**ENCRYPTED NEGATIVE PASSWORD(ENP) AUTHENTIFICATION SYSTEM**

**1st Namrata Barua, 2nd Tanusree Saha, 3d Jui Pattanayak**

1st MCA student of JIS College Of Engineering, 2nd Assistant Professor

1st,2nd,3rdJIS College Of Engineering

1st [namratabarua102@gmail.com](mailto:namratabarua102@gmail.com) , 2nd [tanusree.saha@jiscollege.ac.in](mailto:tanusree.saha@jiscollege.ac.in),

3rd [jui.pattanayak@jiscollege.ac.in](mailto:jui.pattanayak@jiscollege.ac.in)

**Abstract**

Password authentication is one of the most promising authentication systems for security perseverance. Storing passwords securely is one of the most vital challenges. Our paper proposes the Encrypted Negative Password (ENP) password protection technique based on the Negative Database (NDB), cryptographic hash function, and symmetric encryption. This study proposes a password confirmations structure that can be efficiently integrated into current authentication systems and is meant for secure password storage. The client's plain password is encrypted in a hybrid approach. It also presents a password authentication framework based on the ENP. The secret key in the ENP is almost always different and does not need to be specially generated and saved because it is the hash value of each user's password. As a result, symmetric encryption can be utilized to protect passwords .

**Keywords:** *Plain password, Cryptographic hash function, Encrypted negative password (ENP), Symmetric key algorithm (AES)*

**1. Introduction**

A username and password combination is used to secure the majority of our online accounts. Our bank account information, purchasing history, or home address are just a few examples of the information that is protected by these passwords. By employing the social engineering strategies previously covered or by infecting the victim's device with malware that logs their behavior, including the letters they enter, an attacker may, for instance, obtain the victim's password by using one of the methods mentioned above. Sites that protect your accounts more thoroughly than others include some. If you have an online banking account, you might use several strong passwords, but if you create a social network account, you might not take the same security measures. In order to crack the passwords for more secure accounts, attackers will choose weaker accounts as their targets. There are some attacks such as brute force attacks involve the attacker making guesses at passwords until they locate the right one. This could entail guessing a string of characters or, as in the more specialized dictionary attack, compiling a list of passwords starting with the most popular. Passwords based on actual words and word combinations can be found in the dictionary used by attackers. Password cracking is one of the cyberattacks that are most prevalent today. For instance, many users choose their passwords based on familiar words and use the same password across many systems. The attacker employs a variety of techniques to obtain the credentials, including password guessing, shoulder surfing, and the use of password-cracking software to steal critical information. We should use strongly hashed encrypted passwords to get around this issue. It's challenging to separate passwords from ENPs is challenging due to the combined effects of hash functions and encryption work. The study and algorithm comparisons demonstrate that the ENP cloud resists attacks on lookup tables and provides greater password protection against dictionary attacks. Here, we use two procedures to create a strong password: first, we hash the password, and then we encrypt it. This paper describes and implements the Encrypted Negative Password (ENP) password storage and authentication scheme. We analyze the security, performance, and scalability of the scheme in comparison to currently used methods such as salting, key stretching, and peppering. We conclude that it provides a competitive level of security without requiring outside random values such as salt or pepper. We describe the scheme in this paper to perform our analysis and we discuss our implementation of it and our testing of the efficiency of our implementation. Secure password storage and authentication schemes are the backbone of the Internet. Any website which has user accounts should store its users’ data as securely as possible, since any passwords which are broken can be used to log into users’ accounts on other websites due to the commonality of password reuse. In addition, due to users choosing insecure passwords, simply hashing the given password is often not enough to prevent passwords from being discovered if an internal data table is stolen.Due to the importance of this problem, there have been several schemes which have been both proposed and used over the years. Some examples of such schemes beyond just hashing the password are salting, peppering, and key stretching. However, these schemes all have their drawbacks, such as vulnerability to certain attacks in the case of salting and difficulty of use in the case of key stretching.In this paper, we describe and implement the Encrypted Negative Password (ENP) password storage and authentication scheme. We analyse the security, performance, and scalability of the scheme in comparison to currently used methods such as salting, key stretching, and peppering. We conclude that it provides a competitive level of security without requiring outside randomness such as a salt or pepper.

**2. Background**

In this section, we give some examples of common schemes employed today as well as some of their strengths and weaknesses.

## 2.1 Hashed Passwords

One of the most basic forms of security for storing passwords is to simply hide them by hashing them. The hashed value that is obtained from hashing the registered password is directly stored in the database, and every time a user logs in, their password is hashed to determine if it matches the stored value. While extremely simple, this scheme is highly vulnerable against lookup table attacks, in which an attacker has precomputed the hashes of several common passwords and searches for the corresponding hashes in an exposed or insecure database.

