Java Card Platform

Specification Release Notes

Version 3.2

F74152-01

January 2023

Table of Contents

Preface

Java Card technology combines a portion of the Java programming language with a runtime environment optimized for smart cards and related, small-memory embedded devices. The goal of Java Card technology is to bring many of the benefits of the Java programming language to the resource-constrained world of smart cards and secure elements.

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 changes introduced in the Version 3.2 of the specifications.

Audience

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:

`https://www.oracle.com/technetwork/java/embedded/javacard/overview/`.

Typographic Conventions

The following text conventions are used in this document:

Typeface[1]	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; onscreen 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*.

[1] The settings on your browser might differ from these settings.

Related Documents

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

- [JCAPI] - *Application Programming Interface Specification, Version 3.2, Classic Edition*

- [JCVM] - *Virtual Machine Specification, Version 3.2, Classic Edition*
- [JCRE] - *Runtime Environment Specification, Version 3.2, Classic Edition*
- [JLS]*The Java Language Specification Third Edition by James Gosling, Bill Joy, and Guy L. Steele (Addison-Wesley, 2005)*

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 visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

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, v3.2, Classic Edition

Introduction

This release notes describes the list of changes introduced in the Version 3.2 of the Java Card specifications.

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

What's New

This section lists the important changes and features in Java Card Platform Specifications, Version 3.2.

Topics:

- What’s new in the Java Card Runtime Environment
- What's new in the Java Card API

What’s new in the Java Card Runtime Environment

The following table outlines the new features in the Java Card Runtime Environment, Version 3.2.

Table New Features in Java Card Virtual Machine Specifications

Feature	Description
Logical Channel Configuration	A Java Card implementation must support one of the following configurations: • A configuration that provides support for multiple logical channels, either supporting Type 4 channel encoding in APDU commands, or supporting both Type 4 and Type 16 channel encodings. • A configuration that does not provide support for multiple logical channels and uses only the basic logical channel.

What's new in the Java Card API

The following table outlines the new features in the Java Card Application Programming Interface (API) specification version 3.2.

Table New Features in Java Card API Specification

New Feature	Description
Logical channel encoding	Add an API to retrieve the supported logical channel encoding type used by the JCRE .
TLS1.3 and DTLS1.3 key schedule	Add key derivation algorithm and intermediate message digest mechanisms to perform TLS1.3 and DTLS1.3 key schedule operations.
Additional ISO9796 digital signature with message recovery paddings	Support the trailer field option 2 for all schemes defined in the ISO9796 digital signature with message recovery specification.
Extend support to EdDSA digital signature algorithm	Add capability to create a signature instance for predefined edwards25519 and edwards448 curves.
Extend support to SM2 key agreement with confirmation values	Add support to SM2 key exchange with and without confirmation values.
Configure RSA-OAEP cipher scheme	Add the configuration of the message digest algorithm of the Mask Generation function (MGF1) of the RSA OAEP cipher scheme.
Configure RSA-PSS digital signature scheme	Add the configuration of the message digest algorithm of the Mask Generation function (MGF1) of the RSA PSS digital signature scheme.
Retrieve available memory value as a byte array	Add an API to retrieve the memory available for a given type as a convenient byte array parameter

Table (Cont.) New Features in Java Card API Specification

New Feature	Description
Instantiate random generator with external access	Add an API to instantiate a random generator instance that can be used even if the current context is not the context of the currently selected Applet.
Clear a biometric template	Add an API to clear a biometric template.

Detailed Changes

This topic provides comprehensive information about each change made in the specifications for this release.

The list below gives more details about the changes made in this release. They provide information on the following elements:

- **Component -** lists the specifications modified for this feature (Java Card Virtual Machine, Java Card Runtime Environment, Java Card API)
- **Compliance -** describes if the feature is mandatory or optional.
  - A ***mandatory*** feature must be supported by any implementation.
  - An ***optional*** feature is not necessarily supported. However, when the feature is supported, the proposed API, defined based on industry requirements, should be used instead of proprietary APIs to guarantee interoperability and avoid fragmentation.
- **API** - Lists the package or class that supports the feature.

Core – logical channel encoding

Support for logical channel encoding.

