MARATHA VIDHYA PRASARAK SAMAJ NASHIK

K.K. Wagh Arts, Science and Commerce College ,Pimpalgaon Baswant,

Tal. Niphad, Dist.Nashik

RESEARCH WORK-II

Submitted in partial fulfillment of the requirements
For MSc (Computer Science)

Under the guidance of

"Smt. R.P Pagar"
"Smt. J. V.Daund"

RESEARCH TOPIC :

Cloud storage Password Security using Hashing and Salt Functionality

MARATHA VIDHYA PRASARAK SAMAJ NASHIK

K.K. Wagh Arts, Science and Commerce College ,Pimpalgaon Baswant,

Tal. Niphad, Dist.Nashik

CERTIFICATE

This is to certify that the " **Research work -II"** submitted by **1) Mr. Sandip Dnyaneshwar Baste, 2) Mr. Nikhil Adinath Aher** Research Project submitted during 2024-2025 academic year, in partial fulfillment of the requirements for the award of the degree of **MSc(Computer Science)** at K.K. Wagh Arts, Science & Commerce, College Pimpalgaon Baswant.

Internal Examiner **External Examiner** **(HOD)**

M. B.Sonawne

RESEARCH PROJECT REPORT

Name	**1.** Baste Sandip Dnyaneshwar **2.** Aher Nikhil Adinath
Project Title	Cloud storage Password Security using Hashing and Salt Functionality

Sr No.	Date	Task Done	Remark / Sign
1	08-02-2025	Choose the topic	
2	15-02-2025	What is Hashing and Salt Technology	
3	22-02-2025	Objectives of Hash & Salt in in Cloud Storage	
4	01-03-2025	Literature Review	
5	08-03-2025	Data collection	
6	15-03-2025	Actual work done	
7	22-03-2025	Bibliography	

Research project completion remark:

Name and signature of guide:

INDEX

ABSTRACT

In cloud computing, securing user data is paramount, especially when it comes to storing sensitive information like passwords. Storing plain text passwords in cloud databases is highly insecure and vulnerable to cyber attacks. This paper proposes a secure method for storing user passwords by converting them into hash values using a salted hashing technique. The process involves generating a unique salt for each user, appending it to the plain text password, and hashing the combined string using a cryptographic hash function. This ensures that even if the cloud database is compromised, the attacker cannot easily retrieve the original passwords. The methodology also includes a secure login process where the user's input password is hashed with the same salt and compared to the stored hash for authentication. This approach significantly enhances the security of user accounts in cloud environments.

KEYWORDS

Cloud Computing, Password Security, Salted hashing, Cryptographic Hash Function

---

1. INTRODUCTION

With the increasing adoption of cloud computing, the security of user data has become a critical concern. One of the most vulnerable aspects of cloud systems is the storage of user credentials, particularly passwords. Storing passwords in plain text or using weak encryption methods exposes users to significant risks, such as data breaches and unauthorized access. To address this issue, this paper introduces a secure password storage mechanism using salted hashing. The process involves adding a unique salt to each user's password before hashing it, making it computationally infeasible for attackers to reverse-engineer the original password. This method ensures that even if the cloud database is compromised, the stored passwords remain secure. The paper also discusses the implementation of this technique in a cloud environment and its effectiveness in preventing password-related attacks.

