MD5 hash in Java ME

This article explains how to use the Security and Trust Services API (SATSA) to calculate MD5 hashes in a Java ME application.

Introduction

MD5 is a cryptographic algorithm that allows the creation of 16-byte hashes (or digests) of a given input. These hashes are widely used in multiple scenarios where security is a requirement, including:

- storing of password: instead of storing clear-text passwords, an application can store their MD5 hashes to increase security in case of malicious accesses
- data integrity: hashes can be calculated and used to verify that data is transferred without errors and tampering

Sample Java app demostrating MD5 hashing features of JSR177

This article shows how the JSR 177 (SATSA) can be used to produce MD5 hashes.

SATSA - JSR 177

The base class, defined by the *JSR177*, and used to calculate digests is MessageDigest.

An object of this class can be instantiated with its getInstance() static method, by passing the desired hashing algorithm as argument. In this case, the instantiation can be performed as shown below:

```
MessageDigest digest = MessageDigest.getInstance("MD5");
```

If the platform does not support the specified algorithm (MD5 in this case), a NoSuchAlgorithmException is raised, and must be handled accordingly.

Once the `MessageDigest` instance is available, its input can be set via its update method, that adds the specified bytes to the input data. The second and third arguments passed to the `update` method specify the start index and length of the input data to be added.

```
byte[] txtBytes = clearText.getBytes();

digest.update(txtBytes, 0, txtBytes.length);
```


(C) Copyright Nokia 2014. All rights reserved.

When the input data is ready and has been set, the digest computation can be performed by calling the digest method, that takes three arguments:

- a `byte array`, that will contain the hashed value bytes
- an `int` value, specifying the start index within the output byte array
- an `int` value specifying the number of bytes available for the digest within the output byte array

Since an MD5 hash value takes 16 bytes, the process described above can be translated into code as follows:

```
byte[] hashedBytes = new byte[16];

int bytesNum = digest.digest(hashedBytes, 0, hashedBytes.length);
```

Note: The value returned by the `digest` method specifies the number of bytes written in the output array. For MD5 hashes this value is equal to 16

The hashed value is returned from the `digest` method as a byte array, but MD5 hashes are usually represented and used with their hexadecimal form. Such a conversion can be performed as follows:

- each byte is converted to its hexadecimal value with the `Integer.toHexString` method
- if the length of the hexadecimal value is equal to 1 (so, if the corresponding value is less than 16) append a '0' as prefix

```
StringBuffer hexStringBuffer = new StringBuffer();

for (int i = 0; i < bytesNum; i++)
{
 String hex = Integer.toHexString(0xFF & hashedBytes[i]);

 if (hex.length() == 1)
     hexStringBuffer.append('0');

 hexStringBuffer.append(hex);
}
String hexString = hexStringBuffer.toString();
```

Summary

This article shows how an MD5 hash value can be computed on devices supporting the Security and Trust Services API (SATSA) together with the *MD5* hashing algorithm.

On devices not supporting this API (for instance, Series 40 versions previous to 3rd Edition Feature Pack 2), external libraries like BouncyCastle can be used to achieve the same result.

The ZIP file attached to this article shows a complete example of MD5 hashing, allowing the user to enter a custom input text, and generating the corresponding digest: Media:WikiMD5Example.zip

Version Hint

Windows Phone: [[Category:Windows Phone]]
[[Category:Windows Phone 7.5]]
[[Category:Windows Phone 8]]

Nokia Asha: [[Category:Nokia Asha]]
[[Category:Nokia Asha Platform 1.0]]

Series 40: [[Category:Series 40]]
[[Category:Series 40 1st Edition]] [[Category:Series 40 2nd Edition]]
[[Category:Series 40 3rd Edition (initial release)]] [[Category:Series 40 3rd Edition FP1]] [[Category:Series 40 3rd Edition FP2]]
[[Category:Series 40 5th Edition (initial release)]] [[Category:Series 40 5th Edition FP1]]

(C) Copyright Nokia 2014. All rights reserved.

[[Category:Series 40 6th Edition (initial release)]] [[Category:Series 40 6th Edition FP1]] [[Category:Series 40 Developer Platform 1.0]] [[Category:Series 40 Developer Platform 1.1]] [[Category:Series 40 Developer Platform 2.0]]

Symbian: [[Category:Symbian]]
[[Category:S60 1st Edition]] [[Category:S60 2nd Edition (initial release)]] [[Category:S60 2nd Edition FP1]] [[Category:S60 2nd Edition FP2]] [[Category:S60 2nd Edition FP3]]
[[Category:S60 3rd Edition (initial release)]] [[Category:S60 3rd Edition FP1]] [[Category:S60 3rd Edition FP2]]
[[Category:S60 5th Edition]]
[[Category:Symbian^3]] [[Category:Symbian Anna]] [[Category:Nokia Belle]]

(C) Copyright Nokia 2014. All rights reserved.