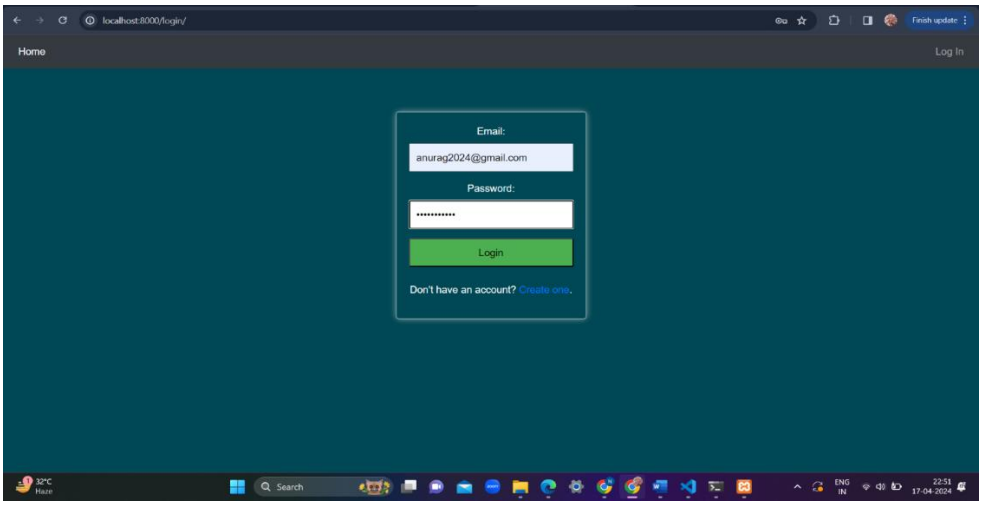


Figure 6.2 : Login Page

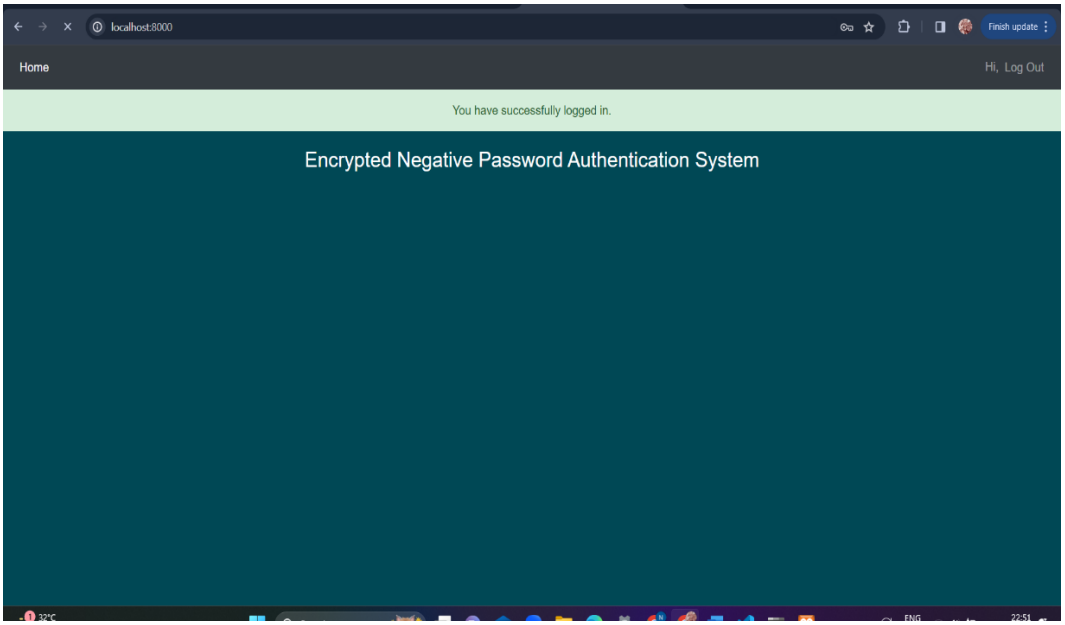


Figure 6.3 : Login Success Page

- **Database**

Mentioned below are the hashes, negative passwords and encrypted negative passwords of the users.

