RSA BSAFE® Crypto-J JSAFE and JCE Software Module 5.0 Security Policy Level 1 Roles, Authentication and Services

This document is a non-proprietary security policy for RSA BSAFE Crypto-J JSAFE and JCE Software Module 5.0 (Crypto-J) security software.

Contents:

Preface

This document is a non-proprietary security policy for the Crypto-J cryptographic toolkit from RSA, the Security Division of EMC (RSA).

This security policy describes how the Crypto-J toolkit meets the Level 1 FIPS 140-2 Security Level 1 validation of the Crypto-J toolkit.

Crypto-J provides both the JSAFE and JCE Application Programming Interfaces (APIs), in the `cryptojFIPS.jar` file. All references to the Crypto-J toolkit apply to both interfaces unless explicitly noted.

FIPS 140-2 (Federal Information Processing Standards Publication 140-2 - Security Requirements for Cryptographic Modules) details the U.S. Government requirements for cryptographic modules. More information about the FIPS 140-2 standard and validation program is available on the `NIST website`.

References

This document deals only with operations and capabilities of the Crypto-J in the technical terms of a FIPS 140-2 cryptographic toolkit security policy. More information on Crypto-J and the entire RSA BSAFE product line is available at:

- `http://www.rsa.com/`, for information on the full line of products and services.
- `http://www.rsa.com/node.aspx?id=1319` for an overview of security tools for Java developers.
- `http://www.rsa.com/node.aspx?id=1204` for an overview of the RSA BSAFE product range.

Terminology

In this document, the term *Crypto-J Module* denotes the Crypto-J 140-2 Security Level 1 validated Cryptographic Module.

The *Crypto-J Module* is also referred to as:

- The Cryptographic Module
- The module.

Document Organization

This Security Policy document is one document in the 140-2 Security Level 1 Validation Submission package. With the exception of the Non-Proprietary Crypto-J Security Policy, the 140-2 Security Level 1 Validation Submission Documentation is RSA Security-proprietary and is releasable only under appropriate non-disclosure agreements. For access to the documentation, please contact RSA Security.

This document explains the Crypto-J features and functionality relevant to FIPS 140-2, and contains the following sections:

- This section, "Preface" on page 2 provides an overview and introduction to the Security Policy.
- "Crypto-J Cryptographic Toolkit" on page 4, describes Crypto-J and how it meets the 140-2 Security Level 1 requirements.
- "Secure Operation of Crypto-J" on page 19, addresses the required configuration for the FIPS140-mode of operation.
- "Acronyms" on page 26, lists the definitions for the acronyms used in this document.

1 Crypto-J Cryptographic Toolkit

This section provides an overview of the Crypto-J toolkit, and contains the following topics:

1.1 Introduction

More than a billion copies of the RSA BSAFE technology are embedded in today's most popular software applications and hardware devices. Encompassing the most widely-used and rich set of cryptographic algorithms as well as secure communications protocols, RSA BSAFE software is a set of complementary security products relied on by developers and manufacturers worldwide.

The Crypto-J software library is the world's most trusted Java-language cryptography component, and is at the heart of the RSA BSAFE product line. It includes a wide range of data encryption and signing algorithms, including AES, Triple-DES, RC5, the RSA Public Key Cryptosystem, the Elliptic Curve Cryptosystem, the DSA government signature algorithm, and the SHA1 and SHA2 message digest routines. Its software libraries, sample code and complete standards-based implementation enable near-universal interoperability for your networked and e-business applications. Any programmer using the RSA BSAFE Crypto-J tools can easily create secure applications without a background in cryptography, mathematics or number theory.

1.2 Toolkit Characteristics

Crypto-J is classified as a FIPS 140-2 multi-chip standalone module. As such, Crypto-J is tested on particular operating systems and computer platforms. The cryptographic boundary includes Crypto-J running on selected platforms that are running selected operating systems.

Crypto-J is validated for FIPS 140-2 Security Level 1 requirements. Crypto-J is packaged in a Java Archive (JAR) file containing all the code for the toolkit. In addition, Crypto-J relies on the physical security provided by the host on which it runs.

Both the JSAFE and JCE APIs of the Crypto-J toolkit are provided in the `cryptojFIPS.jar` file.

Crypto-J is tested on the following platforms:

- Microsoft Windows XP SP3 (32-bit) with Sun™ JRE™ 5.0
- Microsoft Windows XP SP3 (32-bit) with Sun JRE 6.0.

Compliance is maintained on platforms for which the binary executable remains unchanged. This includes (but is not limited to):

- Microsoft
  - Windows 2000 Professional, SP4, Sun JRE 5.0/6.0, IBM® JRE 5.0
  - Windows XP SP2, Sun JRE 5.0/6.0, IBM JRE 5.0, JRockit 5.0/6.0
  - Windows XP Professional (64-bit), Sun JRE 5.0/6.0
  - Windows 2003 Server (32-bit), Sun JRE 5.0/6.0, IBM JRE 5.0, JRockit 5.0/6.0
  - Windows 2003 Server (64-bit), Sun JRE 5.0/6.0, JRockit 5.0/6.0
  - Windows 2008 Server (32-bit), Sun JRE 6.0
  - Windows 2008 Server (64-bit), Sun JRE 6.0
  - Windows Vista® (32-bit), Sun JRE 5.0/6.0, IBM JRE 5.0, JRockit 5.0/6.0
  - Windows Vista (64-bit), Sun JRE 5.0/6.0, JRockit 5.0/6.0.
- Sun
  - Solaris™ 9, UltraSparc v8+ (32-bit), Sun JRE 5.0/6.0, IBM JRE 5.0
  - Solaris 9, UltraSparc v9 (64-bit), Sun JRE 5.0/6.0
  - Solaris 10, UltraSparc v8+ (32-bit), Sun JRE 5.0/6.0, IBM JRE 5.0
  - Solaris 10, UltraSparc v9 (64-bit), Sun JRE 5.0/6.0, IBM JRE 5.0, JRockit 5.0/6.0
  - Solaris 10, x86 (64-bit), Sun JRE 5.0/6.0.

