Java SE Documentation

Java ™ Cryptography Architecture
Standard Algorithm Name Documentation

for Java™ Platform Standard Edition 6

Table of Contents

Note: The Sun Provider Documentation contains specific provider and algorithm information.

Standard Names

The JDK Security API requires and uses a set of standard names for algorithms, certificate and keystore types. This specification establishes the following names as standard names.

In some cases naming conventions are given for forming names that are not explicitly listed, to facilitate name consistency across provider implementations. Items in angle brackets (such as **<digest>** and **<encryption>**) are placeholders to be replaced by a specific message digest, encryption algorithm, or other name.

Note: Standard names are not case-sensitive.

This document includes corresponding lists of standard names relevant to the following security subareas:

- Java PKI Programmer's Guide
- JSSE Reference Guide
- For standard name specifications, See Algorithms.
- Cryptography Architecture
- Single Sign-on Using Kerberos in Java
- The Java SASL API Programming and Deployment Guide
- The XML Digital Signature API Specification

AlgorithmParameterGenerator Algorithms

The algorithm names in this section can be specified when generating an instance of **AlgorithmParameterGenerator**.

Alg. Name	Description
DiffieHellman	Parameters for use with the Diffie-Hellman algorithm.
DSA	Parameters for use with the Digital Signature Algorithm.

AlgorithmParameters Algorithms

The algorithm names in this section can be specified when generating an instance of **AlgorithmParameters**.

Alg. Name	Description
AES	Parameters for use with the AES algorithm.
Blowfish	Parameters for use with the Blowfish algorithm.
DES	Parameters for use with the DES algorithm.
DESede	Parameters for use with the DESede algorithm.
DiffieHellman	Parameters for use with the DiffieHellman algorithm.
DSA	Parameters for use with the Digital Signature Algorithm.
OAEP	Parameters for use with the OAEP algorithm.
PBEWith<digest>And<encryption>	Parameters for use with the PBEWith<digest>And<encryption> algorithm. Examples: **PBEWithMD5AndDES**, and **PBEwithSHA1AndDESede**.
PBE	Parameters for use with the PBE algorithm. *This name should not be used, in preference to the more specific PBE-algorithm names above.*
RC2	Parameters for use with the RC2 algorithm.

CertificateFactory Types

The types in this section can be specified when generating an instance of **CertificateFactory**.

Type	Description
X.509	The certificate type defined in X.509, also available via RFC 3280

CertPathBuilder Algorithms

The algorithms in this section can be specified when generating an instance of **CertPathBuilder**.

Alg. Name	Description
PKIX	The PKIX certification path validation algorithm as defined in the ValidationAlgorithm service attribute. The output of **CertPathBuilder** instances implementing this algorithm is a certification path validated against the PKIX validation algorithm.

CertPath Encodings

The following encodings may be passed to the **getEncoded** method of **CertPath** or the **generateCertPath(InputStream inStream, String encoding)** method of **CertificateFactory**.

Encoding	Description
PKCS7	A PKCS#7 SignedData object, with the only significant field being certificates. In particular, the signature and the contents are ignored. If no certificates are present, a zero-length **CertPath** is assumed. Warning: PKCS#7 does not maintain the order of certificates in a certification path. This means that if a **CertPath** is converted to PKCS#7 encoded bytes and then converted back, the order of the certificates may change, potentially rendering the **CertPath** invalid. Users should be aware of this behavior. See RSA Security for details on PKCS7.
PkiPath	an ASN.1 DER encoded sequence of certificates, defined as follows: **PkiPath ::= SEQUENCE OF Certificate** Within the sequence, the order of certificates is such that the subject of the first certificate is the issuer of the second certificate, etc. Each certificate in **PkiPath** shall be unique. No certificate may appear more than once in a value of **Certificate** in **PkiPath**. The **PkiPath** format is defined in defect report 279 against X.509 (2000) and is incorporated into Technical Corrigendum 1 (DTC 2) for the ITU-T Recommendation X.509 (2000). See the ITU website for details.

CertPathValidator Algorithms

The algorithms in this section can be specified when generating an instance of **CertPathValidator**.

Alg. Name	Description
PKIX	The PKIX certification path validation algorithm as defined in the ValidationAlgorithm service attribute.

CertStore Types

The types in this section can be specified when generating an instance of **CertStore**.

Type	Description
Collection	A **CertStore** implementation that retrieves certificates and CRLs from a **Collection**. This type of **CertStore** is particularly useful in applications where certificates or CRLs are received in a bag or some sort of attachment, such as with a signed email message or in an SSL negotiation.
LDAP	A **CertStore** implementation that fetches certificates and CRLs from an LDAP directory using the schema defined in the LDAPSchema service attribute.

Cipher (Encryption) Algorithms

Cipher Algorithm Names

The following names can be specified as the *algorithm* component in a transformation when requesting an instance of **Cipher**.

