6.4. Password hashing

Table of contents

6.4.1. Overview

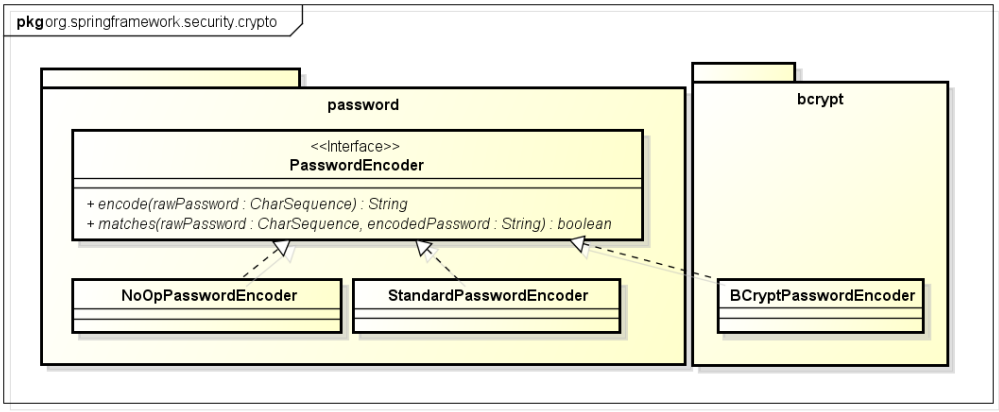
Password hashing is one of the points that must be considered in designing a secure application.
Not impossible that registers the password in plain text in a conventional system, but hashing is essential,
If the strength you have selected the weak algorithm due to "offline brute force attack" and "Rainbow crack"
Would be easily analyzed hashing original data.

Spring Security is, as a mechanism of hashed password
`org.springframework.security.crypto.password.PasswordEncoder` interface is available.
As its implementation class,

- `org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder`
- `org.springframework.security.crypto.password.StandardPasswordEncoder`

Etc., are provided.

`PasswordEncoder` as a mechanism, `encode (String RawPassword)` is performed hashed in the method,
`matches (String RawPassword, String EncodedPassword)` I do verification in the method.



Picture - PasswordEncoder Class Diagram

6.4.2. How to use

In this section, are provided by Spring Security, I will describe how to use the implementation class of PasswordEncoder.

Implementation class list of PasswordEncoder

PasswordEncoder implementation class of	Summary
`org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder`	Encoder to perform the hashing in "bcrypt" algorithm
`org.springframework.security.crypto.password.StandardPasswordEncoder`	Encoder to perform the hashing with "SHA-256" algorithm + 1024 stretch
`org.springframework.security.crypto.password.NoOpPasswordEncoder`	Encoder does not perform a hash (for testing)

If there is no requirement for hashing, `BCryptPasswordEncoder` it is recommended that you use. However, `BCryptPasswordEncoder` because many computation time in order to increase the pair aggression, if you do not meet the performance requirements for authentication `StandardPasswordEncoder` to consider.

Due to the existing systems, and algorithms for hashing, for if there is a limit for salt will be described later `org.springframework.security.authentication.encoding.PasswordEncoder` to be used the implementation class interface. For more information, *How to extend* see.

6.4.2.1. BCryptPasswordEncoder

`BCryptPasswordEncoder` A, `PasswordEncoder` implemented, and is a class that provides the hashed password. Using a salt of random 16-byte, it is encoder using bcrypt algorithm.

> **Note**
>
> Bcrypt algorithm is increased intentionally computational complexity than the generic algorithm. Therefore, from a general-purpose algorithm (SHA, MD5, etc.), it has a strong characteristic in the "offline brute force attack".

6.4.2.1.1. BCryptPasswordEncoder Configuration Examples

- applicationContext.xml

```
<Bean  id = "PasswordEncoder"
    class = "Org.Springframework.Security.Crypto.Bcrypt.BCryptPasswordEncoder"  />
<-! (1) ->
```

No.	Description
(1)	to passwordEncoder of class `BCryptPasswordEncoder` I specify a. To the constructor of the argument, I can specify the number of round hash of Salt. You can specify a value, is up to 4-31. By increasing the value specified, the strength increases, but the number of computations increases exponentially, be noted performance. If you do not specify, "10" is set.

