IBM SSLite for Java FIPS 140-2 Cryptographic Module

Security Policy

IBM SSLite for Java FIPS 140-2 Cryptographic Module	**February 2005**
Revision: 1.9	
NON CONFIDENTIAL	**Status: Final**

Second Edition (February 2005)

This edition applies to the First Edition of the IBM BlueZ – FIPS140-2 SSLite Security Policy and to all subsequent versions until otherwise indicated in new editions.

Contents

1. Introduction

IBM SSLite for Java, or simply SSLite, is a SSL (Secure Socket Layer) v2.0, v3.0 and TLS (Transport Layer Security) v1.0 protocol implementation including PKI (Public Key Infrastructure) functionality for Java. For the purpose of FIPS 140-2 Level 1 validation the implementation is made available in the form of a signed JAR archive file. The cryptographic functions used in SSLite are contained in the IBM CryptoLite for Java module which has been separately evaluated.

SSLite supports most PKI components such as X.509 certificates, Certificate Revocation Lists (CRLs), PKCS #7 tokens, PKCS #11 cryptographic tokens (smart cards, etc.), PKCS #12 tokens, Java KeyStore tokens, MS-CAPI, and certificate repositories such LDAP.

The performance of SSLite is comparable to that of native implementations. It may, however, be augmented with a native booster DLL to make it perform even better.

This document applies to IBM SSLite for Java FIPS builds 3.15 and 3.16.

1.1 Contact Information

For more information about IBM SSLite for Java, IBM CryptoLite for Java, and IBM CryptoLite for C please contact ccc@dk.ibm.com.

2. Security Levels

The SSLite package meets the overall requirements applicable to FIPS 140-2 Level 1 security. The individual security requirements specified for FIPS 140-2 meets the Level specifications indicated in the following table.

Security Requirements Section	Level
Cryptographic Module	1
Ports and Interfaces	1
Roles and Services	1
Finite State Model	1
Physical Security	1
Operational Environment	1
Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

Table 1: FIPS 140-2 Validation Levels

3. Cryptographic Module Specification

SSLite is classified as a multiple-chip standalone cryptographic module for FIPS 140-2 purposes. As such, the SSLite module must be validated upon a particular operating system and computer platform. The SSLite module is packaged in a single Java Archive file, which contains all the classes for the module. SSLite runs upon many other platforms including Windows 95, 98, and NT, Sun Solaris, HP-UX, Linux, and AIX.

As outlined in 4.5 of the Implementation Guidance for FIPS 140-2, the module maintains its compliance on other operating systems, provided:

- The operating system meets the operational environment requirements at the module's Level of validation.
- The module does not require modification to run in the new environment.

Since the SSLite module is a pure Java implementation it should be able to run unmodified on any system which supports a Java Runtime of at least Version 1.3. The above requirements have been demonstrated by testing and validating the IBM CryptoLite for Java package on the platforms shown in Table 2.

Hardware	Operating System	Java VM Version
IBM PC Compatible	Windows 2000, SP3	1.3.1_03
IBM PC Compatible	Red Hat Linux 8.0	1.3.1_07

Table 2: Platforms on which IBM CryptoLite for Java has been tested

3.1 SSLite token interfaces

The SSLite module provides cryptography services through the embedded IBM CryptoLite for Java, or simply CryptoLite, module. CryptoLite contains a proprietary interface for interfacing to an external cryptographic acceleration module. This module, also provided by IBM and FIPS 140-2 Level 1 validated, uses optimized native code to provide high performance speedup for CryptoLite functionality in a totally transparent manner. The module simply has to be present in the same directory as the CryptoLite module at module start-up time. The validity and integrity of the booster module is checked using an approved HMAC method.

The SSLite module also contains industry standard PKCS #11 and MS-CAPI token interfaces.

3.2 Cryptographic Standards

The SSLite module supports the following approved and non-approved FIPS algorithms through the IBM CryptoLite for Java module. Only algorithms listed in *FIPS PUB 140-2 Annex A: Approved Security Functions* have been marked as being FIPS approved. Algorithms marked as non-compliant are FIPS approved but the implementation has not yet been validated.