Alg. Name	Description
AES	Advanced Encryption Standard as specified by NIST in FIPS 197. Also known as the Rijndael algorithm by Joan Daemen and Vincent Rijmen, AES is a 128-bit block cipher supporting keys of 128, 192, and 256 bits.
AESWrap	The AES key wrapping algorithm as described in RFC 3394.
ARCFOUR	A stream cipher believed to be fully interoperable with the RC4 cipher developed by Ron Rivest. For more information, see K. Kaukonen and R. Thayer, "A Stream Cipher Encryption Algorithm 'Arcfour'", Internet Draft (expired), draft-kaukonen-cipher-arcfour-03.txt.
Blowfish	The Blowfish block cipher designed by Bruce Schneier.
DES	The Digital Encryption Standard as described in FIPS PUB 46-3.
DESede	Triple DES Encryption (also known as DES-EDE, 3DES, or Triple-DES). Data is encrypted using the DES algorithm three separate times. It is first encrypted using the first subkey, then decrypted with the second subkey, and encrypted with the third subkey.
DESedeWrap	The DESede key wrapping algorithm as described in RFC 3217 .
ECIES	Elliptic Curve Integrated Encryption Scheme
PBEWith<digest>And<encryption> PBEWith<prf>And<encryption>	The password-based encryption algorithm found in (PKCS5), using the specified message digest (<digest>) or pseudo-random function (<prf>) and encryption algorithm (<encryption>). Examples: • PBEWithMD5AndDES: The password-based encryption algorithm as defined in RSA Laboratories, "PKCS5: Password-Based Encryption Standard," version 1.5, Nov 1993. Note that this algorithm implies *CBC* as the cipher mode and *PKCS5Padding* as the padding scheme and cannot be used with any other cipher modes or padding schemes. • PBEwithSHA1AndDESede: The password-based encryption algorithm as defined in RSA Laboratories, "PKCS5: Password-Based Cryptography Standard," version 2.0, March 1999.
RC2	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc.
RC4	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc. (See note above for ARCFOUR.)
RC5	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc.
RSA	The RSA encryption algorithm as defined in PKCS1

Cipher Algorithm Modes

The following names can be specified as the *mode* component in a transformation when requesting an instance of **Cipher**.

Alg. Name	Description
NONE	No mode.
CBC	Cipher Block Chaining Mode, as defined in FIPS PUB 81.
CFB, CFBx	Cipher Feedback Mode, as defined in FIPS PUB 81. 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.
CTR	A simplification of OFB, Counter mode updates the input block as a counter.
CTS	Cipher Text Stealing, as described in Bruce Schneier's book *Applied Cryptography-Second Edition*, John Wiley and Sons, 1996.
ECB	Electronic Codebook Mode, as defined in FIPS PUB 81.
OFB, OFBx	Output Feedback Mode, as defined in FIPS PUB 81. 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.
PCBC	Propagating Cipher Block Chaining, as defined by Kerberos V4.

Cipher Algorithm Padding

The following names can be specified as the *padding* component in a transformation when requesting an instance of **`Cipher`**.

Alg. Name	Description
NoPadding	No padding.
ISO10126Padding	This padding for block ciphers is described in 5.2 Block Encryption Algorithms in the W3C's "XML Encryption Syntax and Processing" document.
OAEPPadding, OAEPWith<digest>And<mgf>Padding	Optimal Asymmetric Encryption Padding scheme defined in PKCS1, where <digest> should be replaced by the message digest and <mgf> by the mask generation function. Examples: **OAEPWithMD5AndMGF1Padding** and **OAEPWithSHA-512AndMGF1Padding**. If **`OAEPPadding`** is used, **`Cipher`** objects are initialized with a **`javax.crypto.spec.OAEPParameterSpec`** object to suppply values needed for OAEPPadding.
PKCS1Padding	The padding scheme described in PKCS1, used with the RSA algorithm.
PKCS5Padding	The padding scheme described in RSA Laboratories, "PKCS5: Password-Based Encryption Standard," version 1.5, November 1993.

Alg. Name	Description

SSL3Padding	The padding scheme defined in the SSL Protocol Version 3.0, November 18, 1996, section 5.2.3.2 (CBC block cipher):
	
	`block-ciphered struct {`
	`  opaque content[SSLCompressed.length];`
	`  opaque MAC[CipherSpec.hash_size];`
	`  uint8 padding[`
	`  GenericBlockCipher.padding_length];`
	`  uint8 padding_length;`
	`} GenericBlockCipher;`
	
	The size of an instance of a GenericBlockCipher must be a multiple of the block cipher's block length.
	
	The padding length, which is always present, contributes to the padding, which implies that if:
	
	`sizeof(content) + sizeof(MAC) % block_length = 0,`
	
	padding has to be (block_length - 1) bytes long, because of the existence of `padding_length`.
	
	This make the padding scheme similar (but not quite) to PKCS5Padding, where the padding length is encoded in the padding (and ranges from 1 to block_length). With the SSL scheme, the sizeof(padding) is encoded in the always present `padding_length` and therefore ranges from 0 to block_length-1.

Configuration Types

The types in this section can be specified when generating an instance of **javax.security.auth.login.Configuration**.

Type	Description
JavaLoginConfig	The default Configuration implementation from the SUN provider, as described in the ConfigFile class specification. This type accepts `java.security.URIParameter` as a valid `Configuration.Parameter` type. If this parameter is not specified, then the configuration information is loaded from the sources described in the ConfigFile class specification. If this parameter is specified, the configuration information is loaded solely from the specified URI.

Exemption Mechanisms

The following exemption mechanism names can be specified in the permission policy file that accompanies an application considered "exempt" from cryptographic restrictions:

Alg. Name	Description
KeyEscrow	An encryption system with a backup decryption capability that allows authorized persons (users, officers of an organization, and government officials), under certain prescribed conditions, to decrypt ciphertext with the help of information supplied by one or more trusted parties who hold special data recovery keys.
KeyRecovery	A method of obtaining the secret key used to lock encrypted data. One use is as a means of providing fail-safe access to a corporation's own encrypted information in times of disaster.
KeyWeakening	A method in which a part of the key can be escrowed or recovered.

