Secure Password Storage with Password Hashing

Password hashing converts a password into a one-way value suitable for storage. PHP automatically includes a salt, so the same password can produce different valid hashes.

> Use fictional test passwords only. Never enter your school, email, banking or reused personal password.

Generate a hash

```
<?php
$testPassword = "fictional-test-password";
$hash = password_hash($testPassword, PASSWORD_DEFAULT);

echo htmlspecialchars($hash, ENT_QUOTES, "UTF-8");
```

Store the complete hash in a `VARCHAR(255)` database field. Do not shorten it.

Verify a submitted password

```
<?php
$submittedPassword = "fictional-test-password";
$storedHash = '$2y$10$REPLACE_WITH_A_COMPLETE_HASH';

if (password_verify($submittedPassword, $storedHash)) {
    echo "Password accepted.";
} else {
    echo "Password not accepted.";
}
```

Login code should retrieve the stored hash by username and pass it to `password_verify()`. Never hash the submitted password again and compare strings; salts make that unreliable.

Rehash when needed

```
if (password_needs_rehash($storedHash, PASSWORD_DEFAULT)) {
    $newHash = password_hash($submittedPassword, PASSWORD_DEFAULT);
    // Update the stored hash using a prepared statement.
}
```

This allows PHP’s current default algorithm to improve over time.

Safe practice

- Do not print passwords.
- Do not place passwords or connection details in screenshots.
- Do not email or log submitted passwords.
- Use HTTPS on hosted systems.
- Use prepared statements for inserts and updates.
- Keep login failure messages generic.

Check

- ☐ Database stores hashes, not original passwords.
- ☐ `password_hash()` is used during registration.
- ☐ `password_verify()` is used during login.
- ☐ Hash field is `VARCHAR(255)`.
- ☐ Test data contains no real credentials.

Revision #2
Created 2026-06-08 04:29:59 UTC by Mr Napper
Updated 2026-08-18 23:49:05 UTC by Mr Napper