| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Cipher.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/crypto/BadPaddingException.html)   [**NEXT CLASS**](http://docs.google.com/javax/crypto/CipherInputStream.html) | [**FRAMES**](http://docs.google.com/index.html?javax/crypto/Cipher.html)    [**NO FRAMES**](http://docs.google.com/Cipher.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#1ksv4uv) | [METHOD](#2jxsxqh) |

## **javax.crypto**

Class Cipher

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **javax.crypto.Cipher**

**Direct Known Subclasses:** [NullCipher](http://docs.google.com/javax/crypto/NullCipher.html)

public class **Cipher**extends [Object](http://docs.google.com/java/lang/Object.html)

This class provides the functionality of a cryptographic cipher for encryption and decryption. It forms the core of the Java Cryptographic Extension (JCE) framework.

In order to create a Cipher object, the application calls the Cipher's getInstance method, and passes the name of the requested *transformation* to it. Optionally, the name of a provider may be specified.

A *transformation* is a string that describes the operation (or set of operations) to be performed on the given input, to produce some output. A transformation always includes the name of a cryptographic algorithm (e.g., *DES*), and may be followed by a feedback mode and padding scheme.

A transformation is of the form:

* "*algorithm/mode/padding*" or
* "*algorithm*"

(in the latter case, provider-specific default values for the mode and padding scheme are used). For example, the following is a valid transformation:

Cipher c = Cipher.getInstance("*DES/CBC/PKCS5Padding*");

Using modes such as CFB and OFB, block ciphers can encrypt data in units smaller than the cipher's actual block size. When requesting such a mode, you may optionally specify the number of bits to be processed at a time by appending this number to the mode name as shown in the "DES/CFB8/NoPadding" and "DES/OFB32/PKCS5Padding" transformations. If no such number is specified, a provider-specific default is used. (For example, the SunJCE provider uses a default of 64 bits for DES.) Thus, block ciphers can be turned into byte-oriented stream ciphers by using an 8 bit mode such as CFB8 or OFB8.

**Since:** 1.4 **See Also:**[KeyGenerator](http://docs.google.com/javax/crypto/KeyGenerator.html), [SecretKey](http://docs.google.com/javax/crypto/SecretKey.html)

| **Field Summary** | |
| --- | --- |
| static int | [**DECRYPT\_MODE**](http://docs.google.com/javax/crypto/Cipher.html#DECRYPT_MODE)            Constant used to initialize cipher to decryption mode. |
| static int | [**ENCRYPT\_MODE**](http://docs.google.com/javax/crypto/Cipher.html#ENCRYPT_MODE)            Constant used to initialize cipher to encryption mode. |
| static int | [**PRIVATE\_KEY**](http://docs.google.com/javax/crypto/Cipher.html#PRIVATE_KEY)            Constant used to indicate the to-be-unwrapped key is a "private key". |
| static int | [**PUBLIC\_KEY**](http://docs.google.com/javax/crypto/Cipher.html#PUBLIC_KEY)            Constant used to indicate the to-be-unwrapped key is a "public key". |
| static int | [**SECRET\_KEY**](http://docs.google.com/javax/crypto/Cipher.html#SECRET_KEY)            Constant used to indicate the to-be-unwrapped key is a "secret key". |
| static int | [**UNWRAP\_MODE**](http://docs.google.com/javax/crypto/Cipher.html#UNWRAP_MODE)            Constant used to initialize cipher to key-unwrapping mode. |
| static int | [**WRAP\_MODE**](http://docs.google.com/javax/crypto/Cipher.html#WRAP_MODE)            Constant used to initialize cipher to key-wrapping mode. |

| **Constructor Summary** | |
| --- | --- |
| protected | [**Cipher**](http://docs.google.com/javax/crypto/Cipher.html#Cipher(javax.crypto.CipherSpi,%20java.security.Provider,%20java.lang.String))([CipherSpi](http://docs.google.com/javax/crypto/CipherSpi.html) cipherSpi, [Provider](http://docs.google.com/java/security/Provider.html) provider, [String](http://docs.google.com/java/lang/String.html) transformation)            Creates a Cipher object. |