## 2.2 Salted Passwords

Salting is a method used to give more security to basic hashing. Before a password is hashed and stored in the database, salt is added by appending random bits to the beginning or end of the password. The larger the salt that is added, the more security is gained. While salting protects against pre-computation attacks, it still has less than ideal security against dictionary attacks. In a dictionary attack, the attacker will try items in a ”dictionary” (list of possible passwords) in a brute-force fashion until the password is found. Salting helps against dictionary attacks, but more security is needed.

## 2.3 Key Stretching

Key stretching is a method in which stored hidden passwords are made much longer and more random. An example would be applying different hashes multiple times to expand and randomize the key. The number of times that we perform the hash would be denoted as the cost factor for the key stretching algorithm, codifying how computationally expensive it is to calculate an overall hash for each password. While key stretching (when combined with salted passwords) is highly effective in defeating lookup table attacks as well as dictionary attacks, it introduces more parameterization for the implementer and it increases the computational load on the security system.

## 2.4 Peppered Passwords

In a peppered password scheme, after hashing the password we encrypt the hash using a symmetric key, which we call the pepper. The pepper is similar to a salt but is shared across all stored passwords rather than being unique per password like a salt. The pepper is typically a long number (e.g., *>* 128 bits) such that brute force guessing it would take a long time. Importantly, the pepper is not stored in the database, but is instead stored inside a secret vault or hardware security module (similarly to how we might store a secret key used for encryption.

## 2.5 Encrypted Negative Password (ENP)

Encrypted Negative Password is the password storage scheme that we are focusing on in this paper. Like key stretching, it gives high security against lookup table attacks and dictionary attacks. It also attempts to eliminate some of the implementation/usability complexity of key stretching by removing the need for salting and parameterization.

**3. Methodology**

## Existing System

## In the real world, the existing system employs the approach that is the simplest of all the options. The database merely encrypts the plain-text password. Due to this strategy's high vulnerability, it could be quite simple to attack it and obtain the password. Popular hashing algorithms include the secure hash algorithm and the Message-Digest Algorithm. The hashing technique is another main component that remains in use today. This method offers better security than the previous one and only shows the password's hash value, not the actual password. However, the plain password can be obtained from the hashed values produced by both the rainbow table attack and the lookup table assault.

## Proposed System

This project suggests the symmetric encryption method, a cryptographic hash function, and Negative Database (NDB)-based Encrypted Negative Password (ENP) security approach. Additionally, a password authentication system built on the ENP is presented. The NDB is a novel security technique with numerous applications that draw inspiration from biological immune systems. Because it is the hash value of each user's password, the secret key in the ENP is almost always unique and does not need to be separately produced and maintained. As a result, the ENP enables the use of symmetric encryption to safeguard passwords. Our procedure consists of a number of steps. A cryptographic hash technique like SHA-256 is used to first hash the client's plain password. The login information is then hashed, and a negative password is created. The negative password is then further encrypted using hybrid cryptography to create an Encrypted Negative Password (ENP) using a symmetric-key strategy like AES. To further enhance security, many encryption iterations are performed. The suggested framework consists of two phases: registration and authentication. The system designer must first select a cryptographic hash function and a symmetric-key algorithm in order to protect passwords in an authentication data table using our framework. This requirement states that the size of the hash value of the selected cryptographic hash function must match the size of the selected symmetric key algorithm's key. The below figure shows a general block diagram describing the activities performed by this project.