- **Component**: Java Card Runtime Environment – Logical Channels and Applet Selection Java Card Application Programming Interface
- **API**: `javacard.framework.APDU` class
- **Compliance**: Mandatory

The Java Card Runtime Environment must support one of the following configurations:

- A configuration where the Java Card Platform provides support for multiple logical channels. A terminal can open multiple logical channels over any I/O interface and assign each logical channel to an Applet instance. In this configuration, a platform may support the logical channels Type 4 encoding (logical channels 0 to 3) only or both the Type 4 and the Type 16 (logical channels 4 to 19) encodings.
- A configuration where the Java Card platform does not provide support for multiple logical channels. A terminal can only use the basic logical channel (logical channel 0) on each I/O interface and assign it to an Applet instance.

The Java card API provide means to retrieve the supported logical channel encoding of the platform:

New Constant referring to CLA encoding without logical channel information	`javacard.framework.APDU.LC_ENCODING_NO`
New Constant referring a CLA encoding with Type 4 logical channel information	`javacard.framework.APDU.LC_ENCODING_TYPE_4`
New Constant referring to CLA encoding with Type 4 and Type 16 logical channel information	`javacard.framework.APDU.LC_ENCODING_TYPE_4_TYPE_16`
New method to retrieve the logical channel encoding type used by the JCRE to interpret the CLA byte of the current APDU	`javacard.framework.APDU.getLogicalChannelEncoding()`

API - TLS1.3 and DTLS1.3

Support for the TLS1.3 and DTLS1.3 interfaces.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.security.derivation` and `javacard.security.MessageDigest Class`
- **Compliance**: Optional

The API for cryptography is extended to support the TLS1.3 (RFC 8446) and DTLS (RFC 9147) key schedule. For this purpose:

- A new derivation function algorithm is added to perform the HKDF-Expand-Label operation defined in RFC 8446 and to choose the label prefix either for TLS1.3 or for DTLS1.3.
- A new method is added to perform the transcript hash operation defined in RFC 8446: it is possible to perform an intermediate message digest calculation without changing the current state of the message digest instance and continue the operation.

The new classes, interfaces, methods or constants for this feature must be available in any Java Card 3.2 compliant implementation, but the corresponding algorithm implementation is optional and may throw an exception with the following reason code `CryptoException.NO_SUCH_ALGORITHM`.

New method to determine if intermediate hash calculation is supported	`javacard.security.MessageDigest.isIntermediateMessageDigestSupported()`
New method to generate an intermediate hash calculation	`javacard.security.MessageDigest.doIntermediateMessageDigest()`

	`javacard.security.MessageDigest.OneShot.doIntermediateMessageDigest()`
New constant for HKDF-Expand-Label algorithm	`javacardx.derivation.DerivationFunction.ALG_HKDF_EXPAND_LABEL_TLS13`
New interface for HKDF-Expand-Label algorithm	`javacardx.security.derivation.TLSKDFExpandLabelSpec`

API – ISO9796 signature with message recovery with trailer field option 2

Support for ISO9796 signature with message recovery.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.crypto.Cipher class` and `javacard.security.Signature class`
- **Compliance**: Optional

The API for cryptography is extended to support the following paddings for signatures with message recovery:

- padding specified by ISO/IEC 9796-2 for signature scheme 1, signature production function B.6 and giving message recovery with a trailer field option 2.
- padding specified by ISO/IEC 9796-2 for signature scheme 2, signature production function B.6 and giving message recovery with a trailer field option 2.
- padding specified by ISO/IEC 9796-2 for signature scheme 3, signature production function B.6 and giving message recovery with a trailer field option 2.

The new classes, interfaces, methods or constants for this feature must be available in any Java Card 3.2 compliant implementation, but the corresponding algorithm implementation is optional and may throw an exception with the following reason code `CryptoException.NO_SUCH_ALGORITHM`.

New constant for ISO9796 scheme 1 trailer field option 2	`javacardx.crypto.Cipher.PAD_ISO9796_MR_SCHEME_1_OPTION_2`
New constant for ISO9796 scheme 2 trailer field option 2	`javacardx.crypto.Cipher.PAD_ISO9796_MR_SCHEME_2_OPTION_2`
New constant for ISO9796 scheme 3 trailer field option 2	`javacardx.crypto.Cipher.PAD_ISO9796_MR_SCHEME_3_OPTION_2`

API – Extend support to EdDSA digital signature algorithm

Support for EdDSA digital signature algorithm.

