Java MD5 Hashing & Salting: Secure Your Passwords

BY VIRAL PATEL · JUNE 8, 2012

The **MD5** Message-Digest Algorithm is a widely used *cryptographic hash function* that produces a 128-bit (16-byte) hash value. MD5 has been employed in a wide variety of

security applications, and is also commonly used to check data integrity. MD5 was designed by **Ron Rivest** in 1991 to replace an earlier hash function, MD4. An MD5 hash is typically expressed as a 32-digit hexadecimal number.

The Hash

A cryptographic hash function is a hash function, that is, an algorithm that takes an arbitrary block of data and returns a fixed-size bit string, the hash value, such that an change to the data will change the hash value. The data to be encoded is often called the "**message**," and the hash value is sometimes called the **message digest** or simply **digest**.

Java security package `java.security` provides certain useful classes to generate Hash values. Especially the class `java.security.MessageDiges` provides applications the functionality of a message digest algorithm, such as MD5 or SHA.

HTTPS – Self-signed certificate

WordPress – Allow Contributors to Add / Upload Media

Excel Macro: Evaluating Formulas Dynamically

Getting Started With Yeoman (Introduction to Yeoman)

Navigating Spring Security from thick Client to REST Webservice

Below is an example of generating MD5 Hash value for any input in Java using `java.security.MessageDiges`

```
package net.viralpatel

import java.math.BigIr
import java.security.M
import java.security.N

public class JavaMD5Ha

    public static voic

            String pas

            System.out


            System.out
            //= d41d8c


            System.out
                + md5(
            //= 9e107c
    }


    public static Stri

        String md5 = r

        if(null == inp

        try {

        //Create Messa
        MessageDigest

        //Update input
        digest.update(

        //Converts mes
        md5 = new Bigl

        } catch (NoSuc
```

```
            e.printSta
        }
        return md5;
    }
}
```

Above example is quite straight forward. The `main()` method calls `md5()` method to get MD5 hash value of any input. In `md5()` method we used `java.security.MessageDiges` class's object to generate Md5 hash. Note how we used `java.math.BigInteger` to convert the message digest into hex values of base 16.

Alternately, you can also use Apache Commons Codec library to covert any string to Md5 hash. The commons-codec.jar contains class `org.apache.commons.codec.d` that can be used to generate MD5 hash. Following is the code snippet for same:

```
package net.viralpatel

import org.apache.comm

public class JavaMD5Ha

    public static voic

            String pas
```

```
                System.out

        }
    }
```

Thus if your project already uses commons-codec jar then it is advisable to use `DigestUtils.md5Hex()` method to generate MD5 hash values.

The Salt

The wikipedia definition of salt is:

> *In cryptography, a salt consists of random bits, creating one of the inputs to a one-way function. The other input is usually a password or passphrase. The output of the one-way function can be stored rather than the password, and still be used for authenticating users. The one-way function typically uses a cryptographic hash function.*

Thus basically, salt is a random data string that one can append to the password

before hashing it. Consider a scenario where in a system users passwords are stored in database table after hashing them. So the plain text password are hashed using hash algo like Md5 and than stored in database. Now it is possible to perform a dictionary attack based on hash values. A plain hash can be looked up in a lookup table and its corresponding password string can be retrieved.

So instead of directly hashing passwords, we append a random string to it before hashing.

```
String password = "myp
String salt = "Random$
String hash = md5(pass
```

Thus the benefit provided by using a salted password is making a lookup table assisted dictionary attack against the stored values impractical, provided the salt is large enough. That is, an attacker would not be able to create a precomputed lookup table of hashed values (password +

salt), because it would take too much space. A simple dictionary attack is still very possible, although much slower since it cannot be precomputed.

It is generally advisable to have salt value large and random with lot of special characters. Also application wide single salt value must be used so that users can be authenticated irrespective of when accounts were created. The salt value must be stored outside application data and must be loaded with application.

References

- Md5 Wiki article
- Salt_cryptography wiki article
- java.security.MessageDiges Javadoc
- Apache Commons Codec DigestUtils Javadoc

Related Articles

1. **Gravatar: Manage your user avatars for free**
2. **Sending Emails in Java**

using GMail ID
3. Batch Insert In Java – JDBC
4. Read / Write CSV file in Java
5. Creating & Parsing JSON data with Java Servlet/Struts/JSP
6. Calculate Free Disk Space in Java using Apache Commons IO
7. 20 very useful Java code snippets for Java Developers

Tags: core java Java java code

PREVIOUS STORY
Index Skip Scan in Oracle

NEXT STORY
Introduction to FreeMarker Template (FTL)

YOU MAY ALSO LIKE...