GSSAPI Mechanisms

The following mechanisms can be specified when using GSSAPI. Note that Object Identifiers (OIDs) are specified instead of names to be consistent with the GSSAPI standard.

Mechanism OID	Description
1.2.840.113554.1.2.2	The Kerberos v5 GSS-API mechanism defined in RFC 4121.
1.3.6.1.5.5.2	The Simple and Protected GSS-API Negotiation (SPNEGO) mechanism defined in RFC 4178.

KeyAgreement Algorithms

The following algorithm names can be specified when requesting an instance of **KeyAgreement**.

Alg. Name	Description
DiffieHellman	Diffie-Hellman Key Agreement as defined in PKCS3: Diffie-Hellman Key-Agreement Standard, RSA Laboratories, version 1.4, November 1993.
ECDH	Elliptic Curve Diffie-Hellman as defined in ANSI X9.63 and as described in RFC 3278: "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)."
ECMQV	Elliptic Curve Menezes-Qu-Vanstone as defined in "Elliptic Curve Cryptography" from www.secg.org.

KeyFactory Algorithms

(Except as noted, these classes create keys for which ***Key.getAlgorithm()*** *returns the standard algorithm name.)*

The algorithm names in this section can be specified when generating an instance of **KeyFactory**.

Alg. Name	Description
DiffieHellman	Keys for the Diffie-Hellman KeyAgreement algorithm. Note: **key.getAlgorithm()** will return "DH" instead of "DiffieHellman".
DSA	Keys for the Digital Signature Algorithm.
RSA	Keys for the RSA algorithm (Signature/Cipher).
EC	Keys for the Elliptic Curve algorithm.

KeyGenerator Algorithms

The following algorithm names can be specified when requesting an instance of **KeyGenerator**.

Alg. Name	Description
AES	Key generator for use with the AES algorithm.
ARCFOUR	Key generator for use with the ARCFOUR (RC4) algorithm.
Blowfish	Key generator for use with the Blowfish algorithm.
DES	Key generator for use with the DES algorithm.
DESede	Key generator for use with the DESede (triple-DES) algorithm.
HmacMD5	Key generator for use with the HmacMD5 algorithm.
HmacSHA1 HmacSHA256 HmacSHA384 HmacSHA512	Keys generator for use with the various flavors of the HmacSHA algorithms.
RC2	Key generator for use with the RC2 algorithm.

KeyPairGenerator Algorithms

(Except as noted, these classes create keys for which ***Key.getAlgorithm()*** *returns the standard algorithm name.)*

The algorithm names in this section can be specified when generating an instance of **KeyPairGenerator**.

Alg. Name	Description

DiffieHellman	Generates keypairs for the Diffie-Hellman KeyAgreement algorithm. Note: **key.getAlgorithm()** will return "DH" instead of "DiffieHellman".
DSA	Generates keypairs for the Digital Signature Algorithm.
RSA	Generates keypairs for the RSA algorithm (Signature/Cipher).
EC	Generates keypairs for the Elliptic Curve algorithm.

KeyStore Types

The types in this section can be specified when generating an instance of **KeyStore**.

Type	Description
jceks	The proprietary keystore implementation provided by the "SunJCE" provider.
jks	The proprietary keystore implementation provided by the "SUN" provider.
pkcs12	The transfer syntax for personal identity information as defined in PKCS12.

Mac Algorithms

The following algorithm names can be specified when requesting an instance of **Mac**.

Alg. Name	Description
HmacMD5	The HMAC-MD5 keyed-hashing algorithm as defined in RFC 2104 "HMAC: Keyed-Hashing for Message Authentication" (February 1997).
HmacSHA1 HmacSHA256 HmacSHA384 HmacSHA512	The HmacSHA* algorithms as defined in RFC 2104 "HMAC: Keyed-Hashing for Message Authentication" (February 1997) with **SHA-*** as the message digest algorithm.
PBEWith<mac>	Mac for use with the PKCS5 v 2.0 password-based message authentication standard, where <mac> is a Message Authentication Code algorithm name. Example: **PBEWithHmacSHA1**.

MessageDigest Algorithms

The algorithm names in this section can be specified when generating an instance of **MessageDigest**.

Alg. Name	Description
MD2	The MD2 message digest algorithm as defined in RFC 1319.
MD5	The MD5 message digest algorithm as defined in RFC 1321.
SHA-1 SHA-256 SHA-384 SHA-512	Hash algorithms defined in the FIPS PUB 180-2. SHA-256 is a 256-bit hash function intended to provide 128 bits of security against collision attacks, while SHA-512 is a 512-bit hash function intended to provide 256 bits of security. A 384-bit hash may be obtained by truncating the SHA-512 output.

Policy Types

The types in this section can be specified when generating an instance of **Policy**.

Type	Description
JavaPolicy	The default Policy implementation from the SUN provider, as described in the PolicyFile guide. This type accepts **java.security.URIParameter** as a valid **Policy.Parameter** type. If this parameter is not specified, then the policy information is loaded from the sources described in the Default Policy File Locations section of the PolicyFile guide. If this parameter is specified, the policy information is loaded solely from the specified URI.

SaslClient Mechanisms

The mechanisms in this section can be specified when generating an instance of **SaslClient**.