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.security.Signature`
- **Compliance**: optional

The API for pure and pre-hash EdDSA signatures is extended to bind a signature instance to edwards25519 or edwards448 curves prior knowing the related key type.

The new classes, interfaces, methods or constants for this feature must be available in any Java Card 3.2 compliant implementation, but the corresponding algorithm implementation is optional and may throw an exception with the following reason code `CryptoException.NO_SUCH_ALGORITHM`.

New constant for pure EdDSA for the variant Ed25519	`javacard.security.Signature.SIG_CIPHER_EDDSA_ED25519`
New constant for pure EdDSA for the variant Ed448	`javacard.security.Signature.SIG_CIPHER_EDDSA_ED448`
New constant for pre-hash EdDSA for the variant Ed25519ph	`javacard.security.Signature.SIG_CIPHER_EDDSAPH_ED25519`
New constant for pre-hash EdDSA for the variant Ed448ph	`javacard.security.Signature.SIG_CIPHER_EDDSAPH_ED448`

API –Extend support to SM2 key agreement with confirmation value

Support for SM2 key agreements with confirmation value.

- **Component**: Java Card Application Programming Interface
- **API**: `javacardx.security` and `javacard.security.KeyAgreement`
- **Compliance**: Optional

The existing API for SM2 is extended to support the SM2 key agreement protocol with and without confirmation values and for both the initiator and the receiver roles.

The new classes, interfaces, methods or constants for this feature must be available in any Java Card 3.2 compliant implementation, but the corresponding algorithm implementation is optional and may throw an exception with the following reason code `CryptoException.NO_SUCH_ALGORITHM`.

New constant to perform an SM2 key agreement operation using confirmation values in and/or out	`javacard.security.KeyAgreement.ALG_SM2_WITH_CONFIRMATION`
New interface to configure the role and the parameters involved during an SM2 key agreement operation	`javacard.security.SM2KeyAgreementParameterSpec`
New method to initialize a key agreement instance based on algorithm parameters such as for SM2	`javacard.security.KeyAgreement.init()`

API – Configure RSA-OAEP cipher scheme

Configuration of RSA-OAEP cipher scheme.

The RSA-OAEP cipher scheme refers to a message digest algorithm for both the OAEP scheme itself and its underlying mask generation function (MGF1). The API is extended to configure independently the message digest algorithm of the scheme and the message digest algorithm of the MGF1.

The new classes, interfaces, methods or constants for this feature must be available in any Java Card 3.2 compliant implementation, but the corresponding algorithm implementation is optional and may throw an exception with the following reason code `CryptoException.NO_SUCH_ALGORITHM`.

The following table lists the method and interfaces added to configure parameters.

New constant allowing to configure OAEP parameters within Cipher.init()	`javacardx.crypto.PAD_PKCS1_OAEP_EXT_PARAMETERS`

API – Configure RSA-PSS digital signature scheme

Configuration for RSA-PSS digital signature scheme

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.security.Signature` and `javacardx.crypto.Cipher`
- **Compliance**: Optional

The RSA-PSS digital signature scheme refers to a message digest algorithm for both the PSS scheme itself and its underlying mask generation function (MGF1). The API is extended to configure independently the message digest algorithm of the scheme and the message digest algorithm of the MGF1 as well as the salt length.

The new classes, interfaces, methods or constants for this feature must be available in any Java Card 3.2 compliant implementation, but the corresponding algorithm implementation is optional and may throw an exception with the following reason code `CryptoException.NO_SUCH_ALGORITHM`.

New constant allowing to configure PSS parameters within Signature.init()	`javacardx.crypto.PAD_PKCS1_PSS_EXT_PARAMETERS`

API – Retrieve available memory value as byte array

Retrieve available memory as byte array.

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.framework.JCSystem`
- **Compliance**: Mandatory

Two methods already exist to retrieve the memory available for a given type either as a short or in a short[]. The API is extended to also retrieve the value as a byte[].

New method to retrieve memory available as a byte array	`javacard.framework.JCSystem.getAvailableMemory()`

API – Instantiate random generator with external access

Support for instantiating random generator with external access.

- **Component**: Java Card Application Programming Interface
- **API**: `javacard.security.RandomData class`
- **Compliance**: Optional

As for any other cryptographic objects, the random generator API is extended to request explicitly a RandomData instance with external access. Such an instance can be shared among multiple applet instances and/or can also be accessed (via a Shareable interface) when the owner of the RandomData instance is not the currently selected applet.

