

IBM SSLite in Java Security Policy

IBM SSLite in Java Security Policy

May 2003

Revision: 1.07

NON CONFIDENTIAL

Status: Preliminary

First Edition (January 2003)

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. IBM welcomes your comments on this publication. Please address them to: bluez@zurich.ibm.com. When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2002.

All rights reserved. This document may be freely reproduced and distributed in its entirety and without modification.

JCOP, BlueZ and all BlueZ-based trademarks and logos are trademarks or registered trademarks of International Business Machines Corp. in the US and other countries. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems in the US and other countries.



1. Document Information

1.1. Document Scope

This document describes the services that the IBM SSLite in Java Package provides to a population of security officers, and the security policy governing access to those services. Included is a description of the basic security requirements for the SSLite package and a qualitative description of how each of the security requirements is achieved.

1.2. Table of Contents

1.	<i>Document Information</i>	2
1.1.	Document History	2
1.2.	Document Scope	2
1.3.	Table of Contents	2
2.	<i>Applicable documents</i>	4
2.1.	Cryptography	4
2.2.	Protocols	4
3.	<i>SSLite Package</i>	5
3.1.	Module Components	5
4.	<i>Security Levels</i>	5
5.	<i>Cryptographic Module Specification</i>	7
5.1.	SSLite token interfaces	8
5.2.	Cryptographic Standards	8
5.3.	Module Interfaces	10
5.4.	Cryptographic Module Self Tests	10
	Operational Environment	11
5.6.	Module Status	11
6.	<i>Roles and Services</i>	11
6.1.	Roles	11
6.2.	Services	12
7.	<i>Cryptographically Sensitive Material</i>	12
7.1.	Cryptographic Keys	12
8.	<i>Security Rules</i>	15
9.	<i>Notices</i>	17

2. Applicable documents

2.1. Cryptography

RSA Laboratories PKCS #15 v1.0: Cryptographic Token Information Format Standard – April 23, 1999
RSA Laboratories PKCS#15 v1.0 Amendment 1 Draft #1 - October 20, 1999
FIPS 140-2 standard, the *Derived Test Requirements*, and on-line implementation guidelines
Digital Encryption Standard: FIPS PUB 46-3, FIPS PUB 74, and FIPS PUB 81
SHA-1: FIPS PUB 180-1
Digital Signature Standard : FIPS PUB 186-2 27 January 2000
Pseudo-random Number Generation: Appendix 3 of FIPS PUB 186.
Digital Signature Scheme Giving Message Recovery: ISO/IEC 9796
The 3DES standard, ANSI X9.52, Triple Data Encryption Algorithm Modes Of Operation
Advanced Encryption Standard (AES) FIPS Publication 197, November 26, 2001
Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry. ANSI X9.31-1998

2.2. Protocols

The TLS Protocol Version 1.0 RFC 2246, January 1999



3. SSLite Package

3.1. Module Components

The following table lists the module components:

Type	Name	Release	Date	Delivery
Software	SSLite JAR file sslite140.jar	3.15.3232 (FIPS140/ Prod)		
Documentation	SSLite Java Doc	3.15.3232 (FIPS140/ Prod)		

Table 1a: Module Component List for all platforms

3.1.1. Module Description

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, in Java. For the purpose of FIPS 140-2 Level 1 validation the implementation is made available in the form of a signed Jar file. The cryptographic functions used in SSLite are implemented in the FIPS 140-2 validated IBM CryptoLite in Java (See FIPS Certificate #354) module, which is embedded in the SSLite module.

The SSLite package performance is comparable to that of native implementations. The package includes support of PKCS#11 cryptographic tokens (smart cards, etc.), certificate revocation lists (CRL), PKCS#7, PKCS#12, Java KeyStore key and certificate repositories, and PKIX infrastructure.

4. Security Levels

The IBM SSLite package meets the overall requirements applicable to Level 1 security of FIPS 140-2. 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 2: FIPS 140-2 validation levels



5. Cryptographic Module Specification

IBM SSLite is classified as a multi-chip standalone 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. IBM 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 IBM 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 1.1. The above requirements have been demonstrated by testing and validating the IBM SSLite package on the following platforms.

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 3: Platforms on which CryptoLite has been tested

5.1. SSLite token interfaces

The SSLite module provides cryptography services through the embedded IBM CryptoLite in Java module. IBM CryptoLite in Java contains a proprietary interface for interfacing to an external cryptographic acceleration module. This module, also provided by IBM and FIPS140-2 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 startup time. This validity and integrity of the booster module is checked using an approved HMAC method.

The IBM SSLite modules also contains industry standard MSCAPI and PKCS#11 token interfaces.

5.2. Cryptographic Standards

The IBM SSLite module supports the following approved and non approved FIPS algorithms through the IBM CryptoLite in Java module.