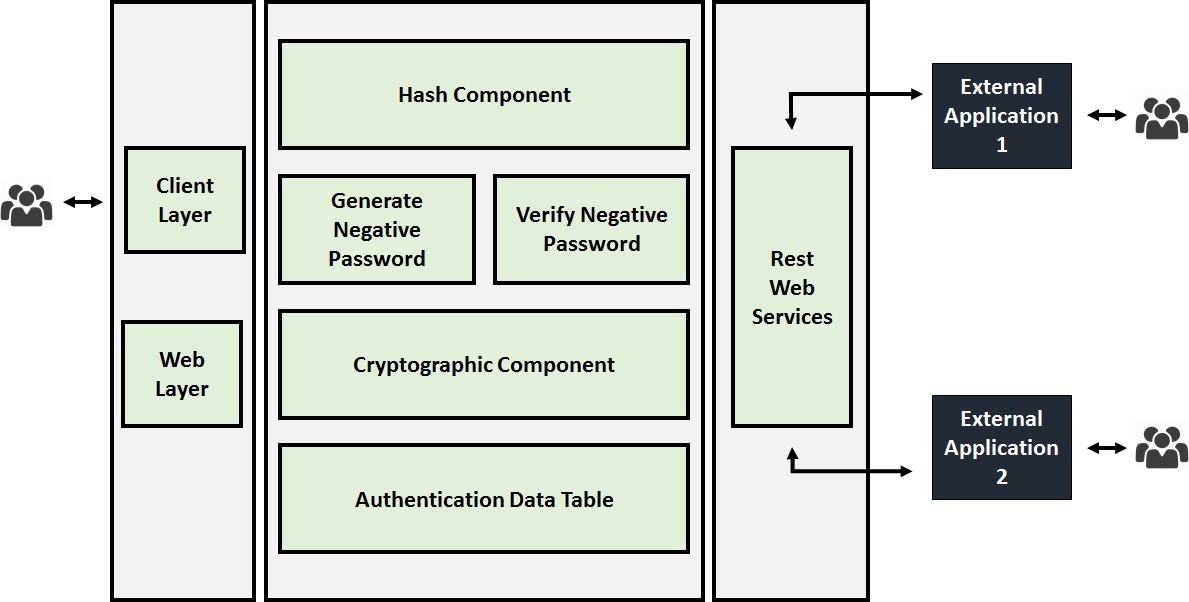


Figure 1: Block Diagram of ENP

## Registration Phase

1. User enters the plain text such as user name and password.

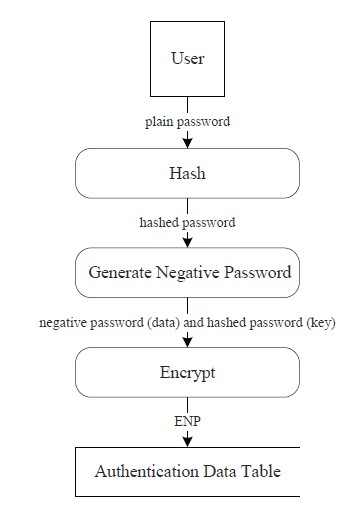
2. The system checks whether the user’s name exists in the database or not.

3. Then received password is hashed through a hashing algorithm such as SHA256.

4. Hashed password converted into negative password using NDB algorithm.

5. Encrypt the negative password using symmetric encryption algorithms such as AES.

6. Finally, store the encrypted password in the authentication table.

Figure 2: Flow diagram for the Registration phase

## Authentication Phase

1. If an existing user, the Username and password are transmitted to the server.

2. Verify the username and password, if existing the user’s name,

3. Search the ENP from the authentication table.

4. ENP is then decrypted it will get the hash value of the plain text.

5. If the hash value matches user can log in.

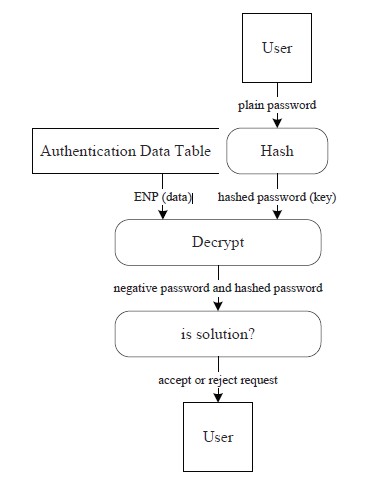


Figure 3: Flow diagram for the Authentication phase

**4. ENP(Encrypted Negative Password)**

The Encrypted Negative Password authentication scheme depends on three main concepts: **hashing, symmetric-key encryption, and negative databases**. The specific cryptographic hash function and encryption algorithm have to match in the sense that the output to the hash function should be the same size as a valid key for the encryption algorithm. This requirement comes from the fact that the hash of the password is used as the key for encryption so it is not necessary to store additional data to be used as a key. The central concept of the ENP scheme is that of a negative database. As such, we will begin by describing negative databases, and then we will describe the steps of the ENP registration and verification algorithms.

**4.1 Negative Databases**

Let *DB* be any database consisting of some set of length-n-bit strings. The central idea of a negative database is to hold the same information by instead storing the complement of the set of entries in *DB*. More precisely, if *U* is the set of all length-*n* bit strings, a negative database will store *U* − *DB*, the set of all length-*n* bit strings not in *DB*.The immediate problem with this idea is that for a constant-size database *DB*, the corresponding negative database as stated above will have exponentially many entries, quickly making it infeasible to store. In order to make this concept feasible, a significant amount of compression is needed. In particular, entries in the negative database *NDB* are stored as length-*n* strings using the symbols 0, 1, and ∗, where a ∗ indicates that either a 0 or a 1 can be put in its place. For example, 1 ∗ 0∗ represents all length-4 bit strings with a 1 as the first bit and a 0 as the third bit, of which there are 4. For the ENP algorithm, we will focus on the specific case in that *DB* contains exactly one entry. In this case, there are multiple efficient algorithms for finding a valid negative database. We will use two algorithms in this paper, the prefix algorithm and a slightly more complicated variant of it. See Table 1 for an example of a negative database given by the prefix algorithm.

