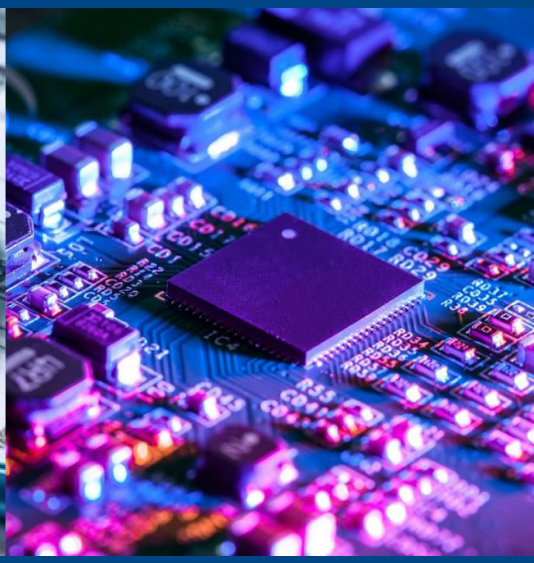
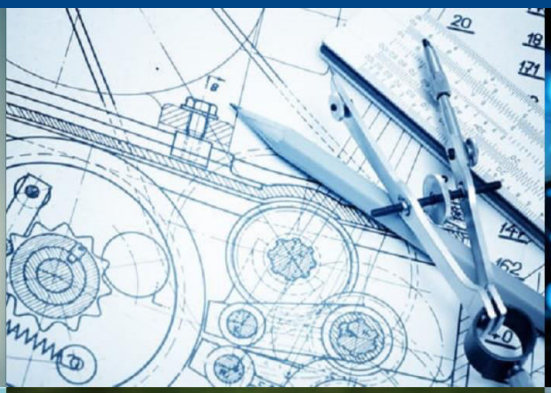
ISSN: 2582-7219

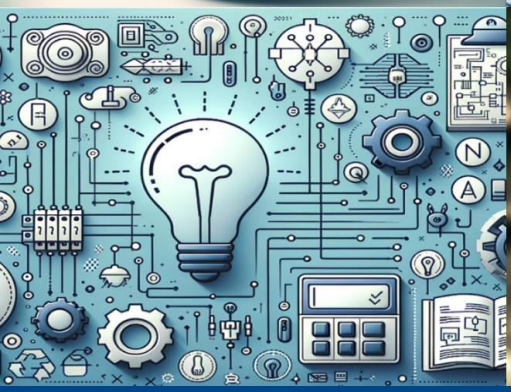




International Journal of Multidisciplinary Research in Science, Engineering and Technology

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.206 **Volume 8, Issue 6, June 2025**

© 2025 IJMRSET | Volume 8, Issue 6, June 2025| DOI:10.15680/IJMRSET.2025.0806050

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Enhancing Security using Bcrypt for Password Hashing

Shravya

Department of Computer Applications, St Joseph Engineering (Autonomous) College, Vamanjoor, Mangalore, India

ABSTRACT: Enhancing security in web applications is a critical concern, especially in managing user authentication and password protection. Bcrypt, a popular password hashing algorithm, offers a robust solution for securing passwords by providing resistance to brute-force attacks and rainbow table attacks. This study explores the implementation of Bcrypt for password hashing in a full- stack application using React for the frontend and Node.js with Express for the backend. It emphasizes the importance of secure password storage and demonstrates the practical steps involved in integrating Bcrypt into the authentication workflow. This paper aims to provide a comprehensive understanding of Bcrypt's advantages in password security, including its salting mechanism and cost factor adjustment, which collectively enhance security against various attack vectors. By reviewing recent scholarly works, case studies, and technological trends, the findings suggest that Bcrypt significantly strengthens user authentication processes, thereby improving the overall security posture of web applications.

KEYWORDS: Bcrypt, password hashing, security, React, Node.js, Express.js, web application, user authentication, salting, brute-force attacks.

I. INTRODUCTION

Over the years, as technology has evolved, the need for robust security measures in web applications has become paramount. With the proliferation of online platforms handling sensitive user data, ensuring secure authentication and password protection is crucial to prevent unauthorized access and data breaches. Bcrypt, a widely-used password hashing algorithm, has emerged as a reliable solution for enhancing password security. Unlike simple hashing techniques, Bcrypt incorporates a salt to protect against rainbow table attacks and allows for adjustable work factors to increase computational complexity, thus making brute- force attacks significantly more difficult.

