IMPLEMENTATION OF PASSWORD HASHING USING BCRYPT IN A WEB-BASED LOGIN SYSTEM

M. Shandito Reynaldi Marwanda
[1]Information System, Faculty of Engineering and Computer Science, Islamic University of Indragiri,
Email: dito7255@gmail.com[1]

ABSTRACT

The evolution of web-based systems has heightened the need for authentication mechanisms capable of safeguarding user data and credentials. A persistent issue in login system development is the storage of passwords in plaintext, which increases the risk of misuse should the database be accessed by unauthorized parties. This study aims to implement the BCrypt algorithm as a password hashing method within a PHP-based web login system. An experimental research methodology was employed, utilizing an approach involving direct implementation and testing, supported by a literature review on password security, hashing, BCrypt, and user authentication. Implementation involved using the `password_hash()` function with the `PASSWORD_BCRYPT` algorithm to generate password hashes prior to database storage, while the `password_verify()` function was used for the verification process. Testing was conducted across several scenarios: registration, logging in with a correct password, logging in with an incorrect password, and testing the same password across different hashing operations. The results demonstrate that BCrypt successfully converts passwords into hash values, ensuring that original passwords are not stored directly in the database. Correct passwords were successfully verified, granting user access, whereas incorrect passwords were rejected. Furthermore, identical passwords yielded different hash values due to the use of salts. Based on these findings, BCrypt serves as a viable password security method for web-based login systems, offering superior protection compared to plaintext password storage.

Keywords: BCrypt, Password Hashing, Login System, Data Security, PHP, Web-based System.

1 INTRODUCTION

The rapid advancement of information technology has driven the adoption of web-based systems across various sectors, including education, government, business, and public services. These systems offer users the convenience of quickly accessing information and services via the internet. In practice, many web systems incorporate a login feature to restrict access to specific data and services. Through the login process, the system identifies the user based on provided information, such as a username, email address, and password. Consequently, the security of the login system is a critical aspect to consider during web application development.

A password is a piece of information used for user authentication. Although simple, passwords are sensitive data because they can be used to gain access to a user's account and personal information. If passwords are stored insecurely, the risk of data leakage increases significantly in the event of an attack or unauthorized access to the database. A common error in application development is storing passwords as plain text. This method allows passwords to be immediately exposed if an unauthorized party gains access to the database. Such a situation can lead to various risks, including account takeover, theft of personal information, and misuse of user data.

To mitigate these risks, a password security method is required to transform the original password into a format that cannot be read directly. Hashing is a widely used approach for this purpose. Unlike encryption—which essentially allows data to be reverted to its original form using a specific key—hashing is designed as a one-way process. A user's password is processed into a hash value, which is then stored in the database. When the user logs in, the entered password undergoes the same processing mechanism, and the result is compared against the

stored hash value. Thus, the system does not need to store the user's actual password in the database.

BCrypt is a hashing algorithm specifically designed for password security. It is based on the Blowfish cipher and incorporates mechanisms such as "salting" and a "cost factor." The use of a salt introduces a unique variable to the hashing process, ensuring that the same password can yield different hash values. This can help reduce the effectiveness of attacks that utilize precomputed hash lists. Furthermore, BCrypt features an adjustable cost factor, allowing the hashing process to be made computationally more intensive. These characteristics make BCrypt suitable for implementation in systems requiring protection for user passwords.

Implementing BCrypt in web-based login systems involves not only the process of converting passwords into hashes but also the integration of this mechanism into the authentication workflow. When a user registers or creates an account, the entered password is not stored directly; instead, it is first processed using BCrypt. The resulting hash value is then stored in the database. Upon login, the system retrieves the stored password hash associated with the account and verifies the password provided by the user. If the provided password matches the stored hash, the user is granted access to the system; otherwise, authentication is denied.

This implementation is increasingly critical as threats to web application security continue to evolve. Attacks on user accounts do not always exploit vulnerabilities in the application's user interface but can instead leverage weaknesses in data management and authentication processes. Database breaches, password reuse across multiple services, and attack techniques such as brute-force and password cracking pose significant threats to account security. Therefore, password security should be integrated into the system design from the initial stages rather than treated as an afterthought once the application is complete. Utilizing an appropriate hashing algorithm is one measure to enhance the protection of user credentials.

