DGTIC UNAM
DIRECCIÓN GENERAL DE CÓMPUTO Y DE TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIÓN

REDEC UNAM

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
DIRECCIÓN GENERAL DE CÓMPUTO Y DE TECNOLOGÍAS
DE INFORMACIÓN Y COMUNICACIÓN

DIPLOMADO

Desarrollo de sistemas con tecnología Java

Módulo 11

Password Encoders

Mtro. Alfonso Gregorio Rivero Duarte

1. Password Encoder

1.1 Manejo de Contraseña

ENCODING VS ENCRYPTION VS HASHING

MINISTRY OF SECURITY

* ENCODING *

WHAT
REVERSIBLE TRANSFORMATION OF DATA IN A DIFFERENT FORMAT.

WHY
TO PRESERVE USABILITY OF DATA.

USE
TRANSMISSION AND STORAGE.

EXAMPLES
- ASCII
- UNICODE
- BASE64

ENCRYPTION

WHAT
CONVERSION OF DATA TO AN UNREADABLE FORMAT BY USING A KEY.

WHY
TO PROTECT CONFIDENTIALITY OF DATA.

USE
KEEPS DATA IN TRANSMISSION AND REST A SECRET.

EXAMPLES
- AES
- RSA
- DIFFIE-HELLMAN

* HASHING *

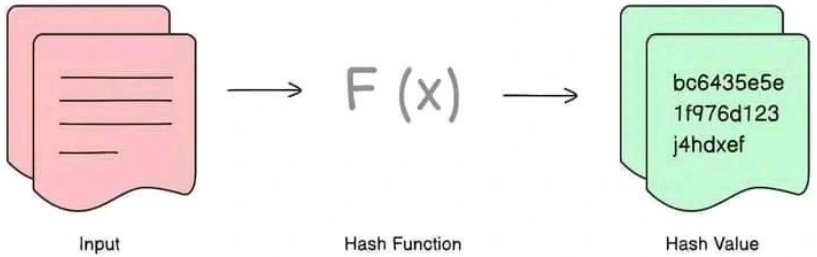
WHAT
PRODUCES A HASH OF THE DATA BY AN ALGORITHM.

WHY
TO VALIDATE THE INTEGRITY OF DATA.

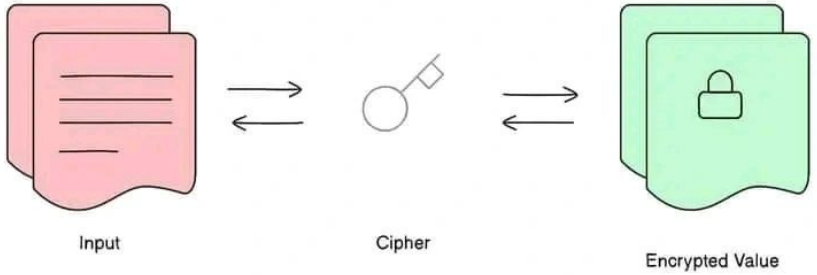
USE
STORE PASSWORD HASHES.