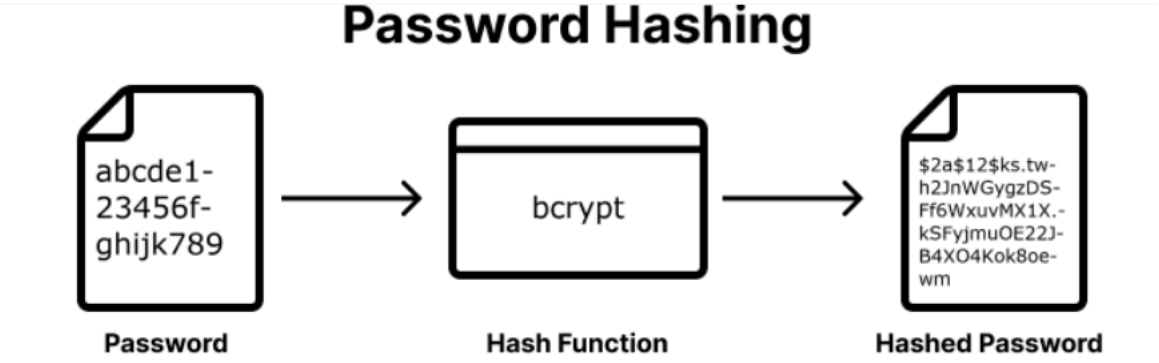
2. RELATED WORK

Several studies have explored methods for securely storing passwords in cloud environments. Traditional methods like MD5 and SHA-1 have been found vulnerable to brute-force and rainbow table attacks. More recent approaches, such as bcrypt, Argon2, and PBKDF2, incorporate salting and key stretching to enhance security. Salting involves adding a random string to the password before hashing, ensuring that even identical passwords produce different hash values. Key stretching increases the computational cost of hashing, making brute-force attacks more difficult. Previous research has demonstrated the effectiveness of these techniques in mitigating password-related vulnerabilities. However, the integration of salted hashing into cloud-based systems remains an area of active research. This paper builds on existing work by proposing a practical implementation of salted hashing for cloud user authentication.

1. Introduction to bcrypt and Salt

Password security is a critical concern in cloud computing, where user credentials must be stored securely to prevent unauthorized access. Traditional hashing methods like MD5 and SHA-1 are

vulnerable to brute-force and rainbow table attacks. To address these vulnerabilities, modern cryptographic techniques such as **bcrypt** combined with **salt** have become industry standards for secure password storage.



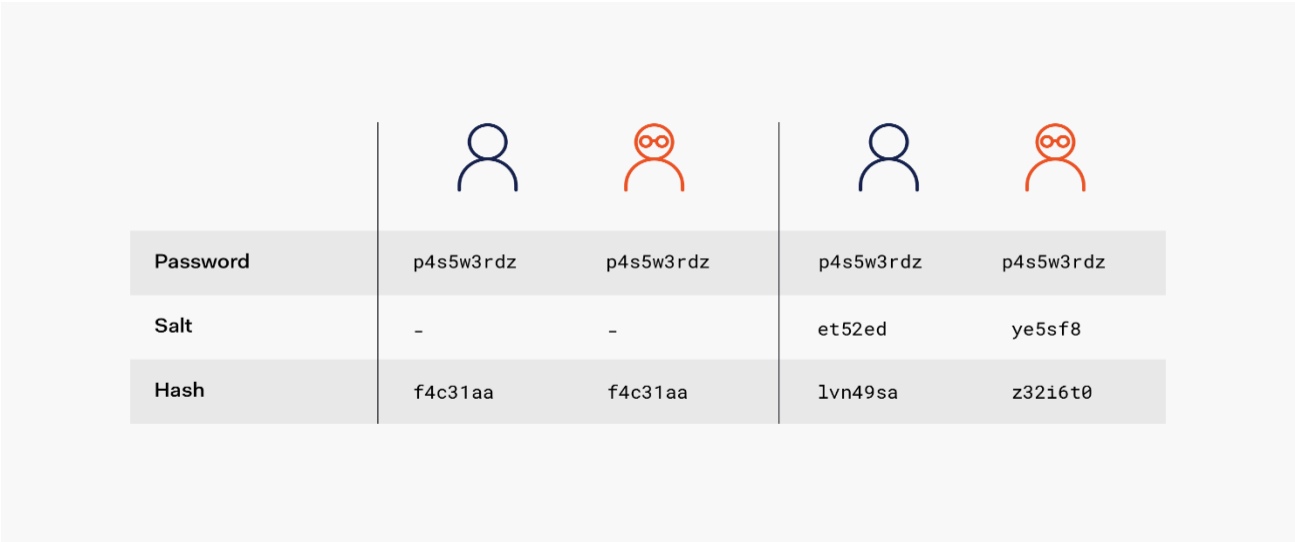
2. bcrypt: A Secure Password Hashing Algorithm

bcrypt is a **key derivation function** designed specifically for password hashing. It incorporates:

- **Salt**: A random string added to each password before hashing to prevent rainbow table attacks.
- **Work Factor (Cost Factor)**: Adjustable computational complexity to slow down brute-force attacks.
- **Adaptive Hashing**: Automatically increases security as hardware improves.