- Linux®
  - Red Hat® Enterprise Linux AS 4.0, x86 (32-bit), Sun JRE 5.0/6.0, IBM JRE 5.0, JRockit 5.0/6.0
  - Red Hat Enterprise Linux AS 4.0, x86 (64-bit), Sun JRE 5.0/6.0, JRockit 5.0/6.0
  - Red Hat Enterprise Linux AS 5.0, x86 (32-bit), Sun JRE 5.0/6.0, IBM JRE 5.0, JRockit 5.0/6.0
  - Red Hat Enterprise Linux AS 5.0, x86 (64-bit), Sun JRE 5.0/6.0, JRockit 5.0/6.0
  - Novell® SUSE® Linux Enterprise Server 9, x86 (32-bit), Sun JRE 5.0/6.0
  - Novell SUSE Linux Enterprise Server 9, x86 (64-bit), Sun JRE 5.0/6.0
  - Novell SUSE Linux Enterprise Server 10, x86 (32-bit), Sun JRE 5.0/6.0
  - Novell SUSE Linux Enterprise Server 10, x86 (64-bit), Sun JRE 5.0/6.0.
- HP
  - HP-UX 11.23, Itanium 2 (32-bit), HP JRE 5.0/6.0
  - HP-UX 11.23, Itanium 2 (64-bit), HP JRE 5.0/6.0
  - HP-UX 11.31, Itanium 2 (32-bit), HP JRE 5.0/6.0
  - HP-UX 11.31, Itanium 2 (64-bit), HP JRE 5.0/6.0.
- IBM
  - AIX 5L™ v5.3, Power PC® (32-bit), IBM JRE 5.0/6.0
  - AIX 5L v5.3, Power PC (64-bit), IBM JRE 5.0/6.0
  - AIX 5L v6.1, Power PC (32-bit), IBM JRE 5.0/6.0
  - AIX 5L v6.1, Power PC (64-bit), IBM JRE 5.0/6.0.

For a resolution on the issue of multi-user modes, see the NIST document Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program.

1.3 Toolkit Interfaces

As a multi-chip standalone toolkit, the physical interface to Crypto-J consists of a keyboard, mouse, monitor, serial ports and network adapters.

The underlying logical interface to the toolkit is the API, documented in the *RSA BSAFE Crypto-J Developer's Guide*. The Crypto-J toolkit provides for Control Input through the API calls. Data Input and Output are provided in the variables passed with API calls, and Status Output is provided in the returns and error codes documented for each call. This is shown in the following diagram.

