Java Card Platform

Options List

Version 3.2

January 2023

Java Card Platform options list, Version 3.2

License Restrictions Warranty/Consequential Damages Disclaimer

Warranty Disclaimer

Restricted Rights Notice

Hazardous Applications Notice

Trademark Notice

Third-Party Content, Products, and Services Disclaimer

Preface

Java Card technology combines a subset of the Java programming language with a runtime environment optimized for secure elements, such as smart cards and other tamper-resistant security chips. Java Card technology offers a secure and interoperable execution platform that can store and update multiple applications on a single resource-constrained device, while retaining the highest certification levels and compatibility with standards. Java Card developers can build, test, and deploy applications and services rapidly and securely. This accelerated process reduces development costs, increases product differentiation, and enhances value to customers.

The Classic Edition of the Java Card platform is defined by three specifications:

- *Java Card Runtime Environment Specification, Java Card Platform, Version 3.2, Classic Edition*,
- *Virtual Machine Specification, Java Card Platform, Version 3.2, Classic Edition*,
- *Application Programming Interface, Java Card Platform, Version 3.2, Classic Edition*.

This document describes the list of options available to implement a Java Card platform, based on the Java Card Specifications.

Who Should Use This Document

This document is intended both for Oracle Java Card licensees who are implementing the Java Card Platform and for application developers who want an understanding of the changes introduced in this release of the Java Card specifications.

Before You Read This Document

Before reading this guide, you should be familiar with the Java programming language, the Java Card technology specifications, and smart card technology. A good resource for becoming familiar with Java technology and Java Card technology located at:

http://www.oracle.com/technetwork/java/javacard/overview/

Typographic Conventions

The settings on your browser might differ from these settings.

Typeface	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	**`%su Password:`**
AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename*.

Related Documentation

A list of related documents that may help in understanding this document are:

[JCAPI] *Application Programming Interface, Java Card Platform, Version 3.2, Classic Edition*

[JCVM] *Virtual Machine Specification, Java Card Platform, Version 3.2, Classic Edition*

[JCRE] *Runtime Specification, Java Card Platform, Version 3.2, Classic Edition*

[JLS] *The Java Language Specification (https://docs.oracle.com/javase/specs/)*

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at:

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit:

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info

Or, if you are hearing impaired, visit:

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs

Oracle Welcomes Your Comments

Oracle is interested in improving its documentation and welcomes your comments and suggestions.

Please include the title of your document with your feedback:

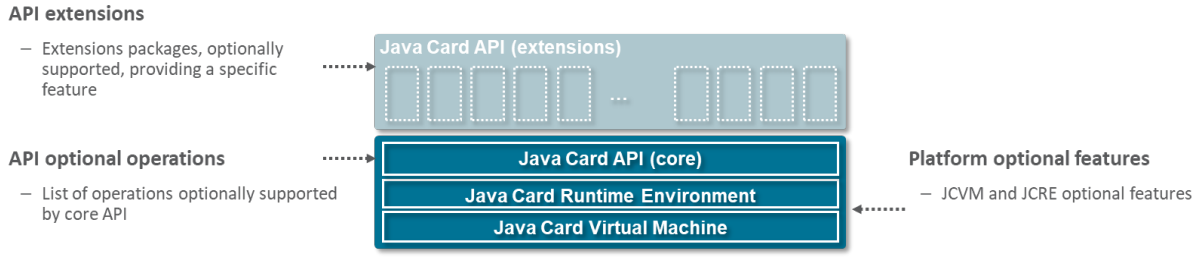
Java Card Platform Options List, Version 3.2, Classic Edition

Table of Contents

1 Overview

The Java Card Platform is extensible and offers a number of possible options for implementers to create products that fulfill a wide range of requirements and applications needs. In addition to these optional features, the specification also supports configuration parameters such as heap and stack sizes, transient memory size or transaction buffer size. All these enable the creation of products for a variety of hardware platforms with different resources constraints.

This document focuses only on optional features categorized as below.



API Extensions

API extensions consist in a set of packages, each defining a feature set. Each extension package might optionally be supported in an implementation. If supported, all the classes and interfaces defined in an extension package must be implemented and embedded in the platform.

See section *3 - API - Extension packages (optional)* for a detailed list of extension packages.

API optional operations

The core API is made of a set of packages with classes or interfaces that must be included in any platform implementation. An application must be able to rely on these classes. However, some operations are specified as being optional and might throw an exception when used. This is specifically the case for cryptographic operations when the algorithm requested by the application is not supported by the implementation.

See section 4 - *API - Optional operations* for a detailed list of optional operations.

Platform optional features

The Java Card Virtual Machine and Java Card Runtime Environment are also optionally supporting some features.

See section 5 - Platform optional features for a detailed list of platform options.