| **Method Summary** | |
| --- | --- |
| byte[] | [**doFinal**](http://docs.google.com/javax/crypto/Cipher.html#doFinal())()            Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized. |
| byte[] | [**doFinal**](http://docs.google.com/javax/crypto/Cipher.html#doFinal(byte%5B%5D))(byte[] input)            Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. |
| int | [**doFinal**](http://docs.google.com/javax/crypto/Cipher.html#doFinal(byte%5B%5D,%20int))(byte[] output, int outputOffset)            Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized. |
| byte[] | [**doFinal**](http://docs.google.com/javax/crypto/Cipher.html#doFinal(byte%5B%5D,%20int,%20int))(byte[] input, int inputOffset, int inputLen)            Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. |
| int | [**doFinal**](http://docs.google.com/javax/crypto/Cipher.html#doFinal(byte%5B%5D,%20int,%20int,%20byte%5B%5D))(byte[] input, int inputOffset, int inputLen, byte[] output)            Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. |
| int | [**doFinal**](http://docs.google.com/javax/crypto/Cipher.html#doFinal(byte%5B%5D,%20int,%20int,%20byte%5B%5D,%20int))(byte[] input, int inputOffset, int inputLen, byte[] output, int outputOffset)            Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. |
| int | [**doFinal**](http://docs.google.com/javax/crypto/Cipher.html#doFinal(java.nio.ByteBuffer,%20java.nio.ByteBuffer))([ByteBuffer](http://docs.google.com/java/nio/ByteBuffer.html) input, [ByteBuffer](http://docs.google.com/java/nio/ByteBuffer.html) output)            Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. |
| [String](http://docs.google.com/java/lang/String.html) | [**getAlgorithm**](http://docs.google.com/javax/crypto/Cipher.html#getAlgorithm())()            Returns the algorithm name of this Cipher object. |
| int | [**getBlockSize**](http://docs.google.com/javax/crypto/Cipher.html#getBlockSize())()            Returns the block size (in bytes). |
| [ExemptionMechanism](http://docs.google.com/javax/crypto/ExemptionMechanism.html) | [**getExemptionMechanism**](http://docs.google.com/javax/crypto/Cipher.html#getExemptionMechanism())()            Returns the exemption mechanism object used with this cipher. |
| static [Cipher](http://docs.google.com/javax/crypto/Cipher.html) | [**getInstance**](http://docs.google.com/javax/crypto/Cipher.html#getInstance(java.lang.String))([String](http://docs.google.com/java/lang/String.html) transformation)            Returns a Cipher object that implements the specified transformation. |
| static [Cipher](http://docs.google.com/javax/crypto/Cipher.html) | [**getInstance**](http://docs.google.com/javax/crypto/Cipher.html#getInstance(java.lang.String,%20java.security.Provider))([String](http://docs.google.com/java/lang/String.html) transformation, [Provider](http://docs.google.com/java/security/Provider.html) provider)            Returns a Cipher object that implements the specified transformation. |
| static [Cipher](http://docs.google.com/javax/crypto/Cipher.html) | [**getInstance**](http://docs.google.com/javax/crypto/Cipher.html#getInstance(java.lang.String,%20java.lang.String))([String](http://docs.google.com/java/lang/String.html) transformation, [String](http://docs.google.com/java/lang/String.html) provider)            Returns a Cipher object that implements the specified transformation. |
| byte[] | [**getIV**](http://docs.google.com/javax/crypto/Cipher.html#getIV())()            Returns the initialization vector (IV) in a new buffer. |
| static int | [**getMaxAllowedKeyLength**](http://docs.google.com/javax/crypto/Cipher.html#getMaxAllowedKeyLength(java.lang.String))([String](http://docs.google.com/java/lang/String.html) transformation)            Returns the maximum key length for the specified transformation according to the installed JCE jurisdiction policy files. |
| static [AlgorithmParameterSpec](http://docs.google.com/java/security/spec/AlgorithmParameterSpec.html) | [**getMaxAllowedParameterSpec**](http://docs.google.com/javax/crypto/Cipher.html#getMaxAllowedParameterSpec(java.lang.String))([String](http://docs.google.com/java/lang/String.html) transformation)            Returns an AlgorithmParameterSpec object which contains the maximum cipher parameter value according to the jurisdiction policy file. |
| int | [**getOutputSize**](http://docs.google.com/javax/crypto/Cipher.html#getOutputSize(int))(int inputLen)            Returns the length in bytes that an output buffer would need to be in order to hold the result of the next update or doFinal operation, given the input length inputLen (in bytes). |
| [AlgorithmParameters](http://docs.google.com/java/security/AlgorithmParameters.html) | [**getParameters**](http://docs.google.com/javax/crypto/Cipher.html#getParameters())()            Returns the parameters used with this cipher. |
| [Provider](http://docs.google.com/java/security/Provider.html) | [**getProvider**](http://docs.google.com/javax/crypto/Cipher.html#getProvider())()            Returns the provider of this Cipher object. |
| void | [**init**](http://docs.google.com/javax/crypto/Cipher.html#init(int,%20java.security.cert.Certificate))(int opmode, [Certificate](http://docs.google.com/java/security/cert/Certificate.html) certificate)            Initializes this cipher with the public key from the given certificate. |
| void | [**init**](http://docs.google.com/javax/crypto/Cipher.html#init(int,%20java.security.cert.Certificate,%20java.security.SecureRandom))(int opmode, [Certificate](http://docs.google.com/java/security/cert/Certificate.html) certificate, [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) random)            Initializes this cipher with the public key from the given certificate and a source of randomness. |
| void | [**init**](http://docs.google.com/javax/crypto/Cipher.html#init(int,%20java.security.Key))(int opmode, [Key](http://docs.google.com/java/security/Key.html) key)            Initializes this cipher with a key. |
| void | [**init**](http://docs.google.com/javax/crypto/Cipher.html#init(int,%20java.security.Key,%20java.security.AlgorithmParameters))(int opmode, [Key](http://docs.google.com/java/security/Key.html) key, [AlgorithmParameters](http://docs.google.com/java/security/AlgorithmParameters.html) params)            Initializes this cipher with a key and a set of algorithm parameters. |
| void | [**init**](http://docs.google.com/javax/crypto/Cipher.html#init(int,%20java.security.Key,%20java.security.spec.AlgorithmParameterSpec))(int opmode, [Key](http://docs.google.com/java/security/Key.html) key, [AlgorithmParameterSpec](http://docs.google.com/java/security/spec/AlgorithmParameterSpec.html) params)            Initializes this cipher with a key and a set of algorithm parameters. |
| void | [**init**](http://docs.google.com/javax/crypto/Cipher.html#init(int,%20java.security.Key,%20java.security.spec.AlgorithmParameterSpec,%20java.security.SecureRandom))(int opmode, [Key](http://docs.google.com/java/security/Key.html) key, [AlgorithmParameterSpec](http://docs.google.com/java/security/spec/AlgorithmParameterSpec.html) params, [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) random)            Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness. |
| void | [**init**](http://docs.google.com/javax/crypto/Cipher.html#init(int,%20java.security.Key,%20java.security.AlgorithmParameters,%20java.security.SecureRandom))(int opmode, [Key](http://docs.google.com/java/security/Key.html) key, [AlgorithmParameters](http://docs.google.com/java/security/AlgorithmParameters.html) params, [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) random)            Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness. |
| void | [**init**](http://docs.google.com/javax/crypto/Cipher.html#init(int,%20java.security.Key,%20java.security.SecureRandom))(int opmode, [Key](http://docs.google.com/java/security/Key.html) key, [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) random)            Initializes this cipher with a key and a source of randomness. |
| [Key](http://docs.google.com/java/security/Key.html) | [**unwrap**](http://docs.google.com/javax/crypto/Cipher.html#unwrap(byte%5B%5D,%20java.lang.String,%20int))(byte[] wrappedKey, [String](http://docs.google.com/java/lang/String.html) wrappedKeyAlgorithm, int wrappedKeyType)            Unwrap a previously wrapped key. |
| byte[] | [**update**](http://docs.google.com/javax/crypto/Cipher.html#update(byte%5B%5D))(byte[] input)            Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part. |
| byte[] | [**update**](http://docs.google.com/javax/crypto/Cipher.html#update(byte%5B%5D,%20int,%20int))(byte[] input, int inputOffset, int inputLen)            Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part. |
| int | [**update**](http://docs.google.com/javax/crypto/Cipher.html#update(byte%5B%5D,%20int,%20int,%20byte%5B%5D))(byte[] input, int inputOffset, int inputLen, byte[] output)            Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part. |
| int | [**update**](http://docs.google.com/javax/crypto/Cipher.html#update(byte%5B%5D,%20int,%20int,%20byte%5B%5D,%20int))(byte[] input, int inputOffset, int inputLen, byte[] output, int outputOffset)            Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part. |
| int | [**update**](http://docs.google.com/javax/crypto/Cipher.html#update(java.nio.ByteBuffer,%20java.nio.ByteBuffer))([ByteBuffer](http://docs.google.com/java/nio/ByteBuffer.html) input, [ByteBuffer](http://docs.google.com/java/nio/ByteBuffer.html) output)            Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part. |
| byte[] | [**wrap**](http://docs.google.com/javax/crypto/Cipher.html#wrap(java.security.Key))([Key](http://docs.google.com/java/security/Key.html) key)            Wrap a key. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [toString](http://docs.google.com/java/lang/Object.html#toString()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Field Detail** |
| --- |