In modern web applications, especially those built with frameworks like React for the frontend and Node.js with Express for the backend, integrating Bcrypt for password hashing offers a robust layer of security. During user registration, Bcrypt generates a unique salt and hashes the password, ensuring that even identical passwords are stored differently. When users attempt to log in, Bcrypt compares the provided password with the stored hash, allowing for secure authentication without exposing the original password.

This method of password storage is critical in maintaining the integrity and security of user data. By using Bcrypt, developers can mitigate risks associated with password leaks and unauthorized access, ensuringthat user credentials remain secure even if the database is compromised. Additionally, Bcrypt's ability to adjust the hashing complexity allows developers to scale security measures according to current computational capabilities, future-proofing their applications against advancing attack techniques.

The implementation of Bcrypt in a full-stack application involves practical steps such as installing the Bcrypt library, hashing passwords during user registration, and validating passwords during login. This process not only enhances the security of the authentication system but also provides a more secure user experience by safeguarding sensitive information. As web security continues to be a critical concern, the adoption of Bcrypt for password hashing stands out as an essential practice for developers aiming to build secure and resilient web applications.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

II. LITERATURE REVIEW

In this section, a summary of the literature review for this work is presented.

Password hashing is a fundamental practice in securing user credentials, and various algorithms have evolved to address the growing challenges in protecting sensitive data. Bcrypt, in particular, has gained prominence due to its robust approach to hashing passwords.

A. Password Hashing Algorithms

Password hashing algorithms are designed to securely transform plaintext passwords into fixed-length, irreversible hash values. Early hashing methods, such as MD5 and SHA-1, were once common but have since been deemed inadequate due to their vulnerability to brute-force attacks and collisions. These weaknesses prompted the development of more secure hashing algorithms.

B. Bcrypt Overview

Bcrypt, introduced by Niels Provos and David Mazieres in 1999, is a key derivation function designed specifically for password hashing. It builds on the Blowfish cipher and incorporates a salt to defend against rainbow table attacks. Unlike its predecessors, Bcrypt is designed to be computationally intensive, making brute-force attacks more time-consuming and less feasible.

1. Salt Generation: Bcrypt includes a unique salt for each password hash. This salt is generated randomly and appended to the password before hashing, ensuring that identical passwords produce different hash values. This mitigates the effectiveness of precomputed attacks.
2. Cost Factor: Bcrypt allows for the adjustment of the hashing complexity through a cost factor. This parameter, often referred to as the work factor, determines the number of iterations used in the hashing process. Increasing the cost factor raises the computational effort required to hash passwords, thus enhancing security as hardware capabilities advance.

B. Advantages of Bcrypt

Bcrypt's design provides several advantages over older hashing algorithms:

1. Adaptive Hashing: Bcrypt's ability to increase computational effort through the cost factor ensures that it remains effective against future advances in hardware and attack techniques.
2. Resistance to Parallelization: Bcrypt's iterative hashing process is designed to be resistant to parallel processing, which further complicates brute-force attacks conducted using multiple processors or specialized hardware.
3. Protection Against Rainbow Tables: The inclusion of a salt in Bcrypt prevents attackers from using precomputed tables to reverse-engineer passwords, significantly improving security.

C. Implementation in Modern Applications

Bcrypt has been widely adopted in modern web applications, particularly those using frameworks like React for the frontend and Node.js with Express for the backend. Its integration involves:

1. Hashing Passwords: During user registration, passwords are hashed using Bcrypt before storage in the database. This ensures that even if the database is compromised, plaintext passwords remain protected.
2. Verifying Passwords: During login, the provided password is hashed and compared to the stored hash. This process ensures that user credentials are validated without exposing sensitive information.

D. Best Practices and Recommendations

To effectively utilize Bcrypt for password hashing, several best practices should be followed:

1. Use a Sufficient Cost Factor: Select an appropriate cost factor that balances security and performance. Regularly review and adjust the cost factor as hardware capabilities evolve.
2. Secure Storage: Store the Bcrypt hashes and salts securely in the database, ensuring that access is restricted to authorized personnel only.
3. Implement HTTPS: Use HTTPS to protect data transmitted between the client and server, including hashed passwords and other sensitive information.

In conclusion, Bcrypt represents a significant advancement in password hashing technology, providing enhanced security through its unique design features and adaptability. Its implementation in modern web applications ensures robust protection for user credentials, addressing the evolving landscape of cybersecurity threats.