The previous method RandomData.getInstance() is deprecated.

The new classes, interfaces, methods, and constants for this feature must be available in any Java Card 3.2 compliant implementation, but the corresponding algorithm implementation is optional and may throw an exception with the following reason code `CryptoExcpetion.NO_SUCH_ALGORITHM` reason code.

New method to instantiate a RandomData object with external access	`javacard.security.RandomData.getInstance()`

API – Clear a biometic template

Support for clearing a biometric template.

The interfaces referring to a biometric template owned by an applet are extended to offer the possibility to clear the biometric template. Doing so, its state becomes uninitialized.

New method to clear a biometric template in owned by an applet	`javacardx.biometry.OwnerBioTemplate.clear()`
New method to clear a biometric template owned by an applet (1:N biometric framework)	`javacardx.biometry1toN.OwnerBioTemplateData.clear()`

Clarifications

Additional information on clarifications and fixes.

This release contains the following clarifications and fixes:

3.1 Java Card Platform Virtual Machine Specification, Classic Edition, Version 3.2

3.1.1 6.16 Static Resource Component

- Fix typo: `static_resource` replaced by `static_resource_info`

3.2 Java Card API Specification, Classic Edition, Version 3.2

3.2.1 javacard.framework.MultiSelectable interface

- Method boolean select(boolean appInstAlreadyActive) – “package” replaced by “group context”. Clarify the behavior when multiple I/O interfaces are supported.
- Method void deselect(boolean appInstStillActive) – “package” replaced by “group context”. Clarify the behavior when multiple I/O interfaces are supported.

3.2.2 javacard.framework.APDU class

- Method byte getCLAChannel() – Clarify that the return value is based on the logical channel encoding type returned by APDU.getLogicalChannelEncoding().
- Method boolean isSecureMessagingCLA() – Clarify that the return value is based on the logical channel encoding type returned by APDU.getLogicalChannelEncoding().
- Method void setOutgoingLength(short len) – Clarify when APDUException.NO_T0_GETRESPONSE and APDUException.NO_T0_REISSUE are thrown when multiple I/O interfaces are supported.
- Method void sendBytesLong(byte[] outData, short bOff, short len) – Clarify when APDUException.NO_T0_GETRESPONSE is thrown when multiple I/O interfaces are supported.

3.2.3 javacard.framework.Applet class

- Method boolean select() – “package” replaced by “group context”. Clarify the behavior when multiple I/O interfaces are supported.
- Method void deselect() – “package” replaced by “group context”. Clarify the behavior when multiple I/O interfaces are supported.

3.2.4 javacard.framework.JCSystem class

- Method Shareable getAppletShareableInterfaceObject(AID serverAID, byte parameter) – Clarify the behavior when multiple I/O interfaces are supported.
- Method static boolean isAppletActive(AID theApplet) – Clarify the behavior when multiple I/O interfaces are supported.

3.2.5 javacard.framework.Resources class

- Method Resources getResources() – Clarify the returned instance is a permanent JCRE Entry Point Object that can be accessed from any applet context.
- Method byte[] getView(short resourceId, short ofs, short len) – Clarify that an IOException is thrown when the value offset + length exceeds the size returned by getSize().

3.2.6 javacard.security.ECKey interface

- Method void setK(short K) – Clarify that the value “1” must be supported by any implementation. A CryptoException.ILLEGAL_VALUE can be thrown when the cofactor value K is different from 1 and not supported by the platform.

3.2.7 javacard.security.GenericSecretKey interface

- Interface description – Clarify that the key can be of any length aligned on a multiple of 8 bits.

3.2.8 javacard.security.HMACKey interface

- Interface description – Clarify that the key can be of any length aligned on a multiple of 8 bits.

3.2.9 javacard.security.Key interface

- Method byte getType() – Clarify that KeyBuilder.TYPE_XEC is returned for keys implementing XECKey interface. Clarify that “0” is returned for keys for which the type is not one of the pre-defined algorithms.

3.2.10 Javacard.security.SignatureMessageRecovery interface

- Interface description – Clarify the list of algorithms used by Signature.getInstance(byte, boolean) and Signature.getInstance(byte, byte, byte, boolean) that return an instance implementing this interface.