When developing web-based systems, the implementation of BCrypt requires careful attention to ensure the authentication process functions correctly. The system must be capable of generating a hash when a password is created or updated and performing accurate verification during user login. Furthermore, selecting the appropriate "cost factor" requires balancing security levels with system performance. A cost factor that is too low may diminish protection against brute-force attacks, whereas a setting that is too high could result in excessive resource consumption and prolonged authentication times. Consequently, BCrypt must be implemented in a manner that aligns with the system's specific requirements and capabilities.

In light of these considerations, a study titled "Implementation of Password Hashing Using BCrypt in Web-Based Login Systems" was conducted to implement a password security mechanism using the BCrypt algorithm within the user authentication process. The study focuses on password hashing to ensure that user password information is not stored as plaintext in the database. This implementation is expected to provide a more secure mechanism for password storage and verification, while also mitigating the risks associated with unauthorized access to the database.

This research aims to provide insight into the application of BCrypt within web-based login systems, covering the entire workflow—from hash generation and storage in the database to password verification during user authentication. The findings are also intended to serve as a reference for developing web applications that prioritize security, particularly regarding the management of user credentials. By implementing an appropriate hashing mechanism, the security of login systems can be enhanced, thereby ensuring better protection for user data and access.

2 RESEARCH METHOD

This study employs an experimental research method involving the direct implementation and testing of a web-based login system. The research entails implementing the BCrypt algorithm for password hashing within an application developed using the PHP programming

language. In addition to direct implementation, a literature review is conducted to gather information and establish a theoretical foundation regarding password security, hashing, the BCrypt algorithm, user authentication, and the application of security functions in PHP. This approach ensures that the research process focuses not only on technical implementation but also rests on a theoretical basis relevant to the problem under investigation.

2.1 Literature Review

The first stage of this research involves conducting a literature review. This review entails gathering and examining various sources relevant to the research topic, such as scientific journals, research articles, official PHP documentation, and reference materials regarding secure password storage. The literature covers concepts such as web-based login systems, user authentication, hashing, salting, the BCrypt algorithm, and secure password storage methods.

The literature review also serves to understand the implementation of the `password_hash()` and `password_verify()` functions in PHP. Official PHP documentation explains that `password_hash()` can be used to generate a password hash using the BCrypt algorithm via the `PASSWORD_BCRYPT` constant, while verification is performed using the `password_verify()` function. PHP also provides an automatic salting mechanism during the hashing process, eliminating the need for developers to generate salts manually.

In addition to PHP documentation, this research draws upon OWASP guidelines for secure password storage. These guidelines state that passwords should not be stored in plain text but must be protected using an appropriate password hashing algorithm. BCrypt is one such algorithm suitable for password storage, with a "work factor" setting that should be adjusted according to the capabilities of the device or server.

2.2 System Design

The next stage involves designing a web-based login system. The system is designed using PHP as the primary programming language, with a database serving as the storage medium for user information. The login system comprises several key processes: user registration, password storage, the login process, and password verification.

During registration, users enter account details such as a username or email and a password. The entered password is not stored directly in the database; instead, the system first processes it using the BCrypt algorithm via PHP's `password_hash()` function. The resulting hash value is then stored in the database. This method ensures that the user's actual password is not stored directly within the database.

During the login process, users enter their username or email and password. The system then searches for user data based on the provided credentials. The entered password is verified using the `password_verify()` function by comparing it against the stored hash. If the password matches, the system grants the user access; conversely, if it does not match, the system rejects the login attempt and indicates that the authentication data is invalid.

2.3 BCrypt Implementation Using PHP

The implementation utilizes PHP's built-in password security functions. During account creation, the user's password is processed using `password_hash()` with the `PASSWORD_BCRYPT` algorithm. This function generates a hash that includes the information necessary for the verification process. PHP automatically generates a random salt if one is not provided manually, thereby simplifying the implementation and aligning with PHP's native mechanisms.