2 API - Core packages (mandatory)

These packages represent the minimal set of classes and interfaces that must be included in any Java Card platform implementation.

Language features
`java.lang`
`java.io`
	Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language.

Application framework `javacard.framework`	Provides a framework of classes and interfaces for building, communicating with and working with Java Card technology-based applets.
Security and Cryptography `javacard.security`	Provides classes and interfaces that contain publicly-available functionality for implementing a security and cryptography framework on the Java Card platform.

3 API - Extension packages (optional)

The packages listed below can optionally be included in a specific Java Card platform implementation. When an optional feature is supported, the proposed API, which is defined based on the industry requirements, must be used instead of any other proprietary APIs to guarantee interoperability and avoid fragmentation.

Package	Description	since
Extended APDU `javacardx.apdu`	Extension package that enables support for ISO7816 specification defined optional APDU related mechanisms. See section 5 - Platform optional features– Extended APDU.	2.2.2
APDU Utilities `javacardx.apdu.util`	Extension package containing helper classes to interpret APDU commands.	3.0.5
Biometry `javacardx.biometry`	Extension package that contains functionality for implementing a biometric framework on the Java Card platform.	2.2.2
Biometry (1toN) `javacardx.biometry1toN`	Extension package that contains functionality for implementing a 1:N biometric framework on the Java Card platform.	3.0.5
Cryptography `javacardx.crypto`	Extension package that contains functionality, which may be subject to export controls, for implementing a security and cryptography framework on the Java Card platform.	2.1
External Memory Access `javacardx.external`	Extension package that provides mechanisms to access memory subsystems which are not directly addressable by the Java Card runtime environment on the Java Card platform (e.g. Mifare).	2.2.2

Package	Description	since
Event framework `javacardx.framework.event`	Extension package that defines a framework to handle different source of events.	3.1
Big Numbers `javacardx.framework.math`	Extension package to perform operations on Big Numbers in either binary form or Binary Coded Decimal (BCD) form and perform parity computations.	2.2.2
NIO buffers `javacardx.framework.nio`	Extension package that defines buffers, which are containers for data.	3.1
String utilities `javacardx.framework.string`	Extension package that contains common utility functions for manipulating UTF-8 encoded character strings.	3.0.4
System time `javacardx.framework.time`	Extension package that defines classes to handle system uptime and perform operations on time durations.	3.1
BER-TLV encoding/decoding `javacardx.framework.tlv`	Extension package for managing the storage of BER TLV formatted data, based on the ASN.1 BER encoding rules of ISO/IEC 8825-1:2002, as well as parsing and editing BER TLV formatted data in I/O buffers.	2.2.2
Array Utilities `javacardx.framework.util`	Extension package that contains common utility functions for manipulating arrays of primitive components - `byte`, `short` or `int`.	2.2.2
int utilities `javacardx.framework.util.intx`	Extension package that contains common utility functions for using `int` components. This package is optional and can only be implemented on a platform also supporting the `int` primitive type. It contains the `JCint` class which provides methods for functionality similar to that of the `javacard.framework.Util` class but with `int` component equivalents, like reading an `int` from a `byte[]`, writing an `int` into a `byte[]` or creating a transient `int[]`.	2.2.2

Package	Description	since

Service framework and RMI
`javacard.framework.service`
`java.rmi`
	Extension package providing a service framework of classes and interfaces that allow a Java Card technology-based applet to be designed as an aggregation of service components.	2.2

Security assertions `javacardx.security`	Extension package that contains functionality for implementing security countermeasures to protect security relevant applet assets on the Java Card platform.	3.0.5
Certificate handling `javacardx.security.cert`	Extension package that provides classes to handle certificates (e.g. X.509 certificates).	3.1
Derivation functions `javacardx.security.derivation`	Extension package that provides classes implementing cryptographic derivation functions (e.g. KDF, PRF).	3.1
Monotonic counters `javacardx.security.util`	Extension package defining utility classes for security framework (e.g. monotonic counters).	3.1

4 API - Optional operations

The operations listed in this section are optional. This means that the class and methods must be included but an implementation may throw an exception (e.g. `CryptoException.NO_SUCH_ALGORITHM`) if the specified combination of parameters is not supported.