3.2.1 Symmetric Key

Algorithm	Specification	FIPS Approved
AES (ECB, CBC)	FIPS 197	YES
DES, Triple-DES (ECB, CBC)	FIPS 46-3	YES

RC2	RFC 2268	NO
RC4	N/A	NO
Blowfish	http://www.schneier.com/blowfish.html	NO

3.2.2 Asymmetric Key

Algorithm	Specification	FIPS Approved
RSA Sign/Verify	PKCS #1 v2.1	YES
RSA Encrypt/Decrypt	PKCS #1 v2.1	NO
DSA Sign/Verify	FIPS 186-2	YES
Diffie-Hellman	N/A	NO

3.2.3 Hash Functions

Algorithm	Specification	FIPS Approved
MD2	RFC 1319	NO
MD5	RFC 1321	NO
SHA-1	FIPS 180-2	YES
SHA-256, SHA-384, SHA-512	FIPS 180-2	Non-compliant
MDC-1, MDC-2, MDC-4	Bruno O. Brachtl, Don Coppersmith, Myrna M. Hyden, Stephen M. Matyas Jr., Carl H. W. Meyer, Jonathan Oseas, Shaiy Pilpel, and Michael Shilling, "Data authentication using modification detection codes based on a public one way encryption function"	NO

3.2.4 Keyed Hash Functions

Algorithm	Specification	FIPS Approved
HMAC	FIPS 198	YES*

* When used with SHA-1.

3.2.5 Random Number Generators

Algorithm	Specification	FIPS Approved
Pseudo Random Number Generator	FIPS 186-2 ANSI X9.31-1998	YES
Random Number Generator*	Patented by IBM Corp., EC Pat. No. EP1081591A2. Used for	NO

	seeding the approved PRNG.	

* Also known as the “Universal Software Based True Random Number Generator”.

3.3 Module Interfaces

As a multiple-chip standalone cryptographic module the SSLite module’s physical interfaces consist of the keyboard, mouse, monitor, serial ports, network adapters, etc. However, the underlying logical interface to the SSLite package is a Java language API documented in the *SSLite User Guide*. The exported public methods comprise the modules control input interface. Data input and output are provided in the variables passed with method calls, and status output is provided in the return and error codes that are documented for each call. The SSLite module is accessed from Java language programs via the inclusion of the package export files and the package class file packaged in JAR format.

3.4 Cryptographic Module Self Tests

The SSLite module relies exclusively on the embedded CryptoLite module for a number of self-tests to check the proper functioning of the module. This includes power-up self-tests and conditional self-tests. Conditional tests are performed when symmetric or asymmetric keys are generated. This also includes a continuous random number generator test and a pair-wise consistency tests of the generated RSA and DSA keys.

3.4.1 Power-up Self-Testing

Power-up self-testing is initiated automatically when the SSLite module starts loading (see the *SSLite Finite State Machine* for more details). These tests are comprised of the software integrity test and the known answer tests (KATs) of cryptographic algorithms. Should any of these tests fail the SSLite module will terminate the loading process and generate an exception. The module cannot be used in this state. The integrity of the module is verified by checking an HMAC on the all of the JAR files classes. The initialization will only succeed if this HMAC is valid.

The SSLite module executes the following cryptographic algorithms tests during power-up:

- DES KAT
- Triple-DES KAT
- AES KAT
- SHA-1 KAT
- SHA-256 KAT
- SHA-384 KAT
- SHA-512 KAT
- RSA sign/verify
- RSA encrypt/decrypt
- DSA parameter generation
- DSA sign/verify
- Diffie-Hellman exponentiation
- RNG KAT

3.4.2 Start-up Recovery

Should the start-up self-tests fail during module initialization the crypto officer should re-initialize the application.