Mechanism	Description
CRAM-MD5	See RFC 2195. This mechanism supports a hashed username/password authentication scheme.
DIGEST-MD5	See RFC 2831. This mechanism defines how HTTP Digest Authentication can be used as a SASL mechanism.
EXTERNAL	See RFC 2222. This mechanism obtains authentication information from an external channel (such as TLS or IPsec).
GSSAPI	See RFC 2222. This mechanism uses the GSSAPI for obtaining authentication information. It supports Kerberos v5 authentication.
PLAIN	See RFC 2595. This mechanism supports cleartext username/password authentication.

SaslServer Mechanisms

The mechanisms in this section can be specified when generating an instance of **SaslServer**.

Mechanism	Description
CRAM-MD5	See RFC 2195. This mechanism supports a hashed username/password authentication scheme.
DIGEST-MD5	See RFC 2831. This mechanism defines how HTTP Digest Authentication can be used as a SASL mechanism.
GSSAPI	See RFC 2222. This mechanism uses the GSSAPI for obtaining authentication information. It supports Kerberos v5 authentication.

SecretKeyFactory Algorithms

The following algorithm names can be specified when requesting an instance of **SecretKeyFactory**.

Alg. Name	Description
AES	Constructs secret keys for use with the AES algorithm.
ARCFOUR	Constructs secret keys for use with the ARCFOUR algorithm.
DES	Constructs secrets keys for use with the DES algorithm.
DESede	Constructs secrets keys for use with the DESede (Triple-DES) algorithm.
PBEWith<digest>And<encryption> PBEWith<prf>And<encryption>	Secret-key factory for use with PKCS5 password-based encryption, where <digest> is a message digest, <prf> is a pseudo-random function, and <encryption> is an encryption algorithm. Examples: • PBEWithMD5AndDES (PKCS5, v 1.5), • PBEwithSHA1AndDESede (PKCS5, v 2.0), and Note: These all use only the low order 8 bits of each password character.
PBKDF2WithHmacSHA1	Constructs secret keys using the Password-Based Key Derivation Function function found in PKCS5 v2.0.

SecureRandom Number Generation (RNG) Algorithms

The algorithm names in this section can be specified when generating an instance of **SecureRandom**.

Alg. Name	Description
SHA1PRNG	The name of the pseudo-random number generation (PRNG) algorithm supplied by the SUN provider. This algorithm uses SHA-1 as the foundation of the PRNG. It computes the SHA-1 hash over a true-random seed value concatenated with a 64-bit counter which is incremented by 1 for each operation. From the 160-bit SHA-1 output, only 64 bits are used.

Service Attributes

A cryptographic service is always associated with a particular algorithm or type. For example, a digital signature service is always associated with a particular algorithm (e.g., DSA), and a **`CertificateFactory`** service is always associated with a particular certificate type (e.g., X.509).

The attributes in this section are for cryptographic services. The service attributes can be used as filters for selecting providers.

Both the attribute name and value are case insensitive.

Attribute	Description
KeySize	The maximum key size that the provider supports for the cryptographic service.
ImplementedIn	Whether the implementation for the cryptographic service is done by software or hardware. The value of this attribute is "software" or "hardware".
ValidationAlgorithm	The name of the specification that defines the certification path validation algorithm that an implementation of **`CertPathBuilder`** or **`CertPathValidator`** supports. RFCs should be specified as "RFC#" (ex: "RFC3280") and Internet Drafts as the name of the draft (ex: "draft-ietf-pkix-rfc2560bis-01.txt"). Values for this attribute that are specified as selection criteria to the **`Security.getProviders`** method will be compared using the **`String.equalsIgnoreCase`** method. All PKIX implementations of **`CertPathBuilder`** and **`CertPathValidator`** should provide a value for this attribute.
LDAPSchema	The name of the specification that defines the LDAP schema that an implementation of an LDAP **`CertStore`** uses to retrieve certificates and CRLs. The format and semantics of this attribute is the same as described for the ValidationAlgorithm attribute. All LDAP implementations of **`CertStore`** should provide a value for this attribute.

For example:

```
map.put("KeyPairGenerator.DSA",
        "sun.security.provider.DSAKeyPairGenerator");
    map.put("KeyPairGenerator.DSA KeySize", "1024");
    map.put("KeyPairGenerator.DSA ImplementedIn", "Software");
```

`Signature` Algorithms

The algorithm names in this section can be specified when generating an instance of **`Signature`**.

Alg. Name	Description
NONEwithRSA	The RSA signature algorithm which does not use a digesting algorithm (e.g. MD5/SHA1) before performing the RSA operation. For more information about the RSA Signature algorithms, please see PKCS1.
MD2withRSA MD5withRSA	The MD2/MD5 with RSA Encryption signature algorithm which uses the MD2/MD5 digest algorithm and RSA to create and verify RSA digital signatures as defined in PKCS1.
SHA1withRSA SHA256withRSA SHA384withRSA SHA512withRSA	The signature algorithm with SHA-* and the RSA encryption algorithm as defined in the OSI Interoperability Workshop, using the padding conventions described in PKCS1.
NONEwithDSA	The Digital Signature Algorithm as defined in FIPS PUB 186-2. The data must be exactly 20 bytes in length. This algorithms is also known under the alias name of rawDSA.
SHA1withDSA	The DSA with SHA-1 signature algorithm which uses the SHA-1 digest algorithm and DSA to create and verify DSA digital signatures as defined in FIPS PUB 186.
NONEwithECDSA SHA1withECDSA SHA256withECDSA SHA384withECDSA SHA512withECDSA *(ECDSA)*	The ECDSA signature algorithms as defined in ANSI X9.62. Note:"ECDSA" is an ambiguous name for the "SHA1withECDSA" algorithm and should not be used. The formal name "SHA1withECDSA" should be used instead.

