# **Java Mac**

The *Java Mac* (javax.crypto.Mac class can create a Message Authentication Code (MAC) from binary data. A MAC is a [**message digest**](http://tutorials.jenkov.com/java-cryptography/messagedigest.html) which has been encrypted with a secret key. Only if you have the secret key can you verify the MAC.

**Creating a Mac Instance**

Before you can use the Java Mac class you must create a Mac instance. Creating a Mac instance is done using the getInstance() method. Here is a Java Mac instantiation example:

Mac mac = Mac.getInstance("HmacSHA256");

The String parameter passed to the Mac getInstance() method contains the name of the MAC algorithm to use. In this case the MAC algorithm is HmacSHA256.

**Initializing the Mac**

Once created, the Java Mac instance must be initialized. You initialize the Mac instance by calling its init()method passing as parameter the secret key to be used by the Mac instance. Here is a Java Macinitialization example:

byte[] keyBytes = new byte[]{0,1,2,3,4,5,6,7,8 ,9,10,11,12,13,14,15};

String algorithm = "RawBytes";

SecretKeySpec key = new SecretKeySpec(keyBytes, algorithm);

mac.init(key);

The Mac init() method takes a Key instance. In this example a SecretKeySpec is used which implements the Key interface.

**Calculating the MAC**

Once the Java Mac instance is initialized you can start calculating MAC values with it. To calculate a MAC value you call the Mac update() or doFinal() method. If you only have a single block of data to calculate the MAC for, you can call doFinal() directly, like this:

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

byte[] macBytes = mac.doFinal(data);

If you have multiple blocks of data to calculate the MAC for, e.g. if you are reading a file block by block, then you must call the update() method with each block, and finish with a call to doFinal(). Here is an example:

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

byte[] data2 = "0123456789".getBytes("UTF-8");

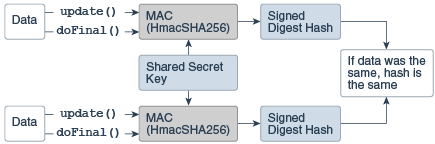
mac.update(data);

mac.update(data2);

byte[] macBytes = mac.doFinal();

## The Mac Class

Similar to a MessageDigest, a Message Authentication Code (MAC) provides a way to check the integrity of information transmitted over or stored in an unreliable medium, but includes a secret key in the calculation. Only someone with the proper key will be able to verify the received message. Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties.



[Description of Figure 8: The Mac Class](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec_image_descriptions.html" \l "mac)

A MAC mechanism that is based on cryptographic hash functions is referred to as HMAC. HMAC can be used with any cryptographic hash function, e.g., SHA-256, in combination with a secret shared key.

The Mac class provides the functionality of a Message Authentication Code (MAC). Please refer to the [code example](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html#HmacEx).

### Creating a Mac Object

Mac objects are obtained by using one of the Mac [getInstance() static factory methods](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html#ProviderImplReq).

### Initializing a Mac Object

A Mac object is always initialized with a (secret) key and may optionally be initialized with a set of parameters, depending on the underlying MAC algorithm.

To initialize a Mac object, call one of its init methods:

public void init(Key key);

public void init(Key key, AlgorithmParameterSpec params);

You can initialize your Mac object with any (secret-)key object that implements the javax.crypto.SecretKey interface. This could be an object returned by javax.crypto.KeyGenerator.generateKey(), or one that is the result of a key agreement protocol, as returned byjavax.crypto.KeyAgreement.generateSecret(), or an instance of javax.crypto.spec.SecretKeySpec.

With some MAC algorithms, the (secret-)key algorithm associated with the (secret-)key object used to initialize the Mac object does not matter (this is the case with the HMAC-MD5 and HMAC-SHA1 implementations of the SunJCE provider). With others, however, the (secret-)key algorithm does matter, and anInvalidKeyException is thrown if a (secret-)key object with an inappropriate (secret-)key algorithm is used.

### Computing a MAC

A MAC can be computed in one step (*single-part operation*) or in multiple steps (*multiple-part operation*). A multiple-part operation is useful if you do not know in advance how long the data is going to be, or if the data is too long to be stored in memory all at once.

To compute the MAC of some data in a single step, call the following doFinal method:

public byte[] doFinal(byte[] input);

To compute the MAC of some data in multiple steps, call one of the update methods:

public void update(byte input);

public void update(byte[] input);

public void update(byte[] input, int inputOffset, int inputLen);

A multiple-part operation must be terminated by the above doFinal method (if there is still some input data left for the last step), or by one of the following doFinal methods (if there is no input data left for the last step):

public byte[] doFinal();

public void doFinal(byte[] output, int outOffset);