|  |  |
| --- | --- |
| DB | NDB |
| 0000 | \*\*\*1  \*1\*0  \*010  1000 |

Table 1: Negative database using prefix algorithm

**4.2 Generation Algorithm**

Input: A password String ‘A’

Step 1: Compute the Hash code of the input String ‘A’ using SHA256

Algorithm hash 🡨sha256(A)

Step 2: Convert the ‘hash’ into binary format

binary 🡨 strToBinary(hash)

Step 3: Compute the Random permutation of ‘binary’

permutedBits 🡨 randomPermutation(binary)

Step 4: m 🡨 permutedBits.length ndb[]🡨empty[m]

for i 🡨 0 to m with a step size of 1

x[] 🡨 CREATESYMBOLS(m);

for j🡨0 to i with step size of 1

x[j] 🡨 permutedBits.charAt(j);

x 🡨 invertPermutation(x)

ndb[i] 🡨 x

result[][] 🡨 empty[m][m]

for i 🡨 0 to m with

step size of 1

result[i]🡨AES.encrypt(n)

return result

**4.3 Verification Algorithm**

Input: A password String ‘A’ and a negative database ‘NDB’

Step 1:Compute the Hash code of the input String ‘A’ using SHA256 Algorithm hash 🡨 sha256 (A)

Step 2:Convert the ‘hash’ into binary format

binary 🡨 strToBinary(hash)

Step 3: m 🡨 binary.length

decryptedNDB 🡨 empty[m][m]

k 🡨 0

for i 🡨 0 to m with step size of 1

decryptedNDB[k][i] 🡨 AES.decrypt(ndb[i])

k++;

Step 4: for i 🡨 0 to m with step size of 1

if (NUMBEROFSP(NDB[i])!= i)

return false;

Step 5: X[] 🡨 empty[m]

for i 🡨 0 to m with step size of 1

index 🡨 INDEXOFSP(NDB[i])

x[index] 🡨NDB[i][index]

for j 🡨 i+1 to m with step size of 1 NDB[j][index] = ‘\*’ if (hash == x)

return true

else return false

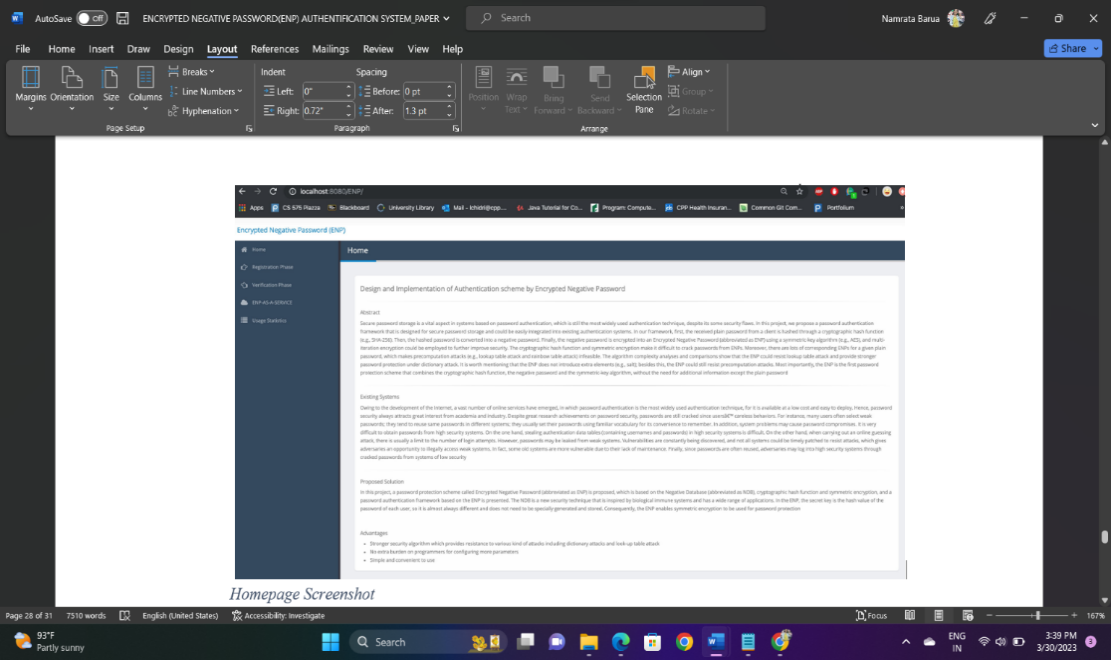
**5. Results and Discussion**

figure:4 home page

This is the home page where there is a description of our project and a description of our service is given from where the user can understand the Encrypted Negative Password security service and how it is working to provide facilities regarding Password Authentication.