Package `javacard.framework`

`javacard.framework.OwnerPINBuilder` class

`OwnerPINBuilder.buildOwnerPIN(byte tryLimit, byte maxPINSize, byte ownerPINType)`	**Since**
`OWNER_PIN`	3.0.5
`OWNER_PIN_X`	3.0.5
`OWNER_PIN_X_WITH_PREDECREMENT`	3.0.5

Note: method throws `SystemException.ILLEGAL_USE` if the specified `ownerPINType` is not supported

`javacard.framework.SensitiveArrays` class

`SensitiveArrays.makeIntegritySensitiveArray(byte type, byte memory, byte length)`	**Since 3.0.5**

Note: method throws `SystemException.ILLEGAL_USE` if the sensitive arrays are not supported

Package `javacard.security`

`javacard.security.Checksum` class

`Checksum.getInstance(byte alg, boolean ext)`	**Since**
`ALG_ISO3309_CRC16`	2.2
`ALG_ISO3309_CRC32`	2.2

`javacard.security.KeyAgreement` class

`KeyAgreement.getInstance(byte alg, boolean ext)`	**Since**
`ALG_DH_PLAIN`	3.0.5
`ALG_EC_PACE_GM`	3.0.5
`ALG_EC_SVDP_DH`	2.2
`ALG_EC_SVDP_DH_KDF`	2.2
`ALG_EC_SVDP_DH_PLAIN`	3.0.1
`ALG_EC_SVDP_DH_PLAIN_XY`	3.0.5
`ALG_EC_SVDP_DHC`	2.2
`ALG_EC_SVDP_DHC_KDF`	2.2
`ALG_EC_SVDP_DHC_PLAIN`	3.0.1
`ALG_XDH`	3.1
`ALG_SM2`	3.1
`ALG_SM2_WITH_CONFIRMATION`	3.2

javacard.security.KeyBuilder class

KeyBuilder.buildKey(byte type, short length, boolean ext)	
Type	**Supported length[1]**
TYPE_AES	LENGTH_AES_xxx (128,192,256)
TYPE_AES_TRANSIENT_DESELECT	LENGTH_AES_xxx (128,192,256)
TYPE_AES_TRANSIENT_RESET	LENGTH_AES_xxx (128,192,256)
TYPE_DES	LENGTH_DES, LENGTH_DES_2KEY, LENGTH_DES_3KEY
TYPE_DES_TRANSIENT_DESELECT	LENGTH_DES, LENGTH_DES_2KEY, LENGTH_DES_3KEY
TYPE_DES_TRANSIENT_RESET	LENGTH_DES, LENGTH_DES_2KEY, LENGTH_DES_3KEY
TYPE_DH_PARAMETERS	LENGTH_DH_xxx (1024..2048)
TYPE_DH_PRIVATE	LENGTH_DH_xxx (1024..2048)
TYPE_DH_PRIVATE_TRANSIENT_DESELECT	LENGTH_DH_xxx (1024..2048)
TYPE_DH_PRIVATE_TRANSIENT_RESET	LENGTH_DH_xxx (1024..2048)
TYPE_DH_PUBLIC	LENGTH_DH_xxx (1024..2048)
TYPE_DH_PUBLIC_TRANSIENT_DESELECT	LENGTH_DH_xxx (1024..2048)
TYPE_DH_PUBLIC_TRANSIENT_RESET	LENGTH_DH_xxx (1024..2048)
TYPE_DSA_PARAMETERS	LENGTH_DSA_xxx (512..1024)
TYPE_DSA_PRIVATE	LENGTH_DSA_xxx (512..1024)
TYPE_DSA_PRIVATE_TRANSIENT_DESELECT	LENGTH_DSA_xxx (512..1024)
TYPE_DSA_PRIVATE_TRANSIENT_RESET	LENGTH_DSA_xxx (512..1024)
TYPE_DSA_PUBLIC	LENGTH_DSA_xxx (512..1024)
TYPE_EC_F2M_PARAMETERS	LENGTH_EC_F2M_xxx (113,131,163,193)
TYPE_EC_F2M_PRIVATE	LENGTH_EC_F2M_xxx (113,131,163,193)
TYPE_EC_F2M_PRIVATE_TRANSIENT_DESELECT	LENGTH_EC_F2M_xxx (113,131,163,193)
TYPE_EC_F2M_PRIVATE_TRANSIENT_RESET	LENGTH_EC_F2M_xxx (113,131,163,193)
TYPE_EC_F2M_PUBLIC	LENGTH_EC_F2M_xxx (113,131,163,193)
TYPE_EC_FP_PARAMETERS	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
TYPE_EC_FP_PRIVATE	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
TYPE_EC_FP_PRIVATE_TRANSIENT_DESELECT	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
TYPE_EC_FP_PRIVATE_TRANSIENT_RESET	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
TYPE_EC_FP_PUBLIC	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
TYPE_GENERIC_SECRET	1..32
TYPE_HMAC	LENGTH_HMAC_xxx (64,128)
TYPE_HMAC_TRANSIENT_DESELECT	LENGTH_HMAC_xxx (64,128)
TYPE_HMAC_TRANSIENT_RESET	LENGTH_HMAC_xxx (64,128)
TYPE_KOREAN_SEED	LENGTH_KOREAN_SEED_128
TYPE_KOREAN_SEED_TRANSIENT_DESELECT	LENGTH_KOREAN_SEED_128
TYPE_KOREAN_SEED_TRANSIENT_RESET	LENGTH_KOREAN_SEED_128
TYPE_RSA_CRT_PRIVATE	LENGTH_RSA_xxx (512..4096)
TYPE_RSA_CRT_PRIVATE_TRANSIENT_DESELECT	LENGTH_RSA_xxx (512..4096)
TYPE_RSA_CRT_PRIVATE_TRANSIENT_RESET	LENGTH_RSA_xxx (512..4096)
TYPE_RSA_PRIVATE	LENGTH_RSA_xxx (512..4096)
TYPE_RSA_PRIVATE_TRANSIENT_DESELECT	LENGTH_RSA_xxx (512..4096)
TYPE_RSA_PRIVATE_TRANSIENT_RESET	LENGTH_RSA_xxx (512..4096)
TYPE_RSA_PUBLIC	LENGTH_RSA_xxx (512..4096)
TYPE_SM4	LENGTH_SM4
~~TYPE_XEC~~	N/A[2] for this method