EXAMPLES
- MD5
- SHA-256
- SHA-3

Hashing

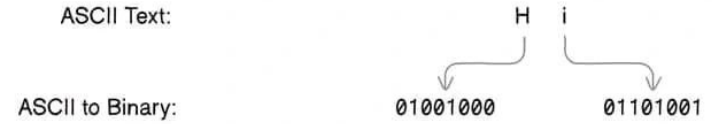


Encryption

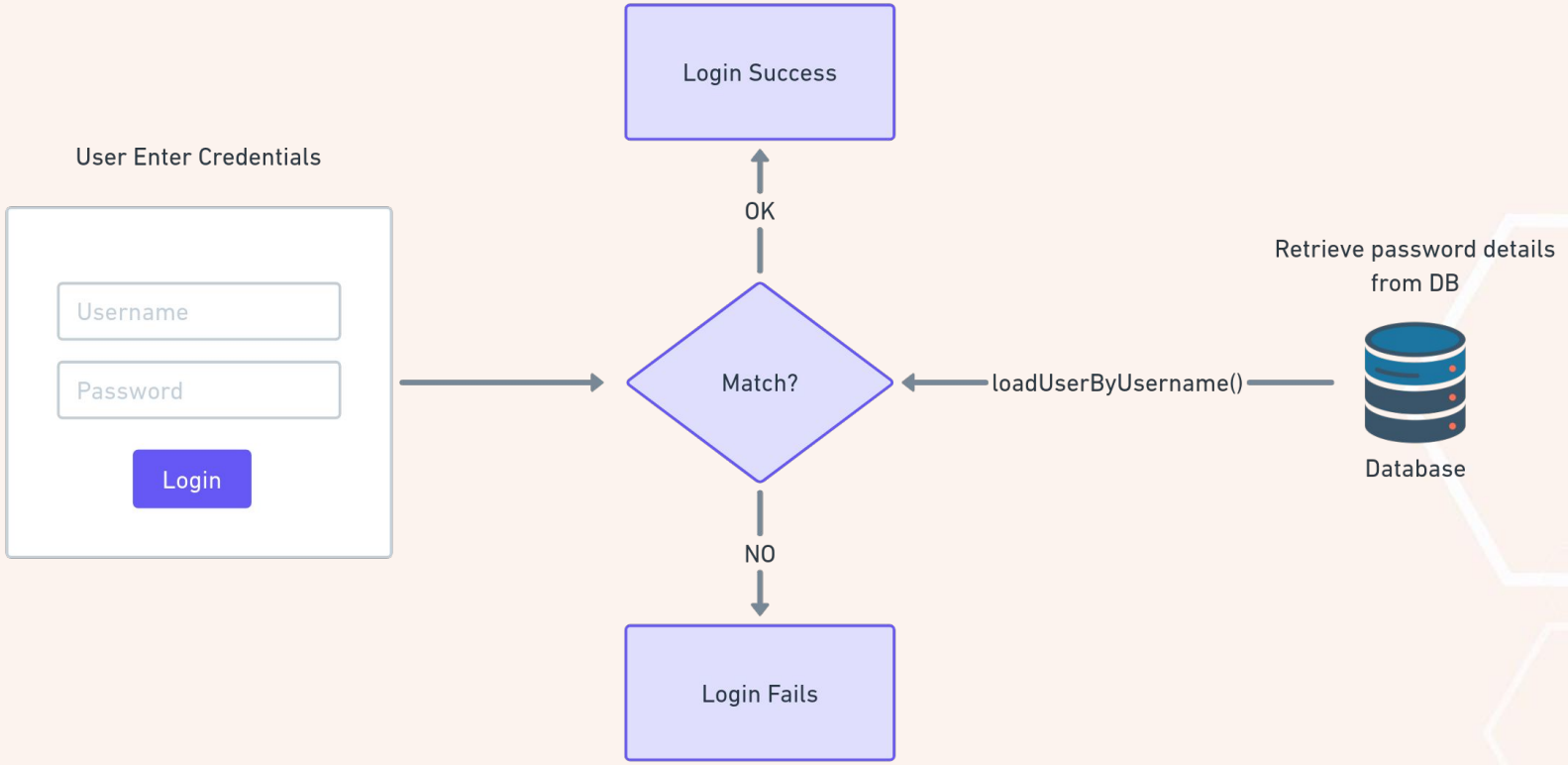


@rauljuncoV

Encoding

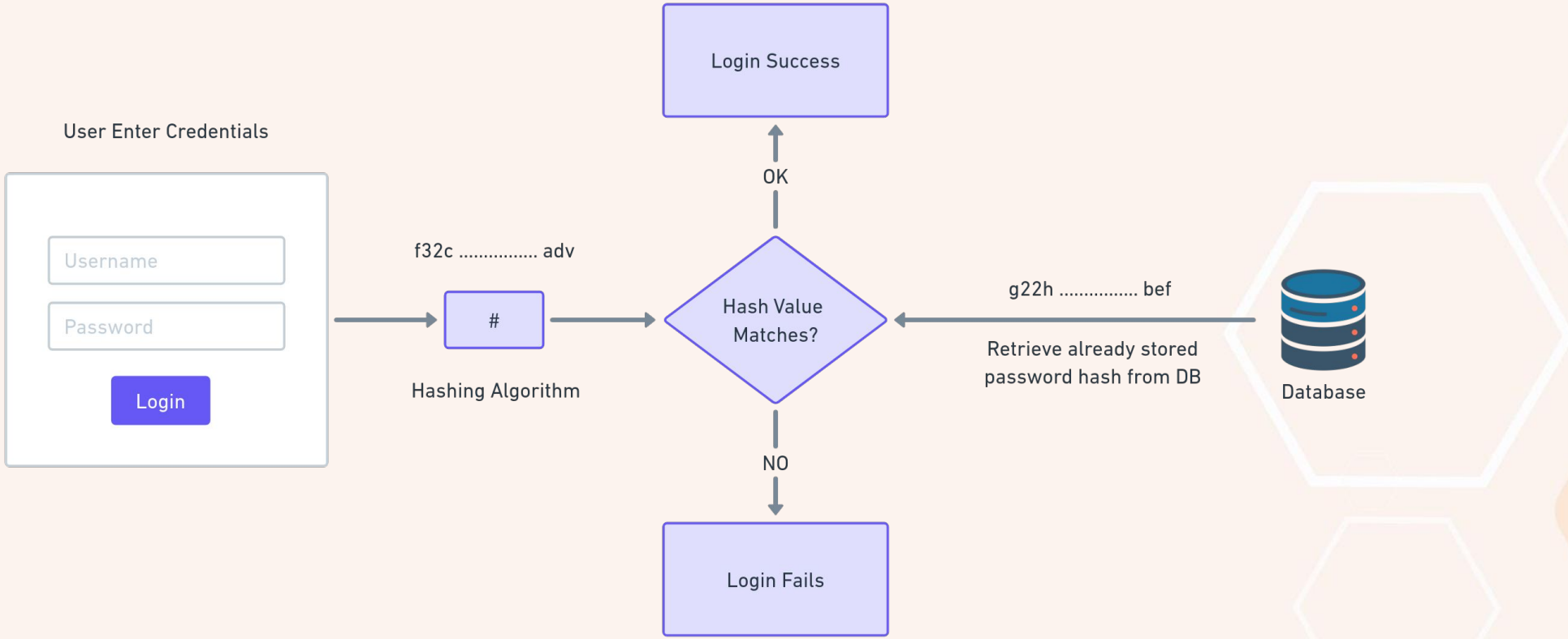


Almacenamiento de contraseñas



Almacenar las contraseñas en texto plano dentro de un sistema de almacenamiento como DB tendrá problemas de integridad y confidencialidad. Por lo tanto, este no es un enfoque recomendado para aplicaciones de producción.

Almacenamiento de contraseñas



Almacenar y administrar las contraseñas con hash es el enfoque recomendado para aplicaciones de producción. Con varios PasswordEncoders disponibles dentro de Spring Security, nos hace la vida más fácil.

Diferentes maneras de Pwd Managing

Codificación (Encoding)

Encoding se define como el proceso de convertir datos de una forma a otra y no tiene nada que ver con la criptografía.

No implica ningún secreto y es completamente reversible.

La codificación no se puede utilizar para proteger los datos. A continuación se muestran los diversos algoritmos disponibles públicamente que se utilizan para la codificación.

Ex: ASCII, BASE64, UNICODE, ...

Cifrado (Encryption)

El cifrado se define como el proceso de transformar datos de tal forma que se garantice la confidencialidad.

Para lograr la confidencialidad, el cifrado requiere el uso de un secreto que, en términos criptográficos, llamamos "clave".

El cifrado puede ser reversible mediante el descifrado con la ayuda de la "clave". Siempre que la "clave" sea confidencial, el cifrado puede considerarse seguro.

Hashing

En el hash, los datos se convierten al valor hash utilizando alguna función hash.

Los datos con un hash no son reversibles. No se pueden determinar los datos originales a partir de un valor hash generado.

Dados algunos datos arbitrarios junto con la salida de un algoritmo hash, se puede verificar si estos datos coinciden con los datos de entrada originales sin necesidad de ver los datos originales.