### ENCRYPT\_MODE

public static final int **ENCRYPT\_MODE**

Constant used to initialize cipher to encryption mode.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.crypto.Cipher.ENCRYPT_MODE)

### DECRYPT\_MODE

public static final int **DECRYPT\_MODE**

Constant used to initialize cipher to decryption mode.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.crypto.Cipher.DECRYPT_MODE)

### WRAP\_MODE

public static final int **WRAP\_MODE**

Constant used to initialize cipher to key-wrapping mode.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.crypto.Cipher.WRAP_MODE)

### UNWRAP\_MODE

public static final int **UNWRAP\_MODE**

Constant used to initialize cipher to key-unwrapping mode.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.crypto.Cipher.UNWRAP_MODE)

### PUBLIC\_KEY

public static final int **PUBLIC\_KEY**

Constant used to indicate the to-be-unwrapped key is a "public key".

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.crypto.Cipher.PUBLIC_KEY)

### PRIVATE\_KEY

public static final int **PRIVATE\_KEY**

Constant used to indicate the to-be-unwrapped key is a "private key".

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.crypto.Cipher.PRIVATE_KEY)

### SECRET\_KEY

public static final int **SECRET\_KEY**

Constant used to indicate the to-be-unwrapped key is a "secret key".

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#javax.crypto.Cipher.SECRET_KEY)

| **Constructor Detail** |
| --- |

### Cipher

protected **Cipher**([CipherSpi](http://docs.google.com/javax/crypto/CipherSpi.html) cipherSpi,  
 [Provider](http://docs.google.com/java/security/Provider.html) provider,  
 [String](http://docs.google.com/java/lang/String.html) transformation)

Creates a Cipher object.

**Parameters:**cipherSpi - the delegateprovider - the providertransformation - the transformation

| **Method Detail** |
| --- |

### getInstance

public static final [Cipher](http://docs.google.com/javax/crypto/Cipher.html) **getInstance**([String](http://docs.google.com/java/lang/String.html) transformation)  
 throws [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html),  
 [NoSuchPaddingException](http://docs.google.com/javax/crypto/NoSuchPaddingException.html)

Returns a Cipher object that implements the specified transformation.

This method traverses the list of registered security Providers, starting with the most preferred Provider. A new Cipher object encapsulating the CipherSpi implementation from the first Provider that supports the specified algorithm is returned.

Note that the list of registered providers may be retrieved via the [Security.getProviders()](http://docs.google.com/java/security/Security.html#getProviders()) method.

**Parameters:**transformation - the name of the transformation, e.g., *DES/CBC/PKCS5Padding*. See Appendix A in the  [Java Cryptography Architecture Reference Guide](http://docs.google.com/technotes/guides/security/crypto/CryptoSpec.html#AppA) for information about standard transformation names. **Returns:**a cipher that implements the requested transformation. **Throws:** [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html) - if transformation is null, empty, in an invalid format, or if no Provider supports a CipherSpi implementation for the specified algorithm. [NoSuchPaddingException](http://docs.google.com/javax/crypto/NoSuchPaddingException.html) - if transformation contains a padding scheme that is not available.**See Also:**[Provider](http://docs.google.com/java/security/Provider.html)

### getInstance

public static final [Cipher](http://docs.google.com/javax/crypto/Cipher.html) **getInstance**([String](http://docs.google.com/java/lang/String.html) transformation,  
 [String](http://docs.google.com/java/lang/String.html) provider)  
 throws [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html),  
 [NoSuchProviderException](http://docs.google.com/java/security/NoSuchProviderException.html),  
 [NoSuchPaddingException](http://docs.google.com/javax/crypto/NoSuchPaddingException.html)

Returns a Cipher object that implements the specified transformation.

A new Cipher object encapsulating the CipherSpi implementation from the specified provider is returned. The specified provider must be registered in the security provider list.

Note that the list of registered providers may be retrieved via the [Security.getProviders()](http://docs.google.com/java/security/Security.html#getProviders()) method.