Tip

Later in How to extend but, DaoAuthenticationProvider is, `Org.Springframework.Security.Crypto.Password.PasswordEncoder` implementation class of, `Org.Springframework.Security.Authentication.Encoding.PasswordEncoder` can be used to set the implementation class both. Therefore, from a conventional PasswordEncoder (authentication package), also when you are migrating to new PasswordEncoder, after password migration of user is completed, can respond by simply changing the passwordEncoder of DaoAuthenticationProvider.

Warning

`DaoAuthenticationProvider` If you have set the authentication provider, `UsernameNotFoundException` if is thrown, in order not to convict that there is no user in the user, `UsernameNotFoundException` after is thrown, you are intentionally hash the password of. (Side-channel attack countermeasures)

In order to create a value to be used for the above-mentioned hashing, when the application starts, `encode` is running once the method internally.

Warning

If you are using a SecureRandom in a Linux environment, delays and processing, there is a case in which a timeout occurs. Cause of this problem are those related to the random number generation is discussed in the following Java Bug Database.

http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6202721

In the b20 later versions of JDK 7, it has been modified.

http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6521844

If this problem occurs, and by setting the following startup argument JVM, can be avoided.

-Djava.security.egd = File: /// dev / urandom

- Java class

```
Inject
PasswordEncoder  PasswordEncoder ;   // (1)

public  String  register ( Customer  Customer ,  String  RawPassword )  {
    // OMITTED
    // Password Hashing
    String  password  =  PasswordEncoder . encode ( RawPassword );  // (2)
    Customer . setPassword ( password );
    // OMITTED
}

public  boolean  matches ( Customer  Customer ,  String  RawPassword )  {
    Return  PasswordEncoder . matches ( RawPassword ,  Customer . getPassword ());
// (3)
}
```

No.	Description
(1)	It was Bean definition, `PasswordEncoder` to injection the.
(2)	Example to hash the password it is possible to specify a clear text password as an argument of the encode method, the hashed

password is the return value.

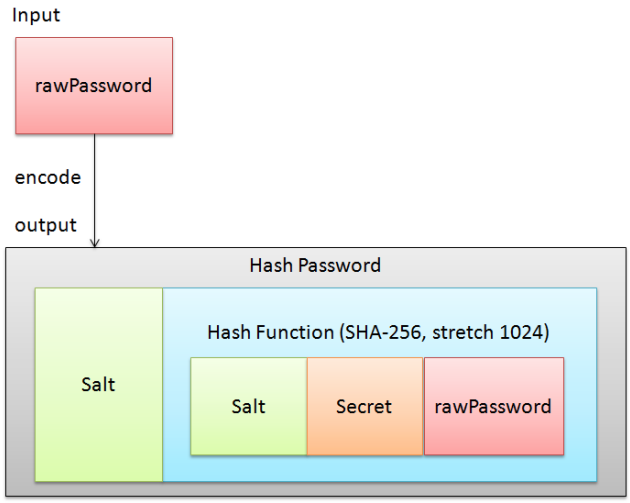
(3)	Example to match the password matches method plaintext password to the first argument, by specifying the hashed passwords in the second argument, Is a method or you can check match.

6.4.2.2. StandardPasswordEncoder

`StandardPasswordEncoder` The hashing algorithm using SHA-256, performs 1024 times stretch.
Also, I have granted the Salt of 8 bytes that is randomly generated.

Below, `StandardPasswordEncoder` of `encode (String RawPassword)` method,
`matches (String RawPassword, String EncodedPassword)` I will explain the mechanism of the method.