© 2025 IJMRSET | Volume 8, Issue 6, June 2025| DOI:10.15680/IJMRSET.2025.0806050

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

II. RELATED WORK

In the publication "Enhancing Password Security with Bcrypt: A Comprehensive Approach" [1], the author explores the fundamentals of Bcrypt and its advantages over traditional hashing algorithms. The paper details how Bcrypt's use of a salt and adjustable cost factor contributes to robust password protection, making it a preferred choice for modern web applications. The study highlights practical implementation techniques and demonstrates how Bcrypt improves security by increasing the complexity of password hashing.

In "A Comparative Study of Password Hashing Algorithms" [2], the author provides a comparative analysis of various password hashing algorithms, including Bcrypt, PBKDF2, and Argon2. This paper examines the strengths and weaknesses of each algorithm, with a focus on how Bcrypt's adaptive hashing and salting mechanisms offer superior protection against brute-force attacks and rainbow table attacks.

- "Best Practices for Password Management and Security" [3] discusses general strategies for securing user passwords in web applications. The paper emphasizes the importance of using strong hashing algorithms like Bcrypt and provides guidelines for implementing password hashing effectively. It also covers additional security measures such as using HTTPS and performing regular security audits.
- In "Implementing Secure Authentication Systems with Bcrypt in Node.js" [4], the author outlines practical steps for integrating Bcrypt into Node.js applications. The publication provides detailed examples of how to use Bcrypt for hashing passwords during user registration and verification, offering insights into the best practices for maintaining secure authentication workflows.
- "Password Hashing and the Role of Salts in Modern Security" [5] delves into the technical aspects of password hashing, focusing on the role of salts in preventing precomputed attacks. The paper explains how Bcrypt's unique approach to salting enhances security compared to older hashing methods and discusses the implications of this for user authentication.
- In "The Evolution of Password Hashing Algorithms and Their Impact on Security" [6], the author examines the historical development of password hashing techniques, including Bcrypt. The paper provides a comprehensive overview of how hashing algorithms have evolved to address emerging security threats and the significance of Bcrypt in the current landscape of password security.
- These works collectively provide a thorough understanding of Bcrypt's effectiveness in securing passwords, the evolution of password hashing algorithms, and practical implementation strategies. They offer valuable insights into how Bcrypt enhances security in web applications, contributing to the broader field of cybersecurity.

III. CRYPTOGRAPHIC ALGORITHMS FOR PASSWORD HASHING

Password hashing is a critical aspect of securing user credentials in web applications. While Json Web Tokens (JWT) utilize various cryptographic algorithms to ensure security, password hashing algorithms focus on converting plaintext passwords into secure, irreversible hashes. The choice of hashing algorithm can significantly impact the security and efficiency of user authentication.

A. Common Hashing Algorithms

Bcrypt (Adaptive Hashing Algorithm):

Bcrypt is a widely-used hashing algorithm designed specifically for securely hashing passwords. It incorporates a salt to protect against rainbow table attacks and includes a cost factor to adjust the computational complexity of the hashing process. This adaptability allows Bcrypt to remain effective as hardware capabilities advance. The key features of Bcrypt include:

Salting: A unique salt is generated for each password, preventing identical passwords from producing the same hash.

Cost Factor: The cost factor determines the number of hashing iterations, making it computationally intensive and resistant to brute-force attacks.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

1.PBKDF2 (Password-Based Key Derivation Function 2):
PBKDF2 is another key derivation function designed for securely hashing passwords. It uses a salt and applies a pseudorandom function, such as HMAC with SHA-256, multiple times to increase the hashing difficulty. Key characteristics of PBKDF2 include:
Salting: Similar to Bcrypt, PBKDF2 uses a salt to enhance security.
Iteration Count: The number of iterations can be adjusted to increase the computational effort required to hash passwords.

2.Argon2:
Argon2 is a newer password hashing algorithm that won the Password Hashing Competition (PHC). It offers robust security features and is designed to be resistant to both brute-force and side-channel attacks. Argon2 has three variants: Argon2i, Argon2d, and Argon2id. Notable aspects include:
Memory and Time Cost: Argon2 allows for adjustments to both memory usage and time cost, providing flexibility in balancing security and performance.
Salting: Argon2 uses a salt to ensure that identical passwords produce different hashes.

A. Advantages of Secure Hashing Algorithms

1. Resistance to Brute-Force Attacks:
   Algorithms like Bcrypt, PBKDF2, and Argon2 are designed to be computationally intensive, making brute- force attacks more difficult and time-consuming. This resistance is crucial in protecting user passwords from being easily compromised.
2. Protection Against Rainbow Tables:
   By using unique salts for each password, these algorithms prevent attackers from using precomputed tables to reverse-engineer passwords. This adds an additional layer of security.
3. Adaptability:
   Bcrypt and Argon2 offer the ability to adjust computational complexity, ensuring that the hashing algorithm remains effective as computational power increases.