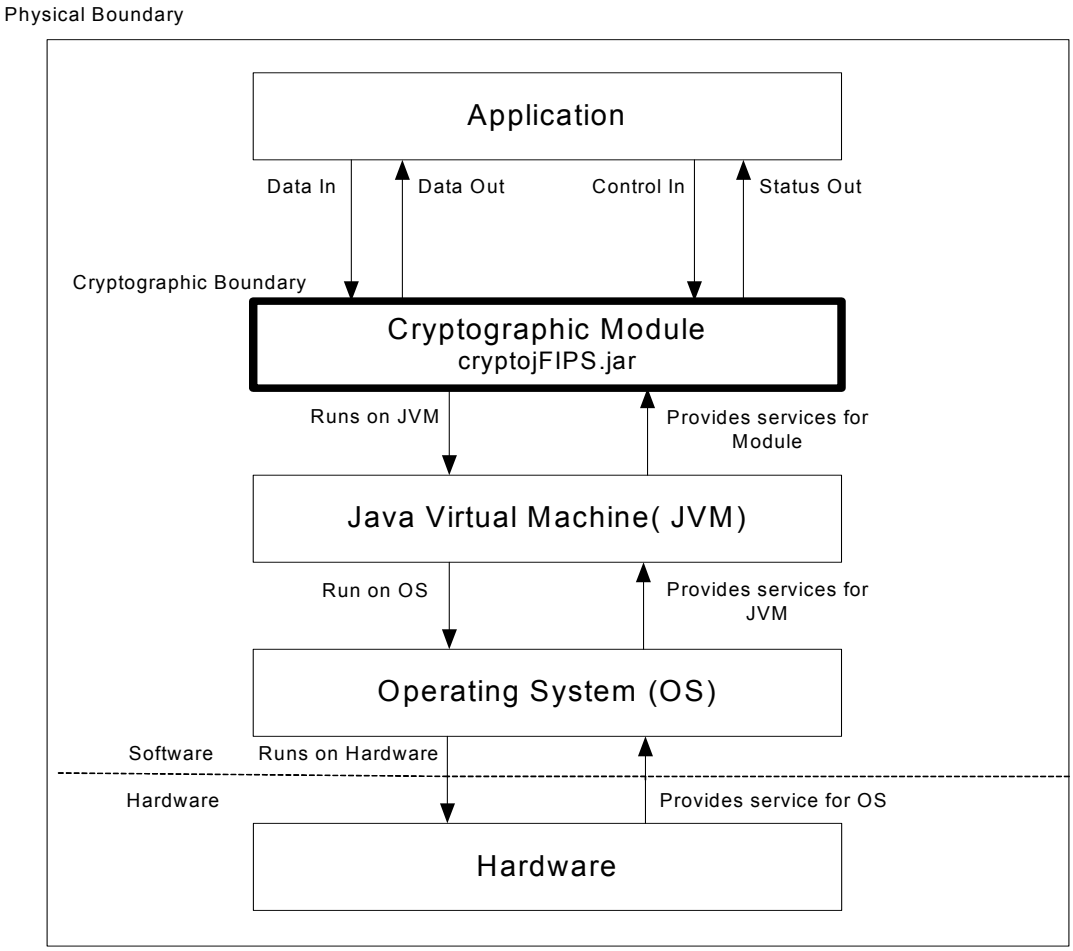

Figure 1 Crypto-J Logical Diagram

1.4 Roles and Services

Crypto-J meets all FIPS140-2 Level 1 requirements, implementing both a Crypto Officer role and a Crypto User role. As allowed by FIPS 140-2, Crypto-J does not require user identification or authentication for these roles.

The API for control of Crypto-J is through the `com.rsa.jsafe.crypto.CryptoJ` class. The API is duplicated in the class `com.rsa.jsafe.CryptoJ`.

1.4.1 Crypto Officer Role

An operator can assume the Crypto Officer Role by invoking the `com.rsa.jsafe.crypto.CryptoJ.setRole()` method with the `CryptoJ.CRYPTO_OFFICER_ROLE` argument. After this invocation, services available to the Crypto Officer Role can be used.

An operator can also assume the Crypto Officer Role by constructing a `FIPS140Context` object where the role is specified as `CryptoJ.CRYPTO_OFFICER_ROLE`. The `FIPS140Context` object can then be input to a Service which is to be used by the Crypto Officer Role.

The Services section provides a list of services available to the Crypto Officer Role.

1.4.2 Crypto User Role

The Crypto User Role is the default operating role for the module. An operator can explicitly assume the Crypto User Role by invoking the `com.rsa.jsafe.crypto.CryptoJ.setRole()` method with the `CryptoJ.USER_ROLE` argument. Once the role is set to Crypto User Role, either by default or explicitly, services available to the Crypto User Role can be used.

An operator can also assume the Crypto User Role by constructing a `FIPS140Context` object where the role is specified as `CryptoJ.USER_ROLE`.The `FIPS140Context` object can then be input to a Service which is to be used by the Crypto User Role.

The Services section provides a list of services available to the Crypto User Role.

1.4.3 Services

The following table lists the un-authenticated services provided by Crypto-J which may be used by either Role in terms of the toolkit interface.

Table 1 Services Available to the Crypto User and Crypto Officer Roles

Services Available to the Crypto User and Crypto Officer Roles	
`CryptoJ.getFIPS140Context`	`CryptoJ.getSeeder`
`CryptoJ.setSeeder`	`CryptoJ.getDefaultRandom`

Table 1 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
CryptoJ.isFIPS140Compliant	CryptoJ.getState
CryptoJ.getMode	CryptoJ.setMode
CryptoJ.getRole	CryptoJ.setRole
CryptoJ.disableLibrary	CryptoJ.selfTestPassed
CryptoJ.runSelfTests	CryptoJ.isInFIPS140Mode
CryptoJ.isInSSLMode	CryptoJ.isInECCMode
CryptoJ.notInFIPS140Mode	CryptoJ.checkIntegrity
CryptoJ.getFIPS140SecurityLevel	CryptoJ.isFIPS140SecurityLevelOne
CryptoJ.isFIPS140SecurityLevelTwo	CryptoJ.getFIPS140ModuleName
CryptoJ.fips186RandomClearQ	CryptoJ.fips186RandomSetQ
CryptoJ.verifyPQGParams	AccessDescription
AlgorithmParameterGenerator	AlgorithmParameters
ArchiveEncryptedKey	ArchiveEncryptedKeyLegacy
ArchiveFromParameters	ArchiveGeneratedPrivKey
ArchiveOptions	Assurance
Attribute	AuthorityKeyIdentifier
CCMParameterSpec	CertComplianceAdjustment
CertConfirmationMessage	CertConfirmationMessageImpl
CertCreationFactory	CertCreationParameterSpec
CertificateFactory	CertJVersion
CertPathBuilder	CertPathValidator
CertPathWithOCSPParameters	CertRequest
CertRequestFactory	CertRequestMessage
CertRequestMessageImpl	CertTemplateSpec
Cipher	CMPController
CMPServerConfig	ControlsSpec
CRLCreationFactory	CRLCreationParameterSpec
CRMFParameterSpec	CRMFProofGenerationParams

Table 1 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
CryptoJVersion	CryptoJVersion
DistributionPoint	DistributionPointName
DRBGInstantiationParameterSpec	DRBGOperationalParameterSpec
DSAGenParameterSpec	ECIESParameterSpec
EncryptedValue	EVChecker
FIPS140Context	GCMParameterSpec
GeneralName	GeneralSubtree
HttpCMPServerConfig	IssuerInformation
IssuingDistributionPoint	JSAFE_AsymmetricCipher
JSAFE_Key	JSAFE_KeyAgree
JSAFE_KeyAttributes	JSAFE_KeyPair
JSAFE_KeyWrapCipher	JSAFE_MAC
JSAFE_MessageDigest	JSAFE_Obfuscator
JSAFE_Object	JSAFE_Parameters
JSAFE_PrivateKey	JSAFE_PublicKey
JSAFE_Recode	JSAFE_SecretKey
JSAFE_SecureRandom	JSAFE_Session
JSAFE_SessionSpec	JSAFE_Signature
JSAFE_SymmetricCipher	JSAFE_VerifyPQG
JsafeJCE	KeyAgreement
KeyFactory	KeyGenerator
KeyIdentifier	KeyPairGenerator
KeyStore	LDAPCertStoreParameters
Mac	MACProtection
MessageDigest	MessageHeaderInfo
MessageProtection	ObjectID
OCSP	OCSPParameters
OCSPResponderConfig	OCSPWithRespondersParameters