3.2.11 javacard.security.InitializedMessageDigest class

- Method void setInitialDigest(byte[] state, short stateOffset, short stateLength, byte[] digestedMsgLenBuf, short digestedMsgLenOffset, short digestedMsgLenLength) – Correct typo “initalDigestLength” into “stateLength” when describing the CryptoException.ILLEGAL_VALUE case.

3.2.12 javacard.security.InitializedMessageDigest.OneShot class

- Method void setInitialDigest(…) – align parameters names with the same method described into the InitializedMessageDigest class.

3.2.13 javacard.security.KeyAgreement class

- Class description – Clarify preserved (e.g. key) and unpreserved instance values after a reset or a tear down.
- Constants ALG_EC_SVDP_DH, ALG_EC_SVDP_DHC – Clarify that those constants are deprecated and should be replaced respectively by ALG_EC_SVDP_DH_KDF and ALG_EC_SVDP_DHC_KDF.
- Constant ALG_SM2 – Clarify that it requires additional algorithm parameters and the use of the init(PrivateKey, AlgoritmParameterSpec) method.
- Method void init(PrivateKey privKey) – Clarify CryptoException.ILLEGAL_VALUE is thrown if KeyAgreement algorithm requires additional algorithm parameters.

- Method generateSecret(byte[] publicData, short publicOffset, short publicLength, byte[] secret, short secretOffset) – Clarify the preserved (e.g. key) and unpreserved instance values after generateSecret operation. CryptoException.UNINITIALIZED_KEY is thrown if the PrivateKey or any other key passed in additional parameters is not initialized. CryptoException.ILLEGAL_VALUE is thrown if the publicData data is inconsistent with the additional algorithm parameters passed in init(PrivateKey, AlgoritmParameterSpec). Clarify the return value and value length when ALG_SM2_WITH_CONFIRMATION algorithm is used.

3.2.14 javacard.security.KeyPair class

- Method void genKeyPair() – Clarify that, for DSA algorithm, default precomputed p, q and g parameters may be used by the platform if not provided by the application. It aligns with the Elliptic curve case.

3.2.15 javacard.security.RandomData class

- Method RandomData getInstance(byte algorithm) – This method is deprecated. getInstance(byte algorithm, boolean externalAccess) should be used instead.

3.2.16 javacard.security.Signature class

- Class description – Clarify the preserved (e.g. key and mode) and unpreserved instance values after a reset or a tear down.
- Constants ALG_RSA_SHA_ISO9796, ALG_RSA_RIPEMD160_ISO9796, ALG_RSA_SHA_ISO9796_MR, ALG_RSA_RIPEMD160_ISO9796_MR – Clarify constants refer to the scheme 1, the signature production function B.6 and trailer field option 1 of the ISO9796 specification.
- Constants ALG_RSA_SHA_PKCS1_PSS, ALG_RSA_MD5_PKCS1_PSS, ALG_RSA_RIPEMD160_PKCS1_PSS, ALG_RSA_SHA_224_PKCS1_PSS, ALG_RSA_SHA_256_PKCS1_PSS, ALG_RSA_SHA_384_PKCS1_PSS, ALG_RSA_SHA_512_PKCS1_PSS – Clarify the default salt length, the scheme digest algorithm and the MGF1 digest algorithm.
- Constants SIG_CIPHER_EDDSA, SIG_CIPHER_EDDSAPH – Clarify to which edDSA variants those constants refer to as per the RFC 8032. Clarify that this cipher algorithm must be associated with message digest algorithm MessageDigest.ALG_NULL and the padding algorithm Cipher.PAD_NULL when calling getInstance(byte, byte, byte, boolean).
- Method void init(Key theKey, byte theMode) – Remove the note about the default salt length value in case of RSA-PSS which is now decribed in the related RSA-PSS constants. Remove the note about “optimal performance”.
- Method init(Key theKey, byte theMode, byte[] bArray, short bOff, short bLen) – Remove the note about “optimal performance”. Clarify the expected behavior for managing RSA PSS and EDDSA Java Card Platform Specification Release Notes, v3.2 Page 23 parameters. CryptoException.NO_SUCH_ALGORITHM can be thrown if the parameters referenced in the byte array parameter are not supported in case of PAD_PKCS1_PSS_EXT_PARAMETERS padding.
- Method short getLength() – Clarify that for DSA and ECDSA, the returned length must be the maximum possible length of the related ASN.1 sequence.

