RESOURCE CENTER

What is password hashing?

Get the full interactive view at
https://bitwarden.com/nl-nl/resources/what-is-password-hashing/

bitwarden

Security affects everyone, from individual users to enterprise organizations. Without robust security measures, data remains vulnerable, supply chains face breach risks, and users constantly react to compromised credentials. Fortunately, most modern operating systems and services implement strong security foundations, with the password hash serving as a cornerstone technology.

While a password hash may not be something users think about daily, it underpins nearly all security mechanisms in modern computing environments.

Understanding what is hashing

A password hash transforms a user's password into a fixed-length string of jumbled characters that cannot be easily reversed or deciphered. When someone creates an account on any platform — whether a website, operating system, email service, or banking portal — their password is typically run through a hash password function before being stored in the service's database.

The essence of a password hash lies in its protective qualities. By converting passwords into seemingly random character strings, systems store authentication data in a format that remains secure even if a database is compromised. Even when hackers gain unauthorized access to servers, they face the significant challenge of reverse-engineering these password hashes — a process requiring substantial computational resources and expertise.

The core characteristics that make a password hash effective include irreversibility, consistency, and collision resistance. Unlike encryption, hashing operates as a one-way process, making it extremely difficult to derive the original password from its hash. The same password always generates identical hash output when processed with the same parameters. Modern algorithms such as bcrypt, Argon2, and SHA-256 minimize the possibility of different passwords producing identical hashes.

A password is typically run through a hash password function before being stored in the service's database.

Understand your account encryption key.

The password hash process

The technical workflow to hash password inputs follows a straightforward but secure pattern. A user creates their plaintext password during account setup. The system processes this password through a password hasher function combined with a random "salt" value to generate the final hash. Only the password hash, not the original password, is stored in the system database. During subsequent login attempts, the system applies the same hashing process to the entered password and compares the result with the stored hash. Login succeeds only when the newly generated hash matches the stored hash exactly.

In this process, a "salt" plays a crucial role. A salt is a random string of characters added to the plaintext password before hashing, resulting in a structure similar to:

Original Password: AyL*fZ%W!C^X@7RC + Salt Value: $random_SaltValue$ = Hashed Password

The critical importance of salt values

Salting significantly enhances password security by protecting against two primary attack vectors. Rainbow table attacks use

precomputed tables of password hashes to reverse-engineer passwords from stolen hash databases. By adding unique salts to each password, systems render these precomputed tables ineffective.

Brute-force attacks employ trial-and-error methods that systematically test possible combinations until finding the correct password. Salting makes this approach exponentially more difficult by requiring attackers to crack each password individually rather than leveraging patterns across multiple accounts.

Evaluate your password security.

Why organizations should implement password hash protection

The password hash serves as the underlying process that significantly increases the difficulty for malicious actors attempting to access systems. Even when attackers employ sophisticated brute-force methods, properly hashed passwords remain secure.

Additionally, password hash protection prevents attackers from directly leveraging stolen credentials in the event of a data breach. Without hashing, compromised passwords would appear in plain text, potentially compromising multiple accounts when users reuse credentials across services. The combination of hashing with salting provides enhanced resistance against both brute-force and rainbow table attacks.

Read more:

Explore NIST guidelines for stronger passwords.

Essential password hash tools

Several proven tools support effective password hash implementations:

Password hashing algorithms

- PBKDF2 (Password-Based Key Derivation Function 2) is a widely adopted and trusted method for deriving cryptographic keys from passwords, using salt values to enhance security.
- Argon2 offers a memory-hard password hashing algorithm specifically designed to resist brute-force and GPU-based attacks.
- Bcrypt provides an open-source, adaptive password hashing algorithm balanced for performance and security.
- SHA-256 and SHA-512 serve as widely used hash functions that convert variable-sized inputs into fixed-size outputs.

Password hash libraries

- Bcrypt offers developers an accessible interface for implementing and verifying passwords with the bcrypt algorithm.
- Argon2-cffi implements the Argon2 password hashing algorithm in a secure, reliable manner.
- Hashlib provides various hash functions, including SHA-256 and SHA-512, for general-purpose hashing.

