Imagen que contiene Texto

Descripción generada automáticamenteIcono

El contenido generado por IA puede ser incorrecto.

**Facultad De Ciencia y Tecnología**

Ingeniería En Sistemas Y Redes Informáticas

**Docente:**

Ing. William Alexis Montes

**Materia:**

Programación computacional 3

**Nombre de la actividad:**

2do Examen parcial

**Estudiantes:**

Karen Esmeralda Portillo Portillo SMSS202223

Yolanda Isabel Marroquín Ulloa SMSS047424

Moisés Abelino Ramírez Rubio **A1** SMSS217724

Erick Josué Rivera Velásquez SMSS147524

**Fecha de entrega:**

1. de octubre del 2025

**Indicé**

[Introducción. 2](#_Toc211785394)

[Librería Bcrypt. 4](#_Toc211785395)

[¿Qué hace exactamente bcrypt? 4](#_Toc211785396)

[Cómo Funciona Bcrypt 5](#_Toc211785397)

[Consejos de Prevención 5](#_Toc211785398)

[Uso Avanzado 6](#_Toc211785399)

[Usos en la vida real de bcrypt 7](#_Toc211785400)

[Funciones más utilizadas 9](#_Toc211785401)

[Bibliografía 10](#_Toc211785402)

# 

# Introducción

En el ámbito de la seguridad informática, la protección de datos sensibles especialmente las contraseñas de los usuarios, constituye una prioridad fundamental para cualquier sistema que maneje información personal. Con el crecimiento exponencial de aplicaciones web y servicios digitales, garantizar que las credenciales de autenticación se almacenen de forma segura se ha convertido en un requisito indispensable para prevenir accesos no autorizados y mitigar los riesgos asociados a filtraciones de datos.

Dentro de este contexto, bcrypt emerge como uno de los algoritmos más robustos y ampliamente adoptados para el *hashing* de contraseñas. Diseñado originalmente por Niels Provos y David Mazières en 1999, bcrypt se basa en el cifrado Blowfish y se distingue por su resistencia a ataques de fuerza bruta gracias a su naturaleza deliberadamente lenta y su uso de *salting* (aleatorización) integrado. En el ecosistema de Python, la librería bcrypt proporciona una interfaz sencilla y eficaz para implementar este algoritmo, permitiendo a los desarrolladores integrar prácticas de seguridad modernas en sus aplicaciones con relativa facilidad.

Este trabajo de investigación tiene como objetivo analizar en profundidad la librería bcrypt de Python: su funcionamiento interno, sus ventajas frente a otros métodos de *hashing*, su correcta implementación en entornos reales, así como las buenas prácticas recomendadas para su uso. A través de este estudio, se busca no solo comprender las capacidades técnicas de esta herramienta, sino también destacar su relevancia en el desarrollo de sistemas seguros y confiables en la era digital actual.

# Librería Bcrypt.

La librería **bcrypt** en Python se utiliza para encriptar contraseñas de manera segura, evitando que puedan ser leídas incluso si alguien accede a tu base de datos.

(Shah, 2023) Define BCrypt como un algoritmo de hash de contraseñas considerado uno de los más seguros en Python. BCrypt está diseñado para ser lento, lo que dificulta que los hackers descifren las contraseñas hasheadas.

# ¿Qué hace exactamente bcrypt?

bcrypt es una función de hash de contraseñas diseñada por Niels Provos y David Mazières. Se basa en el cifrado Blowfish y se presentó en USENIX en 1999. Además de incorporar una sal para proteger contra ataques de tablas arcoíris, bcrypt es una función adaptativa: con el tiempo, se puede aumentar el número de iteraciones para reducir su velocidad, de modo que se mantenga resistente a ataques de búsqueda por fuerza bruta incluso con una mayor potencia de cálculo.