- Method void setInitialDigest(…) – align parameters names with the same method described into the InitializedMessageDigest class.
- Method short sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset) – Clarify the preserved (e.g. key and mode) and unpreserved instance values after sign operation.
- Method short signPreComputedHash(byte[] hashBuff, short hashOffset, short hashLength, byte[] sigBuff, short sigOffset) – Clarify the preserved (e.g. key and mode) and unpreserved instance values after signPreComputedHash operation. Clarify CryptoException.ILLEGAL_USE is thrown for edDSA algorithm.
- Method boolean verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength) – Clarify the preserved (e.g. key and mode) and unpreserved instance values after verify operation.
- Method boolean verifyPreComputedHash(byte[] hashBuff, short hashOffset, short hashLength, byte[] sigBuff, short sigOffset, short sigLength) – Clarify the preserved (e.g. key and mode) and unpreserved instance values after verifyPreComputedHash operation. Clarify CryptoException.ILLEGAL_USE is thrown for edDSA algorithm.

3.2.17 javacard.security.Signature.OneShot class

- Method void setInitialDigest(…) – align parameter names with the same method described into the InitializedMessageDigest class.

3.2.18 javacardx.apdu.util.APDUUTIL class

- Method byte getCLAChannel(byte CLAbyte) – Clarify that this utility method always refer to the type APDU.LC_ENCODING_TYPE_4_TYPE_16.

3.2.19 javacardx.biometry.OwnerBioTemplate interface

- Method short initMatch(byte[] candidate, short offset, short length) – Clarify that the matching session ends in failed state if an exception is thrown during the match.
- Method short match(byte[] candidate, short offset, short length) – Clarify that the matching session ends in failed state if an exception is thrown during the match.

3.2.20 javacardx.biometry1toN.BioMatcher interface

- Method BioTemplateData getBioTemplateData(short index) – Clarify that template indexing starts at value « 1 » (« 0 » is an invalid value).
- Method short getIndexOfLastMatchingBioTemplateData() – Clarify that template indexing starts at value « 1 » (« 0 » is an invalid value).
- Method short initMatch(byte[] candidate, short offset, short length) – Clarify the matching session ends in failed state if an exception is thrown during the match. Also clarify that if the matching session ends, the validated flag remains in the reset state.
- Method short match(byte[] candidate, short offset, short length) – Clarify the matching session ends in failed state if an exception is thrown during the match. Also clarify that the matching session must ignore any enrolled BioTemplateData that is uninitialized. Bio1toNException.NO_BIO_TEMPLATE_ENROLLED is thrown if none of the enrolled BioTemplateData is initialized.

3.2.21 javacardx.biometry1toN.OwnerBioMatcher interface

- Method OwnerBioTemplateData getBioTemplateData(short index) – Clarify that template indexing starts at value « 1 » (« 0 » is an invalid value).
- Method short getIndexOfLastMatchingBioTemplateData() – Clarify that template indexing starts at value « 1 » (« 0 » is an invalid value).
- Method void putBioTemplateData(short index, BioTemplateData templateData) – Clarify that template indexing starts at value « 1 » (« 0 » is an invalid value).

3.2.22 javacardx.crypto.AEADCipher class

- Class description – Clarify the preserved (e.g. key and mode) and unpreserved instance values after a reset or a tear down.
- Method void init(Key theKey, byte theMode) – Remove references to non-AEAD algorithms. Clarify the nonce is a 12 bytes buffer full of zeroes for GCM mode.
- Method void init(Key theKey, byte theMode, byte[] bArray, short bOff, short bLen) – Remove references to non-AEAD algorithms.
- Method short update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) – Clarify the expected output length based on the algorithm. Add reference to array views case when input/output buffers overlap. Remove references to non-AEAD algorithms.
- Method short doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) – Clarify the preserved (e.g. key and mode) and unpreserved instance values after doFinal operation. Remove references to non-AEAD algorithms.
- Method short retrieveTag(byte[] tagBuf, short tagOff, short tagLen) – Clarify CryptoException.ILLEGAL_USE is thrown if the mode is not MODE_ENCRYPT. Clarify CryptoException.ILLEGAL_VALUE if the tag length is not equal to the generated tag length e.g the length passed in init() method for ALD_AES_CCM.
- Method boolean verifyTag(byte[] receivedTagBuf, short receivedTagOff, short receivedTagLen, short requiredTagLen) – Fix typos: “bits” renamed in “bytes”. Clarify that if recievedTagLen and requiredTagLen are valid values and recievedTagLen < requiredTagLen, the method returns false. Clarify CryptoException.ILLEGAL_USE is thrown if the mode is not MODE_DECRYPT.