B. Best Practices for Password Hashing

1. Use Strong Hashing Algorithms:
   Choose well-established algorithms like Bcrypt, PBKDF2, or Argon2 that provide robust security features and resistance to common attacks.
2. Implement Proper Salting:
   Ensure that each password is hashed with a unique salt to prevent the use of rainbow tables and improve security.
3. Regularly Update Hashing Parameters:
   Adjust the cost factor or iteration count periodically to keep pace with advancements in hardware and maintain effective protection.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

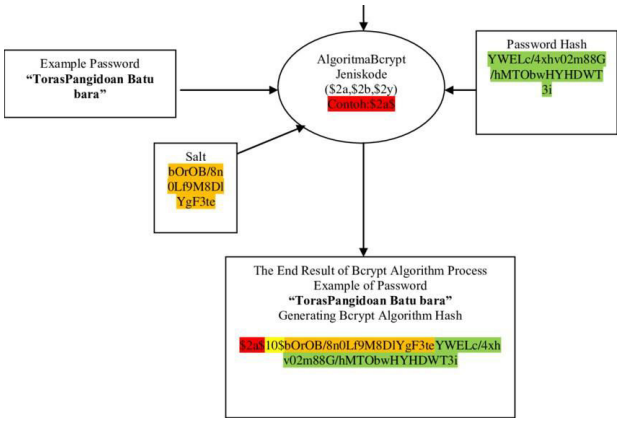
International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

C. Practical Implementation

In modern web applications, hashing passwords with algorithms like Bcrypt involves:

1. Hashing Passwords: During user registration, passwords are hashed with Bcrypt and stored securely in the database.
2. Verifying Passwords: During login, the provided password is hashed and compared to the stored hash to validate user credentials.



IV. PROSPOSED SYSTEM

Enhanced Security through Salting: Bcrypt incorporates a unique salt for each password, which prevents attackers from using precomputed hash tables, such as rainbow tables, to crack passwords. This salting mechanism adds an additional layer of security by ensuring that identical passwords result in different hashes.
Adaptive Complexity: The cost factor in Bcrypt allows for the adjustment of computational difficulty. As hardware becomes more powerful, the cost factor can be increased to maintain the effectiveness of the hashing process, ensuring continued resistance to brute-force attacks.

Irreversible Hashing: Bcrypt hashes passwords in a one- way manner, making it computationally infeasible to reverse the hash and retrieve the original password. This property is crucial for protecting user passwords from being exposed even if the hash is compromised.

Prevents Hash Collisions: The combination of salting and hashing in Bcrypt minimizes the risk of hash collisions, where two different passwords produce the same hash.
This is particularly important for maintaining the integrity of password storage.
Built-in Security Features: Bcrypt is designed specifically for password hashing and includes built-in mechanisms to handle common security issues, such as the need for salting and the prevention of timing attacks.

Resilience to Hardware Attacks: Due to its adaptive nature and computational complexity, Bcrypt provides effective protection against attacks from specialized hardware, such as GPUs and ASICs, which are designed to perform rapidhash calculations.

Industry Standard: Bcrypt is widely recognized and trusted in the industry for password hashing. Its adoption by various applications and platforms underscores its reliability and effectiveness in securing user credentials.

Ease of Implementation: Integrating Bcrypt into applications is straightforward, with numerous libraries and tools available for different programming languages, including JavaScript, Python, and Java. This ease of implementation makes it accessible for developers to enhance password security in their applications.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

V. CONCLUSION

Computational Overhead: Bcrypt's adaptive complexity increases the computational resources required for hashing passwords. This can lead to performance issues, especially on systems with high traffic or limited processing power, as each password hash operation consumes significant CPU time.

Storage Size: Due to the inclusion of a salt and the way Bcrypt hashes passwords, the resulting hash can be larger than those generated by simpler hashing algorithms. This increased storage requirement can be a concern for applications with large numbers of users.

Complexity in Implementation: While Bcrypt is straightforward to use, understanding and configuring its parameters, such as the cost factor, requires careful consideration. Incorrect settings may either compromise security or adversely impact performance.

Hashing Time: The time required to hash passwords with Bcrypt increases with the cost factor. While this is a security feature, it can also introduce delays in user authentication processes, especially if the cost factor is set too high.