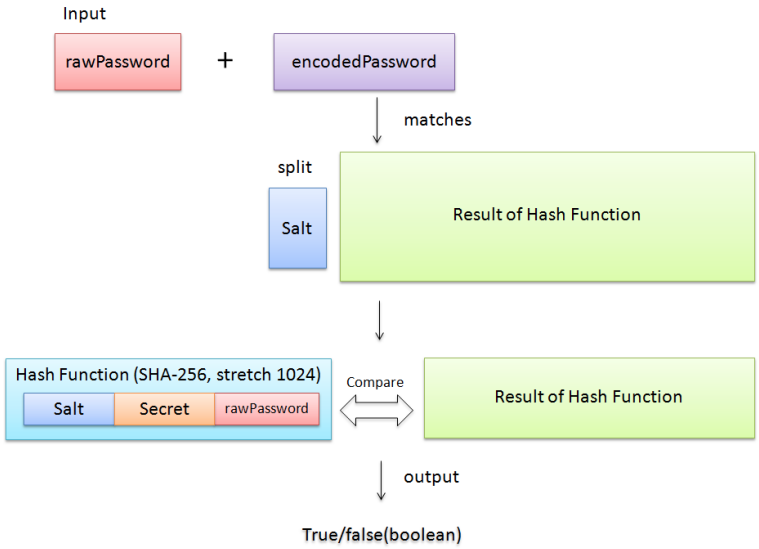
encode (String rawPassword) method



Picture - encode method

8 bytes of salt that is randomly generated + specified in the secret key + argument, it is hashed password.
The hashed value above, the value given at the beginning the salt used for hashing is, the return value of the method.

matches (String rawPassword, String encodedPassword) method



Picture - matches method

Was passed as an argument, and then split the beginning of the salt of encodedPassword, and hashed value in salt + secret + rawPassword
a value obtained by subtracting the first salt of encodedPassword I am performing the comparison process.

6.4.2.2.1. StandardPasswordEncoder Configuration Examples

- applicationContext.xml

```
<Bean  id = "PasswordEncoder"
  class = "Org.Springframework.Security.Crypto.Password.StandardPasswordEncoder" >
  <-! from Properties file ->
  <constructor-Arg  value = "$ {Passoword.Encoder.Secret}" / >  <-! (1) ->
</ bean>
```

No.	Description
(1)	I specify the private key for the hashed (secret). If specified, the hashing process is hashed in "salt generated by the internal" + a "private key given" + "Password". If you do not specify a private key (secret), the strength against attack method using a rainbow table is lowered, it is recommended that you specified. **For private key (secret)** Secret key (secret), it be treated as confidential information. Therefore, the properties file and you do not specify directly to Spring Security configuration file, to get from such environment variable. In this example, Example of acquiring from a property file is enabled. Also be aware of the location of the properties file in a production environment.

Tip

If you want to get the secret key a (secret) from the environment variable

StandardPasswordEncoder bean definitions, `<constructor-Arg>` can be obtained by performing the following settings.

```
<Bean  id = "PasswordEncoder"
  class =
"Org.Springframework.Security.Crypto.Password.StandardPasswordEncoder" >
  <-! from environment variable ->
  <constructor-Arg  value = "# {SystemEnvironment
['PASSWORD_ENCODER_SECRET']}"  />  <-! (1) ->
</ bean>
```

No.	Description
(1)	Environment variable: I get the value from PASSWORD_ENCODER_SECRET.

Java class example `BCryptPasswordEncoder` for similar and, *setting an example of BCryptPasswordEncoder* see.

6.4.2.3. NoOpPasswordEncoder

`NoOpPasswordEncoder` is a encoder to return the values specified in the raw string.
Such as during unit testing, and should not be used other than if you want to use a string that has not been hashed.

Configuration example, the same as that of the BCryptPasswordEncoder, omitted.

6.4.3. How to extend

Some operational requirements described above `PasswordEncoder` it may not be realized in a class that implements.
In particular, in a case where it is desired to follow the hashing scheme that uses the existing account information, the aforementioned `PasswordEncoder` often do not meet the requirements in.

For example, the existing hash method, cases are considered, such as the following.

- Algorithm is SHA-512.
- Stretch count is 1000 times.
- Salt is stored in the column of the account table, `PasswordEncoder` it is necessary to pass from the outside of.

In that case, `Org.Springframework.Security.Crypto.Password.PasswordEncoder` is not a class that implements,
Of different packages `org.springframework.security.authentication.encoding.PasswordEncoder` I recommend the use of a class that implements the.

Warning

Spring Security 3.1.4 and earlier,
`Org.Springframework.Security.Authentication.Encoding.PasswordEncoder` it had been used to hash the class that implements, and has become a Deprecated in 3.1.4 or later. Therefore, I different from the pattern that Spring is recommended.