3.2.23 javacardx.crypto.Cipher class

- Class description – Clarify the preserved (e.g. key and mode) and unpreserved instance values after a reset or a tear down.
- Constants PAD_ISO9796, PAD_ISO9796_MR, PAD_ISO9796_MR_SCHEME_2, PAD_ISO9796_MR_SCHEME_3 – Clarify that these constants refer to the signature production function B.6 and trailer field option 1 of the ISO9796 specification.
- Constant PAD_PKCS1_PSS – Clarify default salt length, scheme digest algorithm and MGF1 digest algorithm.
- Constants ALG_RSA_PKCS1_OAEP, PAD_PKCS1_OAEP, PAD_PKCS1_OAEP_SHA224, PAD_PKCS1_OAEP_SHA256, PAD_PKCS1_OAEP_SHA384, PAD_PKCS1_OAEP_SHA512, PAD_PKCS1_OAEP_SHA3_224, PAD_PKCS1_OAEP_SHA3_256,

PAD_PKCS1_OAEP_SHA3_384, PAD_PKCS1_OAEP_SHA3_512 – Clarify the scheme digest algorithm and the MGF1 digest algorithm.

- Method void init(Key theKey, byte theMode) – Remove the note about "optimal performance" and add reference to SM4.
- Method void init(Key theKey, byte theMode, byte[] bArray, short bOff, short bLen) – Remove the note about "optimal performance". Clarify the input array format for AES XTS mode. Clarify the input array format for SM4 CBC mode. Clarify the expected behavior for managing RSA OAEP parameters. CryptoException.NO_SUCH_ALGORITHM can be thrown if the parameters referenced in the byte array parameter are not supported in case of PAD_PKCS1_OAEP_EXT_PARAMETERS padding.
- Method short update(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) – Clarify the expected output length based on the algorithm. Add reference to array views case when input/output buffers overlap.
- Method short doFinal(byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) – Clarify preserved (e.g. key and mode) and unpreserved instance values after doFinal operation.

3.2.24 javacardx.security.derivation.DerivationFunction class

- Class description – Clarify that after a reset or a tear down, the object state is reset, like after a call to one of the nextBytes() methods, and has to be initialized again using the init(AlgorithmParameterSpec) method.

Supported Platforms

The Java Card specification documents are accessible on any computer system with an Unzip utility, Adobe Acrobat Reader (version 4.0 or later), and a CSS-compliant web browser.

View the HTML files using any of the following CSS-compliant browsers:

- Internet Explorer, version 5.0 or later.
- Mozilla Firefox, version 11.0 or later.

View the PDF files in your web browser with an appropriate plugin or in the Adobe® Acrobat Reader. Most recent browsers include the PDF reader plugin. However, if your browser doesn't have one, then download the plugin from the Install Adobe Acrobat Reader website.

Downloading the Specification Documents

Perform the following steps to download the specifications:

1. Download the specification bundle from the Java Card Technology web site.
2. Unzip the bundle.
3. Browse to the `javacard_specifications-3_2/classic` folder.

   The `classic` directory has the following sub folders:

- `api_classic`: Contains the Java Card API specification for the Classic Edition, Version 3.2 in the Javadoc^{TM} tool HTML format. Use the available browsers to view the APIs. However, the APIs might not render well in Mozilla Firefox, version 3.0.10.
- `jcre_classic`: Contains the Java Card Runtime Environment specification for the Classic Edition, Version 3.2 in the PDF format (JCREspecCLASSIC-3_2.pdf).
- `jcvm_classic`: Contains the Java Card Virtual Machine specification for the Classic Edition, version 3.2 in the PDF format (JCVMspecCLASSIC_3_2.pdf).

Known Issues

There are no known issues in this release of Java Card specifications.

Product Information

The Java Card Technology website provides useful information about the Java Card product.

Visit the Java Card Technology website to access the most up-to-date information on the following:

- Product news and reviews
- Release notes and product documentation

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 visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Java Card Platform Specification Release Notes, Version 3.2
F74152-01

Release notes for Java Card Platform Specifications, Version 3.2.