An example of the password hashing implementation is as follows:

```
$password = $_POST['password'];
$hash = password_hash($password, PASSWORD_BCRYPT);
```

The resulting $hash value is then stored in the password column of the database. Upon login, the system uses the password_verify() function to check whether the password entered by the user matches the stored hash.

```
$password = $_POST['password'];
$hash = $data['password'];
```

```
if (password_verify($password, $hash)) {
    echo "Login berhasil";
} else {
    echo "Password salah";
}
```

This implementation enables the system to perform verification without needing to know or store the user's original password. This aligns with the principle of password storage using one-way hashing functions, where the password is converted into a hash form that is not intended to be reverted to the original password.

2.4 Direct Testing

Once the system implementation is complete, the next phase involves direct system testing. This testing aims to verify that the BCrypt mechanism operates according to the design and to ensure that the registration and login processes correctly handle password hashing and verification.

Testing is conducted using several scenarios. The first scenario involves registering with a specific password and examining the data stored in the database. This test aims to confirm that the password is not stored as plain text but has been converted into a BCrypt hash value.

The second scenario involves logging in with the correct password. In this test, the system is expected to verify the stored hash and grant user access if the entered password matches.

The third scenario involves logging in with an incorrect password. This test aims to ensure the system can distinguish between correct and incorrect passwords and deny access when verification fails.

The fourth scenario involves testing the same password across multiple accounts. This test is conducted to observe the hash values generated by BCrypt. Since BCrypt employs a unique salt during the hashing process, the same password can yield different hash values; this is a crucial characteristic for secure password storage.

The results of each test are recorded and analyzed based on the alignment between the actual outcomes and the expected results. The testing focuses on the success of the hashing process, hash storage, password verification, and the system's ability to reject incorrect passwords.

2.5 Analysis of Test Result

The final stage involves analyzing the test results. Data obtained from the testing is directly compared against the research objectives to determine whether the implementation of BCrypt within the PHP-based login system has been successful. The analysis examines whether user passwords are successfully converted into hashes, whether these hashes are correctly stored in the database, and whether the system accurately verifies correct passwords while rejecting incorrect ones.

The test results serve to illustrate the role of BCrypt in enhancing password storage security within web-based login systems. This research goes beyond merely assessing the functionality of the login feature; it also examines password management throughout the lifecycle, from account creation to authentication. Consequently, this research methodology is expected to demonstrate the practical application of BCrypt in PHP-based applications and highlight the benefits of password hashing in safeguarding user credentials.

Overall, the research process comprises a literature review, system design, BCrypt implementation using PHP, direct testing, and analysis of the test results. These stages are executed sequentially to ensure the research yields data aligned with the objective: evaluating the implementation and outcomes of using BCrypt as a password hashing mechanism in web-based login systems.

3 RESULT AND DISCUSSION

3.1 System Implementation Results

Research results indicate that the BCrypt algorithm can be implemented in web-based login systems developed using PHP. Implementation involves utilizing the `password_hash()` function

to convert passwords into a hashed format before storing them in the database. During the login process, the system uses the `password_verify()` function to compare the entered password against the stored hash value.

The use of BCrypt in this study aims to avoid storing passwords in plain text. The hashing process yields a character string that cannot be directly used to determine the user's original password. PHP provides `PASSWORD_BCRYPT` as an algorithm for the `password_hash()` function, while `password_verify()` is used to verify the password against the stored hash.

During account registration, the user first enters the required information, including a password. The system then processes this password using the BCrypt algorithm. The resulting hash is stored in the user table within the database. With this mechanism, viewing the database directly does not reveal the user's password in its original form.

For instance, if a user enters the password "admin123," the system does not store that value directly. Instead, the password is processed to generate a BCrypt hash value, and it is this hash value that is stored in the database. Consequently, the data stored in the password column does not expose the user's actual password.

3.2 Hashing Process Test Results

The initial test was conducted to verify the proper functioning of the BCrypt password hashing process. Testing involved creating a new account and entering a specific password into the registration form. Following successful registration, the user data in the database was examined to inspect the value stored in the password column.