HASH Functions

Algorithm	Specification	FIPS Approved
MD2	IETF RFC1319 Hash algorithm; hash size: 16 bytes; block size: 16 bytes.	No

	Used only for backward compatibility.	
MD5	IETF RFC 1321 Hash algorithm; hash size: 16 bytes; block size: 64 bytes. Used only for backward compatibility.	No
SHA-1	FIPS180-1 Hash algorithm; hash size: 20 bytes; block size: 64 bytes.	Yes
SHA256	Hash algorithm; hash/block sizes: 32/64, bytes.	No

Table 4: Hash Functions

CIPHER Functions

Algorithm	Specification & Description	FIPS Approved
RC2	IETF: RFC2268 Symmetric block cipher. Block size 8 bytes. Key size 0-1024 bits. RC2 allows adjustment of the effective key strength independent of the input key length.	No
RC4	Stream cipher; key sizes: 0-2048 bits.	No
DES, DES-CBC	FIPS 46-3 Symmetric block cipher; block size: 8 bytes; key size: 56 bits. For legacy systems only.	Yes
3DES, 3DES-CBC	FIPS 46-3 Triple DES has 112/168 bits key length depending on type of key.	Yes
AES AES CBC AES 256	FIPS197, Symmetric block cipher; block sizes: 16,24,32 bytes; key sizes: 16,24,32 bytes.	Yes
HMAC SHA-1	Hashed Message Authentication Codes (HMAC) based on the SHA-1 hash algorithm.	Yes

Table 5: Cipher Functions

Public Key

Algorithm	Specification	FIPS Approved
RSA Sign/Verify	Public key encryption/signature scheme. Typical key/data sizes: 512, 768, 1024 (typical), 2048 bits.	Yes
RSA Encrypt/Decrypt	RSA specification and padding scheme: PKCS#1 OAEP Padding scheme for RSA encryption: RFC2437	No
DSA Sign/Verify	Public key signature scheme. Cannot be used for encryption. Key sizes: 512-1024 bits in steps of 64 bits.	Yes
Diffie-Hellman (DH)	Public key crypto system. Typical key/data sizes: 512, 768, 1024 (typical), 2048 bits. Used for key agreement.	No

Table 6: Public Key Functions



Random Number Generators

Algorithm	Specification	FIPS Approved
PSEUDO Random Number Generator	FIPS 186-2 ANSI X9.31 1998	Yes
Universal Software Based True Random Number Generator	Patented by IBM, EC Pat.No. EP1081591A2, True random number generator that works reliably on variety of platforms without exploiting platform specific features. Entropy evaluation through statistical analysis. Performance: 20-1000 bits/seconds. (Used to seed the Approved PRNG in FIPS mode)	No

Table 7: Random Number Generators

5.3. Module Interfaces

As a multi-chip standalone 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 Application Program Interface (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 returns 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.

5.4. Cryptographic Module Self Tests

The SSLite relies exclusively on the embedded IBM CryptoLite in Java 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. These tests include a continuous random number generator test and pair-wise consistency tests of the generated RSA keys.

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 comprise of the software integrity test and the known answer tests 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 a HMAC of 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:

- DES KAT
- 3DES KAT
- AES KAT
- SHA KAT
- SHA256 KAT
- RSA_SIGN/VERIFICATION
- DSA_PARAMETER GENERATION
- DSA_SIGN/VERIFICATION
- DIFFIE-HELLMAN
- RNG KAT



Startup Recovery

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

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 appropriate manner, for example by retrying the PRNG service.

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 appropriate manner, for example by retrying the key generation service.

5.5. 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 (Version 1.11.1p1) 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 in a FIPS compliant mode, only those FIPS approved algorithms are used.

5.6. 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.

6. Roles and Services

6.1. Roles

The IBM SSLite module supports two roles, a cryptographic officer role and a user role.

- **ROLE_CO:** The Cryptographic 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 8: Roles and Required Identification and Authentication

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

Table 9: Strengths of Authentication Mechanisms

6.2. Services

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

Service	User Role
Certification authority services (com.ibm.SSLite.CA)	Yes
Certificate services (com.ibm.SSLite.CE)	Yes
Certificate request entry services (com.ibm.SSLite.CRE)	Yes
Certificate extension services (com.ibm.SSLite.Extension)	Yes
Lightweight directory access protocol services (com.ibm.SSLite.LDAP)	Yes
Public key infrastructure Services (com.ibm.SSLite.PKI)	Yes
S/MIME services (com.ibm.SSLite.SMIME)	Yes
Certificate revocation list services (com.ibm.SSLite.SSLCRL)	Yes
X509 Version 3 Certificate services (com.ibm.SSLite.SSLCert)	Yes
SSL Context services (com.ibm.SSLite.SSLContext) -	Yes
Java version 2 KeyStore token services (com.ibm.SSLite.SSLKSToken)	Yes
Microsoft CryptoAPI 2.0 token services (com.ibm.SSLite.SSLMSCAPIToken)	Yes
X.500 distinguished name services (com.ibm.SSLite.SSLName)	Yes
PKCS11 token services (com.ibm.SSLite.SSLPKCS11Token)	Yes
PKCS12 token services (com.ibm.SSLite.SSLPKCS12Token)	Yes
PKCS7 token services (com.ibm.SSLite.SSLPKCS7Token)	Yes
SSL socket connection socket services (com.ibm.SSLite.SSLServerSocket)	Yes
SSL session services (com.ibm.SSLite.SSLSession)	Yes
SSL stream socket protocol services (com.ibm.SSLite.SSLSocket)	Yes
SSL general token services (com.ibm.SSLite.SSLToken)	Yes
Signed jar file verification services (com.ibm.SSLite.SignedJarInputStream)	Yes
Signed jar file generation services (com.ibm.SSLite.SignedJarOutputStream)	Yes
X509 certificate name extensions (com.ibm.SSLite.XAltName)	Yes
X509 basic constraints extensions (com.ibm.SSLite.XBasic Constraints)	Yes
X509 key usage extensions (com.ibm.SSLite.XKeyUsage)	Yes
X509 extended key usage extensions (com.ibm.SSLite.XExtKeyUsage)	Yes