**Parameters:**transformation - the name of the transformation, e.g., *DES/CBC/PKCS5Padding*. See Appendix A in the  [Java Cryptography Architecture Reference Guide](http://docs.google.com/technotes/guides/security/crypto/CryptoSpec.html#AppA) for information about standard transformation names.provider - the name of the provider. **Returns:**a cipher that implements the requested transformation. **Throws:** [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html) - if transformation is null, empty, in an invalid format, or if a CipherSpi implementation for the specified algorithm is not available from the specified provider. [NoSuchProviderException](http://docs.google.com/java/security/NoSuchProviderException.html) - if the specified provider is not registered in the security provider list. [NoSuchPaddingException](http://docs.google.com/javax/crypto/NoSuchPaddingException.html) - if transformation contains a padding scheme that is not available. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the provider is null or empty.**See Also:**[Provider](http://docs.google.com/java/security/Provider.html)

### getInstance

public static final [Cipher](http://docs.google.com/javax/crypto/Cipher.html) **getInstance**([String](http://docs.google.com/java/lang/String.html) transformation,  
 [Provider](http://docs.google.com/java/security/Provider.html) provider)  
 throws [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html),  
 [NoSuchPaddingException](http://docs.google.com/javax/crypto/NoSuchPaddingException.html)

Returns a Cipher object that implements the specified transformation.

A new Cipher object encapsulating the CipherSpi implementation from the specified Provider object is returned. Note that the specified Provider object does not have to be registered in the provider list.

**Parameters:**transformation - the name of the transformation, e.g., *DES/CBC/PKCS5Padding*. See Appendix A in the  [Java Cryptography Architecture Reference Guide](http://docs.google.com/technotes/guides/security/crypto/CryptoSpec.html#AppA) for information about standard transformation names.provider - the provider. **Returns:**a cipher that implements the requested transformation. **Throws:** [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html) - if transformation is null, empty, in an invalid format, or if a CipherSpi implementation for the specified algorithm is not available from the specified Provider object. [NoSuchPaddingException](http://docs.google.com/javax/crypto/NoSuchPaddingException.html) - if transformation contains a padding scheme that is not available. [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the provider is null.**See Also:**[Provider](http://docs.google.com/java/security/Provider.html)

### getProvider

public final [Provider](http://docs.google.com/java/security/Provider.html) **getProvider**()

Returns the provider of this Cipher object.

**Returns:**the provider of this Cipher object

### getAlgorithm

public final [String](http://docs.google.com/java/lang/String.html) **getAlgorithm**()

Returns the algorithm name of this Cipher object.

This is the same name that was specified in one of the getInstance calls that created this Cipher object..

**Returns:**the algorithm name of this Cipher object.

### getBlockSize

public final int **getBlockSize**()

Returns the block size (in bytes).

**Returns:**the block size (in bytes), or 0 if the underlying algorithm is not a block cipher

### getOutputSize

public final int **getOutputSize**(int inputLen)

Returns the length in bytes that an output buffer would need to be in order to hold the result of the next update or doFinal operation, given the input length inputLen (in bytes).

This call takes into account any unprocessed (buffered) data from a previous update call, and padding.

The actual output length of the next update or doFinal call may be smaller than the length returned by this method.

**Parameters:**inputLen - the input length (in bytes) **Returns:**the required output buffer size (in bytes) **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not yet been initialized)

### getIV

public final byte[] **getIV**()

Returns the initialization vector (IV) in a new buffer.

This is useful in the case where a random IV was created, or in the context of password-based encryption or decryption, where the IV is derived from a user-supplied password.

**Returns:**the initialization vector in a new buffer, or null if the underlying algorithm does not use an IV, or if the IV has not yet been set.

### getParameters

public final [AlgorithmParameters](http://docs.google.com/java/security/AlgorithmParameters.html) **getParameters**()

Returns the parameters used with this cipher.

The returned parameters may be the same that were used to initialize this cipher, or may contain a combination of default and random parameter values used by the underlying cipher implementation if this cipher requires algorithm parameters but was not initialized with any.

**Returns:**the parameters used with this cipher, or null if this cipher does not use any parameters.

### getExemptionMechanism

public final [ExemptionMechanism](http://docs.google.com/javax/crypto/ExemptionMechanism.html) **getExemptionMechanism**()

Returns the exemption mechanism object used with this cipher.

**Returns:**the exemption mechanism object used with this cipher, or null if this cipher does not use any exemption mechanism.

### init

public final void **init**(int opmode,  
 [Key](http://docs.google.com/java/security/Key.html) key)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html)

Initializes this cipher with a key.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode.

If this cipher requires any algorithm parameters that cannot be derived from the given key, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an InvalidKeyException if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using [getParameters](http://docs.google.com/javax/crypto/Cipher.html#getParameters()) or [getIV](http://docs.google.com/javax/crypto/Cipher.html#getIV()) (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them using the [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) implementation of the highest-priority installed provider as the source of randomness. (If none of the installed providers supply an implementation of SecureRandom, a system-provided source of randomness will be used.)

Note that when a Cipher object is initialized, it loses all previously-acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

**Parameters:**opmode - the operation mode of this cipher (this is one of the following: ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE or UNWRAP\_MODE)key - the key **Throws:** [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if the given key is inappropriate for initializing this cipher, or if this cipher is being initialized for decryption and requires algorithm parameters that cannot be determined from the given key, or if the given key has a keysize that exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files).

### init

public final void **init**(int opmode,  
 [Key](http://docs.google.com/java/security/Key.html) key,  
 [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) random)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html)

Initializes this cipher with a key and a source of randomness.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode.

If this cipher requires any algorithm parameters that cannot be derived from the given key, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an InvalidKeyException if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using [getParameters](http://docs.google.com/javax/crypto/Cipher.html#getParameters()) or [getIV](http://docs.google.com/javax/crypto/Cipher.html#getIV()) (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them from random.