While servers handle the hashing of passwords, password managers help users create, store, and use strong, unique passwords without needing to memorize them.

Security services

- Bitwarden Secrets Manager manages secrets securely, including password storage and handling.
- Google Cloud Key Management Service enables organizations to generate, distribute, and use cryptographic keys for password hashing securely.

Password managers

Password managers complement password hashing systems. While servers handle the hashing of passwords, password managers help users create, store, and use strong, unique passwords without needing to memorize them. Solutions like Bitwarden generate complex passwords that maximize the security benefits of proper hashing algorithms. By encouraging the use of unique passwords for each service, password managers effectively neutralize the risk of credential reuse across multiple platforms — a common vulnerability that even strong hashing cannot protect against when users employ identical passwords across services.

> Try out the password security checker tool!

Scaling password hashing for growing organizations

As applications grow in size and complexity, organizations face increasing challenges in managing password hashing securely. Effective scaling strategies include implementing distributed password hashing systems that allow applications to scale by distributing password hashes across multiple servers or databases. Authentication platforms like Auth0 and Stytch support this approach effectively.

Effective scaling of password hashing requires organizations to address several factors:

- Organizations must handle large volumes of user data efficiently while maintaining security and privacy.
- Plan for increased traffic and authentication requests, and regularly monitor system performance.
- Staying current with security patches and updates is essential, as is using secure, modern hash functions.
- Implementing unique salt values for each password and iterating the hashing process appropriately provides additional protection.
- Many organizations benefit from leveraging specialized services like Authgear and compromised password detection tools like Enzoic.
- Encouraging organization-wide password manager adoption complements server-side hashing for a comprehensive security approach.

Password hash best practices

Organizations seeking to implement effective password hashing systems should follow the following essential practices, many of which work harmoniously with password manager adoption.

Select strong, proven password hashing functions

Organizations should choose well-established algorithms like bcrypt, Argon2, or PBKDF2 that have undergone extensive testing and review by the cryptographic community as reliable password hashing options.

Read more:

Learn about master password best

practices.

Balance security and performance with work factors

Adjusting the computational requirements of hash functions allows organizations to balance security against performance needs. Higher work factors increase resistance to brute-force attacks, but reduce overall system performance. Finding the optimal balance depends on each organization's specific requirements.

Implement unique salt values

Using randomly generated salt values for each user's password prevents attackers from leveraging precomputed hash databases and makes brute-force attempts significantly less efficient.

Store salt values properly

Organizations should generate a separate random value for each user's password and store it alongside the hashed password to prevent rainbow table attacks.

Apply iterative hashing

Increasing computational overhead through multiple hashing iterations slows down brute-force attempts while maintaining reasonable performance for legitimate authentication.

Employ memory-hard hash functions

Algorithms like Argon2 consume computational resources in ways that resist GPU-based attacks, ensuring stronger password protection.

Maintain updated dependencies

Organizations must regularly update and patch all dependencies used in password hashing implementation, especially libraries or frameworks that might contain security vulnerabilities.

By implementing comprehensive password hash strategies, organizations create a solid foundation for their broader security architecture, protecting both their systems and their users' sensitive information from increasingly sophisticated cyber threats.

For optimal results, organizations should encourage employees and users to adopt password managers alongside these password hashing implementations. This creates a powerful two-pronged approach where strong passwords are both generated and stored securely by users while being properly protected through hashing on the server side.

Get started with Bitwarden

Password managers like Bitwarden complement password hashing technologies by addressing security from the user side, while hashing protects from the server side. These tools work together by enabling the generation of complex, unique passwords that maximize the effectiveness of hashing algorithms while solving the human tendency to create weak or reused passwords.

Read more:

Discover the encryption methods behind Bitwarden.

Password managers implement their own encryption before passwords are transmitted, creating a zero-knowledge security model that parallels the server's hashing protection. Many password managers also include breach monitoring to alert users when credentials appear in data breaches, providing an additional protection layer beyond server-side hashing. This creates a comprehensive security approach — strong, unique passwords protected by client-side encryption before being secured again through server-side hashing, effectively closing security gaps that either solution alone cannot address.