3.4.3 Conditional Self-Testing

This includes continuous PRNG testing. The very first output block generated by the PRNG is never used for any purpose other than initiating the continuous PRNG test which compares every newly generated block with the previously generated block. The test fails if newly generated PRNG output block matches the previously generated block. In such a case, the module generates an exception to the calling application. It is the responsibility of the calling application to handle the exception in a FIPS 140-2 appropriate manner, for example by retrying the PRNG service.

3.4.4 Pair-wise Consistency Self-Testing

The test is run whenever private key is generated by the SSLite module. The private key structure of the module always contains either the data of the corresponding public key or information sufficient for computing the corresponding public key. If the test fails the module generates an exception to the calling application. It is the responsibility of the calling application to handle the exception in a FIPS 140-2 appropriate manner, for example by retrying the key generation service.

3.5 Operational Environment

The SSLite module is written entirely in the Java programming language that allows for extensive review to confirm security. Applications using SSLite functionality are secure from each other due to the fact that each runs in a “Java sandbox” where the firewall protects applet objects from illegal access by other applications. SSLite is developed and maintained according to IBM’s internal development standards and tools including CVS (cvs-1.11.2-25) are used for configuration management. The CryptoLite module implements both approved and non-approved services. The calling application controls the cryptographic material as well as the services that use them. It is the applications responsibility to ensure that when used in a FIPS 140-2 compliant mode, only those FIPS 140-2 approved algorithms are used.

3.6 Module Status

The module communicates any error status asynchronously through the use of exceptions. It is the responsibility of the calling application to handle these exceptions.

4. Roles and Services

4.1 Roles

The SSLite module supports two roles:

- **ROLE_CO**: The Crypto Officer Role is purely an administrative role and does not involve the use of any of the modules cryptographic services. The role is not explicitly authenticated but assumed implicitly on implementation of the modules installation and usage sections defined in the security rules section.
- **ROLE_USER**: The User Role has access to all of the modules services. The role is not explicitly authenticated but assumed implicitly on access of any of the modules services.

Role	Type of Authentication	Authentication Data
Cryptographic Officer Role	None	None
User Role	None	None

Table 3: Roles and Required Identification and Authentication

Authentication Mechanism	Strength of Mechanism
There are no role or user authentication mechanisms	N/A

Table 4: Strengths of Authentication Mechanisms

4.2 Services

The modules services are accessed through API interfaces from the calling application.

Service	User Role
ASN.1 handling services (com.ibm.sslite140.ASN1)	YES
Certificate authority services (com.ibm.sslite140.CA)	YES
Certificate entry services (com.ibm.sslite140.CE)	YES
CryptoLite for Java (com.ibm.sslite140.CL3)	YES
Certificate request entry services (com.ibm.sslite140.CRE)	YES
Directory services (com.ibm.sslite140.Directory)	YES
Certificate extension services (com.ibm.sslite140.Extension)	YES
LDAP services (com.ibm.sslite140.LDAP)	YES
PKI services (com.ibm.sslite140.PKI)	YES
Certificate revocation entry services (com.ibm.sslite140.RE)	YES
JAR file signature verification services (com.ibm.sslite140.SignedJarInputStream)	YES
JAR file signature generation services (com.ibm.sslite140.SignedJarOutputStream)	YES
S/MIME services (com.ibm.sslite140.SMIME)	YES