Note that when a Cipher object is initialized, it loses all previously-acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

**Parameters:**opmode - the operation mode of this cipher (this is one of the following: ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE or UNWRAP\_MODE)key - the encryption keyrandom - the source of randomness **Throws:** [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if the given key is inappropriate for initializing this cipher, or if this cipher is being initialized for decryption and requires algorithm parameters that cannot be determined from the given key, or if the given key has a keysize that exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files).

### init

public final void **init**(int opmode,  
 [Key](http://docs.google.com/java/security/Key.html) key,  
 [AlgorithmParameterSpec](http://docs.google.com/java/security/spec/AlgorithmParameterSpec.html) params)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html),  
 [InvalidAlgorithmParameterException](http://docs.google.com/java/security/InvalidAlgorithmParameterException.html)

Initializes this cipher with a key and a set of algorithm parameters.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode.

If this cipher requires any algorithm parameters and params is null, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an InvalidAlgorithmParameterException if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using [getParameters](http://docs.google.com/javax/crypto/Cipher.html#getParameters()) or [getIV](http://docs.google.com/javax/crypto/Cipher.html#getIV()) (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them using the [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) implementation of the highest-priority installed provider as the source of randomness. (If none of the installed providers supply an implementation of SecureRandom, a system-provided source of randomness will be used.)

Note that when a Cipher object is initialized, it loses all previously-acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

**Parameters:**opmode - the operation mode of this cipher (this is one of the following: ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE or UNWRAP\_MODE)key - the encryption keyparams - the algorithm parameters **Throws:** [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if the given key is inappropriate for initializing this cipher, or its keysize exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files). [InvalidAlgorithmParameterException](http://docs.google.com/java/security/InvalidAlgorithmParameterException.html) - if the given algorithm parameters are inappropriate for this cipher, or this cipher is being initialized for decryption and requires algorithm parameters and params is null, or the given algorithm parameters imply a cryptographic strength that would exceed the legal limits (as determined from the configured jurisdiction policy files).

### init

public final void **init**(int opmode,  
 [Key](http://docs.google.com/java/security/Key.html) key,  
 [AlgorithmParameterSpec](http://docs.google.com/java/security/spec/AlgorithmParameterSpec.html) params,  
 [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) random)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html),  
 [InvalidAlgorithmParameterException](http://docs.google.com/java/security/InvalidAlgorithmParameterException.html)

Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode.

If this cipher requires any algorithm parameters and params is null, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an InvalidAlgorithmParameterException if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using [getParameters](http://docs.google.com/javax/crypto/Cipher.html#getParameters()) or [getIV](http://docs.google.com/javax/crypto/Cipher.html#getIV()) (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them from random.

Note that when a Cipher object is initialized, it loses all previously-acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

**Parameters:**opmode - the operation mode of this cipher (this is one of the following: ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE or UNWRAP\_MODE)key - the encryption keyparams - the algorithm parametersrandom - the source of randomness **Throws:** [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if the given key is inappropriate for initializing this cipher, or its keysize exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files). [InvalidAlgorithmParameterException](http://docs.google.com/java/security/InvalidAlgorithmParameterException.html) - if the given algorithm parameters are inappropriate for this cipher, or this cipher is being initialized for decryption and requires algorithm parameters and params is null, or the given algorithm parameters imply a cryptographic strength that would exceed the legal limits (as determined from the configured jurisdiction policy files).

### init

public final void **init**(int opmode,  
 [Key](http://docs.google.com/java/security/Key.html) key,  
 [AlgorithmParameters](http://docs.google.com/java/security/AlgorithmParameters.html) params)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html),  
 [InvalidAlgorithmParameterException](http://docs.google.com/java/security/InvalidAlgorithmParameterException.html)

Initializes this cipher with a key and a set of algorithm parameters.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode.

If this cipher requires any algorithm parameters and params is null, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an InvalidAlgorithmParameterException if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using [getParameters](http://docs.google.com/javax/crypto/Cipher.html#getParameters()) or [getIV](http://docs.google.com/javax/crypto/Cipher.html#getIV()) (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them using the [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) implementation of the highest-priority installed provider as the source of randomness. (If none of the installed providers supply an implementation of SecureRandom, a system-provided source of randomness will be used.)

Note that when a Cipher object is initialized, it loses all previously-acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

**Parameters:**opmode - the operation mode of this cipher (this is one of the following: ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE or UNWRAP\_MODE)key - the encryption keyparams - the algorithm parameters **Throws:** [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if the given key is inappropriate for initializing this cipher, or its keysize exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files). [InvalidAlgorithmParameterException](http://docs.google.com/java/security/InvalidAlgorithmParameterException.html) - if the given algorithm parameters are inappropriate for this cipher, or this cipher is being initialized for decryption and requires algorithm parameters and params is null, or the given algorithm parameters imply a cryptographic strength that would exceed the legal limits (as determined from the configured jurisdiction policy files).

### init

public final void **init**(int opmode,  
 [Key](http://docs.google.com/java/security/Key.html) key,  
 [AlgorithmParameters](http://docs.google.com/java/security/AlgorithmParameters.html) params,  
 [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) random)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html),  
 [InvalidAlgorithmParameterException](http://docs.google.com/java/security/InvalidAlgorithmParameterException.html)

Initializes this cipher with a key, a set of algorithm parameters, and a source of randomness.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode.

If this cipher requires any algorithm parameters and params is null, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an InvalidAlgorithmParameterException if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using [getParameters](http://docs.google.com/javax/crypto/Cipher.html#getParameters()) or [getIV](http://docs.google.com/javax/crypto/Cipher.html#getIV()) (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them from random.

Note that when a Cipher object is initialized, it loses all previously-acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

**Parameters:**opmode - the operation mode of this cipher (this is one of the following: ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE or UNWRAP\_MODE)key - the encryption keyparams - the algorithm parametersrandom - the source of randomness **Throws:** [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if the given key is inappropriate for initializing this cipher, or its keysize exceeds the maximum allowable keysize (as determined from the configured jurisdiction policy files). [InvalidAlgorithmParameterException](http://docs.google.com/java/security/InvalidAlgorithmParameterException.html) - if the given algorithm parameters are inappropriate for this cipher, or this cipher is being initialized for decryption and requires algorithm parameters and params is null, or the given algorithm parameters imply a cryptographic strength that would exceed the legal limits (as determined from the configured jurisdiction policy files).