Alg. Name	Description
<digest>with<encryption>	Use this to form a name for a signature algorithm with a particular message digest (such as MD2 or MD5) and algorithm (such as RSA or DSA), just as was done for the explicitly-defined standard names in this section (MD2withRSA, etc.). For the new signature schemes defined in PKCS1 v 2.0, for which the <digest>with<encryption> form is insufficient, <digest>with<encryption>and<mgf> can be used to form a name. Here, <mgf> should be replaced by a mask generation function such as MGF1. Example: MD5withRSAandMGF1.

SSLContext Algorithms

The algorithm names in this section can be specified when generating an instance of **SSLContext**.

Alg. Name	Description
SSL	Supports some version of SSL; may support other versions
SSLv2	Supports SSL version 2 or higher; may support other versions
SSLv3	Supports SSL version 3; may support other versions
TLS	Supports some version of TLS; may support other versions
TLSv1	Supports RFC 2246: TLS version 1.0; may support other versions
TLSv1.1	Supports RFC 4346: TLS version 1.1 since Java SE 6u111; may support other versions
TLSv1.2	Supports RFC 5246: TLS version 1.2 since Java SE 6u121; may support other versions

TrustManagerFactory Algorithms

The algorithm names in this section can be specified when generating an instance of **TrustManagerFactory**.

Alg. Name	Description
PKIX	A factory for X509TrustManagers which validate certificate chains according to the rules defined by the IETF PKIX working group in RFC 3280 or its successor. The TrustManagerFactory must support initialization using the class `javax.net.ssl.CertPathTrustManagerParameters`.

XML Signature (**XMLSignatureFactory**/**KeyInfoFactory**/**TransformService**) Mechanisms

The mechanisms in this section can be specified when generating an instance of **XMLSignatureFactory**, **KeyInfoFactory**, or **TransformService**. The mechanism identifies the XML processing mechanism that an implementation uses internally to parse and generate XML signature and KeyInfo structures. Also, note that each **TransformService** instance supports a specific transform algorithm in addition to a mechanism. The standard names for the transform algorithms are defined in the next section.

Mechanism	Description
DOM	The Document Object Model. See DOM Mechanism Requirements for additional requirements for DOM implementations.

XML Signature Transform (**TransformService**) Algorithms

The algorithms in this section can be specified when generating an instance of **TransformService**. Note that URIs are specified instead of names to be consistent with the XML Signature standard. API

constants have been defined for each of these URIs, and these are listed in parentheses after each URI in the table below.

Algorithm URI	Description
http://www.w3.org/TR/2001/REC-xml-c14n-20010315 (**`CanonicalizationMethod.INCLUSIVE`**)	The Canonical XML (without comments) canonicalization algorithm.
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments (**`CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS`**)	The Canonical XML with comments canonicalization algorithm.
http://www.w3.org/2001/10/xml-exc-c14n# (**`CanonicalizationMethod.EXCLUSIVE`**)	The Exclusive Canonical XML (without comments) canonicalization algorithm.
http://www.w3.org/2001/10/xml-exc-c14n#WithComments (**`CanonicalizationMethod.EXCLUSIVE_WITH_COMMENTS`**)	The Exclusive Canonical XML with comments canonicalization algorithm.
http://www.w3.org/2000/09/xmldsig#base64 (**`Transform.BASE64`**)	The Base64 transform algorithm.
http://www.w3.org/2000/09/xmldsig#enveloped-signature (**`Transform.ENVELOPED`**)	The Enveloped Signature transform algorithm.
http://www.w3.org/TR/1999/REC-xpath-19991116 (**`Transform.XPATH`**)	The XPath transform algorithm.
http://www.w3.org/2002/06/xmldsig-filter2 (**`Transform.XPATH2`**)	The XPath Filter 2 transform algorithm.
http://www.w3.org/TR/1999/REC-xslt-19991116 (**`Transform.XSLT`**)	The XSLT transform algorithm.

Additional JSSE Standard Names

The **keyType** parameter passed to the **chooseClientAlias**, **chooseServerAlias**, **getClientAliases**, and **getServerAliases** methods of **X509KeyManager** specify the public key type(s). Each row of the table below lists the standard name that should be used for **keyType**, given the specified certificate type.

Name	Certificate Type
RSA	RSA
DSA	DSA
DH_RSA	Diffie-Hellman with RSA signature
DH_DSA	Diffie-Hellman with DSA signature
EC	Elliptic Curve
EC_EC	Elliptic Curve with ECDSA signature
EC_RSA	Elliptic Curve with RSA signature

The **protocols** parameter passed to the **setEnabledProtocols** method of **SSLSocket** specifies the protocol versions to be enabled for use on the connection. The table below lists the standard names that can be passed to **setEnabledProtocols** or that may be returned by the **SSLSocket getSupportedProtocols** and **getEnabledProtocols** methods.

Name	Protocol
SSLv2	SSL version 2 protocol
SSLv3	SSL version 3 protocol
TLSv1	TLS version 1.0 protocol (defined in RFC 2246)
TLSv1.1	TLS version 1.1 protocol (defined in RFC 4346); supported since Java SE 6u111
TLSv1.2	TLS version 1.2 protocol (defined in RFC 5246); supported since Java SE 6u121
SSLv2Hello	Currently, the SSLv3, TLSv1, and TLSv1.1 protocols allow you to send SSLv3, TLSv1, and TLSv1.1 hellos encapsulated in an SSLv2 format hello. For more details on the reasons for allowing this compatibility in these protocols, see Appendix E in the appropriate RFCs (above). Note that some SSL/TLS servers do not support the v2 hello format and require that client hellos conform to the SSLv3 or TLSv1 client hello formats.