Certificate services (com.ibm.sslite140.SSLCert)	YES
SSL context services (com.ibm.sslite140.SSLContext)	YES
Certificate revocation list services (com.ibm.sslite140.SSLCRL)	YES
GSKit certificate services (com.ibm.sslite140.SSLKDBToken)	YES
Java KeyStore certificate services (com.ibm.sslite140.SSLKSCert)	YES
Java KeyStore services (com.ibm.sslite140.SSLKSToken)	YES
MS-CAPI token services (com.ibm.sslite140.SSLMSCAPIToken)	YES
X.500 name services (com.ibm.sslite140.SSLName)	YES
Netscape token services (com.ibm.sslite140.SSLNSToken)	YES
PKCS #11 token services (com.ibm.sslite140.SSLPKCS11Token)	YES
PKCS #12 token services (com.ibm.sslite140.SSLPKCS12Token)	YES
PKCS #7 token services (com.ibm.sslite140.SSLPKCS7Token)	YES
SSL server services (com.ibm.sslite140.SSLServerSocket)	YES
SSL session services (com.ibm.sslite140.SSLSession)	YES
SSL socket services (com.ibm.sslite140.SSLSocket)	YES
SSL general token services (com.ibm.sslite140.SSLToken)	YES
Transaction layer services (com.ibm.sslite140.TL)	YES
Certificate SubjectAltName/IssuerAltName extension services (com.ibm.sslite140.XAltName)	YES
Certificate AuthorityInformationAccess extension services (com.ibm.sslite140.XAuthorityInfoAccess)	YES
Certificate BasicConstaints extension services (com.ibm.sslite140.XBasicConstraints)	YES
Certificate CRL distribution points extension services (com.ibm.sslite140.XCRLDist)	YES
Certificate CRL distribution point services (com.ibm.sslite140.XCRLDist.Point)	YES
Certificate ExtKeyUsage extension services (com.ibm.sslite140.XExtKeyUsage)	YES
Certificate SubjectKeyIdentifier/AuthorityKeyIdentifier extension services (com.ibm.sslite140.XKeyIdentifier)	YES
Certificate KeyUsage extension services (com.ibm.sslite140.XKeyUsage)	YES
Certificate policy services (com.ibm.sslite140.XPolicyInfo)	YES
Certificate policy qualifier services (com.ibm.sslite140.XPolicyInfo.Qualifier)	YES
CryptoLite for Java exception services (com.ibm.sslite140.CL3Exception)	YES
SSL exception services (com.ibm.sslite140.SSLException)	YES
SSL runtime exception services (com.ibm.sslite140.SSLRuntimeException)	YES

5. Cryptographically Sensitive Material

5.1 Cryptographic Keys

5.1.1 Key Storage

The SSLite module does not provide long-term cryptographic key storage. If an application program makes use of SSLite service to implement cryptographic key storage functionality, it is the responsibility of the application program developer to ensure FIPS 140-2 compliance of key storing techniques they implement.

5.1.2 Key establishment

SSLite provides protocol services for SSLv2.0, SSLv3.0 and SSLv3.1/TLSv1.0. Each of these protocols involves the generation of key material based on elements within the handshake protocol. SSLv3.0/TLSv1.0 depends on SHA-1 for the generation of key material. SSLv2.0 and SSLv3.0 generally depend on MD5 for key material generation and thus are not FIPS 140-2 compliant. There is an exception for 2 SSLv3.0 cipher suites, namely 0xFEFE and 0xFEFF, where SSLite will use the SSLv3.1/TLSv1.0 method for key generation which is based on SHA-1.

5.1.3 Key Protection

The management and allocation of memory is the responsibility of the operating system. It is assumed that a unique process space is allocated for each request, and that the operating system and the underlying central processing unit (CPU) hardware control access to that space. Each instance of the cryptographic module is self-contained within a process space. All keys are associated with the user role. It is the responsibility of application program developers to protect keys exported from the SSLite module.

5.1.4 Key Generation

Key generation is handled using the CryptoLite subsystem which uses a FIPS 140-2 approved RNG algorithm based on SHA-1. The RNG has a maximum number of internal states of 2^{160}; this being limited by the compression function in SHA-1. The RSA and DH key generation algorithms use the RNG engine seeded with 20 bytes of true random data. This true random generator is based on IBM patented technology where statistical analysis used to estimate the entropy of the clock jitter. The internal RNG engine is enhanced using an automatic reseeding policy that insert a true random byte every 128 bytes of output if more than 30 seconds passed since last being reseeded. Applications can additionally provide their own seeding data and also increase the automatic reseeding policy of the internal RNG engine for example to add true random data every 8th byte without time constraint.