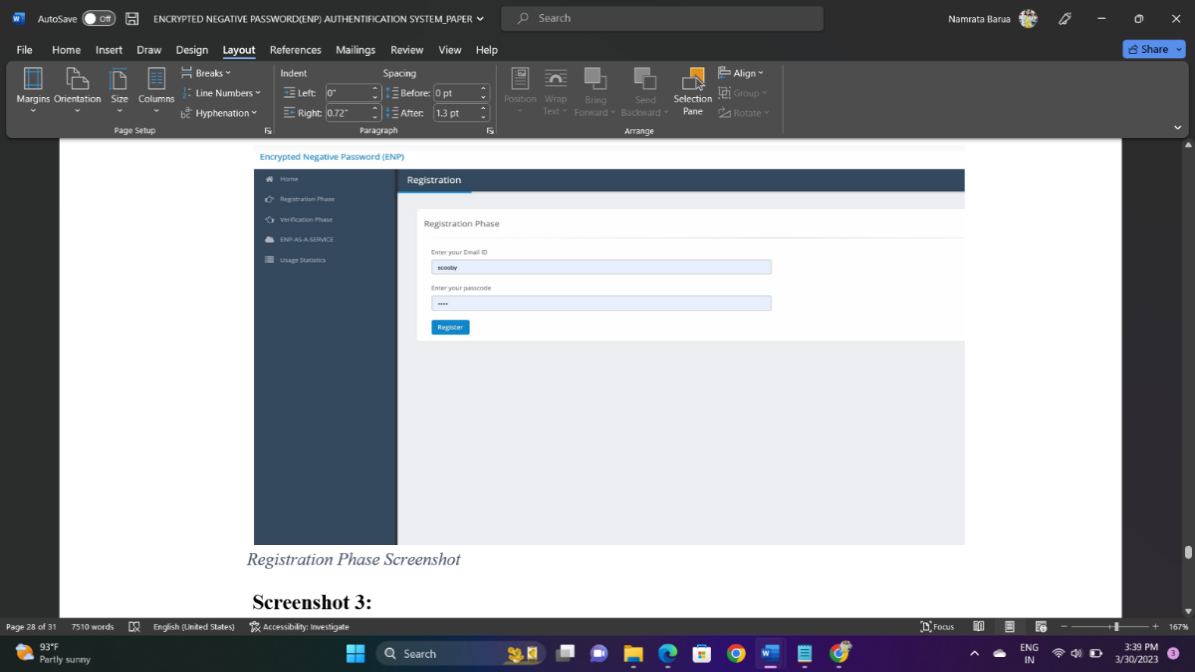


figure:5 Registration page

Result 2, where the user enters the plain text such as user name and password. The system checks whether the user’s name exists in the database or not. Then received password is hashed through a hashing algorithm such as SHA256. Hashed password converted into negative password using NDB algorithm. Encrypt the negative password using symmetric encryption algorithms such as AES. Finally, store the encrypted password in the authentication table.