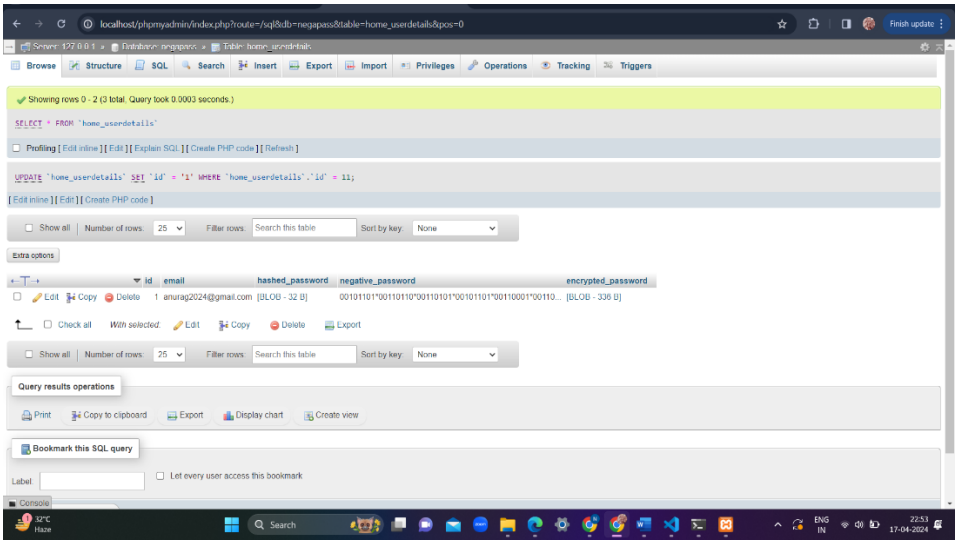


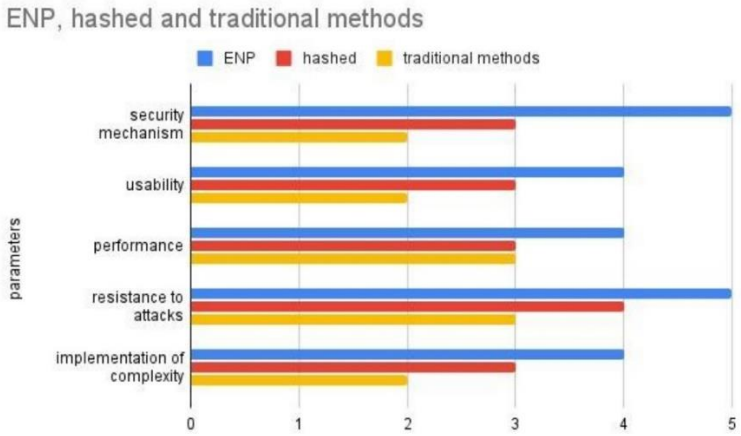
Figure 6.4 : Database page

7. Parameters:

- **Hashing Formula**: HashedPassword= HashFunction(Password+Salt).
- **Encryption Formula**: EncryptedData= EncryptionAlgo(Data, EncryptionKey)
- **Negative Authentication Criteria**: Define conditions like FailedAttempts > Threshold or SuspiciousBehaviour == True
- **Monitoring Criteria**: Define criteria for triggering alerts, such as Multiple_Failed_Logins or Unsusual_Access_Pattern.
- **Key Management:** KeyRotationPeriod= Interval for changing encryption/decryption keys.
- **Number of ENP Generated**: The number of ENPs converted from a plain password is calculated by

$$N_{ENPII} = \left\{ \binom{m-1}{1} * \binom{m-2}{1} * \binom{m-2}{1} \right\} * \binom{m-2}{1} * \prod_{i=2}^{m-1} \binom{i}{2} * m!,$$

- **Performances of ENP :** The time complexity of the generation of ENP is O(m2), and the time complexity of the verification of ENP is also O(m2), where m is the length of the hashed password. Since the length of the hashed passwords in the ENP is smaller, the generation and verification of the ENP are efficient.



Graph 7.1 : Parameter comparison to previous methods

8. CONCLUSION

In this project, we proposed a password protection scheme called ENP, and presented a password authentication framework based on the ENP. In our framework, the entries in the authentication data table are ENPs. In the end, we analyzed and compared the attack complexity of hashed password, salted password, key stretching and the ENP. The results show that the ENP could resist lookup table attack and provide stronger password protection under dictionary attack. It is worth mentioning that the ENP does not need extra elements (e.g., salt) while resisting lookup table attack. we provided an analysis and implementation of the ENP scheme, contributing an open-source proof of concept. In our analysis of the scheme, we conclude that it provides theoretical and practical security on par with existing widely used schemes. It does not seem to offer a significant edge over such schemes, but could be useful in scenarios where a database administrator is concerned about insecurely generating and/or storing external randomness. In future work, it could be good to evaluate how well ENP might pair as a layer with other schemes, and to conduct experiments on larger authentication databases.