5.1.5 Key zeroization

Key objects are normally zeroed and any associated data discarded when the key object is garbage collected through the finalizer method. The CryptoLite module provides an additional mechanism which helps to ensure key zeroization through a dispose method. An application can explicitly call this method in order to clear and release key material associated with a key object without waiting for a possible pending invocation of the finalizer method.

5.1.6 Key Import/Export

The SSLite module provides a series of services for applications to access cryptographic material contained within various long term storage elements or tokens. These key repositories and tokens are outside of SSLite's cryptographic boundary. The SSLite module temporarily holds and uses key material on behalf of the calling applications and processes. Key material imported is stored internally in token key rings. This temporary internal storage of key material and its subsequent use is on behalf of the calling applications.

5.2 Cryptographic Tokens

5.2.1 PKCS #7

PKCS #7 describes a general syntax for data that may have cryptography applied to it, such as digital signatures and digital envelopes. The syntax admits recursion, so that, for example, one envelope can be nested inside another, or one party can sign some previously enveloped digital data. It also allows arbitrary attributes, such as signing time, to be authenticated along with the content of a message, and provides for other attributes such as countersignatures to be associated with a signature. This token is a soft token and can be retrieved from different media. It contains a set of certificates and, optionally, associated CRLs. Keys cannot be stored in this type of repository. This repository does not require authentication. Certificates and CRLs are protected by a signature. This type of token is used when the expected set of items is defined by some context.

5.2.2 PKCS #11

PKCS #11 specifies an application programming interface (API), called "Cryptoki," to devices which hold cryptographic information and perform cryptographic functions. Cryptoki, "follows a simple object-based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a "cryptographic token". The standard specifies the data types and functions available to an application requiring cryptographic services using the ANSI C programming language. PKCS #11 tokens can store keys and certificates. Storage of CRLs is not supported. Access to a token is protected by a personal identification number (PIN).

5.2.3 PKCS #12

PKCS #12 is a standard format for exchange of private keys and certificates supported by most browsers and cryptographic applications. It describes a transfer syntax for personal identity information, including private keys, certificates, miscellaneous secrets, and extensions. Machines, applications, browsers, Internet kiosks, and so on, that support this standard will allow a user to import, export, and exercise a single set of personal identity information. This standard supports direct transfer of personal information under several privacy and integrity modes. This token is a soft token and can be retrieved from different media. It contains private keys, certificates, and associated CRLs. The content is protected by a user pass-phrase. The public items (certificates, CRLs) and the private items (keys) can be protected by algorithms with different strengths.

5.2.4 Microsoft CryptoAPI

Microsoft CryptoAPI (MS-CAPI) provides services that enable application developers to add security based on cryptography to applications. MS-CAPI includes functionality for encoding to and decoding from ASN.1, hashing, encrypting and decrypting data, for authentication using digital certificates, and for

managing certificates in certificate stores. Encryption and decryption are provided both using both session keys and with public/private key pairs. MS-CAPI functions use cryptographic service providers (CSP's) to perform encryption and decryption, and to provide key storage and security. These CSP's are independent modules. Ideally, CSP's are written to be independent of a particular application, so that any application will run with a variety of CSP's.

MS-CAPI support is available on Microsoft Windows operating systems only (95/98, NT, 2000, or above). An intermediate system DLL is required which mediates the Java calls to the underlying operating system APIs.

5.2.5 Java KeyStore

Java KeyStore is a database of private keys and their associated certificates or certificate chains. The certificate chains aid in authenticating end entity certificates. The Java Cryptography Architecture (JCA) provides extensible architecture to manage keys. This architecture is embodied in java.security as a KeyStore. The Java KeyStore follows the existing JCA architecture which provides a framework and implementations for a KeyStore.

6. Security Rules

6.1 Operating System

The cryptographic module is dependant on the operating system environment being set up in accordance with FIPS 140-2 specifications. This includes that the host operating system be restricted to a single operator mode. An additional requirement for this cryptographic provider is the availability of a valid commercial grade installation of a Java SDK 1.3.1 or greater JVM.