### init

public final void **init**(int opmode,  
 [Certificate](http://docs.google.com/java/security/cert/Certificate.html) certificate)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html)

Initializes this cipher with the public key from the given certificate.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode.

If the certificate is of type X.509 and has a *key usage* extension field marked as critical, and the value of the *key usage* extension field implies that the public key in the certificate and its corresponding private key are not supposed to be used for the operation represented by the value of opmode, an InvalidKeyException is thrown.

If this cipher requires any algorithm parameters that cannot be derived from the public key in the given certificate, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or ramdom values) if it is being initialized for encryption or key wrapping, and raise an InvalidKeyException if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using [getParameters](http://docs.google.com/javax/crypto/Cipher.html#getParameters()) or [getIV](http://docs.google.com/javax/crypto/Cipher.html#getIV()) (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them using the SecureRandom implementation of the highest-priority installed provider as the source of randomness. (If none of the installed providers supply an implementation of SecureRandom, a system-provided source of randomness will be used.)

Note that when a Cipher object is initialized, it loses all previously-acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

**Parameters:**opmode - the operation mode of this cipher (this is one of the following: ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE or UNWRAP\_MODE)certificate - the certificate **Throws:** [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if the public key in the given certificate is inappropriate for initializing this cipher, or this cipher is being initialized for decryption or unwrapping keys and requires algorithm parameters that cannot be determined from the public key in the given certificate, or the keysize of the public key in the given certificate has a keysize that exceeds the maximum allowable keysize (as determined by the configured jurisdiction policy files).

### init

public final void **init**(int opmode,  
 [Certificate](http://docs.google.com/java/security/cert/Certificate.html) certificate,  
 [SecureRandom](http://docs.google.com/java/security/SecureRandom.html) random)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html)

Initializes this cipher with the public key from the given certificate and a source of randomness.

The cipher is initialized for one of the following four operations: encryption, decryption, key wrapping or key unwrapping, depending on the value of opmode.

If the certificate is of type X.509 and has a *key usage* extension field marked as critical, and the value of the *key usage* extension field implies that the public key in the certificate and its corresponding private key are not supposed to be used for the operation represented by the value of opmode, an InvalidKeyException is thrown.

If this cipher requires any algorithm parameters that cannot be derived from the public key in the given certificate, the underlying cipher implementation is supposed to generate the required parameters itself (using provider-specific default or random values) if it is being initialized for encryption or key wrapping, and raise an InvalidKeyException if it is being initialized for decryption or key unwrapping. The generated parameters can be retrieved using [getParameters](http://docs.google.com/javax/crypto/Cipher.html#getParameters()) or [getIV](http://docs.google.com/javax/crypto/Cipher.html#getIV()) (if the parameter is an IV).

If this cipher (including its underlying feedback or padding scheme) requires any random bytes (e.g., for parameter generation), it will get them from random.

Note that when a Cipher object is initialized, it loses all previously-acquired state. In other words, initializing a Cipher is equivalent to creating a new instance of that Cipher and initializing it.

**Parameters:**opmode - the operation mode of this cipher (this is one of the following: ENCRYPT\_MODE, DECRYPT\_MODE, WRAP\_MODE or UNWRAP\_MODE)certificate - the certificaterandom - the source of randomness **Throws:** [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if the public key in the given certificate is inappropriate for initializing this cipher, or this cipher is being initialized for decryption or unwrapping keys and requires algorithm parameters that cannot be determined from the public key in the given certificate, or the keysize of the public key in the given certificate has a keysize that exceeds the maximum allowable keysize (as determined by the configured jurisdiction policy files).

### update

public final byte[] **update**(byte[] input)

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

The bytes in the input buffer are processed, and the result is stored in a new buffer.

If input has a length of zero, this method returns null.

**Parameters:**input - the input buffer **Returns:**the new buffer with the result, or null if the underlying cipher is a block cipher and the input data is too short to result in a new block. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized)

### update

public final byte[] **update**(byte[] input,  
 int inputOffset,  
 int inputLen)

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

The first inputLen bytes in the input buffer, starting at inputOffset inclusive, are processed, and the result is stored in a new buffer.

If inputLen is zero, this method returns null.

**Parameters:**input - the input bufferinputOffset - the offset in input where the input startsinputLen - the input length **Returns:**the new buffer with the result, or null if the underlying cipher is a block cipher and the input data is too short to result in a new block. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized)

### update

public final int **update**(byte[] input,  
 int inputOffset,  
 int inputLen,  
 byte[] output)  
 throws [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html)

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

The first inputLen bytes in the input buffer, starting at inputOffset inclusive, are processed, and the result is stored in the output buffer.

If the output buffer is too small to hold the result, a ShortBufferException is thrown. In this case, repeat this call with a larger output buffer. Use [getOutputSize](http://docs.google.com/javax/crypto/Cipher.html#getOutputSize(int)) to determine how big the output buffer should be.

If inputLen is zero, this method returns a length of zero.

Note: this method should be copy-safe, which means the input and output buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

**Parameters:**input - the input bufferinputOffset - the offset in input where the input startsinputLen - the input lengthoutput - the buffer for the result **Returns:**the number of bytes stored in output **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html) - if the given output buffer is too small to hold the result

### update

public final int **update**(byte[] input,  
 int inputOffset,  
 int inputLen,  
 byte[] output,  
 int outputOffset)  
 throws [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html)

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

The first inputLen bytes in the input buffer, starting at inputOffset inclusive, are processed, and the result is stored in the output buffer, starting at outputOffset inclusive.