The test results revealed that the password was not stored as plain text but had been converted into a hash value. The value's format indicated the use of BCrypt; specifically, PHP documentation notes that the `PASSWORD_BCRYPT` option generates a hash with the `$2y$` identifier, allowing the format to serve as an indicator that the password was processed using BCrypt.

These findings demonstrate that the implementation aligns with the research objective: to protect passwords before they are stored in the database. Storing passwords as hashes is a more secure practice than storing them as plain text. Furthermore, OWASP emphasizes that passwords should not be stored in plain text and recommends using password hashing algorithms specifically designed to slow down password-guessing attempts.

3.3 Login Test Results with the Correct Password

A second test was conducted using a registered account and entering the correct password on the login page. During this process, the system retrieves user data based on the entered username or email. Once the data is located, the password entered by the user is verified using the `password_verify()` function.

The test results demonstrate that the system successfully recognizes the password matching the stored hash and grants access to the user. This process shows that the system does not need to store the actual password to perform authentication; instead, it simply uses the password entered during login and compares it against the stored hash using the BCrypt verification mechanism.

Using this method offers security advantages because the actual password does not need to be retrieved or displayed from the database. Verification is performed against the hash value, ensuring that password information remains protected during storage. This aligns with the principle of one-way hashing, where the password is processed using a function not designed to reverse the hash back into the original password.

3.4 Login Test Results with an Incorrect Password

The subsequent test involved entering a password different from the one used during registration. This test aimed to evaluate the system's ability to reject invalid authentication attempts.

When an incorrect password was entered, the `password_verify()` function returned a mismatch, resulting in the system denying user access. The system then displayed a notification indicating that the login attempt had failed or that the entered password was incorrect.

These results demonstrate that the verification mechanism functioned as intended. The system not only checks for the existence of the user account but also validates the provided password. Consequently, a user knowing only the username or email address cannot access the system without the correct password.

This test is a crucial component of the authentication system, as the login process must be capable of distinguishing between valid and invalid credentials. If the system were to grant access despite an incorrect password, the authentication mechanism would fail to provide adequate protection for user accounts.

3.5 Test Results for Login Using the Same Password

Subsequent testing involved using the same password for the hashing process on multiple occasions. The objective of this test was to observe BCrypt's characteristics in generating hash values.

The test results showed that the same password could yield different hash values when processed at different times. This variation occurs because BCrypt employs a "salt" during the hashing process. A salt is a random value used in hash generation, ensuring that the same password does not consistently produce an identical hash value. OWASP explains that using a unique salt for each password helps prevent attacks utilizing precomputed tables—such as rainbow tables—and makes the process of cracking multiple hashes more difficult.

Despite generating different hashes, the original password can still be correctly verified using `password_verify()`. This demonstrates that the system does not compare passwords based on the similarity of the hash text itself, but rather utilizes information contained within the hash to perform the verification process.

3.6 Discussion on BCrypt Implementation Security

Based on the test results, implementing BCrypt enhances security compared to plaintext password storage mechanisms. In systems using plaintext storage, anyone gaining access to the database can immediately view users' passwords. Conversely, in systems using BCrypt, the data obtained is in the form of a hash, meaning the original password is not directly accessible. OWASP states that storing passwords in plaintext can compromise the system and recommends using specialized password hashing algorithms to protect passwords.

BCrypt is characterized as a password hashing algorithm designed to be computationally more intensive than standard, high-speed hashing functions. This characteristic is significant because password attacks typically involve repeatedly attempting numerous possible passwords. A slower algorithm increases the computational cost for attackers attempting to brute-force the obtained hashes. OWASP recommends using a work factor appropriate for the server's capabilities when employing BCrypt.

However, implementing BCrypt does not render the system entirely free from security risks. Login system security is also influenced by other factors, such as user password strength, protection against repeated login attempts, connection security, session management, input validation, and protection against other web application vulnerabilities. Therefore, BCrypt should be viewed as one layer of protection within the system's overall security mechanism.