	The SSLv2Hello option controls the SSLv2 encapsulation. If SSLv2Hello is disabled on the client, then all outgoing messages will conform to the SSLv3/TLSv1 client hello format. If SSLv2Hello is disabled on the server, then all incoming messages must conform to the SSLv3/TLSv1 client hello format.

The **authType** parameter passed to the **checkClientTrusted** and **checkServerTrusted** methods of **X509TrustManager** indicates the authentication type. The table below specifies what standard names should be used for the client or server certificate chains.

Client or Server Certificate Chain	Authentication Type Standard Name
Client	Determined by the actual certificate used. For instance, if RSAPublicKey is used, the **authType** should be "RSA".
Server	The key exchange algorithm portion of the cipher suites represented as a String, such as "RSA" or "DHE_DSS". Note: For some exportable cipher suites, the key exchange algorithm is determined at run time during the handshake. For instance, for TLS_RSA_EXPORT_WITH_RC4_40_MD5, the **authType** should be "RSA_EXPORT" when an ephemeral RSA key is used for the key exchange, and "RSA" when the key from the server certificate is used. Or it can take the value "UNKNOWN".

Over time, various groups have added new ciphersuites definitions to the SSL/TLS namespace. Some ciphersuite names were defined before TLSv1.0 was finalized, and were therefore given the **SSL_*** prefix. The names mentioned in the TLS RFC prefixed with **TLS_*** are functionally equivalent to the JSSE ciphersuites prefixed with **SSL_***.

The following table lists the standard cipher suite names:

CipherSuite
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA
SSL_DH_DSS_WITH_DES_CBC_SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_RSA_WITH_DES_CBC_SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA
SSL_DHE_DSS_WITH_RC4_128_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
SSL_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA

CipherSuite
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_RC4_128_MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_FORTEZZA_DMS_WITH_NULL_SHA
SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_IDEA_CBC_SHA
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_RSA_EXPORT1024_WITH_RC4_56_SHA
SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA
SSL_RSA_FIPS_WITH_DES_CBC_SHA
SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
TLS_KRB5_WITH_3DES_EDE_CBC_MD5
TLS_KRB5_WITH_3DES_EDE_CBC_SHA
TLS_KRB5_WITH_DES_CBC_MD5
TLS_KRB5_WITH_DES_CBC_SHA
TLS_KRB5_WITH_IDEA_CBC_SHA
TLS_KRB5_WITH_IDEA_CBC_MD5
TLS_KRB5_WITH_RC4_128_MD5
TLS_KRB5_WITH_RC4_128_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
TLS_KRB5_EXPORT_WITH_RC2_CBC_40_SHA
TLS_KRB5_EXPORT_WITH_RC2_CBC_40_MD5
TLS_KRB5_EXPORT_WITH_RC4_40_MD5
TLS_KRB5_EXPORT_WITH_RC4_40_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ECDSA_WITH_RC4_128_SHA

CipherSuite
TLS_ECDH_ECDSA_WITH_NULL_SHA
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_RSA_WITH_RC4_128_SHA
TLS_ECDH_RSA_WITH_NULL_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_NULL_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_NULL_SHA
TLS_ECDH_anon_WITH_AES_128_CBC_SHA
TLS_ECDH_anon_WITH_AES_256_CBC_SHA
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_anon_WITH_RC4_128_SHA
TLS_ECDH_anon_WITH_NULL_SHA
TLS_EMPTY_RENEGOTIATION_INFO_SCSV *

* This is a new pseudo-ciphersuite to support RFC 5746. Please see the Transport Layer Security (TLS) Renegotiation Issue section of the JSEE Reference Guide for more information.

Algorithms

This Section specifies details concerning some of the algorithms defined in this document. Any provider supplying an implementation of the listed algorithms must comply with the specifications in this appendix.

To add a new algorithm not specified here, you should first survey other people or companies supplying provider packages to see if they have already added that algorithm, and, if so, use the definitions they published, if available. Otherwise, you should create and make available a template, similar to those found in this Section, with the specifications for the algorithm you provide.

Specification Template

The following table shows the fields of the algorithm specifications.

Field	Description

Name	The name by which the algorithm is known. This is the name passed to the `getInstance` method (when requesting the algorithm), and returned by the `getAlgorithm` method to determine the name of an existing algorithm object. These methods are in the relevant engine classes: **`Signature`**, **`MessageDigest`**, **`KeyPairGenerator`**, and `AlgorithmParameterGenerator` .
Type	The type of algorithm: **`Signature`**, **`MessageDigest`**, **`KeyPairGenerator`**, or **`ParameterGenerator`**.
Description	General notes about the algorithm, including any standards implemented by the algorithm, applicable patents, etc.
`KeyPair` Algorithm (*optional*)	The keypair algorithm for this algorithm.
Keysize (*optional*)	For a keyed algorithm or key generation algorithm: the legal keysizes.
Size (*optional*)	For an algorithm parameter generation algorithm: the legal "sizes" for algorithm parameter generation.
Parameter Defaults (*optional*)	For a key generation algorithm: the default parameter values.
`Signature` Format (*optional*)	For a **`Signature`** algorithm, the format of the signature, that is, the input and output of the verify and sign methods, respectively.