Table 1 Services Available to the Crypto User and Crypto Officer Roles (continued)

Services Available to the Crypto User and Crypto Officer Roles	
OCSPWithResponseParameters	PKCS10ParameterSpec
PKCS10SigningParameters	PolicyChecker
POPEncryptedKey	POPEncryptedKeyLegacy
POPPrivateKeySpec	POPRAVerifiedSpec
POPSigningKeySpec	POPSubsequentMessage
ProofGenerationParameters	ProofOfPossessionSpec
PSSParameterSpec	RegInfoSpec
RevocationDetails	RevocationRequestMessage
RevocationRequestMessageImpl	RevokedCertificate
RSAGenParameterSpec	RSAJCP
SecureRandom	SecureRandomEx
SensitiveData	Signature
SignatureProtection	StatusInfo
SuiteBChecker	TcpCMPServerConfig
ValidateEncryptedKeyParams	ValidateParameters
ValidateSignKeyParams	X500PrincipalBuilder
X509CRLEntryExtensionSpec	X509CRLExtensionSpec
X509ExtensionRequestSpec	X509ExtensionSpec
X509V1BuilderParameters	X509V1BuilderResult
X509V1ValidatorParameters	X509V1ValidatorResult
X942DHParameterSpec	X942DHPrivateKeySpec
X942DHPublicKeySpec	XTSParameterSpec

For more information on each function, see the *RSA BSAFE Crypto-J Developer's Guide*.

1.5 Cryptographic Key Management

1.5.1 Key Generation

The Crypto-J toolkit supports the generation of the DSA, RSA, and Diffie-Hellman (DH) and ECC public and private keys. The toolkit also employs a Federal Information Processing Standard 186-2, Digital Signature Standard (FIPS 186-3) Approved Random Number Generator, a FIPS Approved HMAC Deterministic Random Bit Generator (HMAC DRBG SP800-90), as well as a FIPS Approved Dual Elliptic Curve Deterministic Random Bit Generator (ECDRBG SP 800-90) for generating asymmetric and symmetric keys used in algorithms such as AES, Triple-DES, RSA, DSA, DH and ECC.

1.5.2 Key Protection

All key data resides in internally allocated data structures and can only be output using the Crypto-J API. The operating system and the Java Runtime Environment (JRE) safeguards memory and process space from unauthorized access.

1.5.3 Key Access

An authorized operator of the module has access to all key data created during Crypto-J operation.

Note: The User and Officer roles have equal and complete access to all keys.

The following table lists the different services provided by the toolkit with the type of access to keys or CSPs.

Table 2 Key and CSP Access

Service	Key or CSP	Type of Access
Encryption and decryption	Symmetric keys (AES, Triple-DES)	Read/Execute
Digital signature and verification	Asymmetric keys (DSA, RSA, ECDSA)	Read/Execute
Hashing	None	N/A
MAC	HMAC keys	Read/Execute
Random number generation	FIPS 186-2 seed and key X931Random seed and number of streams HMAC DRBG entropy, strength, and seed EC DRBG entropy, strength, and seed	Read/Write/Execute
Key establishment primitives	Asymmetric keys (RSA, DH, ECDH)	Read/Execute

Table 2 Key and CSP Access (continued)

Service	Key or CSP	Type of Access
Key generation	Symmetric keys (AES, Triple-DES) Asymmetric keys (DSA, EC DSA, RSA, DH, ECDH) MAC keys (HMAC)	Write
Self-test	Hard coded keys (AES, Triple-DES, RSA, DSA, ECDSA and HMAC)	Read/Execute
Show status	None	N/A
Zeroization	All	Read/Write

1.5.4 Key Zeroization

Users can ensure sensitive data is properly zeroized by making use of the `SensitiveData.clear()` method for clearing sensitive data. The toolkit ensures that all ephemeral sensitive data is cleared within the toolkit. For more information about clearing sensitive data, see Clearing Sensitive Data in the *RSA BSAFE Crypto-J Developer's Guide*.

1.5.5 Key Storage

Crypto-J does not provide long-term cryptographic key storage. Storage of keys is the responsibility of the user of Crypto-J.

The following table shows how the storage of keys and Critical Security Parameters (CSPs) are handled. The Crypto User and Crypto Officer roles have equal and complete access to all keys and CSPs.

Table 3 Key and CSP Storage

Item	Storage
AES keys	In volatile memory only (plaintext)
Triple-DES keys	In volatile memory only (plaintext)
HMAC with SHA1 and SHA2 keys	In volatile memory only (plaintext)
EC public keys	In volatile memory only (plaintext)
EC private keys	In volatile memory only (plaintext)
DH public key	In volatile memory only (plaintext)
DH private key	In volatile memory only (plaintext)
RSA public key	In volatile memory only (plaintext)
RSA private key	In volatile memory only (plaintext)
DSA public key	In volatile memory only (plaintext)
DSA private key	In volatile memory only (plaintext)
PRNG seeds (FIPS 186-2)	In volatile memory only (plaintext)
PRNG Keys (FIPS 186-2)	In volatile memory only (plaintext)
EC DRBG Entropy	In volatile memory only (plaintext)
EC DRBG S Value	In volatile memory only (plaintext)
EC DRBG init_seed	In volatile memory only (plaintext)
HMAC DRBG Entropy	In volatile memory only (plaintext)
HMAC DRBG V Value	In volatile memory only (plaintext)
HMAC DRBG Key	In volatile memory only (plaintext)
HMAC DRBG init_seed	In volatile memory only (plaintext)

1.6 Cryptographic Algorithms

Crypto-J meets FIPS 140-2 requirements by implementing algorithm enforcement, such that when operating in `FIPS140_MODE`, only FIPS 140-approved algorithms are available for use.

