# **Java MessageDigest**

The *Java MessageDigest* class represents a cryptographic hash function which can calculate a message digest from binary data. When you receive some encrypted data you cannot see from the data itself whether it was modified during transportation. A message digest can help alleviate that problem.

To be able to detect if the encrypted data has been modified in transport, the sender can calculate a message digest from the data and send that along with the data. When you receive the encrypted data and message digest you can recalculate the message digest from the data and check if the message calculated digest matches the message digest received with the data. If the two message digests match there is a probability that the encrypted data was not modified during transport.

There are a couple of conditions that have to be met for a message digest to be useful as a modification detection mechanism. However, the exact conditions are part of cryptographic theory, so you will have to visit that theory to read how to use message digests correctly. This tutorial only explains how to use the Java Cryptography API representation of a message digest in the MessageDigest class.

**Creating a MessageDigest Instance**

To create a Java MessageDigest instance you call the static getInstance() method of the MessageDigestclass. Here is an example of creating a MessageDigest instance:

MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");

The text parameter passed to the getInstance() method is the name of the concrete message digest algorithm to use.

**Message Digest Algorithms**

The Java Cryptography API supports the following message digest algorithms (although your concrete cryptography provider might support more!):

|  |
| --- |
| **Algorithm Name** |
| MD2 |
| MD5 |
| SHA-1 |
| SHA-256 |
| SHA-384 |
| SHA-512 |

Not all of these message digest algorithms are equally secure. At the time of writing it is recommended that you use SHA-256 or higher to get the highest possible amount of security.

**Calculate Message Digest**

Once you have created a Java MessageDigest instance you can use it to calculate a message digest from data. If you have a single block of data to calculate a message digest from, use the digest() method. Here is how calculating a message digest from a single block of data looks:

byte[] data1 = "0123456789".getBytes("UTF-8");

MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");

byte[] digest = messageDigest.digest(data1);

If you have multiple blocks of data to include in the same message digest, call the update() method and finish off with a call to digest(). Here is how calculating a message digest from multiple blocks of data looks:

byte[] data1 = "0123456789".getBytes("UTF-8");

byte[] data2 = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");

MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");

messageDigest.update(data1);

messageDigest.update(data2);

byte[] digest = messageDigest.digest();

### Creating a MessageDigest Object

The first step for computing a digest is to create a message digest instance. MessageDigest objects are obtained by using one of the [getInstance() static factory methods](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html" \l "ProviderImplReq) in the MessageDigest class. The factory method returns an initialized message digest object. It thus does not need further initialization.

### Updating a Message Digest Object

The next step for calculating the digest of some data is to supply the data to the initialized message digest object. It can be provided all at once, or in chunks. Pieces can be fed to the message digest by calling one of the update methods:

void update(byte input)

void update(byte[] input)

void update(byte[] input, int offset, int len)

### Computing the Digest

After the data chunks have been supplied by calls to update, the digest is computed using a call to one of the digest methods:

byte[] digest()

byte[] digest(byte[] input)

int digest(byte[] buf, int offset, int len)

The first method return the computed digest. The second method does a final update(input) with the input byte array before calling digest(), which returns the digest byte array. The last method stores the computed digest in the provided buffer buf, starting at offset. len is the number of bytes in buf allotted for the digest, the method returns the number of bytes actually stored in buf. If there is not enough room in the buffer, the method will throw an exception.

Please see the [Computing a MessageDigest](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html#MDEx) example in the [Code Examples](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html#Examples) section for more details.