KeyBuilder.buildKey(byte alg, byte memtype, short length, boolean ext) **KeyBuilder.buildKeyWithSharedDomain(byte alg,byte memtype, Key params, boolean ext)**	
Type	**Supported length1**
ALG_TYPE_AES	LENGTH_AES_xxx (128,192,256)
ALG_TYPE_DES	LENGTH_DES, LENGTH_DES_2KEY, LENGTH_DES_3KEY
ALG_TYPE_DH_PARAMETERS	LENGTH_DH_xxx (1024..2048)
ALG_TYPE_DH_PRIVATE	LENGTH_DH_xxx (1024..2048)
ALG_TYPE_DH_PUBLIC	LENGTH_DH_xxx (1024..2048)
ALG_TYPE_DSA_PARAMETERS	LENGTH_DSA_xxx (512..1024)
ALG_TYPE_DSA_PRIVATE	LENGTH_DSA_xxx (512..1024)
ALG_TYPE_DSA_PUBLIC	LENGTH_DSA_xxx (512..1024)
ALG_TYPE_EC_F2M_PARAMETERS	LENGTH_EC_F2M_xxx (113,131,163,193)

[1] The LENGTH_XXX constants are examples, an implementation may support more.

[2] This constant is returned by Key.getType() for XEC keys created with KeyBuilder.buildXECKey(NamedParameterSpec, short, boolean)

ALG_TYPE_EC_F2M_PRIVATE	LENGTH_EC_F2M_xxx (113,131,163,193)
ALG_TYPE_EC_F2M_PUBLIC	LENGTH_EC_F2M_xxx (113,131,163,193)
ALG_TYPE_EC_FP_PARAMETERS	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
ALG_TYPE_EC_FP_PRIVATE	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
ALG_TYPE_EC_FP_PUBLIC	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
ALG_TYPE_GENERIC_SECRET	1..32
ALG_TYPE_HMAC	LENGTH_HMAC_xxx (64,128)
ALG_TYPE_KOREAN_SEED	LENGTH_KOREAN_SEED_128
ALG_TYPE_RSA_CRT_PRIVATE	LENGTH_RSA_xxx (512..4096)
ALG_TYPE_RSA_PRIVATE	LENGTH_RSA_xxx (512..4096)
ALG_TYPE_RSA_PUBLIC	LENGTH_RSA_xxx (512..4096)
ALG_TYPE_SM4	LENGTH_SM4

KeyBuilder.buildXECKey(NamedParameterSpec, short, boolean)	**Since 3.1**
NamedParameterSpec.BRAINPOOLP192R1	3.1
NamedParameterSpec.BRAINPOOLP192T1	3.1
NamedParameterSpec.BRAINPOOLP224R1	3.1
NamedParameterSpec.BRAINPOOLP224T1	3.1
NamedParameterSpec.BRAINPOOLP256R1	3.1
NamedParameterSpec.BRAINPOOLP256T1	3.1
NamedParameterSpec.BRAINPOOLP320R1	3.1
NamedParameterSpec.BRAINPOOLP320T1	3.1
NamedParameterSpec.BRAINPOOLP384R1	3.1
NamedParameterSpec.BRAINPOOLP384T1	3.1
NamedParameterSpec.BRAINPOOLP512R1	3.1
NamedParameterSpec.BRAINPOOLP512T1	3.1
NamedParameterSpec.ED25519	3.1
NamedParameterSpec.ED448	3.1
NamedParameterSpec.FRP256V1	3.1
NamedParameterSpec.SECP192R1	3.1
NamedParameterSpec.SECP224R1	3.1
NamedParameterSpec.SECP256R1	3.1
NamedParameterSpec.SECP384R1	3.1
NamedParameterSpec.SECP521R1	3.1
NamedParameterSpec.SM2	3.1
NamedParameterSpec.X25519	3.1
NamedParameterSpec.X448	3.1