Manejo de Contraseñas

Si queremos autenticar al usuario en el lado del servidor, tenemos que seguir estos pasos:

1. Obtenga el nombre de usuario y la contraseña del usuario que desea autenticarse.
2. Busque el nombre de usuario en el almacenamiento, normalmente una base de datos.
3. Compare la contraseña que proporcionó el usuario con la contraseña del usuario de la base de datos.

Echemos un vistazo a algunas de las mejores (y peores) prácticas sobre cómo hacerlo.

Guardar contraseñas como texto sin formato

Tenemos que lidiar con el hecho de que tenemos que guardar las contraseñas de los usuarios en nuestro sistema para compararlas durante la autenticación.

Obviamente, es una mala idea guardar las contraseñas como texto sin formato en la base de datos.

Debemos suponer que un atacante puede robar la base de datos con contraseñas u obtener acceso a las contraseñas mediante otros métodos como la inyección SQL.

En este caso, el atacante podría utilizar la contraseña inmediatamente para acceder a la aplicación. Por lo tanto, debemos guardar las contraseñas de una forma que el atacante no pueda usar para la autenticación.

Hashing

Hashing resuelve el problema del acceso inmediato al sistema con contraseñas expuestas.

El hash es una función unidireccional que convierte la entrada en una línea de símbolos. Normalmente la longitud de esta línea es fija.

Si los datos tienen hash, es muy difícil volver a convertir el hash a la entrada original y también es muy difícil encontrar la entrada para obtener el resultado deseado.

Hashing

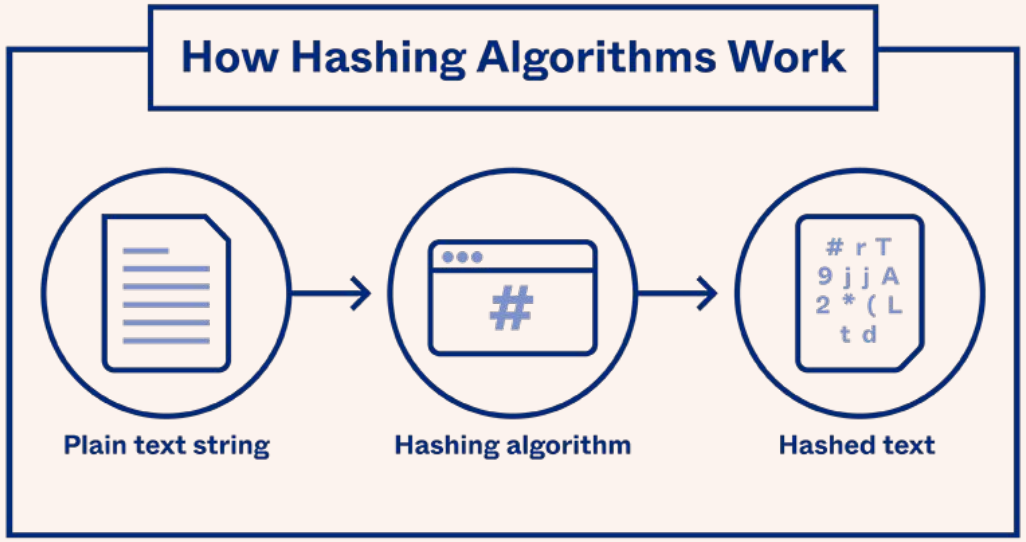
Tenemos que cifrar la contraseña en dos casos:

1. Cuando el usuario se registra en la aplicación, codificamos la contraseña y la guardamos en la base de datos.
2. Cuando el usuario quiere autenticarse, aplicamos un hash a la contraseña proporcionada y la comparamos con el hash de la contraseña de la base de datos.

Ahora, cuando los atacantes obtienen el hash de una contraseña, no pueden utilizarlo para acceder al sistema. Cualquier intento de encontrar el texto sin formato a partir del valor hash requiere un gran esfuerzo por parte del atacante. Un ataque de fuerza bruta puede resultar muy caro si el hash es lo suficientemente largo.