The following table lists the FIPS 140-approved algorithms provided by Crypto-J, when operating in `FIPS140_MODE`.

Table 4 Crypto-J FIPS-approved Algorithms

Algorithm	Validation Certificate
AES ECB, CBC, CFB (128), OFB (128), CTR - [128, 192, 256 bit key sizes] CCM, GCM, XTS	Certificate #1465
Triple-DES ECB, CBC, CFB (64bit) and OFB (64 bit)	Certificate #988
DH	Non-Approved (Allowed in FIPS mode)
DSA	Certificate #464
Dual EC DRBG (SP800-90)	Certificate #57
EC-Diffie-Hellman, EC-Diffie-Hellman with Cofactor	Non-Approved (Allowed in FIPS mode)
EC-DSA, EC-DSA-SHA1	Certificate #182
FIPS 186-2 PRNG (Change Notice 1-with and without the mod q step)	Certificate #802
HMAC DRBG (SP800-90)	Certificate #57
HMAC-SHA1, SHA224, SHA256, SHA384, SHA512	Certificate #863
RSA encrypt/decrypt	Non-Approved (Allowed in FIPS mode for key transport)
RSA X9.31, PKCS #1 V.1.5, PKC S#1 V.2.1 (SHA256 - PSS)	Certificate #717
SHA-1	Certificate #1328
SHA-224, 256, 384, 512	Certificate #1328

The following are the non-FIPS 140-approved algorithms provided by Crypto-J, when operating in `NON_FIPS140_MODE`.

- DES
- DESX
- ECAES
- ECIES
- MD2
- MD5
- PBE
- Random Number Generators (ANSI X9.31 (non-compliant), MD5Random and SHA1Random)
- RC2® block cipher
- RC4® stream cipher
- RC5® block cipher
- PBE with SHA1 and Triple-DES
- RSA OAEP for key transport
- Raw RSA encryption and decryption
- RSA Keypair Generation MultiPrime (2 or 3 primes)
- RIPEMD160
- HMAC-MD5.

1.7 Self-tests

Crypto-J performs power-up and conditional self-tests to ensure proper operation. If the power-up self-test fails, the toolkit is disabled and throws a `SecurityException`. The toolkit can only leave the disabled state by restarting the JVM. If the conditional self-test fails, the toolkit throws a `SecurityException` and aborts the operation. A conditional self test failure does NOT disable the toolkit.

1.7.1 Power-up Self-tests

The following FIPS-140 required power-up self-tests are implemented in Crypto-J:

- FIPS186 PRNG KAT
- AES KAT
- TDES KAT
- SHA-1 KAT
- SHA-224 KAT
- SHA-256 KAT
- SHA-384 KAT
- SHA-512 KAT
- HMAC SHA-1 KAT
- HMAC SHA-224 KAT
- HMAC SHA-256 KAT
- HMAC SHA-384 KAT
- HMAC SHA-512 KAT
- HMAC DRBG Self-Test
- ECDRBG Self-Test
- ECDSA KAT
- Software/firmware integrity check
- DSA KAT
- DSA, RSA, EC pair-wise consistency test
- RSA (signature) KAT.

The following non-FIPS-140 power-up self-tests are implemented in Crypto-J:

- MD5 KAT
- HMAC MD5 KAT
- ECIES KAT
- ECAES KAT.

Power-up self-tests are executed automatically when Crypto-J is loaded into memory.

1.7.2 Conditional Self-tests

Crypto-J performs two conditional self-tests:

- Pair-wise consistency tests each time the toolkit generates a DSA, RSA or EC public/private key pair.
- Continuous RNG (CRNG) test each time the toolkit produces random data, as per the FIPS 186-2 standard. The CRNG test is performed on all approved and non-approved RNGs.

1.7.3 Mitigation of Other Attacks

RSA key operations implement blinding by default. Blinding is a reversible way of modifying the input data, so as to make the RSA operation immune to timing attacks. Blinding has no effect on the algorithm other than to mitigate attacks on the algorithm.

RSA Blinding is implemented through blinding modes, for which the following options are available:

- Blinding mode off
- Blinding mode with no update, where the blinding value is squared for each operation
- Blinding mode with full update, where a new blinding value is used for each operation.

2 Secure Operation of Crypto-J

The following guidance must be followed in order to operate the Crypto-J module in a FIPS 140 mode of operation, in conformance with FIPS 140-2 requirements.

2.1 Module Configuration

To operate the Crypto-J module in compliance with FIPS 140-2 requirements, the `com.rsa.cryptoj.kat.strategy` Java security property must be set to the value `on.load` at start-up of the module. See the *RSA BSAFE Crypto-J Installation Guide* for more information about setting this property.

The property is read once only, on start-up, and any subsequent changes to the property are ignored. Setting this property ensures that all module self-tests are run during module start-up, as specified by FIPS 140-2 requirements. If any self-test fails, the module is disabled.