7. Cryptographically Sensitive Material

7.1. Cryptographic Keys

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 a responsibility of the application program developers to ensure FIPS140-2 compliance of key storing techniques they implement.

Key establishment

SSLite provides protocol services for SSLv2.0, SSLv3.0 and SSLv3.1/TLS1.0. 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 compliant. There is an exception for 2 SSLv3.0 cipher suites 0xFEFE and 0xFEFF where SSLite will use the SSLv3.1/TLS1.0 method for key generation which is based on SHA-1.

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.

Key Generation

Key generation is handled using the IBM CryptoLite subsystem which uses a the FIPS approved RNG algorithm which is 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.

Key zeroization

Key objects are normally zeroed and any associated data discarded when the key object is garbage collected through the finalizer method. The IBM CryptoLite sub-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.

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.

SSLite supports the following token types.

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.

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).

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

Microsoft CryptoAPI (an application programming interface) provides services that enable application developers to add security based on cryptography to applications. CryptoAPI 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. CryptoAPI 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.

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

Java KeyStore is 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.



8. Security Rules

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.

Application Usage

The application shall ensure that keys are exchange in a FIPS 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

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

- o SSLv3.0/SSL_RSA_FIPS_WITH_DES_CBC_SHA (0xFEFE)*
- o 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:

SSLite DES based FIPS approved supported algorithms

- o SSLv3.0/SSL_RSA_FIPS_WITH_DES_CBC_SHA (0xFEFE)*
- o TLSv1/SSL_RSA_FIPS_WITH_DES_CBC_SHA (0x0009) *
- o TLSv1/SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA (0x0011)*
- o TLSv1/SSL_DHE_DSS_WITH_DES_CBC_SHA (0x0012)*
- o TLSv1/SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0014)*
- o TLSv1/SSL_DHE_RSA_WITH_DES_CBC_SHA (0x0015)*
- o TLSv1/SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA (0x0019)*
- o TLSv1/SSL_DH_anon_WITH_DES_CBC_SHA (0x001A)*
- o TLSv1/ SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA (0x0062)*
- o TLSv1/SSL_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA (0x0063)*

SSLite TDES based FIPS approved supported algorithms

- o SSLv3.0/SSL_RSA_FIPS_WITH_3DES_EDE_SHA (0xFEFF)
- o TLSv1/SSL_RSA_FIPS_WITH_3DES_CBC_SHA (0x000A)
- o TLSv1/ SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x0013)
- o TLSv1/SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
- o TLSv1/SSL_DH_anon_WITH_3DES_EDE_CBC_SHA (0x001B)

SSLite AES128 based FIPS approved supported algorithms

- o TLSv1/ SSL_RSA_WITH_AES_128_CBC_SHA (0x002F)
- o TLSv1/SSL_DHE_DSS_WITH_AES_128_CBC_SHA (0x0032)
- o TLSv1/ SSL_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
- o TLSv1/SSL_DH_anon_DSS_WITH_AES_128_CBC_SHA (0x0034)

SSLite AES256 based FIPS approved supported algorithms

- o TLSv1/ SSL_RSA_WITH_AES_256_CBC_SHA (0x0035)
- o TLSv1/ SSL_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
- o TLSv1/SSL_DHE_DSS_WITH_AES_256_CBC_SHA (0x0038)
- o TLSv1/ SSL_DH_anon_DSS_WITH_AES_256_CBC_SHA (0x003A)

* = Only to be used for backwards compatibility

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 FIP S Approved. For FIPS purposes, these values are considered to be in plaintext.

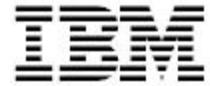
Tokens that are used to supply cryptographic services in addition to key storage are recommended to be FIPS140 level 2 validated.

All soft tokens should be configured local to the computer.

Single User Guidelines

The following explains how to configure a Unix system for single user. The general idea is the same across all Unix variants:

- Remove all login accounts except "root" (the superuser).
- Disable NIS and other name services for users and groups.
- Turn off all remote login, remote command execution, and file transfer daemons.



9. Notices

AIX, Everyplace, 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.

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

© 2002 International Business Machines Corporation. All rights reserved.