Hashing

Sin embargo, al utilizar tablas de arco iris, los atacantes aún pueden tener éxito. Una tabla arcoíris es una tabla con hashes precalculados para muchas contraseñas. Hay muchas tablas de arcoíris disponibles en Internet y algunas de ellas contienen millones de contraseñas.



1. Password Encoder

1.2 Password Salting

Password Salting

Para evitar un ataque con tablas arcoíris podemos utilizar contraseñas “salteadas”. Un salt es una secuencia de bytes generados aleatoriamente que se codifica junto con la contraseña. El salt se almacena y no es necesario protegerla.

Cada vez que el usuario intenta autenticarse, la contraseña del usuario se codifica con el salt guardado y el resultado debe coincidir con la contraseña almacenada.

La probabilidad de que la combinación de la contraseña y el salt se calcule previamente en una tabla de arcoíris es muy pequeña. Si el salt es lo suficientemente largo y aleatorio, es imposible encontrar el hash en una tabla de arcoíris.

Password Salting

Sin embargo, dado que el salt no es un secreto, los atacantes aún pueden iniciar un ataque de fuerza bruta.

Un salt puede dificultar el ataque del atacante, pero el hardware se está volviendo más eficiente. Debemos asumir un hardware en rápida evolución con el que el atacante pueda calcular miles de millones de hashes por segundo.

Por lo tanto, el hashing y el salt son necesarios, pero no suficientes.

1. Password Encoder

1.3 Funciones de Hash

Funciones de hash de contraseñas

Las funciones hash no se crearon para codificar únicamente contraseñas. El inventor de las funciones hash hizo un muy buen trabajo e hizo que la función hash fuera muy rápida.

Sin embargo, si podemos codificar contraseñas muy rápido, entonces un atacante también puede ejecutar el ataque de fuerza bruta muy rápido.

La solución es hacer que el hash de contraseñas sea lento.

¿Pero qué tan lento puede ser? No debe ser tan lento como para resultar inaceptable para el usuario, pero sí lo suficientemente lento como para que un ataque de fuerza bruta lleve un tiempo infinito.

Funciones de hash de contraseñas

No necesitamos desarrollar el hashing lento por nuestra cuenta. Se han desarrollado varios algoritmos especialmente para el hash de contraseñas:

- bcrypt,
- scrypt,
- PBKDF2,
- argon2,
- y otros.

Utilizan un algoritmo criptográfico complicado y asignan recursos como CPU o memoria deliberadamente.


Argon2
bcrypt
vs
scrypt

1. Password Encoder

1.4 Password Encoders

Password Encoders

Primero, echemos un vistazo a los codificadores de contraseñas de Spring Security. Todos los codificadores de contraseñas implementan la interfaz PasswordEncoder.

Esta interfaz define el método encode() para convertir la contraseña simple al formato codificado y el método matches() para comparar una contraseña simple con la contraseña codificada.

Cada codificador tiene un constructor predeterminado que crea una instancia con el factor de trabajo predeterminado. Podemos utilizar otros constructores para ajustar el factor de trabajo.

1. Password Encoder

1.5 BCrypt

BCryptPasswordEncoder

Este es el codificador más usado en la actualidad.

La mayoría de los ejemplos en Internet traen este como codificador principal.

Es muy bueno! Podemos decir que es el estándar hoy en día.

Modo de uso:

```
int strength = 10; // work factor of bcrypt
BCryptPasswordEncoder bCryptPasswordEncoder =
 new BCryptPasswordEncoder(strength, new SecureRandom());
String encodedPassword = bCryptPasswordEncoder.encode(plainPassword);
```