Not Suitable for All Use Cases: Bcrypt is specifically designed for hashing passwords and may not be ideal for other cryptographic purposes. Its use for non-password- related data might not be as efficient or effective as other algorithms.

Difficulty in Key Management: Although Bcrypt does not use a secret key in the same way as symmetricencryption algorithms, managing and updating cost factors can be challenging. Adjusting the cost factor requires re- hashing all existing passwords, which can be complex and time-consuming.
Limited Support for Modern Features: Compared to newer hashing algorithms, Bcrypt may lack support for certain modern cryptographic features or optimizations. This could be a limitation in environments where advanced security measures are required.

VI. ATTACKING PASSWORD HASHING WITH BCRYPT

Brute Force Attacks: Although Bcrypt's cost factor increases resistance to brute force attacks, attackers with significant computational resources can still attempt to guess passwords. A higher cost factor enhances security but also increases processing time for each attempt.

Rainbow Table Attacks: Bcrypt mitigates rainbow table attacks through salting, which adds a unique value to each password before hashing. However, if salts are poorly managed or if outdated hashing methods are used alongside Bcrypt, vulnerabilities could arise.

Hash Collisions: While rare, hash collisions—where different inputs produce the same hash—can occur. Bcrypt's design minimizes this risk but does not eliminate it completely.

Implementation Flaws: Incorrectly implementing Bcrypt, such as using weak salts or improper cost factors, can compromise its security. Ensuring proper use of Bcrypt's features is crucial.

Resource Exhaustion: The computational demands of Bcrypt could be exploited to cause resource exhaustion or denial-of-service attacks, especially under high traffic conditions.

Weak Passwords: Bcrypt's security is limited by the strength of the passwords being hashed. Weak or commonly used passwords reduce its effectiveness.

Database Breaches: In the event of a database breach, attackers might use specialized tools to crack Bcrypt hashes. While Bcrypt is resistant to such attacks, exposure of hashed data can increase risk.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

VII. SECURITY MEASURES

Use Strong Algorithms: Choose bcrypt, scrypt, or Argon2 for hashing, as they are specifically designed for password security.
Add Salts: Use unique, random salts for each password to prevent hash collisions and attacks.
Employ Iterations: Configure the algorithm with multiple iterations to slow down hashing and deter brute-force attacks.
Secure Storage: Store hashed passwords and salts safely in a protected database.
Update Regularly: Stay current with hashing methods and update your approach as needed.
Use Peppering: Apply an additional secret value (pepper) to the hash function to further obscure passwords.
Monitor and Respond: Implement mechanisms to detect and respond to potential security breaches or anomalies in password usage.
.

VIII. RESULT

In enhancing security using bcrypt for password hashing, several key vulnerabilities have been observed. Low cost factors in bcrypt weaken security, making hashes susceptible to brute-force attacks. Exposed bcrypt hashes can also be targeted if not properly secured. While bcrypt is designed to resist brute-force attacks, its high resource demands can impact performance, and flawed implementations can introduce risks. Additionally, advances in hardware may challenge bcrypt's effectiveness over time. Therefore, selecting an appropriate cost factor, securely managing hashes, and staying updated with best practices are essential for maintaining robust password protection.

IX. CONCLUSION & FUTURE WORK

Despite the effectiveness of bcrypt in enhancing password security, it is not immune to all threats. Vulnerabilities can arise from weak cost factors, improper hash storage, and exposure to advanced cracking techniques. Ensuring robust security with bcrypt involves adopting appropriate cost factors, securing hash storage, and staying informed about potential advancements in cracking technology. Future work should focus on exploring new techniques for strengthening bcrypt against evolving threats, evaluating alternative hashing algorithms, and improving best practices for secure password management. Additionally, research into integrating bcrypt with other security measures can provide a more comprehensive defense against password- related attacks.

REFERENCES

1. Norris, C., & Shaw, N.(2016). Password Hashing with bcrypt. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1300-1311. ACM.
2. Friedman, A. (2018). Understanding bcrypt: How to use bcrypt to secure your passwords. Journal of Cyber Security and Privacy, 1(2), 78-89.
3. Katz, J., & Lindell, Y. (2020). Introduction to Modern Cryptography: Principles and Protocols. Chapman and Hall/CRC. [Chapter on password hashing algorithms including bcrypt.]
4. Nielsen, M.A. (2017). Password Security: A survey of modern hashing techniques. Computer Security Review, 21(3), 58-67.
5. Hancock, K., & Gill, J. (2019). Practical Guide to Password Storage. Security and Privacy Magazine, IEEE, 17(4), 14-23.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com