

FIPS 140-2 Security Policy

SIEMENS PLM Software Teamcenter Cryptographic Module

SIEMENS PLM Software
5800 Granite Parkway, Suite 600
Plano, TX 75024
USA

Jun 10, 2014

Document Version 2.0

Based on OpenSSL

This product includes cryptographic software written by

Eric Young (eay@cryptsoft.com)
Tim Hudson (tjh@cryptsoft.com)

The Siemens logo, consisting of the word "SIEMENS" in a bold, teal, sans-serif font.

Non-Proprietary and Unrestricted:

This document contains information that is non-proprietary to Siemens PLM Software Inc. and its use is unrestricted.

Trademarks:

Siemens and the Siemens logo are registered trademarks of Siemens AG. Teamcenter is a trademark or registered trademark of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in the United States and in other countries. All other trademarks, registered trademarks, or service marks belong to their respective holders.

FIPS 140-2 Security Policy

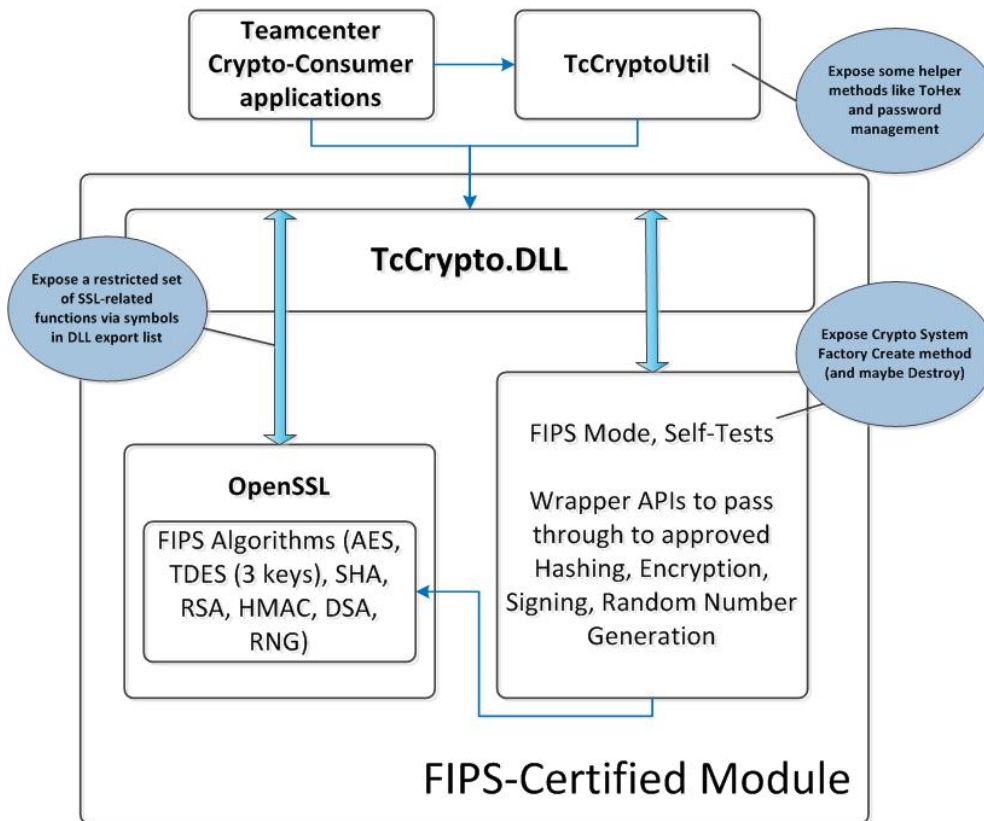
SIEMENS PLM Software Teamcenter Cryptographic Module

1. Introduction

The following describes the security policy for the SIEMENS PLM Software Teamcenter Cryptographic Module (TCM). This module provides FIPS-validated encryption, hashing, digital signatures, random number generation, and Secure Sockets Layer (SSL) / Transport Layer Security (TLS) encryption for HTTPS. This Security policy contains the details for both TCM version 1.1.1 and TCM version 2.0.