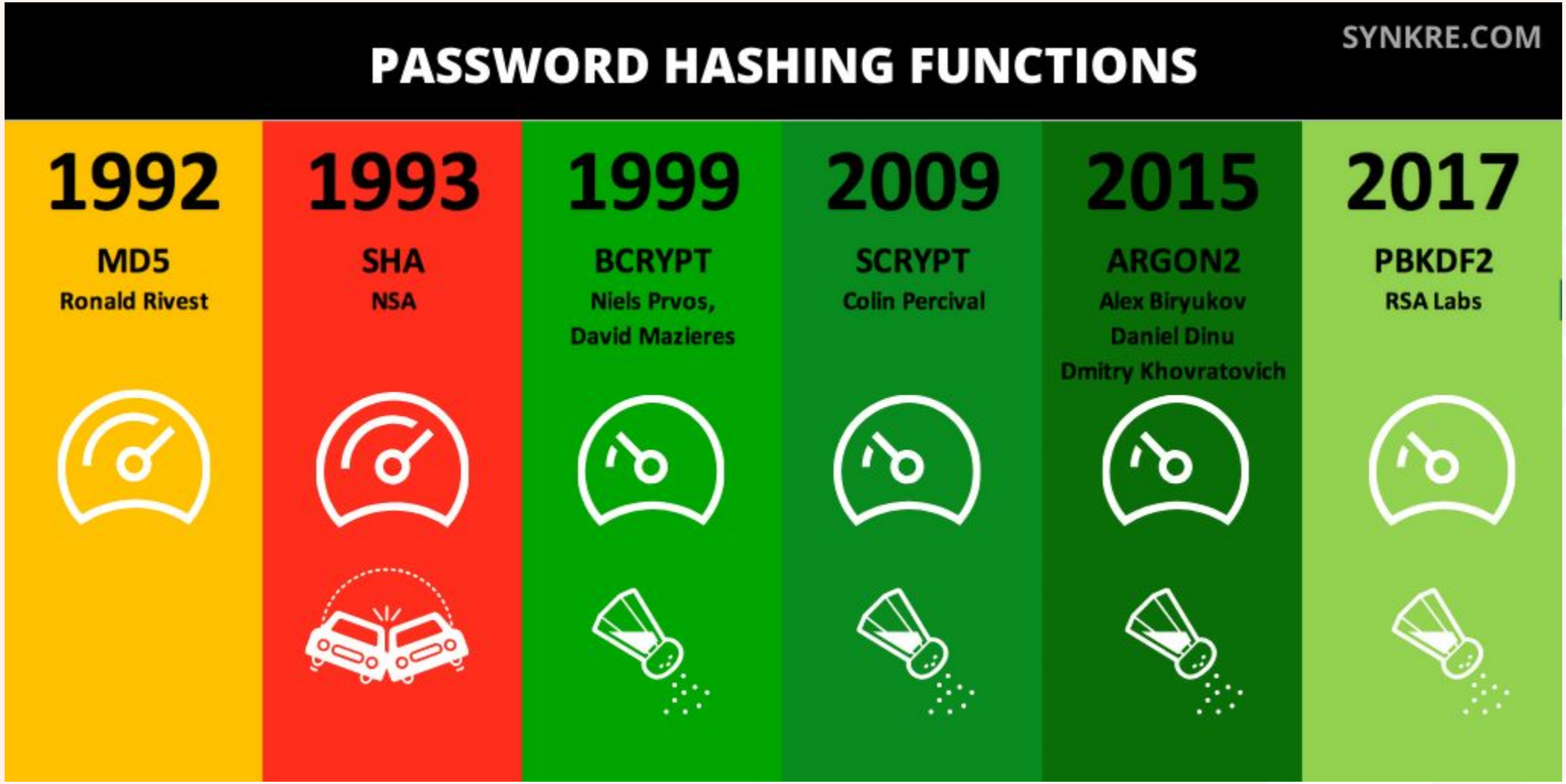
BCryptPasswordEncoder

BCryptPasswordEncoder tiene el parámetro fuerza. El valor predeterminado en Spring Security es 10 (pero se puede modificar). Se recomienda utilizar SecureRandom como generador de sal, ya que proporciona un número aleatorio criptográficamente fuerte.

La salida se ve así:

```
$2a$10$EzbrJCN8wj8M8B5aQiRmiuWqVvnxna73Ccvm38aoneiJb88kkwlH2
```

Tenga en cuenta que, a diferencia de los algoritmos hash simples como SHA-256 o MD5, la salida de bcrypt contiene metainformación sobre la versión del algoritmo, el factor de trabajo y la sal. No necesitamos guardar esta información por separado.


