**Algorytmy Kryptograficzne do Szyfrowania Hasła w Bazie Danych**

**1. Algorytmy Skrótu (Hashing):**

SHA-256:

<?php

function hash\_password($password) {

$hashed\_password = hash('sha256', $password);

return $hashed\_password;

}

?>

**2. Algorytmy Uwierzytelniania (Key Derivation Functions - KDF):**

PBKDF2:

<?php

function hash\_password\_pbkdf2($password, $salt) {

$key = hash\_pbkdf2("sha256", $password, $salt, 100000, 32);

return $key;

}

?>

**3. Algorytmy Szyfrowania Symetrycznego (Symmetric Encryption):**

AES:

<?php

function encrypt\_password\_aes($password, $key) {

$cipher = "aes-256-cbc";

$ivlen = openssl\_cipher\_iv\_length($cipher);

$iv = openssl\_random\_pseudo\_bytes($ivlen);

$encrypted\_password = openssl\_encrypt($password, $cipher, $key, 0, $iv);

return $encrypted\_password;

}

?>

**Uwaga:**

W przypadku używania funkcji kryptograficznych, aby unikać błędów związanych z bezpieczeństwem, warto używać odpowiednich bibliotek i funkcji języka programowania. Dlatego warto skorzystać z gotowych bibliotek PHP, takich jak Libsodium, do obsługi operacji kryptograficznych.

**Kod do Zapisu Zaszyfrowanych Danych w Bazie:**

<?php

// Przykładowy kod zapisu zaszyfrowanego hasła do bazy danych

function save\_encrypted\_password\_to\_db($username, $encrypted\_password) {

// Kod zapisu do bazy danych (pominięte dla czytelności)

// Przy użyciu PDO do obsługi bazy danych, np. MySQL

$pdo = new PDO("mysql:host=localhost;dbname=twoja\_baza", "nazwa\_uzytkownika", "haslo");

$stmt = $pdo->prepare("INSERT INTO uzytkownicy (nazwa, haslo) VALUES (?, ?)");

$stmt->execute([$username, $encrypted\_password]);

}

?>

**Kod Sprawdzający Dane Logowania z Szyfrowaniem:**

<?php

// Przykładowy kod sprawdzający dane logowania z zaszyfrowanym hasłem

function check\_login\_credentials($username, $provided\_password) {

// Pobranie zapisanego zaszyfrowanego hasła z bazy danych

$saved\_encrypted\_password = get\_encrypted\_password\_from\_db($username);

// Porównanie zaszyfrowanych haseł

if ($saved\_encrypted\_password == hash\_password($provided\_password)) {

echo "Login successful!";

} else {

echo "Login failed.";

}

}

// Funkcja do pobrania zaszyfrowanego hasła z bazy danych

function get\_encrypted\_password\_from\_db($username) {

// Kod pobierania hasła z bazy danych (pominięte dla czytelności)

// Przy użyciu PDO do obsługi bazy danych, np. MySQL

$pdo = new PDO("mysql:host=localhost;dbname=twoja\_baza", "nazwa\_uzytkownika", "haslo");

$stmt = $pdo->prepare("SELECT haslo FROM uzytkownicy WHERE nazwa = ?");

$stmt->execute([$username]);

$row = $stmt->fetch();

return $row['haslo'];

}

?>

Początek formularza

**Czym się różni SHA256 od SHA512?**

SHA256 i SHA512 to różne wersje algorytmu funkcji skrótu, znanej jako Secure Hash Algorithm (SHA). Główną różnicą między nimi jest długość wygenerowanego skrótu. SHA256 generuje 256-bitowy skrót, podczas gdy SHA512 generuje 512-bitowy. Większa długość skrótu oznacza potencjalnie większe bezpieczeństwo, ale również może wymagać więcej zasobów obliczeniowych.

**Za co odpowiada długość klucza w algorytmach?**

Długość klucza w algorytmach kryptograficznych odgrywa kluczową rolę w zapewnianiu bezpieczeństwa. Im dłuższy klucz, tym trudniej go złamać za pomocą ataków siłowych. Długość klucza wpływa na przestrzeń klucza, czyli ilość możliwych kombinacji, co determinuje czas potrzebny do skutecznego złamania zabezpieczeń.

**Jaki klucz wybiera się do zapisania hasła?**

Przy wyborze klucza do zapisywania hasła ważne jest uwzględnienie używanego algorytmu. Na przykład, w kontekście algorytmu AES, klucz powinien być wystarczająco długi i generowany w sposób bezpieczny. Klucz do zapisywania hasła powinien być traktowany jako tajemnica, nieprzewidywalna i trudna do odgadnięcia.

**Jeśli chce się użyć różnych kluczy to jakich i dlaczego?**

Jeśli decydujemy się używać różnych kluczy, można rozważyć zastosowanie jednego klucza do szyfrowania danych, takiego jak AES, a innego do generowania skrótu hasła, np. SHA256 lub PBKDF2. To podejście może zwiększyć bezpieczeństwo systemu, gdyby jeden z kluczy został skompromitowany, drugi nadal zapewniałby ochronę.

**Po co szyfrować dane kluczem, który daje się złamać?**

Szyfrowanie danych kluczem, który jest łatwy do złamania, jest ryzykowne i niewystarczające w kontekście zabezpieczania poufnych informacji. Wybieranie silnych i bezpiecznych algorytmów oraz kluczy jest kluczowe dla utrzymania integralności i poufności danych. Używanie łatwo złamanych kluczy naraża system na ryzyko nieautoryzowanego dostępu i utraty danych.