javacard.security.KeyPair class

KeyPair(byte algorithm, short length)	
Algorithm	**Supported length[3]**
ALG_DH	LENGTH_DH_xxx (1024..2048)
ALG_DSA	LENGTH_DSA_xxx (512..1024)
ALG_EC_F2M	LENGTH_EC_F2M_xxx (113,131,163,193)
ALG_EC_F2P	LENGTH_EC_FP_xxx (112,128,160,192,224,256,384,521)
ALG_RSA	LENGTH_RSA_xxx (512..4096)
ALG_RSA_CRT	LENGTH_RSA_xxx (512..4096)

javacard.security.MessageDigest class

MessageDigest.getInstance(byte alg, boolean ext)	**Since 2.1**
ALG_MD5	2.1
~~ALG_NULL~~ N/A[4]	3.0.4
ALG_RIPEMD160	2.1
ALG_SHA	2.1
ALG_SHA_224	3.0.1

[3] The LENGTH_XXX constants are examples, an implementation may support more.

[4] Not Applicable for this method. Only used as digest parameter used to create Signature instances.

ALG_SHA_256	2.2.2
ALG_SHA_384	2.2.2
ALG_SHA_512	2.2.2
ALG_SHA3_224	3.0.5
ALG_SHA3_256	3.0.5
ALG_SHA3_384	3.0.5
ALG_SHA3_512	3.0.5
ALG_SM3	3.1

javacard.security.RandomData class

RandomData.getInstance(byte alg) [deprecated] **RandomData.getInstance(byte alg, boolean ext)**	**Since** **2.1** **3.2**
ALG_FAST	3.0.5
ALG_KEYGENERATION	3.0.5
ALG_PRESEEDED_DRBG	3.0.5
~~ALG_PSEUDO_RANDOM~~ (deprecated)	2.1
~~ALG_SECURE_RANDOM~~ (deprectaed)	2.1
ALG_TRNG	3.0.5

javacard.security.Signature class

Signature.getInstance(byte alg, boolean ext)			**Since 2.1**
algorithm	**Corresponding <digest, cipher, padding>**		
ALG_AES_CMAC_128	ALG_NULL,	SIG_CIPHER_AES_CMAC128,	PAD_NULL
ALG_AES_MAC_128_NOPAD	ALG_NULL,	SIG_CIPHER_AES_MAC128,	PAD_NULL
ALG_DES_MAC4_ISO9797_1_M1_ALG3	ALG_NULL,	SIG_CIPHER_DES_MAC4,	PAD_ISO9797_1_M1_ALG3
ALG_DES_MAC4_ISO9797_1_M2_ALG3	ALG_NULL,	SIG_CIPHER_DES_MAC4,	PAD_ISO9797_1_M2_ALG3
ALG_DES_MAC4_ISO9797_M1	ALG_NULL,	SIG_CIPHER_DES_MAC4,	PAD_ISO9797_M1
ALG_DES_MAC4_ISO9797_M2	ALG_NULL,	SIG_CIPHER_DES_MAC4,	PAD_ISO9797_M2
ALG_DES_MAC4_NOPAD	ALG_NULL,	SIG_CIPHER_DES_MAC4,	PAD_NOPAD
ALG_DES_MAC4_PKCS5	ALG_NULL,	SIG_CIPHER_DES_MAC4,	PAD_PKCS5
ALG_DES_MAC8_ISO9797_1_M1_ALG3	ALG_NULL,	SIG_CIPHER_DES_MAC8,	PAD_ISO9797_1_M1_ALG3
ALG_DES_MAC8_ISO9797_1_M2_ALG3	ALG_NULL,	SIG_CIPHER_DES_MAC8,	PAD_ISO9797_1_M2_ALG3
ALG_DES_MAC8_ISO9797_M1	ALG_NULL,	SIG_CIPHER_DES_MAC8,	PAD_ISO9797_M1
ALG_DES_MAC8_ISO9797_M2	ALG_NULL,	SIG_CIPHER_DES_MAC8,	PAD_ISO9797_M2
ALG_DES_MAC8_NOPAD	ALG_NULL,	SIG_CIPHER_DES_MAC8,	PAD_NOPAD
ALG_DES_MAC8_PKCS5	ALG_NULL,	SIG_CIPHER_DES_MAC8,	PAD_PKCS5
ALG_DSA_SHA	ALG_SHA,	SIG_CIPHER_DSA,	PAD_NULL
ALG_ECDSA_SHA	ALG_SHA,	SIG_CIPHER_ECDSA,	PAD_NULL
ALG_ECDSA_SHA_224	ALG_SHA_224,	SIG_CIPHER_ECDSA,	PAD_NULL
ALG_ECDSA_SHA_256	ALG_SHA_256,	SIG_CIPHER_ECDSA,	PAD_NULL
ALG_ECDSA_SHA_384	ALG_SHA_384,	SIG_CIPHER_ECDSA,	PAD_NULL
ALG_ECDSA_SHA_512	ALG_SHA_512,	SIG_CIPHER_ECDSA,	PAD_NULL
ALG_HMAC_MD5	ALG_MD5,	SIG_CIPHER_HMAC,	PAD_NULL
ALG_HMAC_RIPEMD160	ALG_RIPEMD160,	SIG_CIPHER_HMAC,	PAD_NULL
ALG_HMAC_SHA_256	ALG_SHA_256,	SIG_CIPHER_HMAC,	PAD_NULL
ALG_HMAC_SHA_384	ALG_SHA_384,	SIG_CIPHER_HMAC,	PAD_NULL
ALG_HMAC_SHA_512	ALG_SHA_512,	SIG_CIPHER_HMAC,	PAD_NULL
ALG_HMAC_SHA1	ALG_SHA,	SIG_CIPHER_HMAC,	PAD_NULL
ALG_KOREAN_SEED_MAC_NOPAD	ALG_NULL,	SIG_CIPHER_KOREAN_SEED_MAC,	PAD_NOPAD
ALG_RSA_MD5_PKCS1	ALG_MD5,	SIG_CIPHER_RSA,	PAD_PKCS1
ALG_RSA_MD5_PKCS1_PSS	ALG_MD5,	SIG_CIPHER_RSA,	PAD_PKCS1_PSS
ALG_RSA_MD5_RFC2409	ALG_MD5,	SIG_CIPHER_RSA,	PAD_RFC2409
ALG_RSA_RIPEMD160_ISO9796	ALG_RIPEMD160,	SIG_CIPHER_RSA,	PAD_ISO9796