Algorithm Specifications

SHA-1 Message Digest Algorithm

Name	SHA-1
Type	**`MessageDigest`**
Description	The message digest algorithm as defined in NIST's FIPS 180-2. The output of this algorithm is a 160-bit digest.

MD2 Message Digest Algorithm

Name	MD2
Type	**`MessageDigest`**
Description	The message digest algorithm as defined in RFC 1319. The output of this algorithm is a 128-bit (16 byte) digest.

MD5 Message Digest Algorithm

Name	MD5
Type	**`MessageDigest`**
Description	The message digest algorithm as defined in RFC 1321. The output of this algorithm is a 128-bit (16 byte) digest.

The Digital Signature Algorithm

Name	SHA1withDSA
Type	**`Signature`**
Description	This algorithm is the signature algorithm described in NIST FIPS 186, using DSA with the SHA-1 message digest algorithm.
`KeyPair` **Algorithm**	DSA
Signature Format	ASN.1 sequence of two INTEGER values: **`r`** and **`s`**, in that order: **`SEQUENCE ::= { r INTEGER, s INTEGER }`**

RSA-based Signature Algorithms, with MD2, MD5 or SHA-1

Names	MD2withRSA, MD5withRSA and SHA1withRSA
Type	**`Signature`**
Description	These are the signature algorithms that use the MD2, MD5, and SHA-1 message digest algorithms (respectively) with RSA encryption.
`KeyPair` **Algorithm**	RSA
Signature Format	DER-encoded PKCS1 block as defined in RSA Laboratory's *Public Key*

	Cryptography Standards Note #1. The data encrypted is the digest of the data signed.

DSA KeyPair Generation Algorithm

Name	DSA
Type	`KeyPairGenerator`
Description	This algorithm is the key pair generation algorithm described in NIST FIPS 186 for DSA.
Keysize	The length, in bits, of the modulus `p`. This must range from 512 to 1024, and must be a multiple of 64. The default keysize is 1024.

Parameter Defaults	The following default parameter values are used for keysizes of 512, 768, and 1024 bits:
	
	512-bit Key Parameters
	
	`SEED = b869c82b 35d70e1b 1ff91b28 e37a62ec dc34409b`
	`counter = 123`
	`p = fca682ce 8e12caba 26efccf7 110e526d b078b05e decbcd1e b4a208f3 ae1617ae 01f35b91 a47e6df6 3413c5e1 2ed0899b cd132acd 50d99151 bdc43ee7 37592e17`
	`q = 962eddcc 369cba8e bb260ee6 b6a126d9 346e38c5`
	
	`g = 678471b2 7a9cf44e e91a49c5 147db1a9 aaf244f0 5a434d64 86931d2d 14271b9e 35030b71 fd73da17 9069b32e 2935630e 1c206235 4d0da20a 6c416e50 be794ca4`
	
	768-bit key parameters
	
	`SEED = 77d0f8c4 dad15eb8 c4f2f8d6 726cefd9 6d5bb399`
	`counter = 263`
	`p = e9e64259 9d355f37 c97ffd35 67120b8e 25c9cd43 e927b3a9 670fbec5 d8901419 22d2c3b3 ad248009 3799869d 1e846aab 49fab0ad 26d2ce6a 22219d47 0bce7d77 7d4a21fb e9c270b5 7f607002 f3cef839 3694cf45 ee3688c1 1a8c56ab 127a3daf`
	`q = 9cdbd84c 9f1ac2f3 8d0f80f4 2ab952e7 338bf511`
	`g = 30470ad5 a005fb14 ce2d9dcd 87e38bc7 d1b1c5fa cbaecbe9 5f190aa7 a31d23c4 dbbcbe06 17454440 1a5b2c02 0965d8c2 bd2171d3 66844577 1f74ba08 4d2029d8 3c1c1585 47f3a9f1 a2715be2 3d51ae4d 3e5a1f6a 7064f316 933a346d 3f529252`
	
	1024-bit key parameters
	
	`SEED = 8d515589 4229d5e6 89ee01e6 018a237e 2cae64cd`
	`counter = 92`
	`p = fd7f5381 1d751229 52df4a9c 2eece4e7 f611b752 3cef4400 c31e3f80 b6512669 455d4022 51fb593d 8d58fabf c5f5ba30 f6cb9b55 6cd7813b 801d346f f26660b7 6b9950a5 a49f9fe8 047b1022 c24fbba9 d7feb7c6 1bf83b57 e7c6a8a6 150f04fb 83f6d3c5 1ec30235 54135a16 9132f675 f3ae2b61 d72aeff2 2203199d d14801c7`
	`q = 9760508f 15230bcc b292b982 a2eb840b f0581cf5`
	
	`g = f7e1a085 d69b3dde cbbcab5c 36b857b9 7994afbb fa3aea82 f9574c0b 3d078267 5159578e bad4594f e6710710 8180b449 167123e8 4c281613 b7cf0932 8cc8a6e1 3c167a8b 547c8d28 e0a3ae1e 2bb3a675 916ea37f 0bfa2135 62f1fb62 7a01243b cca4f1be a8519089 a883dfe1 5ae59f06 928b665e 807b5525 64014c3b fecf492a`

RSA KeyPair Generation Algorithm

Names	RSA
Type	`KeyPairGenerator`
Description	This algorithm is the key pair generation algorithm described in PKCS1.
Strength	Any integer that is a multiple of 8, greater than or equal to 512.