Advantages of bcrypt:

✓ **Built-in Salting** – bcrypt automatically generates and stores a unique salt for each password.
✓ **Future-Proof** – The work factor can be increased to maintain security against evolving threats.



3. The Role of Salt in Password Security

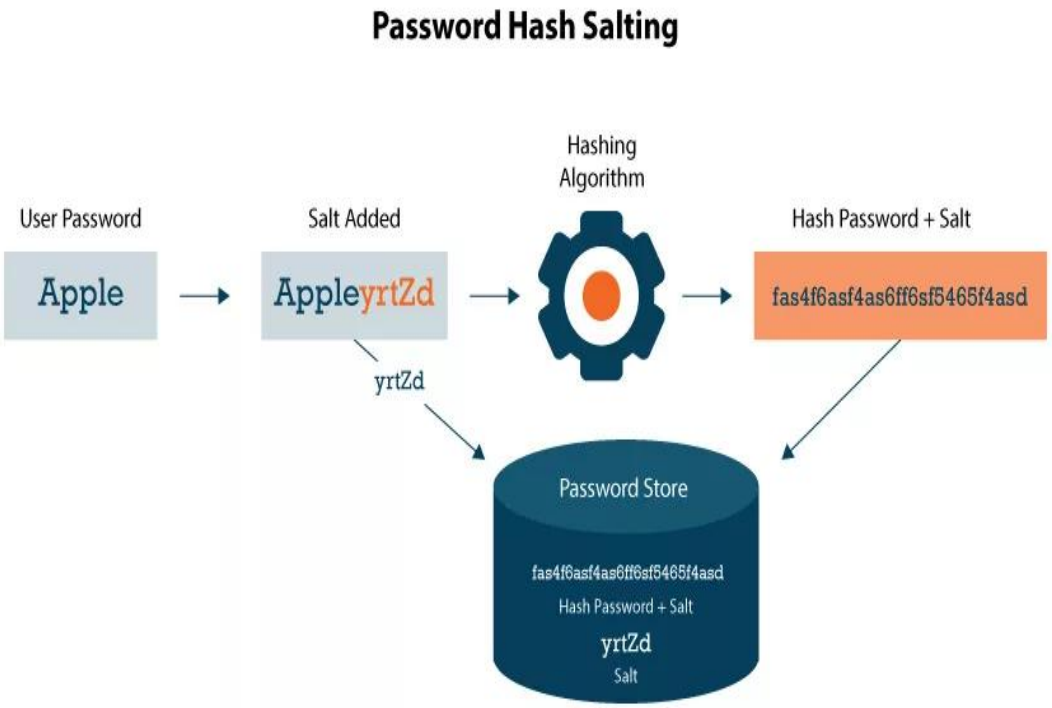
Salt is a **randomly generated string** appended to a password before hashing. It ensures:

- **Uniqueness**: Even if two users have the same password, their hashes will differ due to different salts.
- **Rainbow Table Resistance**: Attackers cannot use precomputed hash tables to reverse-engineer passwords.

- **Brute-Force Mitigation**: Each password requires individual cracking attempts.