2.2 Security Roles, Authentication, and Services Operation

To operate the cryptographic module in FIPS 140 Security Level 1 Roles, Authentication, and Services mode, ensure that the Java Security property `com.rsa.cryptoj.fips140auth in the <jre>/lib/security/java.security` file is NOT set. That is, ensure that the property does not exist in the `java.security` file.

No role authentication is required for operation of the module in Security Level 1 mode. When in Security Level 1 mode, invocation of methods which are particular to Security Level 2 Roles, Authentication and Services will result in an error. These methods include constructors for `FIPS140Context` objects which accept a PIN, as well as PIN management APIs on the `CryptoJ` class.

2.3 Crypto User Guidance

This section provides guidance to the toolkit user to ensure that the toolkit is used in a FIPS 140-2 compliant way.

Section 2.3.1 provides algorithm-specific guidance. The requirements listed in this section are not enforced by the module and must be ensured by the module user.

Section 2.3.2 provides guidance on obtaining assurances for Digital Signature Applications.

Section 2.3.3 provides general crypto user guidance.

2.3.1 Crypto User Guidance on Algorithms

- The Crypto User must only use algorithms approved for use in a FIPS 140 mode of operation, as listed in Table 4, “Crypto-J FIPS-approved Algorithms,” on page 15.
- When using GCM feedback mode for symmetric encryption, the authentication tag length and authenticated data length may be specified as input parameters, but the Initialization Vector (IV) must not be specified. It must be generated internally.
- RSA keys used for signing shall not be used for any other purpose other than digital signatures.
- For RSASSA-PSS, the length of the salt (`sLen`) shall be `0<=sLen<=hlen` where `hlen` is the length of the hash function output block.
- Bit lengths for the Diffie-Hellman[1] key agreement must be between 1024 and 2048 bits. Diffie Hellman shared secret provides between 80 bits and 112 bits of encryption strength.
- Bit lengths for an HMAC key must be one half of the block size.
- For RSA digital signature generation either Dual EC DRBG or HMACDRBG have to be used.
- EC key pairs must have domain parameters from the set of NIST-recommended named curves (P192, P224, P256, P384, P521, B163, B233, B283, B409, B571, K163, K233, K283, K409, and K571). The domain parameters can be specified by name or can be explicitly defined. The module limits possible curves for Dual EC DRBG to P-256, P-384, and P-521 in accordance with SP 800-90.
- EC Diffie-Hellman primitives must use curve domain parameters from the set of NIST recommended named curves listed above. The domain parameters can be specified by name, or can be explicitly defined. Using the NIST-recommended curves, the computed Diffie-Hellman shared secret provides between 80 bits and 256 bits of encryption strength.
- When using an Approved RNG to generate keys or DSA parameters, the RNG’s requested security strength must be at least as great as the security strength of the key being generated. That means that either Dual EC DRBG or HMACDRBG

[1] Using the minimum allowed modulus size, the minimum strength of encryption provided is 80 bits.

with an appropriate strength have to be used. For more information on requesting the RNG security strength, see the Random Number Generation sections of the *RSA BSAFE Crypto-J Developer's Guide*.

- SHA1 is a FIPS approved hash algorithm for the generation of digital signatures only until the end of 2010.

More information on the algorithm strength and keysize is provided in the *RSA BSAFE Crypto-J Release Notes*.

2.3.2 Crypto User Guidance on Obtaining Assurances for Digital Signature Applications

Crypto-J 5.0 has added support for the FIPS 186-3 standard for digital signatures in FIPS 140-2 mode. The following gives an overview of the assurances required by FIPS 186-3.

NIST Special Publication 800-89: "*Recommendation for Obtaining Assurances for Digital Signature Applications"* provides the methods to obtain these assurances.

The tables below describes the FIPS 186-3 requirements for signatories and verifiers and the corresponding Crypto-J toolkit capabilities and recommendations.

Table 5 Signatory Requirements

FIPS 186-3 Requirement	Crypto-J toolkit capabilities and recommendations
Obtain appropriate DSA and ECSDSA parameters when using DSA or ECDSA.	The generation of DSA parameters is in accordance with the FIP 186-3 standard for the generation of probable primes. For ECDSA, use the NIST recommended curves as defined in section 2.3.1.
Obtain assurance of the validity of those parameters.	The toolkit provides APIs to validate DSA parameters for probable primes as described in FIPS 186-3. For the JSAFE API, `com.rsa.jsafe.JSAFE_VerifyPQG.verifyPQGParams()` For the JCE API, `com.rsa.jsafe.provider.Assurance.verifyDSAParameters()` For ECDSA, use the NIST recommended curves as defined in section 2.3.1.
Obtain a digital signature key pair that is generated as specified for the appropriate digital signature algorithm.	The toolkit generates the digital signature key pair according to the required standards. Choose a FIPS approved RNG like HMACDRBG or Dual-ECDRBG to generate the key pair.
Obtain assurance of the validity of the public key.	The toolkit provides APIs to explicitly validate the public key according to NIST Special Publication 800-89. For the JSAFE API, `com.rsa.jsafe.JSAFE_VerifyPQG.verifyPQGParams()` For the JCE API, `com.rsa.jsafe.provider.Assurance.isValidPublicKey()`

Table 5 Signatory Requirements (continued)

FIPS 186-3 Requirement	Crypto-J toolkit capabilities and recommendations
Obtain assurance that the signatory actually possesses the associated private key.	The toolkit verifies the signature created using the private key, but all other assurances are outside the scope of the toolkit.

Table 6 Verifier Requirements

FIPS 186-3 Requirement	Crypto-J toolkit capabilities and recommendations
Obtain assurance of the signatory's claimed identity	The toolkit verifies the signature created using the private key, but all other assurances are outside the scope of the toolkit.