3.7 Analysis of Research Results

Overall, the test results indicate that the implementation of BCrypt in a PHP-based login system successfully performs the expected core functions: converting passwords into hashes, storing hashes in the database, verifying correct passwords, and rejecting incorrect ones. Utilizing PHP's built-in functions also simplifies implementation, as the hashing and verification processes are already provided by the programming language itself.

These results demonstrate that implementing password hashing is a crucial step in building a more secure authentication system. Passwords are no longer stored as human-readable data when the database is inspected. Furthermore, BCrypt’s use of salts ensures that identical passwords generate unique hashes, thereby offering additional protection against attacks relying on precomputed hashes.

Thus, this research shows that BCrypt is a viable method for securing passwords in PHP-based web login systems. The implementation successfully executed password hashing and verification processes in accordance with system requirements. Empirical testing results also corroborate literature findings suggesting that passwords should not be stored in plain text but rather through hashing mechanisms specifically designed for authentication purposes.

Nevertheless, there remains scope for further development. Future research could compare BCrypt with other password hashing algorithms—such as Argon2id—in terms of processing time, resource utilization, and security levels. Additionally, testing could be expanded to include broader scenarios, such as repeated login attempts, brute-force attacks, "work factor" adjustments, and the impact of user volume on system performance.

4 CONCLUSION

Based on research regarding the implementation of password hashing using BCrypt in web-based login systems, it can be concluded that the BCrypt algorithm is effectively applicable to login systems developed using PHP. Implementation involves utilizing the `password_hash()` function to convert user passwords into a hashed format before database storage, and the `password_verify()` function to verify passwords during the login process.

Direct testing demonstrates that user passwords are no longer stored as plain text but rather as hashes that are not directly readable. Testing also confirms that correct passwords are successfully verified—granting user access—while incorrect passwords are rejected by the system. Furthermore, BCrypt's use of a salt mechanism ensures that identical passwords yield different hash values across separate hashing operations, thereby providing an additional layer of protection against the misuse of hashed data.

Based on the literature review and implementation results, employing BCrypt enhances password management security in web-based login systems compared to storing passwords directly in the database. However, BCrypt is not the sole determinant of system security. Login security also relies on input validation, robust session management, secure connections, login attempt limits, and strong password policies.

Thus, this research demonstrates that implementing BCrypt with PHP is a viable method for securing user passwords in web-based login systems. This approach provides a more secure mechanism for password storage and verification and can serve as a foundation for developing authentication systems that prioritize user data security.

AKNOWLADGEMENT

The author extends sincere gratitude to the academic supervisor for providing guidance, direction, feedback, and support throughout the preparation and completion of this research. Thanks to the knowledge, time, and advice provided, the research titled "Implementation of Password Hashing Using BCrypt in a Web-Based Login System" was successfully completed.

The author also wishes to thank everyone who provided assistance and support—whether directly or indirectly—during the research and the preparation of this journal article. May all the help and kindness extended be rewarded in the best possible way.

REFERENCE

[1] Nugroho Dwi Aji, Tomi Tri Sujaka, Husain, Ondi Asroni, and Kurniadin Abd. Latif, "Analisis Dan Implementasi Algoritma Bcrypt Dengan Affine Cipher Untuk Pengamanan Password Pada Aplikasi Web," *Cyber Secur. dan Forensik Digit.*, vol. 8, no. 1, pp. 1–9, 2025, doi: 10.14421/csecurity.2025.8.1.5076.

[2] N. Jannah, L. Istianah, M. Tahir, and K. Kunci, "Implementasi Cryptography Based Multilayer Authentication pada Sistem Login Berbasis Web Menggunakan Bcrypt Hashing dan One Time Password (OTP) STATUS ARTIKEL," *Surabaya J. Sist. Cerdas dan Rekayasa*, vol. 8, no. 1, pp. 2656–7504, 2026.

[3] A. Saputra and R. Kurniati, "Aplikasi Penjualan Udang Pepai Berbasis Web Dengan

Keamanan Hash Password Menggunakan Algoritma Bcrypt," *JATI (Jurnal Mhs. Tek. Inform.*, vol. 10, no. 1, pp. 726–733, 2026, doi: 10.36040/jati.v10i1.16883.