The TCM is a software module that is dynamically linked as a DLL by Teamcenter applications that require cryptographic capabilities. Any Teamcenter product that uses this library as its sole source of cryptographic functionality and that has adhered to the guidelines of this document may be operated in a FIPS-compliant manner.

FIPS Library Architecture



1.1. Purpose

This document covers the secure operation of the TCM including the initialization, roles, and responsibilities of operating the product in a secure, FIPS-compliant manner.

1.2. References

OpenSSL	http://www.openssl.org/

1.3. Glossary

Term/Acronym	Description
TCM	Teamcenter Cryptographic Module
CO	Cryptographic-officer or Crypto-officer

2. Roles, Services, and Authentication

The TCM provides a single role and a set of services. The TCM will start up with an application calling an initialize function, and then provides cryptographic capabilities on behalf of the user.

2.1. Roles

All operations occur on behalf of the application running operations for the user of the application software. For a complete description of all services, please see the Teamcenter Cryptographic Module Application Program Interface documentation (TCM API Guide).

The TCM supports both User and Crypto-officer roles. Both of these roles have access to all services of the TCM.

2.2. Authentication Mechanisms and Strength

No authentication is performed, since the TCM simply provides cryptographic primitives for use by higher-level Teamcenter applications.

2.3. Algorithms

2.3.1. Approved

Algorithm	TCM Version 2.0 (Cert#)	TCM Version 1.1.1 (Cert#)
Triple-DES (3-Keys)	1694	443
AES	2834	410
SHA	2376	477
HMAC(key length \geq 112 bits)	1776	183
RNG	1277	204
RSA(key length \geq 2048 bits & SHA \geq 224 bits)	1476	150
DSA(Signature Verification)	852	170

2.3.2. Non-Approved (non-FIPS mode only)

- MD5
- DSA (Signature Generation)
- DES
- HMAC (key length $<$ 112 bits)
- RSA (key length $<$ 2048 bits & SHA-1)






2.4. Exportable Functions











The following table contains a list of the exportable functions that provide access to the TCM.









Notes:

- The functions prefixed with *TcCrypto_System_* control the operation of the library (these are listed last in the table below).
- The library may be initialized in a FIPS or non-FIPS (Domestic) mode. The table below is a complete list of functions, some of which have restricted functionality when the library is operating in a FIPS-compliant mode.
- The library also exports all of the symbols from the OpenSSL library, except for a limited set that would allow too much control over the operation of the library.
 - Non-exported OpenSSL symbols: `BIO_set_cipher`, `ENGINE_set_ciphers`, `SSL_CTX_new`, `SSL_CTX_set_cipher_list`, `SSL_CTX_set_ssl_version`, `SSL_set_cipher_list`, `SSL_set_ssl_method`, `RAND_cleanup`, `EVP_cleanup`, `ERR_free_strings`, `CRYPTO_cleanup_all_ex_data`
 - All other OpenSSL functions are exported. For information on the OpenSSL functionality, please visit <http://www.openssl.org/> and read the documentation.






Public Instance Methods

 TcCrypto_Asymmetric_GetContext	Called to initialize a new TcCryptoContext for use in asymmetric (public / private key) encryption.
 TcCrypto_Asymmetric_GetOutputSize	Can be called after TcCrypto_Asymmetric_SetPublicKey to determine the maximum buffer size needed to hold encrypted data.
 TcCrypto_Asymmetric_PrivateKey_Decrypt	Performs decryption of a value encrypted with TcCrypto_Asymmetric_PublicKey_Encrypt . This should only be used to decrypt crypto keys, not data.
 TcCrypto_Asymmetric_PublicKey_Encrypt	Performs asymmetric encryption, using the public key. Only the recipient of the private key will be able to decrypt the resulting message with with TcCrypto_Asymmetric_PrivateKey_Decrypt . This should only be used to encrypt crypto keys, not data.
 TcCrypto_Asymmetric_SetPrivateKey	Loads the TcCryptoContext object with a private key and password to decrypt the private key, in preparation for decrypting data encrypted via a call to TcCrypto_Asymmetric_PublicKey_Encrypt