La función bcrypt es el algoritmo hash de contraseña predeterminado para OpenBSD y también lo era para algunas distribuciones de Linux como SUSE Linux.Existen implementaciones de bcrypt en C, C++, C#, Embarcadero, Delphi, Elixir, Go, Java, JavaScript, Perl, PHP, Ruby, Python, Rust, V (Vlang), Zig y otros lenguajes**.**

Blowfish destaca entre los cifradores de bloque por su costosa fase de configuración de claves. Comienza con subclaves en un estado estándar, luego utiliza este estado para realizar un cifrado de bloque utilizando parte de la clave y, con el resultado de dicho cifrado (que es más preciso en el hash), reemplaza algunas de las subclaves. Después, utiliza este estado modificado para cifrar otra parte de la clave y, con el resultado, reemplaza más subclaves. Procede de esta manera, utilizando un estado progresivamente modificado para aplicar el hash a la clave y reemplazar fragmentos de estado, hasta que se hayan configurado todas las subclaves.Provos y Mazières aprovecharon esta ventaja y la llevaron más allá. Desarrollaron un nuevo algoritmo de configuración de claves para Blowfish, al que denominaron "Eksblowfish" ('Blowfish, programa de claves costoso'). La configuración de claves comienza con una forma modificada de la configuración de claves estándar de Blowfish, en la que tanto la sal como la contraseña se utilizan para configurar todas las subclaves. A continuación, se aplican varias rondas del algoritmo estándar de codificación de claves de Blowfish, utilizando alternativamente la sal y la contraseña como clave. Cada ronda comienza con el estado de la subclave de la ronda anterior. En teoría, esto no es más robusto que el programa de claves estándar de Blowfish, pero el número de rondas de regeneración de claves es configurable; por lo tanto, este proceso puede ralentizarse arbitrariamente, lo que ayuda a disuadir ataques de fuerza bruta contra el hash o la sal.

# Cómo Funciona Bcrypt

Bcrypt emplea un proceso de múltiples pasos para garantizar la seguridad de las contraseñas de los usuarios. Aquí tienes una explicación paso a paso de cómo funciona:

1. **Generación de una Sal Aleatoria**: Cuando un usuario crea o actualiza su contraseña, Bcrypt genera una sal aleatoria. Una sal es un valor aleatorio que se añade a la contraseña antes de hacer el hash, asegurando que la misma contraseña no producirá el mismo hash. Esto añade una capa adicional de seguridad contra los ataques.
2. **Hasheado de la Contraseña con la Sal**: Luego, Bcrypt combina la sal aleatoria con la contraseña del usuario y aplica una función de hashing múltiples veces. El número de iteraciones es un parámetro que se puede ajustar para incrementar o disminuir el tiempo requerido para calcular el hash. Esta ralentización intencional del proceso de hashing lo hace altamente resistente a ataques de fuerza bruta.
3. **Almacenamiento de la Contraseña Hasheada en la Base de Datos**: El hash resultante, junto con la sal generada aleatoriamente, se almacena en la base de datos del sistema. Al almacenar la sal junto con el hash, Bcrypt asegura que la información necesaria esté disponible para la verificación de la contraseña durante los intentos de inicio de sesión.
4. **Verificación de la Contraseña del Usuario**: Cuando un usuario intenta iniciar sesión en el sistema, Bcrypt recupera el hash y la sal almacenados de la base de datos. Luego aplica la misma función de hashing y el proceso de iteración a la contraseña ingresada utilizando la sal recuperada. El hash resultante se compara con el hash almacenado, y se concede el acceso solo si los dos hashes coinciden.

# Consejos de Prevención

Para utilizar Bcrypt de manera efectiva y maximizar la seguridad de las contraseñas, considera los siguientes consejos:

* **Implementa Bcrypt u Otro Algoritmo de Hashing Seguro**: Almacenar contraseñas en bases de datos, siempre opta por un algoritmo de hashing robusto como Bcrypt. Su proceso de hashing adaptativo y parámetros configurables proporcionan una defensa sólida contra ataques de fuerza bruta.
* **Usa Sales Únicas para Cada Contraseña**: Para aumentar aún más la seguridad de las contraseñas, genera una sal única para cada contraseña de usuario. Esta práctica evita que los atacantes creen tablas precomputadas (tablas de arco iris) para la recuperación eficiente de contraseñas.
* **Actualiza Regularmente los Sistemas**: Mantén tus sistemas actualizados para asegurarte de que estás utilizando las versiones más recientes de Bcrypt u otros algoritmos de hashing recomendados. Las versiones actualizadas suelen abordar posibles vulnerabilidades y garantizan el más alto nivel de seguridad.