[4] S. Erdi, P. Sohidin, H. Prasetyo Utomo, and A. Ahmadi, "Penerapan Algoritma Bcrypt untuk Pengamanan Password pada Sistem Informasi Akademik (SIAK) (Studi Kasus : Universitas Langlangbuana)," *Infosecure*, vol. 6, no. 2, pp. 1–5, 2025, [Online]. Available: https://jurnal-pasca.unla.ac.id/infosecure/article/view/v6n2_01

[5] R. M. Liauren, B. Zaman, and S. Bahri, "Implementasi Algortima Aes Dan Bcrypt Untuk Pengamanan Data Pengguna Pada Website Jahitku," *KHARISMA Tech*, vol. 20, no. 1, pp. 57–71, 2025, doi: 10.55645/kharismatech.v20i1.535.

[6] G. D. M. Zulma, H. B. Seta, and T. Yuniati, "Implementasi Algoritma Aes Dan Bcrypt Untuk Pengamanan File Dokumen," *Inform. J. Ilmu Komput.*, vol. 18, no. 2, p. 163, 2022, doi: 10.52958/iftk.v18i2.4667.

[7] A. Rahayu, G. Abdillah, and H. Ashaury, "Pengamanan Menggunakan Algoritma Aes (Advanced Encryption Standard) Dan Bcrypt (Blowfish Crypt) Pada File Dokumen," *JATI (Jurnal Mhs. Tek. Inform.*, vol. 8, no. 6, pp. 11627–11634, 2024, doi: 10.36040/jati.v8i6.11498.

[8] M. M. Subagio, R. Mumpuni, and A. L. Nurlaili, "Design of Web-Based Decision Support System Using AHP and bcrypt Security," *bit-Tech*, vol. 8, no. 3, pp. 3987–3998, 2026, doi: 10.32877/bt.v8i3.3769.

[9] R Dermawan, and R Rahman, "Penerapan Keamanan Hashing Password Pada Sistem Penjualan Bibbi Shop Berbasis Web," vol. 1, no. 2, pp. 66–69, 2025.

[10] N. Ramadani and E. maiyana, "Perancangan dan Implementasi Sistem Informasi Perpustakaan BerbasisWeb Menggunakan Framework Laravel untuk Optimalisasi PengelolaanBuku dan Transaksi Peminjaman," *Neotech J. Inf. Technol. Innov.*, vol. 1, no. 01, pp. 01–17, 2026.

[11] T. Mary and N. Febriyani, "Peningkatan Keamanan Sistem Informasi Berbasis Laravel 12 dengan Rate Limiting dan Role-Based Access Control (RBAC)," *J. Teknol. Dan Sist. Inf. Bisnis*, vol. 7, no. 3, pp. 473–481, 2025, doi: 10.47233/jteksis.v7i3.1976.

[12] I. Q. Lubis and A. Zulherry, "Implementasi keamanan website dari serangan Cross Site Request Forgery menggunakan algoritma HMAC-SHA256 pada framework Laravel," *Hello World J. Ilmu Komput.*, vol. 5, no. 1, pp. 47–58, 2026.

[13] G. Maulana and U. I. Indragiri, "Analisis Keamanan Sistem Autentikasi Login Berbasis Framework Laravel," vol. 1, no. 1, pp. 18–25, 2026.

[14] A. Fitriyani, H. Lubis, Y. Yaddarabullah, and R. Sari, "Keamanan Sistem Login Menggunakan Multifactor Authentication dan Algoritma Hashing," *J. Inf. Syst. Informatics Comput.*, vol. 9, no. 2, p. 263, 2025, doi: 10.52362/jisicom.v9i2.2137.

[15] D. Febrian *et al.*, "Implementation of Bcrypt Algorithm on Website-Based Hashing Generator Using Laravel Framework," *J. Inf. Syst. Informatics Comput.*, vol. 7, no. 2, p. 199, 2023, doi: 10.52362/jisicom.v7i2.1130.