ALG_RSA_RIPEMD160_ISO9796_MR	ALG_RIPEMD160,	SIG_CIPHER_RSA,	PAD_ISO9796_MR
ALG_RSA_RIPEMD160_PKCS1	ALG_RIPEMD160,	SIG_CIPHER_RSA,	PAD_PKCS1
ALG_RSA_RIPEMD160_PKCS1_PSS	ALG_RIPEMD160,	SIG_CIPHER_RSA,	PAD_PKCS1_PSS
ALG_RSA_SHA_224_PKCS1	ALG_SHA_224,	SIG_CIPHER_RSA,	PAD_PKCS1
ALG_RSA_SHA_224_PKCS1_PSS	ALG_SHA_224,	SIG_CIPHER_RSA,	PAD_PKCS1_PSS
ALG_RSA_SHA_256_PKCS1	ALG_SHA_256,	SIG_CIPHER_RSA,	PAD_PKCS1
ALG_RSA_SHA_256_PKCS1_PSS	ALG_SHA_256,	SIG_CIPHER_RSA,	PAD_PKCS1_PSS
ALG_RSA_SHA_384_PKCS1	ALG_SHA_384,	SIG_CIPHER_RSA,	PAD_PKCS1
ALG_RSA_SHA_384_PKCS1_PSS	ALG_SHA_384,	SIG_CIPHER_RSA,	PAD_PKCS1_PSS
ALG_RSA_SHA_512_PKCS1	ALG_SHA_512,	SIG_CIPHER_RSA,	PAD_PKCS1
ALG_RSA_SHA_512_PKCS1_PSS	ALG_SHA_512,	SIG_CIPHER_RSA,	PAD_PKCS1_PSS
ALG_RSA_SHA_ISO9796	ALG_SHA,	SIG_CIPHER_RSA,	PAD_ISO9796
ALG_RSA_SHA_ISO9796_MR	ALG_SHA,	SIG_CIPHER_RSA,	PAD_ISO9796_MR
ALG_RSA_SHA_PKCS1	ALG_SHA,	SIG_CIPHER_RSA,	PAD_PKCS1
ALG_RSA_SHA_PKCS1_PSS	ALG_SHA,	SIG_CIPHER_RSA,	PAD_PKCS1_PSS
ALG_RSA_SHA_RFC2409	ALG_SHA,	SIG_CIPHER_RSA,	PAD_RFC2409