Performance Monitoring using Glassbox

Eclipse: This project needs to migrate WTP metadata

Dynamic Class Loading using Java Reflection API

HTML5 Server-Sent Events with Java Servlets example

9 COMMENTS

Cristian Rivera

18 June, 2012, 0:54

Hello, I am wondering why you would use MD5, an algorithm thats has already been broken instead of something like GrandCentral which creates a password digest based on the time of day using SHA-512. Just a suggestion.
http://code.google.com/p/grandcentral/

Reply

pradeep

27 August, 2012, 13:53

Really useful, thanks to viral patel.

Reply

simar

20 December, 2012, 14:29

i believe that it
String hash =
md5(password + salt;

has to be changed to
String hash =
md5(password + salt);

and usually you have to keep salt somewhere and for more security for each passwod to have a new salt.
i do preref
Sting hash =
md5(password+salt) + salt;

Reply

simar

20 December, 2012, 14:34

pay attention to
apache.common
MessageDigest digest =
MessageDigest.getInstance("MD5");
this plaform specific and
could not work on some.

better to pay attention to
apache.common
there are useful class:
RandomUtilString to
generate string (numeric,
alphanumeric and so on)
and as well class
DigestUtils with a certain
amoutn of supported
algorithms and diff method
to get hashed string
hashed bytes

Reply

gli00001
21 March, 2013, 3:44
//Update input string in
message digest
digest.update(input.getBytes(),
0, input.length());
shouldn't it be
input.getBytes().length() ?

Reply

Sif 27 April, 2013, 10:56
This is horrifyingly bad
advice. MD5 is NOT
APPROPRIATE for
password storage (it never
was). Nor are any of the
SHA algorithms (the
unreleased SHA-3 might
become an exception).

MD5 & the SHA family are
horribly unsuited for
password storage due to
being far too fast. A

modern GPU can do 10+ billion MD5 calculations per second. That's enough guesses to wreck a massive database in no time at all, especially with how intelligent modern cracking tools like oclHashcat are (it's not just brute forcing all possible combinations… hackers have a better idea how you formulate your passwords than you do!).

Use bcrypt, scrypt, or PBKDF2. Pretty much every language out there easily supports one or more of those three, usually in their standard library. They're designed for password storage, widely recommended by actual security experts, and can be configured to be as slow as you deem necessary.

Likewise, DO NOT HAVE A SINGLE SALT FOR ALL PASSWORDS! You defeat the primary purpose of a salt by doing that. A salt's primary purpose is NOT to defeat rainbow table attacks or any similar attack. It's to kill an attacker's ability to attack multiple hashes simultaneously. A common salt does not do that. Every time you write the password to your database (account creation, user

password change) you should salt it with a new random salt, created by a cryptographically secure random number generator (32 – 64 bits should be adequate). And store this salt.

So, in summary: Do not use MD5/SHA, use a key derivation function designed for passwords (bcrypt, scrypt, pbkdf2 being the most popular and widely supported). Do not have a single site-wide salt, have a *per-user* salt. Seriously, doing the right thing is easy, and will increase your security an absurd amount.

Reply

Rajesh Perul

23 April, 2014, 12:16

Hi I got an error……

Method md5(String input) is not work cool with a string that will generate and md5 string with zero at left side.

(eg:-this method will generat a string “sandeep” to “DCF16D903E5890AABA465B0B1BA51F ” than the actual “00DCF16D903E5890AABA465B0B1BA51F”

use this method instead..
String md5(String s) {
try {
// Create MD5 Hash
MessageDigest digest =

```
java.security.MessageDigest.getInstance("MD5");
digest.update(s.getBytes());
byte messageDigest[] =
digest.digest();

// Create Hex String
StringBuffer hexString =
new StringBuffer();
for (int i = 0; i <
messageDigest.length; i++)
{
String h =
Integer.toHexString(0xFF &
messageDigest[i]);
while (h.length() < 2)
h = "0" + h;
hexString.append(h);
}
return hexString.toString();

} catch
(NoSuchAlgorithmException
e) {
e.printStackTrace();
}
return "";
}
```

Reply

Keyur Bhatt

8 January, 2015, 8:43

Hi, nice article.
I want to know that is there any algorithm available that will generate same result in all programming languages. I am trying to hash string using salt.
Thank You.

Reply

Rakesh

13 August, 2015, 16:54

nice tutorial and very

useful for encryption.:)

Reply

LEAVE A REPLY

Name * **Email ***

Website

Comment

Post Comment