 TcCrypto_Asymmetric_SetPublicKey	Loads the TcCryptoContext object with a public key, in preparation for data encryption.
 TcCrypto_Cipher_Final	Finishes the encryption or decryption operation. This effectively flushes any buffered data from a partial final block into the output buffer, and shuts down the cipher.
 TcCrypto_Cipher_GetBlockLength	Returns the cipher block length, which is 8 for DES or 3DES, and 16 for AES.
 TcCrypto_Cipher_GetContext	Creates a TcCryptoContext that can be used for encryption. The cipher associated with the requested <code>cipherType</code> parameter may not always be available, depending on the system mode (FIPS, Domestic).
 TcCrypto_Cipher_Init	Sets the key and initialization vector (IV) to be used by the cipher, as well as configuring whether the cipher is set to encrypt or decrypt data that is processed.
 TcCrypto_Cipher_SetPadding	Allows cipher padding to be turned on or off.
 TcCrypto_Cipher_Update	Called to encrypt or decrypt data (based on how the cipher was configured in TcCrypto_Cipher_Init). Can be called one or more times to encrypt data into the output buffer.
 TcCrypto_Digest_Final	Called after all data to hash has been passed through TcCrypto_Digest_Update . This produces a hash, which is a unique bit pattern based on the bits of data passed through TcCrypto_Digest_Update . If even a single bit of input changes, the entire output value will change in approximately 50% of the bits.
 TcCrypto_Digest_GetContext	Called to initialize a new TcCryptoContext for use in hashing. The context contains hash-related state.
 TcCrypto_Digest_Init	Initializes the digest state to prepare it for computing a new hash value via TcCrypto_Digest_Update .
 TcCrypto_Digest_Update	Processes one or more blocks of data through the hash function. This method can be called one or more times with all the data that is to be hashed.
 TcCrypto_Hmac_Final	Called after all data to hash has been passed through TcCrypto_Hmac_Update . This produces

	<p>an HMAC hash, which is a unique bit pattern based on the bits of data passed through <code>TcCrypto_Hmac_Update</code>, using the secret HMAC key supplied in <code>TcCrypto_Hmac_Init</code>.</p>
<p> TcCrypto_Hmac_GetContext</p>	<p>Called to initialize a new <code>TcCryptoContext</code> for use in keyed hashing, using the HMAC construct defined in RFC 2104, http://www.faqs.org/rfcs/rfc2104.html.</p>
<p> TcCrypto_Hmac_Init</p>	<p>Initializes the hmac and underlying digest state to prepare it for computing a new hash value via <code>TcCrypto_Hmac_Update</code>.</p>
<p> TcCrypto_Hmac_Update</p>	<p>Processes one or more blocks of data through the HMAC function. This method can be called one or more times with data that is to be hashed.</p>
<p> TcCrypto_Legacy_NTLM_DES_ecb_encrypt</p>	<p>Legacy method. Should only be used to support NTLM headers in applications (like <code>cUtl</code>) that require legacy NTLM support.</p>
<p> TcCrypto_Legacy_NTLM_DES_set_key</p>	<p>Legacy method. Should only be used to support NTLM headers in applications (like <code>cUtl</code>) that require legacy NTLM support.</p>
<p> TcCrypto_Legacy_NTLM_DES_set_odd_parity</p>	<p>Legacy method. Should only be used to support NTLM headers in applications (like <code>cUtl</code>) that require legacy NTLM support.</p>
<p> TcCrypto_Rand_GetBytes</p>	<p>Uses a cryptographically strong random number generation technique to produce as many random bytes as requested.</p>
<p> TcCrypto_Rand_Init</p>	<p>Initializes the random number subsystem. This operation can take a bit, while the random number generator 'warms' up, accumulating entropy.</p>
<p> TcCrypto_Rand_Seed</p>	<p>Accumulates variable data into the Pseudo-Random Number Generator (PRNG), allowing the output to become more random.</p>
<p> TcCrypto_Rand_SetPrngKey</p>	<p>Loads an application-specific key into the Pseudo-Random Number Generator (PRNG).</p>
<p> TcCrypto_Rand_Status</p>	<p>Indicates if the random number generator has been seeded with enough entropy so that it can be used to generate random output bytes.</p>