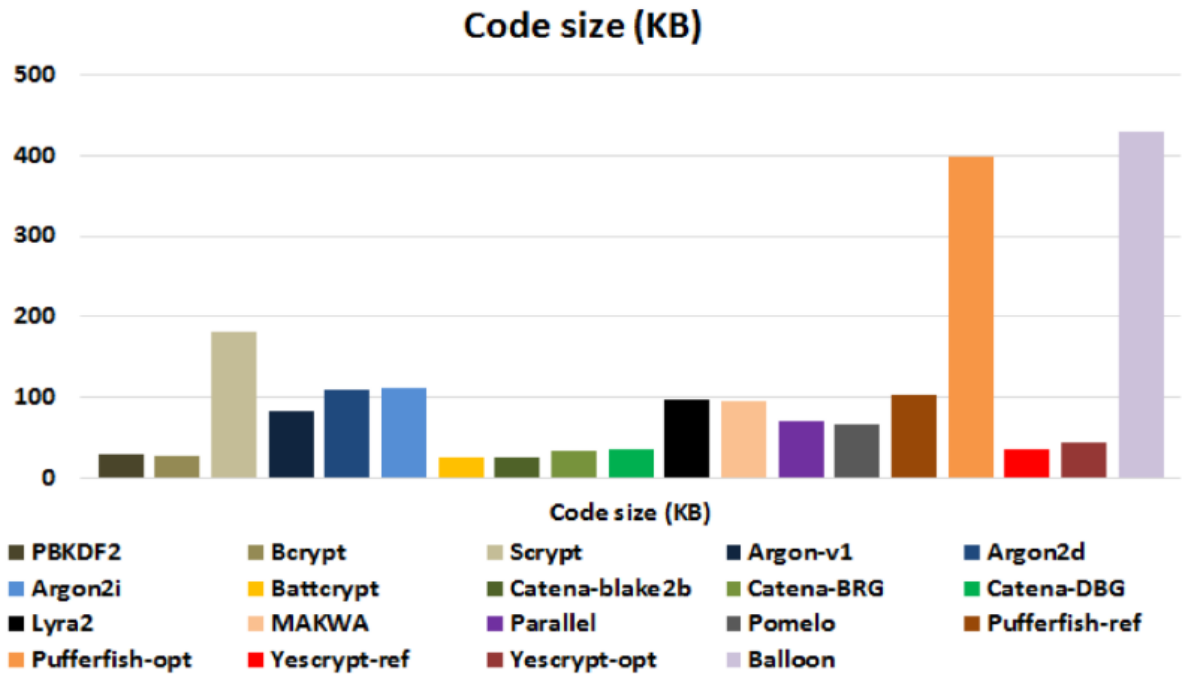
How bcrypt Uses Salt:

1. **During Registration**:
    - User submits a plain-text password (e.g., "mypassword123").
    - bcrypt generates a unique salt (e.g., "$2a$10$N9qo8uLOickgx2ZMRZoMy").
    - The password and salt are combined and hashed (e.g., hash = bcrypt("mypassword123" + salt)).
    - The **hash + salt** is stored in the database.



2. **During Login**:
    - User enters their password.
    - The system retrieves the stored salt from the database.
    - bcrypt rehashes the input password with the same salt.
    - If the new hash matches the stored hash, authentication succeeds.

4. Comparison with Other Hashing Methods



- **Brute-Force Resistance**: bcrypt's adjustable cost factor slows down attacks.
- **Rainbow Table Immunity**: Unique salts prevent precomputed hash attacks.
- **Database Breach Protection**: Even if hackers access hashes, they cannot reverse them.

5. Security Analysis

- **Brute-Force Resistance**: bcrypt's adjustable cost factor slows down attacks.
- **Rainbow Table Immunity**: Unique salts prevent precomputed hash attacks.
- **Database Breach Protection**: Even if hackers access hashes, they cannot reverse them.

6. Implementation in Cloud Environments

Cloud providers (AWS, Azure, GCP) recommend bcrypt for:

- **User Authentication Services** (e.g., AWS Cognito, Firebase Auth).
- **Database Security** (e.g., encrypting passwords in MongoDB, PostgreSQL).
- **Serverless Functions** (e.g., AWS Lambda for password hashing).