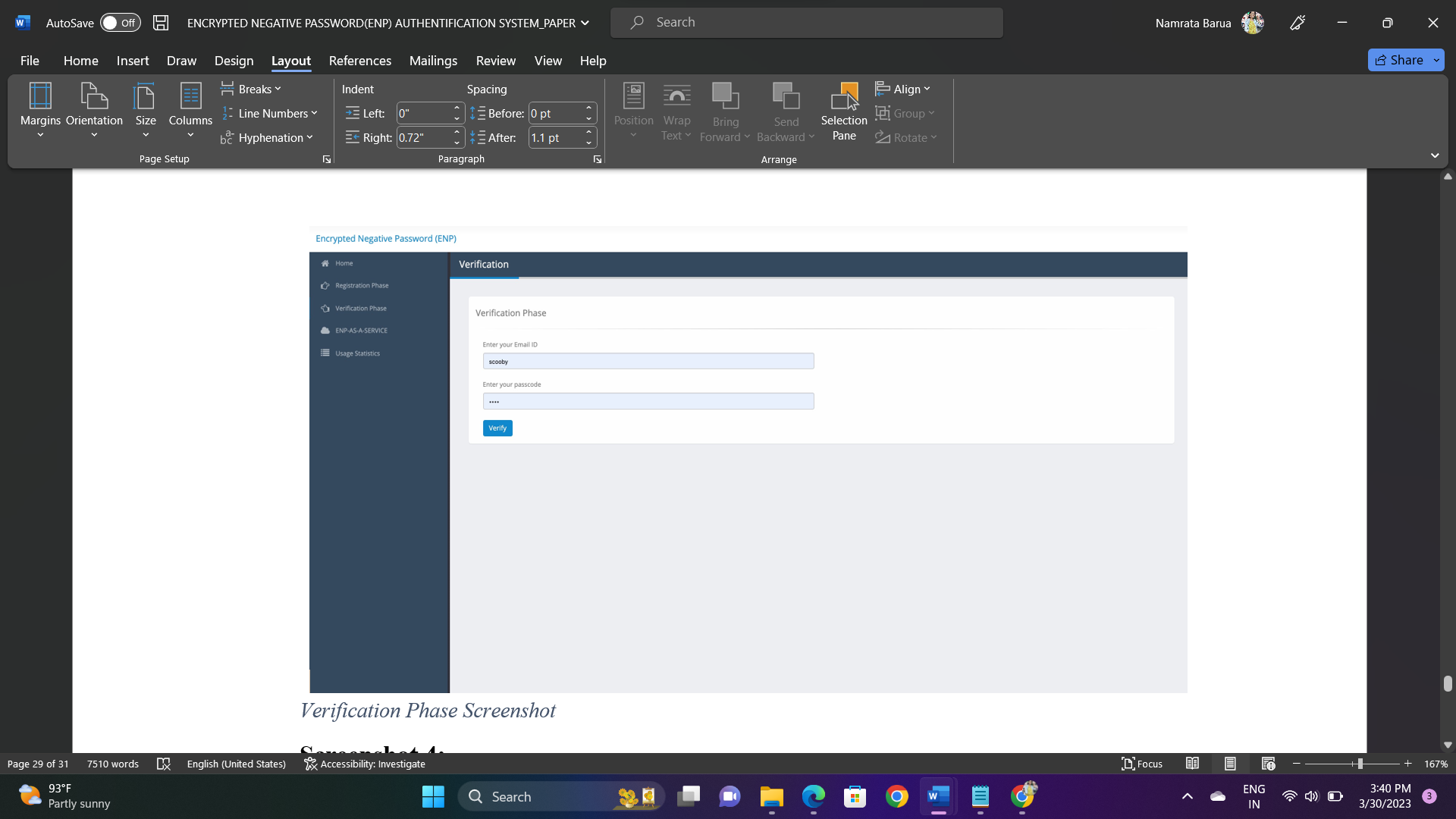


figure:6 Verification page

Result 3, here if an existing user, the Username and password are transmitted to the server. Verify the username and password, if existing the user’s name. Search the ENP from the authentication table. ENP is then decrypted it will get the hash value of the plain text. If the hash value matches user can log in.