6.2 Application Usage

The application shall ensure that keys are exchange in a FIPS 140-2 compliant manner The application shall be ensure that cryptographically sensitive material is not inadvertently output over physical ports.

The application shall ensure that:

- SSLv2.0 is not used
- SSLv3.0 is only used with the following cipher suites:
    - SSLv3.0/SSL_RSA_FIPS_WITH_DES_CBC_SHA (0xFEFE)*
    - SSLv3.0/SSL_RSA_FIPS_WITH_3DES_EDE_SHA (0xFEFF)

The application shall ensure that SSLv3.1/ TLS1.0 is only used with the following cipher suites:

- SSLv3.0/SSL_RSA_FIPS_WITH_DES_CBC_SHA (0xFEFE)*
- TLSv1/SSL_RSA_FIPS_WITH_DES_CBC_SHA (0x0009) *
- TLSv1/SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA (0x0011)*
- TLSv1/SSL_DHE_DSS_WITH_DES_CBC_SHA (0x0012)*
- TLSv1/SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0014)*
- TLSv1/SSL_DHE_RSA_WITH_DES_CBC_SHA (0x0015)*
- TLSv1/SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA (0x0019)*
- TLSv1/SSL_DH_anon_WITH_DES_CBC_SHA (0x001A)*
- TLSv1/ SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA (0x0062)*
- TLSv1/SSL_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA (0x0063)*
- SSLv3.0/SSL_RSA_FIPS_WITH_3DES_EDE_SHA (0xFEFF)
- TLSv1/SSL_RSA_FIPS_WITH_3DES_CBC_SHA (0x000A)
- TLSv1/ SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x0013)
- TLSv1/SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
- TLSv1/SSL_DH_anon_WITH_3DES_EDE_CBC_SHA (0x001B)
- TLSv1/ SSL_RSA_WITH_AES_128_CBC_SHA (0x002F)
- TLSv1/SSL_DHE_DSS_WITH_AES_128_CBC_SHA (0x0032)
- TLSv1/ SSL_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
- TLSv1/SSL_DH_anon_DSS_WITH_AES_128_CBC_SHA (0x0034)
- TLSv1/ SSL_RSA_WITH_AES_256_CBC_SHA (0x0035)

- TLSv1/ SSL_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
- TLSv1/SSL_DHE_DSS_WITH_AES_256_CBC_SHA (0x0038)
- TLSv1/ SSL_DH_anon_DSS_WITH_AES_256_CBC_SHA (0x003A)

* Only to be used for backwards compatibility.

6.3 Tokens

All tokens used for storing private cryptographic keys should be password protected. The password should follow generally accepted guidelines for password security. Please note that encryption of keys using a password-based key generation is not FIPS 140-2 approved. For FIPS 140-2 purposes, these values are considered to be in plaintext.

Tokens that are used to supply cryptographic services in addition to key storage are recommended to be FIPS 140-2 Level 2 validated.

All soft tokens should be configured local to the computer.

6.4 Single User Guidelines

The following list provides some high-Level guidelines for configuring a UNIX system for single user. The general idea is the same across all UNIX variants:

1. Remove all login accounts except "root" (the super user)
2. Disable NIS and other name services for users and groups
3. Turn off all remote login, remote command execution, and file transfer daemon

For other operating systems please refer to the appropriate manuals.

7. Notices

MD2, MD5, RC2, RC4, RC5, RC6 and RSA are registered trademarks of RSA Security Inc.

IBM, the IBM logo, and AIX are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered Sun Microsystems, Inc. in the United States, other countries, or both.

HP-UX is a registered trademark Hewlett-Packard Development Company.

AIX, Everyplace, z/OS, AS/400 and IBM are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both.

Pentium and X-Scale are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Red Hat is a trademark of Red Hat, Inc.

SuSE is a registered trademark of SuSE AG.

Other company, product, and service names may be trademarks or service marks of others.