DSA Parameter Generation Algorithm

Names	DSA
Type	`ParameterGenerator`
Description	This algorithm is the parameter generation algorithm described in NIST FIPS 186 for DSA.
Strength	The length, in bits, of the modulus `p`. This must range from 512 to 1024, and must be a multiple of 64. The default size is 1024.

Implementation Requirements

This section defines the security algorithm requirements for Java SE 6 implementations. These requirements are intended to improve the interoperability of Java SE 6 implementations and applications that use these algorithms.

The implementation requirements are one of two types:

- RECOMMENDED: Each implementation of Java SE 6 should, by default (as installed), support the specified algorithm(s).
- REQUIRED: Each implementation of Java SE 6 must, by default (as installed), support the specified algorithm(s).

Note that the requirements in this section are not a measure of the strength or security of the algorithm. For example, recent advances in cryptanalysis have found weaknesses in the strength of the MD5 MessageDigest algorithm. It is your responsibility to determine whether the algorithm meets the security requirements of your application.

If an algorithm or engine is not listed, then it is not subject to the specified requirement.

Java Cryptography and PKI Algorithms

Each implementation of Java SE 6 should, by default (as installed), support the specified algorithms (all RECOMMENDED) in the table below. For the next release of Java SE (7), the plan is that each algorithm in the table below will be changed to REQUIRED. These requirements do not apply to 3rd party providers.

RECOMMENDED

Engine	Algorithm Name(s)	Requirement
`AlgorithmParameters`	DSA	RECOMMENDED
`CertificateFactory`	X.509	RECOMMENDED
`CertPathBuilder`	PKIX	RECOMMENDED
`CertPathValidator`	PKIX	RECOMMENDED
`CertStore`	Collection	RECOMMENDED
`Cipher` (the algorithms are specified as transformations). Implementations must support up to the key size in parentheses.	AES/CBC/NoPadding (128) AES/CBC/PKCS5Padding (128) AES/ECB/NoPadding (128) AES/ECB/PKCS5Padding (128) DES/CBC/NoPadding (56) DES/CBC/PKCS5Padding (56) DES/ECB/NoPadding (56) DES/ECB/PKCS5Padding (56) DESede/CBC/NoPadding (168) DESede/CBC/PKCS5Padding (168) DESede/ECB/NoPadding (168) DESede/ECB/PKCS5Padding (168) RSA/ECB/PKCS1Padding (2048) RSA/ECB/OAEPPadding (2048)	RECOMMENDED
`KeyFactory` (implementations must support up to the key size in parentheses)	DSA (1024) RSA (2048)	RECOMMENDED

Engine	Algorithm Name(s)	Requirement
`KeyGenerator` (implementations must support up to the key size in parentheses)	AES (128) DES (56) DESede (168) HmacMD5 (128) HmacSHA1 (160) HmacSHA256 (160)	RECOMMENDED
`KeyPairGenerator` (implementations must support up to the key size in parentheses)	DSA (1024) RSA (2048)	RECOMMENDED
`KeyStore`	jks pkcs12	RECOMMENDED
`Mac` (implementations must support up to the key size in parentheses)	HmacMD5 (128) HmacSHA1 (160) HmacSHA256 (160)	RECOMMENDED
`MessageDigest`	MD5 SHA-1 SHA-256	
`SecretKeyFactory` (implementations must support up to the key size in parentheses)	AES (128) DES (56) DESede (168)	RECOMMENDED
`Signature`	MD5withRSA SHA1withDSA SHA1withRSA SHA256withRSA	RECOMMENDED

XML Signature Algorithms

The following table lists the algorithm requirements of XML Signature (JSR 105) implementations.

Engine	Algorithm Name(s)	Requirement
`TransformService`	http://www.w3.org/2001/10/xml-exc-c14n# (`CanonicalizationMethod.EXCLUSIVE`)	REQUIRED
`TransformService`	http://www.w3.org/2001/10/xml-exc-c14n#WithComments (`CanonicalizationMethod.EXCLUSIVE_WITH_COMMENTS`)	RECOMMENDED
`TransformService`	http://www.w3.org/TR/2001/REC-xml-c14n-20010315 (`CanonicalizationMethod.INCLUSIVE`)	REQUIRED
`TransformService`	http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments (`CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS`)	RECOMMENDED
`TransformService`	http://www.w3.org/2000/09/xmldsig#base64 (`Transform.BASE64`)	REQUIRED
`TransformService`	http://www.w3.org/2000/09/xmldsig#enveloped-signature (`Transform.ENVELOPED`)	REQUIRED
`TransformService`	http://www.w3.org/TR/1999/REC-xpath-19991116 (`Transform.XPATH`)	RECOMMENDED
`TransformService`	http://www.w3.org/2002/06/xmldsig-filter2 (`Transform.XPATH2`)	RECOMMENDED
`XMLSignatureFactory`	http://www.w3.org/2000/09/xmldsig#sha1 (`DigestMethod.SHA1`)	REQUIRED
`XMLSignatureFactory`	http://www.w3.org/2000/09/xmldsig#hmac-sha1 (`SignatureMethod.HMAC_SHA1`)	REQUIRED
`XMLSignatureFactory`	http://www.w3.org/2000/09/xmldsig#dsa-sha1 (`SignatureMethod.DSA_SHA1`)	REQUIRED
`XMLSignatureFactory`	http://www.w3.org/2000/09/xmldsig#rsa-sha1 (`SignatureMethod.RSA_SHA1`)	RECOMMENDED