Signature.getInstance(byte digest, byte cipher, byte padding, boolean ext)						**since 3.0.4**
Digest	**X**	**Cipher**		**X**	**Padding**	
ALG_MD5		SIG_CIPHER_AES_CMAC128			PAD_ISO9796	
ALG_RIPEMD160		SIG_CIPHER_AES_MAC128			PAD_ISO9796_MR	
ALG_SHA		SIG_CIPHER_DES_MAC4			PAD_ISO9796_MR_SCHEME_1_OPTION_2	3.2
ALG_SHA_224		SIG_CIPHER_DES_MAC8			PAD_ISO9796_MR_SCHEME_2_OPTION_2	3.2
ALG_SHA_256		SIG_CIPHER_DSA			PAD_ISO9796_MR_SCHEME_3_OPTION_2	3.2
ALG_SHA_384		SIG_CIPHER_ECDSA			PAD_ISO9797_1_M1_ALG3	
ALG_SHA_512		SIG_CIPHER_ECDSA_PLAIN			PAD_ISO9797_1_M2_ALG3	
ALG_SHA3_224		SIG_CIPHER_EDDSA			PAD_ISO9797_M1	
ALG_SHA3_256		SIG_CIPHER_EDDSAPH			PAD_ISO9797_M2	
ALG_SHA3_384		SIG_CIPHER_EDDSA_ED25519	3.2		PAD_NOPAD	
ALG_SHA3_512		SIG_CIPHER_EDDSA_ED448	3.2		PAD_NULL	
		SIG_CIPHER_EDDSAPH_ED25519	3.2		PAD_PKCS1	
		SIG_CIPHER_EDDSAPH_ED448	3.2		~~PAD_PKCS1_OAEP~~ N/A[5]	
		SIG_CIPHER_HMAC			~~PAD_PKCS1_OAEP_EXT_PARAMETERS~~ N/A[5]	3.2
		SIG_CIPHER_KOREAN_SEED_MAC			~~PAD_PKCS1_OAEP_SHA224~~ N/A[5]	
		SIG_CIPHER_RSA			~~PAD_PKCS1_OAEP_SHA256~~ N/A[5]	
		SIG_CIPHER_SM2			~~PAD_PKCS1_OAEP_SHA3_224~~ N/A[5]	
		SIG_CIPHER_SM4_MAC128			~~PAD_PKCS1_OAEP_SHA3_256~~ N/A[5]	
					~~PAD_PKCS1_OAEP_SHA3_384~~ N/A[5]	
					~~PAD_PKCS1_OAEP_SHA3_512~~ N/A[5]	
					~~PAD_PKCS1_OAEP_SHA384~~ N/A[5]	
					~~PAD_PKCS1_OAEP_SHA512~~ N/A[5]	
					PAD_PKCS1_PSS	
					PAD_PKCS1_PSS_EXT_PARAMETERS	3.2
					PAD_PKCS5	
					PAD_RFC2409	

[5] Padding Not Applicable for Signature operations.

Package `javacardx.crypto`

`javacardx.crypto.Cipher class`

`Cipher.getInstance(byte alg, boolean ext)`		**Since 2.1**
algorithm	**Corresponding <cipher, padding>**	
ALG_AES_BLOCK_128_CBC_NOPAD	CIPHER_AES_CBC,	PAD_NOPAD
ALG_AES_BLOCK_128_ECB_NOPAD	CIPHER_AES_ECB,	PAD_NOPAD
~~ALG_AES_BLOCK_192_CBC_NOPAD~~	Deprecated	
~~ALG_AES_BLOCK_192_ECB_NOPAD~~	Deprecated	
~~ALG_AES_BLOCK_256_CBC_NOPAD~~	Deprecated	
~~ALG_AES_BLOCK_256_ECB_NOPAD~~	Deprecated	
ALG_AES_CBC_ISO9797_M1	CIPHER_AES_CBC,	PAD_ISO9797_M1
ALG_AES_CBC_ISO9797_M2	CIPHER_AES_CBC,	PAD_ISO9797_M2
ALG_AES_CBC_PKCS5	CIPHER_AES_CBC,	PAD_PKCS5
ALG_AES_CFB	CIPHER_AES_CFB,	PAD_NULL
ALG_AES_CTR	CIPHER_AES_CTR,	PAD_NULL
ALG_AES_ECB_ISO9797_M1	CIPHER_AES_CFB,	PAD_ISO9797_M1
ALG_AES_ECB_ISO9797_M2	CIPHER_AES_CFB,	PAD_ISO9797_M2
ALG_AES_ECB_PKCS5	CIPHER_AES_CFB,	PAD_PKCS5
ALG_AES_XTS	CIPHER_AES_XTS,	PAD_NULL
ALG_DES_CBC_ISO9797_M1	CIPHER_DES_CBC,	PAD_ISO9797_M1
ALG_DES_CBC_ISO9797_M2	CIPHER_DES_CBC,	PAD_ISO9797_M2
ALG_DES_CBC_NOPAD	CIPHER_DES_CBC,	PAD_NOPAD
ALG_DES_CBC_PKCS5	CIPHER_DES_CBC,	PAD_PKCS5
ALG_DES_ECB_ISO9797_M1	CIPHER_DES_ECB,	PAD_ISO9797_M1
ALG_DES_ECB_ISO9797_M2	CIPHER_DES_ECB,	PAD_ISO9797_M2
ALG_DES_ECB_NOPAD	CIPHER_DES_ECB,	PAD_NOPAD
ALG_DES_ECB_PKCS5	CIPHER_DES_ECB,	PAD_PKCS5
ALG_KOREAN_SEED_CBC_NOPAD	CIPHER_KOREAN_SEED_CBC,	PAD_NOPAD
ALG_KOREAN_SEED_ECB_NOPAD	CIPHER_KOREAN_SEED_ECB,	PAD_NOPAD
ALG_RSA_ISO14888	CIPHER_RSA,	PAD_ISO14888
ALG_RSA_ISO9796	CIPHER_RSA,	PAD_ISO9796
ALG_RSA_NOPAD	CIPHER_RSA,	PAD_NOPAD
ALG_RSA_PKCS1	CIPHER_RSA,	PAD_PKCS1
ALG_RSA_PKCS1_OAEP	CIPHER_RSA,	PAD_PKCS1_OAEP