PASSWORD HASHING FUNCTIONS
SYNKRE.COM
1992
MD5
Ronald Rivest
1993
SHA
NSA
1999
BCRYPT
Niels Prvos,
David Mazieres
2009
SCRYPT
Colin Percival
2015
ARGON2
Alex Biryukov
Daniel Dinu
Dmitry Khovratovich
2017
PBKDF2
RSA Labs

¿Es bcrypt el mejor algoritmo de hash de contraseñas?

Bcrypt fue diseñado como una mejora del algoritmo de hash de contraseñas de Blowfish, específicamente para reducir la probabilidad de que

- ataques de fuerza bruta
- ataques de tabla arcoíris tengan éxito

Agrega un colchón adicional de seguridad al modificar la configuración de la clave Blowfish de tal manera que requiere más tiempo para producir una clave.

¿Es bcrypt el mejor algoritmo de hash de contraseñas?

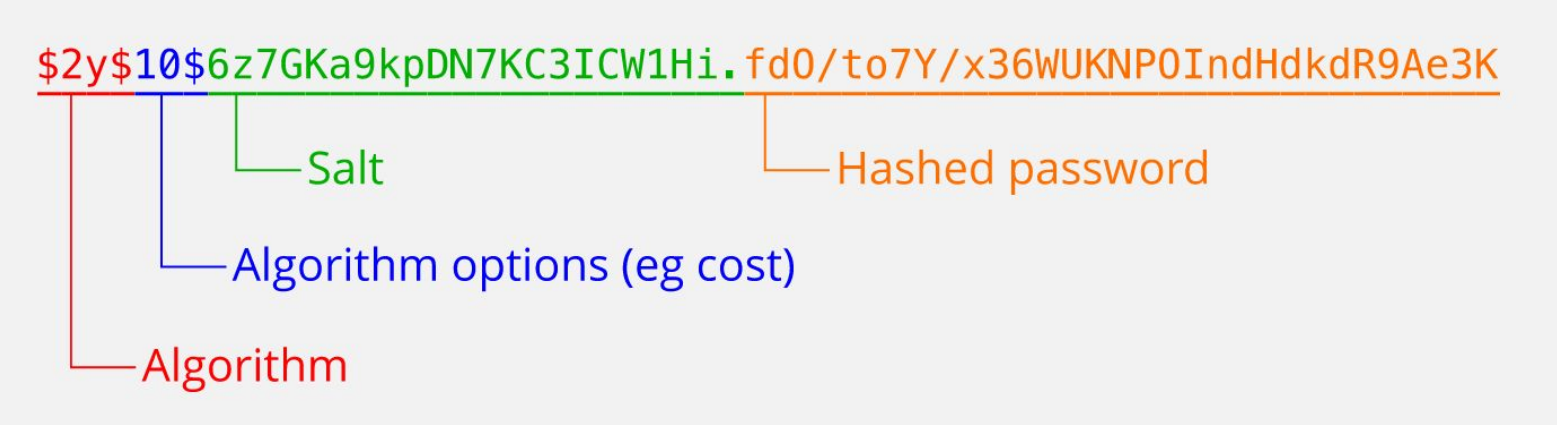
Bcrypt efectivamente agregó más rondas en su función de hash al calcular el hash al hacer que el número de rondas sea configurable y, por lo tanto, convertirlo en un hash más lento y fortalecer efectivamente la clave.

Redujo la probabilidad de ataques basados en diccionarios hash al simplificar el uso de salt, que se sabe que actúa en su contra.

¿Cuántos bytes puede tener el hash bcrypt?

Normalmente, 72 caracteres es el límite superior para las contraseñas que utilizan bcrypt, porque los fundamentos matemáticos del algoritmo lo establecen como el límite superior. En realidad, en las implementaciones se suelen utilizar 56 caracteres.

Bcrypt es un algoritmo de hashing de contraseñas que utiliza una variante del esquema de cifrado Blowfish. BCrypt es del año 1999 y es resistente por diseño a ataques de GPU-ASIC, ya que también es una función de refuerzo de la memoria: no solo requiere un uso intensivo de la CPU, sino también de la RAM para ejecutar un hash de bCrypt.