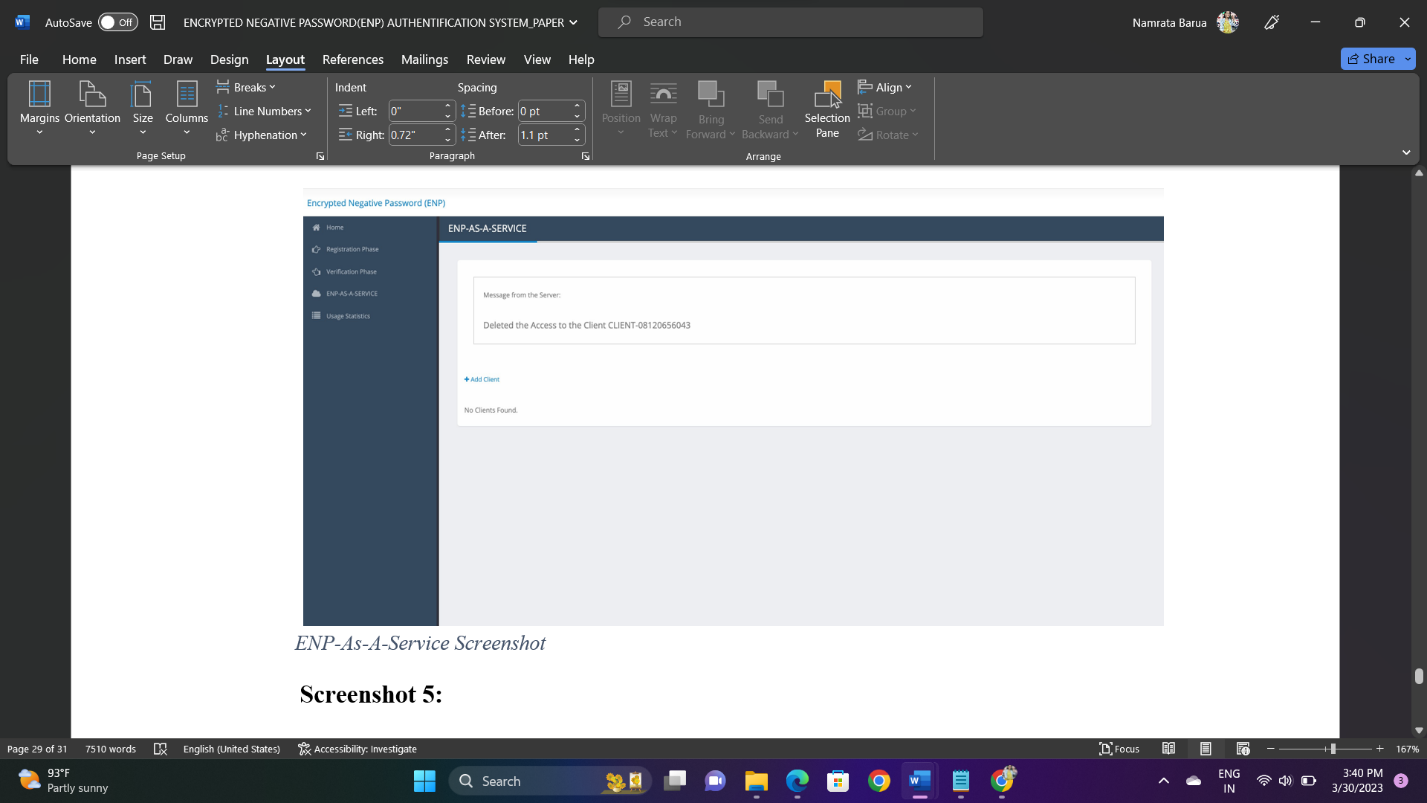


figure:7 ENP service page

Result 4, here the portal enables the owner of the product to share the solution we have proposed to the external applications. The user just has to add a new client by entering the client’s email ID. The portal will then generate the Client Identifier and the unique PIN and sends it across an email to the client. The client will also get the API details that have to be invoked to consume the web services we are exposing. The client identifier and the client PIN are mandatory to be included in each client request.