Obtain assurance of the validity of the domain parameters for DSA and ECDSA.	The toolkit provides APIs to validate DSA parameters for probable primes as described in FIPS 186-3.
	For the JSAFE API,
	`com.rsa.jsafe.JSAFE_VerifyPQG.verifyPQGParams()`
	For the JCE API,
	`com.rsa.jsafe.provider.Assurance.verifyDSAParameters()`
	For ECDSA, use the NIST recommended curves as defined in section 2.3.1.

Obtain assurance of the validity of the public key	The toolkit provides APIs to explicitly validate the public key according to NIST Special Publication 800-89. For the JSAFE API, `com.rsa.jsafe.JSAFE_VerifyPQG.verifyPQGParams()` For the JCE API, `com.rsa.jsafe.provider.Assurance.isValidPublicKey()`
Obtain assurance that the claimed signatory actually possessed the private key that was used to generate the digital signature at the time that the signature was generated	Outside the scope of the toolkit.

For more details on the requirements, see the *FIPS 186-3 and NIST Special Publication 800-89*.

2.3.3 General Crypto User Guidance

Crypto-J users should take care to zeroize CSPs when they are no longer needed. For more information on clearing sensitive data, see *Clearing Sensitive Data* in the *RSA BSAFE Crypto-J Developer's Guide*.

2.4 Crypto Officer Guidance

The Crypto Officer is responsible for installing the toolkit. Installation instructions are provided in the *RSA BSAFE Crypto-J Installation Guide*.

2.5 Role Changes

If a user of Crypto-J needs to operate the toolkit in different roles, then the user must ensure that all instantiated cryptographic objects are destroyed before changing from the Crypto User role to the Crypto Officer role, or unexpected results could occur.

2.6 Operating the Cryptographic Module

The Cryptographic Module operates in `FIPS140_MODE` by default after module power-up. The initial mode can be configured through the use of the `com.rsa.cryptoj.fips140initialmode` security property. See the *RSA BSAFE Crypto-J Installation Guide* for details on how to set this property. If the property is not set, then the default mode `FIPS140_MODE` is used. The current mode of the cryptographic module can be determined with a call to the `CryptoJ.getMode()` method. The mode of the cryptographic module can be changed by using the function `CryptoJ.setMode()` with an information identifier from Table 7, “Mode of Operation Values,” on page 24.

When changing to an approved mode of operation, the toolkit causes the power-up self-tests to run.

After setting the cryptographic module into a FIPS approved mode, the Cryptographic Module ensures that only the FIPS approved algorithms listed in “Services” on page 8 are available to operators. To disable FIPS mode, call the `CryptoJ.setMode()` method with the mode identifier `NON_FIPS140_MODE`.

The Service `CryptoJ.runSelfTests()` is restricted to operation by the Crypto Officer.

2.7 Modes of Operation

There are five modes of operation:

- `FIPS140_MODE`
- `FIPS140_SSL_MODE`
- `NON_FIPS140_MODE`
- `FIPS140_ECC_MODE`
- `FIPS140_SSL_ECC_MODE.`

The following table lists the values that can be used in the `setMode()` method to change the mode of operation, and the algorithms available in that mode.

Table 7 Mode of Operation Values

Value	Algorithms Available
`CryptoJ.FIPS140_MODE` FIPS 140-2 approved.	Provides the cryptographic algorithms listed inTable 4, "Crypto-J FIPS-approved Algorithms," on page 15. This is the Crypto-J default mode on start up.
`CryptoJ.FIPS140_SSL_MODE` FIPS 140-2 approved if used with TLS protocol implementations.	Provides the same algorithms as `CryptoJ.FIPS140_MODE`, plus the MD5 message digest. This mode can be used in the context of the key establishment phase in the TLSv1, TLSv1.1 and TLSv1.2 protocols. For more information, see section 7.1 Acceptable Key Establishment Protocols in `Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program`. The implementation guidance disallows the use of the SSLv2 and SSLv3 versions. Cipher suites that include non-FIPS 140-2-approved algorithms are unavailable. This mode allows implementations of the TLS protocol to operate Crypto-J in a FIPS 140-2-compliant manner.
`CryptoJ.NON_FIPS140_MODE` Not FIPS 140-2 approved.	Allows users to operate Crypto-J without any cryptographic algorithm restrictions.
`CryptoJ.FIPS140_ECC_MODE` Not FIPS 140-2 approved.	Provides the same algorithms as `CryptoJ.FIPS140_MODE`, plus ECAES and ECIES.
`CryptoJ.FIPS140_SSL_ECC_MODE` Not FIPS 140-2 approved.	Provides the same algorithms as `CryptoJ.FIPS140_SSL_MODE`, plus ECAES and ECIES. The same restrictions with respect to protocol versions and cipher suites as in `CryptoJ.FIPS140_SSL_MODE` apply.

Note: Refer to Random Number Generator for details of the default random number generator for the available algorithms.

Cryptographic algorithms can be created in different modes using the `com.rsa.jsafe.crypto.FIPS140Context` class for the JSAFE API. For more information about operating in FIPS 140 mode, see the *RSA BSAFE Crypto-J JSAFE and JCE Software Module 5.0 Developers Guide*.

2.8 Random Number Generator

Crypto-J provides a default RNG. This default RNG can be set to one of the FIPS-140 approved RNG, (ECDRBG, HMACDRBG, FIPS186-2 RNG) using the property `com.rsa.crypto.default.random`. For the correct configuration settings, see the *RSA BSAFE Crypto-J Developer's Guide.*

If the property is not set, the default RNG used is Dual ECDRBG.