1. Password Encoder

1.6 Pbkdf2PasswordEncoder

Pbkdf2PasswordEncoder

El algoritmo PBKDF2 no fue diseñado para codificar contraseñas sino para derivar claves a partir de una contraseña. La derivación de clave generalmente es necesaria cuando queremos cifrar algunos datos con una contraseña, pero la contraseña no es lo suficientemente segura como para usarla como clave de cifrado.

Pbkdf2PasswordEncoder ejecuta el algoritmo hash sobre la contraseña simple muchas veces. También genera un salt. Podemos definir cuánto tiempo puede durar la salida y, además, usar un secreto llamado `pepper` para hacer que la codificación de la contraseña sea más segura.

Pbkdf2PasswordEncoder

Funciona de la siguiente manera:

```
String pepper = "pepper"; // secret key used by password encoding
int iterations = 200000;  // number of hash iteration
int hashWidth = 256;      // hash width in bits

Pbkdf2PasswordEncoder pbkdf2PasswordEncoder =
  new Pbkdf2PasswordEncoder(pepper, iterations, hashWidth);
pbkdf2PasswordEncoder.setEncodeHashAsBase64(true);
String encodedPassword = pbkdf2PasswordEncoder.encode(plainPassword);
```

La salida se ve de la siguiente manera:

```
lLDINGz0YLUUFQuuj5ChAsq0GNM9yHeUAJiL2Be7WUh43Xo3gmXNaw==
```

Pbkdf2PasswordEncoder

El salt se guarda internamente, pero tenemos que guardar el número de iteraciones y el ancho del hash por separado. El `pepper` debe mantenerse en secreto.

El número predeterminado de iteraciones es 185000 y el hash predeterminado es 256.

1. Password Encoder

1.7 SCryptPasswordEncoder

SCryptPasswordEncoder

```
int cpuCost = (int) Math.pow(2, 14); // factor to increase CPU costs
int memoryCost = 8;       // increases memory usage
int parallelization = 1; // currently not supported by Spring Security
int keyLength = 32;       // key length in bytes
int saltLength = 64;      // salt length in bytes

SCryptPasswordEncoder sCryptPasswordEncoder = new SCryptPasswordEncoder(
  cpuCost,
  memoryCost,
  parallelization,
  keyLength,
  saltLength);
String encodedPassword = sCryptPasswordEncoder.encode(plainPassword);
```

SCryptPasswordEncoder

El algoritmo scrypt no solo puede configurar el costo de la CPU sino también el costo de la memoria. De esta forma podemos encarecer aún más un ataque.

La salida se ve así:

```
$e0801$jRlFuIUd6eAZcuM1wKrzswD8TeKPed9wuWf3lwsWkStxHs0DvdpOZQB32cQJnf0lq/dxL+Qs
```

De hecho, es un string sumamente largo.

Este codificador coloca el parámetro para el factor de trabajo y el salt en la cadena de resultados, por lo que no hay información adicional para guardar.

1. Password Encoder

1.8 Argon2PasswordEncoder

Argon2PasswordEncoder

Argon2 es el ganador del Concurso de Hashing de Contraseñas en 2015. Este algoritmo también nos permite ajustar los costos de CPU y memoria. El codificador Argon2 guarda todos los parámetros en la cadena de resultado. Si queremos utilizar este codificador de contraseñas, tendremos que importar la biblioteca criptográfica BouncyCastle.

Conclusión

Spring Security admite muchos codificadores de contraseñas, tanto para algoritmos antiguos como modernos. Además, Spring Security proporciona métodos para trabajar con múltiples codificaciones de contraseñas en la misma aplicación. Podemos cambiar el factor de trabajo de las codificaciones de contraseñas o migrar de una codificación a otra sin afectar a los usuarios.

https://github.com/thombergs/code-examples/tree/master/spring-boot/password-encoding/

Contacto

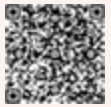
Mtro. Alfonso Gregorio Rivero Duarte

Senior Data Manager - CBRE

devil861109@gmail.com

Tels: (+52) 55 289970 69

Redes sociales:



https://www.linkedin.com/in/alfonso-gregorio-rivero-duarte-139a9225/