# Uso Avanzado

Si bien Bcrypt es un algoritmo de hashing de contraseñas ampliamente recomendado, existen usos avanzados y consideraciones a tener en cuenta:

**Generación de Sal**

Bcrypt genera automáticamente una sal aleatoria durante el proceso de hashing de la contraseña. Sin embargo, para requisitos específicos, podrías considerar generar tu propia sal utilizando generadores de números aleatorios criptográficamente seguros.

**Factor de Costo**

El factor de costo, también conocido como el factor de trabajo, determina el número de iteraciones que Bcrypt realiza durante el proceso de hashing. Cuanto mayor sea el factor de costo, más tiempo consume el proceso de hashing. Ajustar el factor de costo te permite equilibrar la seguridad y el rendimiento según tus requisitos específicos.

**Compatibilidad**

Bcrypt se ha convertido en el estándar de facto para el hashing de contraseñas en muchos sistemas debido a su adopción generalizada y seguridad probada. Es compatible con la mayoría de los lenguajes de programación y plataformas, lo que lo convierte en una opción confiable para la encriptación en diferentes entornos.

**Alternativas**

Si bien Bcrypt es una opción robusta para el hashing de contraseñas, otros algoritmos, como Argon2 y Scrypt, proporcionan características de seguridad similares. Estos algoritmos ofrecen diversas compensaciones en términos de rendimiento y requisitos de memoria, por lo que es esencial evaluar tus necesidades específicas al elegir un algoritmo de hashing de contraseñas.

**Términos Relacionados**

* **Ataque de Fuerza Bruta**: Un ataque de fuerza bruta es un método de prueba y error utilizado por los atacantes para obtener contraseñas o claves de cifrado. En estos ataques, los atacantes chequean sistemáticamente todas las posibles contraseñas o claves hasta encontrar la correcta.
* **Ataque de Tablas de Arco Iris**: Un ataque de tablas de arco iris implica el uso de tablas precomputadas para la recuperación eficiente de contraseñas. Estas tablas contienen una lista de contraseñas cifradas y sus valores en texto plano correspondientes, lo que permite a los atacantes encontrar rápidamente coincidencias para contraseñas hasheadas.
* **Sal**: En el contexto del hashing de contraseñas, una sal es un valor aleatorio que se añade a cada contraseña antes de hacer el hash. El propósito de una sal es asegurar que la misma contraseña no produzca el mismo hash, incluso si es utilizada por múltiples usuarios. Esto añade una capa adicional de seguridad y previene que los atacantes utilicen tablas precomputadas u otros métodos eficientes para la recuperación de contraseñas.

# Usos en la vida real de bcrypt

**Ejemplo en Python:**

import bcrypt

# Registro de usuario: hashear la contraseña antes de guardar

def hash\_password(plain\_password: str) -> bytes:

# bcrypt.gensalt toma como argumento “rounds” (cost factor)

salt = bcrypt.gensalt(rounds=12)

hashed = bcrypt.hashpw(plain\_password.encode('utf-8'), salt)

return hashed

# Autenticación: comparar contraseña ingresada con el hash guardado

def check\_password(plain\_password: str, hashed: bytes) -> bool:

return bcrypt.checkpw(plain\_password.encode('utf-8'), hashed)

# Uso

if \_\_name\_\_ == "\_\_main\_\_":

pw = "miClaveSegura123"

hashed\_pw = hash\_password(pw)

print("Hash:", hashed\_pw)

# Simular login

entrada = "miClaveSegura123"

print("¿Coincide?", check\_password(entrada, hashed\_pw))

**Ejemplo en Node.js**

const bcrypt = require('bcrypt');

async function hashPassword(plainPassword) {

const saltRounds = 12;

const salt = await bcrypt.genSalt(saltRounds);

const hash = await bcrypt.hash(plainPassword, salt);

return hash;

}

async function checkPassword(plainPassword, hash) {

return await bcrypt.compare(plainPassword, hash);

}

// Uso

(async () => {

const pw = "miClaveSegura123";

const hash = await hashPassword(pw);

console.log("Hash:", hash);

const ok = await checkPassword("miClaveSegura123", hash);

console.log("¿Coincide?", ok);

})();

# Funciones más utilizadas

Como ya se ha explicado anteriormente bcrypt es una librería enfocada en el hashing seguro de contraseñas, sus funciones más utilizadas son las siguientes:

1. **bcrypt.gensalt(rounds=12, prefix=b“2b”):** Esta función genera una cadena de sal(salt) criptográficamente segura que usará para “salar” la contraseña antes de hashearla.