Users in FIPS 140-2 mode can select either the FIPS 186-2, ECDRBG or HMAC DRBG when creating a RNG object and setting this object against the operation requiring random number generation (for example key generation).

Users in non-FIPS 140-2 mode can use any RNG and set it against the operation requiring random number generation.

For more information on each function, see the *RSA BSAFE Crypto-J Developer's Guide*

3 Acronyms

The following table lists the acronyms used with Crypto-J and their definitions.

Table 8 Acronyms used with Crypto-J

Acronym	Definition
3DES	Refer to Triple-DES
AES	Advanced Encryption Standard. A fast block cipher with a 128-bit block, and keys of lengths 128, 192 and 256 bits. This will replace DES as the US symmetric encryption standard.
API	Application Programming Interface.
Attack	Either a successful or unsuccessful attempt at breaking part or all of a cryptosystem. Attack types include an algebraic attack, birthday attack, brute force attack, chosen ciphertext attack, chosen plaintext attack, differential cryptanalysis, known plaintext attack, linear cryptanalysis, middleperson attack and timing attack.
CBC	Cipher Block Chaining. A mode of encryption in which each ciphertext depends upon all previous ciphertexts. Changing the IV alters the ciphertext produced by successive encryptions of an identical plaintext.
CFB	Cipher Feedback. A mode of encryption that produces a stream of ciphertext bits rather than a succession of blocks. In other respects, it has similar properties to the CBC mode of operation.
CRNG	Continuous Random Number Generation.
CSP	Critical Security Parameters.
DES	Data Encryption Standard. A symmetric encryption algorithm with a 56-bit key.
Diffie-Hellman	The Diffie-Hellman asymmetric key exchange algorithm. There are many variants, but typically two entities exchange some public information (for example, public keys or random values) and combines them with their own private keys to generate a shared session key. As private keys are not transmitted, eavesdroppers are not privy to all of the information that composes the session key.
DRBG	Deterministic Random Bit Generator.
DSA	Digital Signature Algorithm. An asymmetric algorithm for creating digital signatures.
EC	Elliptic Curve.
ECAES	Elliptic Curve Asymmetric Encryption Scheme.

Table 8 Acronyms used with Crypto-J (continued)

Acronym	Definition
ECB	Electronic Code Book. A mode of encryption in which identical plaintexts are encrypted to identical ciphertexts, given the same key.
ECC	Elliptic Curve Cryptography.
ECDH	Elliptic Curve Diffie-Hellman.
ECDHC	Elliptic Curve Diffie-Hellman with Components.
ECDSA	Elliptic Curve Digital Signature Algorithm.
ECIES	Elliptic Curve Integrated Encryption Scheme.
Encryption	The transformation of plaintext into an apparently less readable form (called ciphertext) through a mathematical process. The ciphertext may be read by anyone who has the key that decrypts (undoes the encryption) the ciphertext.
FIPS	Federal Information Processing Standards.
HMAC	Keyed-Hashing for Message Authentication Code.
IV	Initialization Vector. Used as a seed value for an encryption operation.
JCE	Java Cryptography Extension.
JVM	Java Virtual Machine.
KAT	Known Answer Test.
Key	A string of bits used in cryptography, allowing people to encrypt and decrypt data. Can be used to perform other mathematical operations as well. Given a cipher, a key determines the mapping of the plaintext to the ciphertext. Various types of keys include: distributed key, private key, public key, secret key, session key, shared key, subkey, symmetric key, and weak key.
MD5	A secure hash algorithm created by Ron Rivest. MD5 hashes an arbitrary-length input into a 16-byte digest.
NIST	National Institute of Standards and Technology. A division of the US Department of Commerce (formerly known as the NBS) which produces security and cryptography-related standards.
OFB	Output Feedback. A mode of encryption in which the cipher is decoupled from its ciphertext.
OS	Operating System.
PC	Personal Computer.

Table 8 Acronyms used with Crypto-J (continued)

Acronym	Definition
private key	The secret key in public key cryptography. Primarily used for decryption but also used for encryption with digital signatures.
PRNG	Pseudo-random Number Generator.
RC2	Block cipher developed by Ron Rivest as an alternative to the DES. It has a block size of 64 bits and a variable key size. It is a legacy cipher and RC5 should be used in preference.
RC4	Symmetric algorithm designed by Ron Rivest using variable length keys (usually 40 bit or 128 bit).
RC5	Block cipher designed by Ron Rivest. It is parameterizable in its word size, key length and number of rounds. Typical use involves a block size of 64 bits, a key size of 128 bits and either 16 or 20 iterations of its round function.
RNG	Random Number Generator.
RSA	Public key (asymmetric) algorithm providing the ability to encrypt data and create and verify digital signatures. RSA stands for Rivest, Shamir, and Adleman, the developers of the RSA public key cryptosystem.
SHA	Secure Hash Algorithm. An algorithm which creates a hash value for each possible input. SHA takes an arbitrary input which is hashed into a 160-bit digest.
SHA-1	A revision to SHA to correct a weakness. It produces 160-bit digests. SHA-1 takes an arbitrary input which is hashed into a 20-byte digest.
SHA-2	The NIST-mandated successor to SHA-1, to complement the Advanced Encryption Standard. It is a family of hash algorithms (SHA-256, SHA-384 and SHA-512) which produce digests of 256, 384 and 512 bits respectively.
TDES	Refer to Triple-DES
Triple-DES	A symmetric encryption algorithm which uses either two or three DES keys. The two key variant of the algorithm provides 80 bits of security strength while the three key variant provides 112 bits of security strength.