If the output buffer is too small to hold the result, a ShortBufferException is thrown. In this case, repeat this call with a larger output buffer. Use [getOutputSize](http://docs.google.com/javax/crypto/Cipher.html#getOutputSize(int)) to determine how big the output buffer should be.

If inputLen is zero, this method returns a length of zero.

Note: this method should be copy-safe, which means the input and output buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

**Parameters:**input - the input bufferinputOffset - the offset in input where the input startsinputLen - the input lengthoutput - the buffer for the resultoutputOffset - the offset in output where the result is stored **Returns:**the number of bytes stored in output **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html) - if the given output buffer is too small to hold the result

### update

public final int **update**([ByteBuffer](http://docs.google.com/java/nio/ByteBuffer.html) input,  
 [ByteBuffer](http://docs.google.com/java/nio/ByteBuffer.html) output)  
 throws [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html)

Continues a multiple-part encryption or decryption operation (depending on how this cipher was initialized), processing another data part.

All input.remaining() bytes starting at input.position() are processed. The result is stored in the output buffer. Upon return, the input buffer's position will be equal to its limit; its limit will not have changed. The output buffer's position will have advanced by n, where n is the value returned by this method; the output buffer's limit will not have changed.

If output.remaining() bytes are insufficient to hold the result, a ShortBufferException is thrown. In this case, repeat this call with a larger output buffer. Use [getOutputSize](http://docs.google.com/javax/crypto/Cipher.html#getOutputSize(int)) to determine how big the output buffer should be.

Note: this method should be copy-safe, which means the input and output buffers can reference the same block of memory and no unprocessed input data is overwritten when the result is copied into the output buffer.

**Parameters:**input - the input ByteBufferoutput - the output ByteByffer **Returns:**the number of bytes stored in output **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if input and output are the same object [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - if the output buffer is read-only [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html) - if there is insufficient space in the output buffer**Since:** 1.5

### doFinal

public final byte[] **doFinal**()  
 throws [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html),  
 [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html)

Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized.

Input data that may have been buffered during a previous update operation is processed, with padding (if requested) being applied. The result is stored in a new buffer.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to init. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to init) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

**Returns:**the new buffer with the result **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html) - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided. [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html) - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

### doFinal

public final int **doFinal**(byte[] output,  
 int outputOffset)  
 throws [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html),  
 [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html),  
 [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html)

Finishes a multiple-part encryption or decryption operation, depending on how this cipher was initialized.

Input data that may have been buffered during a previous update operation is processed, with padding (if requested) being applied. The result is stored in the output buffer, starting at outputOffset inclusive.

If the output buffer is too small to hold the result, a ShortBufferException is thrown. In this case, repeat this call with a larger output buffer. Use [getOutputSize](http://docs.google.com/javax/crypto/Cipher.html#getOutputSize(int)) to determine how big the output buffer should be.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to init. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to init) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

**Parameters:**output - the buffer for the resultoutputOffset - the offset in output where the result is stored **Returns:**the number of bytes stored in output **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html) - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided. [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html) - if the given output buffer is too small to hold the result [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html) - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

### doFinal

public final byte[] **doFinal**(byte[] input)  
 throws [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html),  
 [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html)

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The bytes in the input buffer, and any input bytes that may have been buffered during a previous update operation, are processed, with padding (if requested) being applied. The result is stored in a new buffer.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to init. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to init) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

**Parameters:**input - the input buffer **Returns:**the new buffer with the result **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html) - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided. [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html) - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

### doFinal

public final byte[] **doFinal**(byte[] input,  
 int inputOffset,  
 int inputLen)  
 throws [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html),  
 [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html)

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The first inputLen bytes in the input buffer, starting at inputOffset inclusive, and any input bytes that may have been buffered during a previous update operation, are processed, with padding (if requested) being applied. The result is stored in a new buffer.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to init. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to init) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

**Parameters:**input - the input bufferinputOffset - the offset in input where the input startsinputLen - the input length **Returns:**the new buffer with the result **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html) - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided. [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html) - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

### doFinal

public final int **doFinal**(byte[] input,  
 int inputOffset,  
 int inputLen,  
 byte[] output)  
 throws [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html),  
 [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html),  
 [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html)

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The first inputLen bytes in the input buffer, starting at inputOffset inclusive, and any input bytes that may have been buffered during a previous update operation, are processed, with padding (if requested) being applied. The result is stored in the output buffer.

If the output buffer is too small to hold the result, a ShortBufferException is thrown. In this case, repeat this call with a larger output buffer. Use [getOutputSize](http://docs.google.com/javax/crypto/Cipher.html#getOutputSize(int)) to determine how big the output buffer should be.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to init. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to init) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Note: this method should be copy-safe, which means the input and output buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

**Parameters:**input - the input bufferinputOffset - the offset in input where the input startsinputLen - the input lengthoutput - the buffer for the result **Returns:**the number of bytes stored in output **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html) - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided. [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html) - if the given output buffer is too small to hold the result [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html) - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

### doFinal

public final int **doFinal**(byte[] input,  
 int inputOffset,  
 int inputLen,  
 byte[] output,  
 int outputOffset)  
 throws [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html),  
 [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html),  
 [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html)

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

The first inputLen bytes in the input buffer, starting at inputOffset inclusive, and any input bytes that may have been buffered during a previous update operation, are processed, with padding (if requested) being applied. The result is stored in the output buffer, starting at outputOffset inclusive.