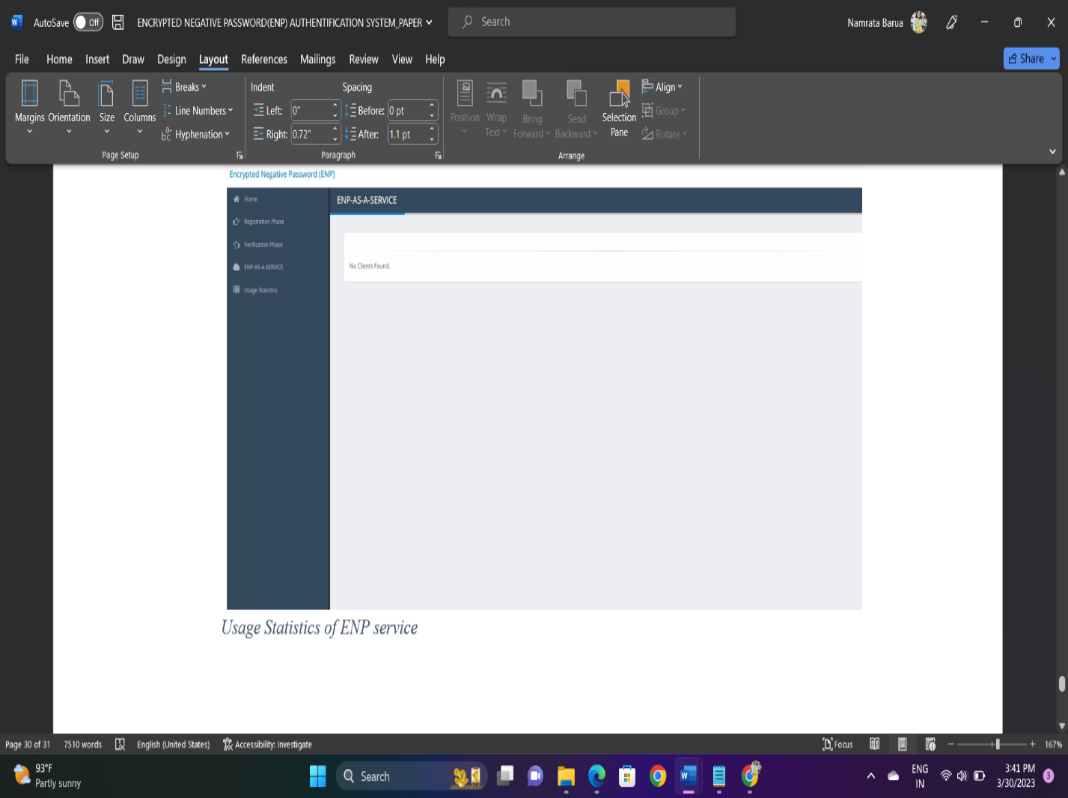


figure:8 User Statistics of ENP service page

Result 5, here the portal enables the users of the product to get access to the usage statistics of the clients on the APIs they have shared with them. Fundamentally, each and every client who requests access through the RESTFul web service will be logged into the database. The owner of the product will then be shown a cumulative report of the number of times each APIs have been consumed by each client. This report is useful especially when the owner of the product wants to charge the clients over the pay-as-you-go model.

**6. Conclusion**

We introduced a password authentication framework based on the ENP as well as a password protection method named ENP in this project. Entries in the authentication data table in our framework are ENPs. The attack complexity of the hashed password, salted password, key stretching, and ENP were all compared and evaluated in the conclusion. The outcomes demonstrate that the ENP could defend against dictionary and lookup table attacks while offering stronger password security. It's important to note that the ENP can withstand a lookup table assault without the requirement for additional components (like salt).To further strengthen password security, other NDB generation techniques will be researched and included to the ENP in the future. In addition, other methods will be added to our framework for password authentication, including multi-factor authentication and challenge-response authentication.

**7. References:**

1. J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, “Passwords and the evolution of imperfect authentication,” Communications of the ACM, vol. 58, no. 7, pp.

78–87, Jun. 2015.

1. M. A. S. Gokhale and V. S. Waghmare, “The shoulder surfing resistant graphical password authentication technique,” Procedia Computer Science, vol. 79, pp. 490–498, 2016.
2. J. Ma, W. Yang, M. Luo, and N. Li, “A study of probabilistic password models,” in Proceedings of 2014 IEEE Symposium on Security and Privacy, May 2014, pp. 689–704. [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.
3. E. H. Spafford, “Opus: Preventing weak password choices,” Computers & Security, vol. 11, no. 3, pp. 273–278, 1992.
4. 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.
5. 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.
6. 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.
7. 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.
8. H. M. Sun, Y. H. Chen, and Y. H. Lin, “oPass: A user authentication protocol resistant to password stealing and password reuse attacks,” IEEE Transactions on Information Forensics and Security, vol. 7, no. 2, pp. 651–663, Apr. 2012.
9. M. Zviran and W. J. Haga, “Password security: An empirical study,” Journal of Management Information Systems, vol. 15, no. 4, pp. 161– 185, 1999.
10. P. Andriotis, T. Tryfonas, and G. Oikonomou, “Complexity metrics and user strength perceptions of the pattern-lock graphical authentication method,” in Proceedings of Human Aspects of Information Security, Privacy, and Trust. Springer International Publishing, 2014, pp. 115– 126.
11. D. P. Jablon, “Strong password-only authenticated key exchange,” SIGCOMM Computer Communication Review, vol. 26, no. 5, pp. 5–26, Oct. 1996.
12. J. Jose, T. T. Tomy, V. Karunakaran, A. K. V, A. Varkey, and N. C. A., “Securing passwords from dictionary attack with character-tree,” in Proceedings of 2016 International Conference on Wireless Communications, Signal Processing and Networking, Mar. 2016, pp. 2301–2307.
13. A. Arora, A. Nandkumar, and R. Telang, “Does information security attack frequency increase with vulnerability disclosure? an empirical analysis,” Information Systems Frontiers, vol. 8, no. 5, pp. 350–362, Dec. 2006.
14. R. Song, “Advanced smart card based password authentication protocol,” Computer Standards & Interfaces, vol. 32, no. 5, pp. 321–325, 2010.
15. M. C. Ah Kioon, Z. S. Wang, and S. Deb Das, “Security analysis of MD5 algorithm in password storage,” in Proceedings of Instruments, Measurement, Electronics, and Information Engineering. Trans Tech Publications, Oct. 2013, pp. 2706–2711.
16. P. Oechslin, “Making a faster cryptanalytic time-memory trade-off,” in Proceedings of Advances in Cryptology - CRYPTO 2003. Springer Berlin Heidelberg, 2003, pp. 617– 630.
17. S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O’Hare, and K. Prole, “Advances in topological vulnerability analysis,” in Proceedings of 2009 Cybersecurity Applications Technology Conference for Homeland Security, Mar. 2009, pp. 124–129.