7.1 Future Enhancements

In an era where digital security is paramount, password protection remains a critical concern. The Authentication by Encrypted Negative Password (ENP) framework introduces a novel approach to bolster password security. In this paper, we explore potential enhancements and integrations that can further fortify ENP, ensuring robust authentication while maintaining usability. From biometrics to blockchain, these strategies pave the way for a more secure and user-friendly authentication landscape. Some of the future enhancements of our project are as follows:

- Biometric Authentication Integration:

  Enhance security by integrating biometric authentication methods such as fingerprint recognition, facial recognition, or iris scanning.

Biometrics can complement ENP, providing an additional layer of user verification.

- Two-Factor Authentication (2FA):

  Implement 2FA alongside ENP.

  Users can receive a one-time code via SMS, email, or authenticator apps to verify their identity in addition to their ENP.

- Adaptive Authentication:

  Monitor user behavior and adjust authentication requirements accordingly.

  For example, if a user logs in from an unfamiliar location, prompt them for additional verification.

- Password Recovery Mechanism:

  Develop a secure password recovery process.

  Consider options like security questions, recovery email, or SMS-based recovery codes.

- Enhanced Negative Password Generation:

  Explore more sophisticated rules for creating negative passwords.

  For instance, use context-aware transformations based on user behavior.

- Blockchain-Based Authentication:

  Leverage blockchain technology for secure authentication.

  Store authentication data in a decentralized manner, reducing reliance.

- Behavioral Biometrics:

  Analyze user behavior patterns (typing speed, mouse movements) to verify identity. Behavioral biometrics can be used alongside ENP for continuous authentication.

- Passwordless Authentication:

  Investigate passwordless alternatives.

Options include WebAuthn (FIDO2), where users authenticate using biometrics or hardware tokens.

- Integration with Single Sign-On (SSO):

  Enable users to log in once and access multiple services seamlessly.

  SSO streamlines user experience while maintaining security.

- Usability Enhancements:

  Improve the user experience.

  Provide password strength feedback during account creation.

  Allow users to change their negative passwords easily.

- Security Audits and Penetration Testing:

  Regularly assess the system’s security.

  Conduct penetration tests to identify vulnerabilities.

- Mobile App Integration:

  Develop a mobile app for ENP-based authentication.

  Mobile apps can enhance usability and security.

9. REFERENCES

[1] Wenjian Luo, Yamin Hu, Hao Jiang, and Junteng Wang. Authentication by encrypted negative password. IEEE Transactions on Information Forensics and Security, 14(1):114–128, 2019. doi: 10.1109/TIFS.2018.2844854.

[2] Udi Manber. A simple scheme to make passwords based on one-way functions much harder to crack. Computers & Security, 15(2):171–176, 1996. ISSN 0167-4048. doi: https://doi.org/10.1016/0167-4048(96)00003-X. URL https://www.sciencedirect.com/science/article/pii/016740489600003X.

[3] Craig Webster. Securing passwords with salt, pepper and rainbows. Barking Iguana, 2009. URL http://www.barkingiguana.com/2009/08/03/securing-passwords-with-salt-pepper-and-rainbows/.

[4] A. Adams and M. A. Sasse, "Users are not the enemy," Communications of the ACM, vol. 42, no. 12, pp. 40–46, Dec. 1999.

[5] E. H. Spafford, "Opus: Preventing weak password choices," Computers & Security, vol. 11, no. 3, pp. 273–278, 1992

[6] Y. Li, H. Wang, and K. Sun, "Personal information in passwords and its security implications," IEEE Transactions on Information Forensics and Security, vol. 12, no. 10, pp. 2320–2333, Oct. 2017.

[7] D. Florencio and C. Herley, "A large-scale study of web password habits," in Proceedings of the 16th International Conference on World Wide Web. ACM, 2007, pp. 657–666.

[8] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor, "Designing password policies for strength and usability," ACM Transactions on Information and System Security, vol. 18, no. 4, pp. 13:1–13:34, May 2016.

[9] D. Wang, D. He, H. Cheng, and P. Wang, “fuzzyPSM: A new password strength meter using fuzzy probabilistic context-free grammars,” in Proceedings of 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Jun. 2016, pp. 595–606.