If the output buffer is too small to hold the result, a ShortBufferException is thrown. In this case, repeat this call with a larger output buffer. Use [getOutputSize](http://docs.google.com/javax/crypto/Cipher.html#getOutputSize(int)) to determine how big the output buffer should be.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to init. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to init) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Note: this method should be copy-safe, which means the input and output buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

**Parameters:**input - the input bufferinputOffset - the offset in input where the input startsinputLen - the input lengthoutput - the buffer for the resultoutputOffset - the offset in output where the result is stored **Returns:**the number of bytes stored in output **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html) - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided. [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html) - if the given output buffer is too small to hold the result [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html) - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes

### doFinal

public final int **doFinal**([ByteBuffer](http://docs.google.com/java/nio/ByteBuffer.html) input,  
 [ByteBuffer](http://docs.google.com/java/nio/ByteBuffer.html) output)  
 throws [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html),  
 [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html),  
 [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html)

Encrypts or decrypts data in a single-part operation, or finishes a multiple-part operation. The data is encrypted or decrypted, depending on how this cipher was initialized.

All input.remaining() bytes starting at input.position() are processed. The result is stored in the output buffer. Upon return, the input buffer's position will be equal to its limit; its limit will not have changed. The output buffer's position will have advanced by n, where n is the value returned by this method; the output buffer's limit will not have changed.

If output.remaining() bytes are insufficient to hold the result, a ShortBufferException is thrown. In this case, repeat this call with a larger output buffer. Use [getOutputSize](http://docs.google.com/javax/crypto/Cipher.html#getOutputSize(int)) to determine how big the output buffer should be.

Upon finishing, this method resets this cipher object to the state it was in when previously initialized via a call to init. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to init) more data.

Note: if any exception is thrown, this cipher object may need to be reset before it can be used again.

Note: this method should be copy-safe, which means the input and output buffers can reference the same byte array and no unprocessed input data is overwritten when the result is copied into the output buffer.

**Parameters:**input - the input ByteBufferoutput - the output ByteBuffer **Returns:**the number of bytes stored in output **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized) [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if input and output are the same object [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - if the output buffer is read-only [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html) - if this cipher is a block cipher, no padding has been requested (only in encryption mode), and the total input length of the data processed by this cipher is not a multiple of block size; or if this encryption algorithm is unable to process the input data provided. [ShortBufferException](http://docs.google.com/javax/crypto/ShortBufferException.html) - if there is insufficient space in the output buffer [BadPaddingException](http://docs.google.com/javax/crypto/BadPaddingException.html) - if this cipher is in decryption mode, and (un)padding has been requested, but the decrypted data is not bounded by the appropriate padding bytes**Since:** 1.5

### wrap

public final byte[] **wrap**([Key](http://docs.google.com/java/security/Key.html) key)  
 throws [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html),  
 [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html)

Wrap a key.

**Parameters:**key - the key to be wrapped. **Returns:**the wrapped key. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized). [IllegalBlockSizeException](http://docs.google.com/javax/crypto/IllegalBlockSizeException.html) - if this cipher is a block cipher, no padding has been requested, and the length of the encoding of the key to be wrapped is not a multiple of the block size. [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if it is impossible or unsafe to wrap the key with this cipher (e.g., a hardware protected key is being passed to a software-only cipher).

### unwrap

public final [Key](http://docs.google.com/java/security/Key.html) **unwrap**(byte[] wrappedKey,  
 [String](http://docs.google.com/java/lang/String.html) wrappedKeyAlgorithm,  
 int wrappedKeyType)  
 throws [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html),  
 [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html)

Unwrap a previously wrapped key.

**Parameters:**wrappedKey - the key to be unwrapped.wrappedKeyAlgorithm - the algorithm associated with the wrapped key.wrappedKeyType - the type of the wrapped key. This must be one of SECRET\_KEY, PRIVATE\_KEY, or PUBLIC\_KEY. **Returns:**the unwrapped key. **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this cipher is in a wrong state (e.g., has not been initialized). [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html) - if no installed providers can create keys of type wrappedKeyType for the wrappedKeyAlgorithm. [InvalidKeyException](http://docs.google.com/java/security/InvalidKeyException.html) - if wrappedKey does not represent a wrapped key of type wrappedKeyType for the wrappedKeyAlgorithm.

### getMaxAllowedKeyLength

public static final int **getMaxAllowedKeyLength**([String](http://docs.google.com/java/lang/String.html) transformation)  
 throws [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html)

Returns the maximum key length for the specified transformation according to the installed JCE jurisdiction policy files. If JCE unlimited strength jurisdiction policy files are installed, Integer.MAX\_VALUE will be returned. For more information on default key size in JCE jurisdiction policy files, please see Appendix E in the  [Java Cryptography Architecture Reference Guide](http://docs.google.com/technotes/guides/security/crypto/CryptoSpec.html#AppE).

**Parameters:**transformation - the cipher transformation. **Returns:**the maximum key length in bits or Integer.MAX\_VALUE. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if transformation is null. [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html) - if transformation is not a valid transformation, i.e. in the form of "algorithm" or "algorithm/mode/padding".**Since:** 1.5

### getMaxAllowedParameterSpec

public static final [AlgorithmParameterSpec](http://docs.google.com/java/security/spec/AlgorithmParameterSpec.html) **getMaxAllowedParameterSpec**([String](http://docs.google.com/java/lang/String.html) transformation)  
 throws [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html)

Returns an AlgorithmParameterSpec object which contains the maximum cipher parameter value according to the jurisdiction policy file. If JCE unlimited strength jurisdiction policy files are installed or there is no maximum limit on the parameters for the specified transformation in the policy file, null will be returned.

**Parameters:**transformation - the cipher transformation. **Returns:**an AlgorithmParameterSpec which holds the maximum value or null. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if transformation is null. [NoSuchAlgorithmException](http://docs.google.com/java/security/NoSuchAlgorithmException.html) - if transformation is not a valid transformation, i.e. in the form of "algorithm" or "algorithm/mode/padding".**Since:** 1.5

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Cipher.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/crypto/BadPaddingException.html)   [**NEXT CLASS**](http://docs.google.com/javax/crypto/CipherInputStream.html) | [**FRAMES**](http://docs.google.com/index.html?javax/crypto/Cipher.html)    [**NO FRAMES**](http://docs.google.com/Cipher.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#1ksv4uv) | [METHOD](#2jxsxqh) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).