6.4.3.1. Examples were used ShaPasswordEncoder

If business requirements is less than or equal to,
The algorithm uses the SHA-512, performs 1000 times stretching.
Authentication I described,
Have been used DaoAuthenticationProvider, to explain the authentication process as an example.

- applicationContext.xml

```
<Bean  id = "PasswordEncoder"
    class =
"Org.Springframework.Security.Authentication.Encoding.ShaPasswordEncoder" >  <-!
(1) ->
    <constructor-Arg  value = "512"  />  <-! (2 ) ->
    <property  name = "iterations"  value = "1000"  />  <-! (3) ->
</ bean>
```

No.	Description
(1)	The PasswordEncoder, `Org.Springframework.Security.Authentication.Encoding.ShaPasswordEncoder` I specify a. specified in passwordEncoder, class be changed to suit the algorithm to be used.
(2)	To the constructor of the argument, to set the type of SHA algorithm Possible values are "1,256,384,512". If it is omitted, "1" is set.
(3)	I specify the number of stretching at the time of hashing. If omitted, it becomes 0 times.

- spring-mvc.xml

```
<Bean  id = "AuthenticationProvider"
    class =
"Org.Springframework.Security.Authentication.Dao.DaoAuthenticationProvider" >
    <-! OMITTED ->
    <property  name = "SaltSource"  ref = "SaltSource"  />  <-! ( 1) ->
    <property  name = "UserDetailsService"  ref = "UserDetailsService"  />
    <property  name = "PasswordEncoder"  ref = "PasswordEncoder"  />  <-! (2) ->
</ bean>

<Bean  id = "SaltSource"
    class = "Org.Springframework.Security.Authentication.Dao.ReflectionSaltSource"
>  <-! (3) ->
    <property  name = "UserPropertyToUse"  value = "username"  />  <! - - (4) ->
</ bean>
```

No.	Description
(1)	If you want to external definition Salt, `Org.Springframework.Security.Authentication.Dao.SaltSource` the I set the BeanId of implementation class. In this example, I get the value set in the user information class in the reflection, `org.springframework.security.authentication.dao.ReflectionSaltSource` I have defined.
(2)	The PasswordEncoder, `Org.Springframework.Security.Authentication.Encoding.ShaPasswordEncoder` I specify a. specified in passwordEncoder, class be changed to suit the algorithm to be used.
(3)	Determine the Salt how to create `org.springframework.security.authentication.dao.SaltSource` I specify a. Here `UserDetails` to get in reflection properties of the object `ReflectionSaltSource` I use.
(4)	`UserDetails` object `usernamte` I use the property as a salt.

- Java class

```
Inject
PasswordEncoder  PasswordEncoder ;

public  String  register ( Customer  Customer ,  String  RawPassword ,  String
UserSalt )  {
    // OMITTED
    String  password  =  PasswordEncoder . encodePassword ( RawPassword ,
            UserSalt );  // (1)
    Customer . setPassword ( password );
    // OMITTED
}

public  boolean  matches ( Customer  Customer ,  String  RawPassword ,  String
UserSalt )  {
    Return  PasswordEncoder . IsPasswordValid ( Customer . getPassword (),
               RawPassword ,  UserSalt );  // (2)
}
```

No.	Description
(1)	If you want to hash the password, `org.springframework.security.authentication.encoding.PasswordEncoder` in the class that implements the `encodePassword` I specify a password, the Salt string argument of the method.
(2)	If you want to match the password, `isPasswordValid` Using the method, hash to the argument of passwords, By specifying plaintext password, the salt string, and compares the hashed passwords and clear-text password.

6.4.4. Appendix

Note

Stretch and the

By repeating the calculation of the hash function is to encrypt repeatedly store information about the password. As a countermeasure to the password brute forcing, is performed to prolong the time required for analysis password. However, since stretch affects the performance of the system, it is necessary to determine the number of stretching in view of the performance of the system.

Note

Salt The

Is a string to be added to the data to be original to be encrypted. By applying a salt to the password, apparently, a longer password length, is utilized to hard password analysis such as Rainbow Crack. Note that when utilizing the same salt to multiple users, when the user has set the same password exist, it would be found to be identical to the password from the hash value. Therefore, Salt is recommended that you set a different value for each user (random values, etc.).