Cipher.getInstance(byte cipher, byte padding, boolean ext**)**				**since 3.0.4**
cipher	**X**	**padding**		
CIPHER_AES_CBC		PAD_ISO9796		
CIPHER_AES_ECB		~~PAD_ISO9796_MR~~	N/A[7]	
CIPHER_AES_CTR		~~PAD_ISO9796_MR_SCHEME_1_OPTION_2~~	N/A[7]	3.2
CIPHER_AES_CFB		~~PAD_ISO9796_MR_SCHEME_2_OPTION_2~~	N/A[7]	3.2
CIPHER_AES_XTS		~~PAD_ISO9796_MR_SCHEME_3_OPTION_2~~	N/A[7]	3.2
CIPHER_DES_CBC		PAD_ISO9797_1_M1_ALG3		
CIPHER_DES_ECB		PAD_ISO9797_1_M2_ALG3		
CIPHER_KOREAN_SEED_CBC		PAD_ISO9797_M1		
CIPHER_KOREAN_SEED_ECB		PAD_ISO9797_M2		
CIPHER_RSA		PAD_NOPAD		
CIPHER_SM2		PAD_NULL[6]		
CIPHER_SM4_CBC		PAD_PKCS5		
CIPHER_SM4_ECB		PAD_PKCS1_OAEP		
		PAD_PKCS1_OAEP_EXT_PARAMETERS		3.2
		PAD_PKCS1_OAEP_SHA224		
		PAD_PKCS1_OAEP_SHA256		
		PAD_PKCS1_OAEP_SHA3_224		
		PAD_PKCS1_OAEP_SHA3_256		
		PAD_PKCS1_OAEP_SHA3_384		
		PAD_PKCS1_OAEP_SHA3_512		
		PAD_PKCS1_OAEP_SHA384		
		PAD_PKCS1_OAEP_SHA512		
		PAD_PKCS1		
		~~PAD_RFC2409~~	N/A[7]	
		~~PAD_PKCS1_PSS~~	N/A[7]	
		~~PAD_PKCS1_PSS_EXT_PARAMETERS~~	N/A[7]	3.2

Package `javacardx.security.derivation`

`javacardx.security.derivation.DerivationFunction class`

DerivationFunction.getInstance(byte alg, boolean ext)	**Since 3.1**
algorithm	
ALG_KDF_COUNTER_MODE	3.1
ALG_KDF_DPI_MODE	3.1
ALG_KDF_FEEDBACK_MODE	3.1
ALG_PRF_TLS11	3.1
ALG_PRF_TLS_12	3.1
ALG_KDF_IEEE_1363	3.1
ALG_KDF_ICAO_MRTD	3.1
ALG_KDF_ANSI_X9_63	3.1
ALG_KDF_HKDF	3.1
ALG_HKDF_EXPAND_LABEL_TLS13	3.2

[6] Only used for algorithms where padding is not applicable (AES_CTR, AES_CFB, AES_XTS)

[7] Not Applicable for Cipher operations.

5 Platform optional features

APDU Logical channels	A Java Card platform implementation must support 1 basic logical channel and may optionally support up to 20 logical channels on each physical I/O interface. See *[JCRE]* specification section 4.
Extended APDU	The support for extended APDU length (i.e. APDU payload greater than 255 bytes) is optional. If the extended length is supported by an implementation, the `javacardx.apdu` package must be included in the API allowing applications to use this feature. See *[JCRE]* specification section 9.4.
32-bit integer support	A Java Card platform implementation may optionally support 32-bit `int` type. See *[JCVM]* specification section 2.2.3.
Garbage Collector	A Java Card platform implementation may optionally support Garbage Collection. See *[JCVM]* specification section 2.2.3.
Extended CAP format	A Java Card platform implementation must support the compact CAP file format. Support for extended CAP file format is optional. The Extended CAP file format is used to deploy applications and libraries made of multiple public or private packages and can embed a much larger amount of code compared to compact format. See *[JCVM]* specification section 2.2.3.