7. Challenges and Future Work

- **Performance Overhead**: bcrypt is slower than SHA-256, but this is intentional for security.
- **Quantum Computing Threats**: Future research may explore post-quantum hashing algorithms.
- **Multi-Factor Integration**: Combining bcrypt with OTP or biometrics for enhanced security.

3. METHODOLOGY

The proposed methodology for secure password storage and authentication in cloud computing involves the following steps:

1. **User Registration**:
    - The user creates an account and provides a plain text password.

- The backend system generates a unique salt using a cryptographically secure function like gensalt().
- The salt is appended to the plain text password, and the combined string is hashed using a secure hash function (e.g., bcrypt).
- The hash and salt are stored in the cloud database.

2. **User Login**:
   - During login, the user enters their password.
   - The backend retrieves the stored salt for the user from the database.
   - The salt is appended to the input password, and the combined string is hashed using the same hash function.
   - The resulting hash is compared to the stored hash. If they match, the user is authenticated.
3. **Security Measures**:
   - Use of strong cryptographic hash functions resistant to collision attacks.
   - Unique salts for each user to prevent rainbow table attacks.
   - Secure storage of salts and hashes in the cloud database.
   - Implementation of rate limiting and account lockout mechanisms to prevent brute-force attacks.
4. **Cloud Integration**:
   - The hashing and salting process is implemented on the backend server.
   - The cloud database stores only the hashes and salts, ensuring that plain text passwords are never exposed.

4. CONCLUSION

The proposed salted hashing technique provides a robust solution for securely storing user passwords in cloud environments. By generating unique salts for each user and hashing the combined password-salt string, the method ensures that even if the cloud database is compromised, the original passwords remain protected. The implementation of this technique in cloud systems significantly enhances security and mitigates the risks associated with password-related attacks. Future work could explore the integration of advanced cryptographic techniques, such as multi-factor authentication and zero-knowledge proofs, to further strengthen cloud security.

The use of salted hashing in cloud computing not only protects user data but also builds trust in cloud-based services. As cloud adoption continues to grow, the importance of robust security measures cannot be overstated. This paper demonstrates that salted hashing is a practical and effective solution for securing user passwords in cloud environments. By adopting this approach, cloud service providers can ensure the confidentiality and integrity of user data, even in the face of sophisticated cyberattacks.

5. ACKNOWLEDGEMENT

The authors would like to thank the developers of cryptographic libraries and cloud platforms for providing the tools necessary to implement secure password storage mechanisms. Special thanks to the research community for their contributions to the field of cloud security. This work was supported by my College teachers Team and my Friends.

We are grateful to our colleagues and peers for their valuable feedback and suggestions during the preparation of this paper. Their insights have greatly contributed to the quality and depth of this research. We also acknowledge the support of our families and friends, who have encouraged us throughout this project.

Finally, we extend our gratitude to the open-source community for developing and maintaining the cryptographic libraries used in this research. Their efforts have made it possible to implement secure and efficient password storage solutions in cloud environments. This work would not have been possible without their contributions.

6. REFERENCES

1. Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice*. Pearson.
2. Paar, C., & Pelzl, J. (2010). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer.
3. Provos, N., & Mazieres, D. (1999). "A Future-Adaptable Password Scheme." *Proceedings of the USENIX Annual Technical Conference*.
4. Biryukov, A., & Khovratovich, D. (2015). "Argon2: The Memory-Hard Function for Password Hashing." *IACR Cryptology ePrint Archive*.
5. OWASP Foundation. (2021). "Password Storage Cheat Sheet." Retrieved from https://owasp.org/www-project-cheat-sheets/.
6. NIST. (2020). "Digital Identity Guidelines: Authentication and Lifecycle Management." *Special Publication 800-63B*.
7. Krawczyk, H. (2016). "Salted Password Hashing - Doing it Right." *Cryptography Engineering*.
8. Ferguson, N., Schneier, B., & Kohno, T. (2010). *Cryptography Engineering: Design Principles and Practical Applications*. Wiley.
9. Percival, C. (2009). "Stronger Key Derivation via Sequential Memory-Hard Functions." *BSDCan Conference*.