Dicha función tiene los siguientes parámetros:

* **rounds:** numero de iteraciones (logaritmo), el valor por defecto es 12, el rango típico es de 4 a 31 cada incremento duplica el tiempo de computo, se recomienda usar 12 o 13 para equilibrio entre seguridad y rendimiento.
* **prefix:** normalmente no se cambia, b“2b”es el estándar moderno

1. **bcrypt.hashpw(password, salt):** Esta función toma una contraseña en bytes y una sal, y devuelve un hash seguro que incluye la sal + el hash real en un solo valor. El resultado de **hashpw()** ya incluye la sal dentro del hash, lo que significa que no es necesario guardar la sal por separado en una base de datos.

Dicha función tiene los siguientes parámetros:

* **password:** debe ser de tipo bytes, no string, en caso de tener un string hay que cambiarlo usando **.encode(‘utf-8’).**
* **salt:** generado con **gensalt().**

1. **bcrypt.checkpw(password, hashed):** Esta función compara una contraseña en texto plano con un hash almacenado para verificar si coinciden.La forma en la que esta función funcióna internamente es, primero extrae la sal del hash alamcenado (está incluido en él), despues aplica bcrypt a la contraseña proporcionada usando esa misma sal luego comparael resultado con el hash original, finalmente devuelve “True” o “False”.

Dicha función tiene los siguientes parámetros:

* **password:** bytes con la contraseña que el usuario ingresó
* **hashed:** bytes con el hash guardado en la base de datos.

1. **bcrypt.kdf(password, salt, desired\_key\_bytes, rounds)**  
   Deriva una clave criptográfica a partir de una contraseña y un salt, útil para aplicaciones que requieren claves seguras.

**5-bcrypt.revision()**  
 Devuelve la versión de bcrypt utilizada, lo que puede ser útil para verificar compatibilidad.

# Bibliografía

Shah, A. (18 de abril de 2023). *tutorialspoint*. Obtenido de Hashing Passwords in Python with BCrypt: https://www.tutorialspoint.com/hashing-passwords-in-python-with-bcrypt

AcademiaLab. (s. f.). *Bcrypt*. *AcademiaLab*. Recuperado de <https://academia-lab.com/enciclopedia/bcrypt/>

VPN Unlimited. (s. f.). Qué es Bcrypt – Términos y definiciones de ciberseguridad. *VPN Unlimited*. Recuperado de <https://www.vpnunlimited.com/es/help/cybersecurity/bcrypt?srsltid=AfmBOoryTxtlbNaLGRtkC5jni0Wl2kfu0iZA_L_UVcab2F5-iS01lxDH> (consultado en octubre de 2025)

Etese, O., & Adesina, A.-A. (2025, julio 14). A review and comparative analysis of password hashing techniques: Evaluating bcrypt and Argon2 (SSRN No. 5363433). *SSRN*. <https://doi.org/10.2139/ssrn.5363433>

Zhao, Y., Zhang, L., & Liu, J. (2020). A study on the application of digital twin technology in the production process. *Mathematics, 11*(4), 450. <https://doi.org/10.3390/math11040450>

The Python Cryptographic Authority. (2025, 25 de septiembre). *bcrypt* [Paquete para Python]. PyPI. Recuperado de [https://pypi.org/project/bcrypt/](https://pypi.org/project/bcrypt/?utm_source=chatgpt.com) [pypi.org](https://pypi.org/project/bcrypt/?utm_source=chatgpt.com)

The Python Cryptographic Authority. (s. f.). *pyca/bcrypt* [Repositorio en GitHub]. Recuperado de [https://github.com/pyca/bcrypt/](https://github.com/pyca/bcrypt/?utm_source=chatgpt.com)