 TcCrypto_Signature_GetContext	Called to initialize a new TcCryptoContext for use in digital signatures using the RSA or DSA algorithms.
 TcCrypto_Signature_SetPrivateKey	Loads the TcCryptoContext object with a private key and password to decrypt the private key, in preparation for computing a digital signature.
 TcCrypto_Signature_SetPublicKey	Loads the TcCryptoContext object with a public key, in preparation for verifying a digital signature.
 TcCrypto_Signature_SignFinal	Called after all data has been signed using TcCrypto_Signature_SignUpdate , producing the final digital signature.
 TcCrypto_Signature_SignInit	Initializes the TcCryptoContext to prepare it for computing a digital signature.
 TcCrypto_Signature_SignUpdate	Called one or more times to supply data that is to be 'signed'.
 TcCrypto_Signature_VerifyFinal	Called after all data has been verified using TcCrypto_Signature_SignUpdate , producing the final digital signature. If this value matches a pre-computed digital signature, then the data is valid.
 TcCrypto_Signature_VerifyInit	Initializes the TcCryptoContext to prepare it for verifying a pre-computed digital signature.
 TcCrypto_Signature_VerifyUpdate	Called one or more times to supply data that is to be 'verified'.
 TcCrypto_SSL_CTX_new	Creates a new SSL context which can be used to create an SSL connection info.
 TcCrypto_System_FreeContext	Releases the resources associated with a TcCryptoContext , freeing any allocated memory and zeroing out the internal state.
 TcCrypto_System_GetDigestSize	Returns the number of bytes needed to store the output digest from the hash function that has been initialized via TcCrypto_Digest_GetContext , TcCrypto_Hmac_GetContext , or TcCrypto_Signature_GetContext
 TcCrypto_System_GetLastError	Retrieves information about the last error that occurred for the system.

<p> TcCrypto_System_GetType</p>	<p>Retrieves an enum that indicates the type of system that is currently allocated.</p>
<p> TcCrypto_System_GetTypeString</p>	<p>Retrieves a string that indicates the type of system that is currently allocated.</p>
<p> TcCrypto_System_Initialize</p>	<p>Factory method to create a crypto system. The system contains the mode the library will run in ("Fips", "Domestic"), as well as the random number generator state.</p>
<p> TcCrypto_System_SetLastError</p>	<p>Sets the thread's last error state, retrievable by TcCrypto_System_GetLastError.</p>
<p> TcCrypto_System_Shutdown</p>	<p>Shutdown releases any allocated memory and frees up system resources consumed by the crypto library. This should be called before the application exits. Note that only the first call to this method will have an affect.</p>
<p>TcCrypto_Watermark_Generate TcCrypto_Watermark_GetData TcCrypto_Watermark_IsValid</p>	<p>These functions are not intended to be called by any consumers of this library – they are used internally and exported for use by our watermarking utility, which is beyond the scope of this document.</p>

3. Secure Operation and Security Rules

In order to operate the Teamcenter Cryptographic Module securely, the operator should be aware of the security rules enforced by the module and should adhere to the physical security rules and secure operation rules required.

3.1. Security Rules

The security rules enforced by the TCM result from the security requirements of FIPS 140-2.

FIPS 140-2 Security Rules

The following are security rules needed to operate the module securely, that stem from the requirements of FIPS PUB 140-2. The module enforces these requirements when initialized into FIPS mode.

1. When initialized to operate in FIPS mode, the TCM shall only use FIPS-approved cryptographic algorithms.
2. The TCM shall employ the FIPS-approved pseudo random number generator based on ANSI931 Standard (TDES2 Algorithm) whenever generating keys.
3. The replacement or modification of the Module by unauthorized intruders is prohibited.
4. The Operating System enforces authentication method(s) to prevent unauthorized access to Module services.
5. All Critical Security Parameters are verified as correct and are securely generated, stored, and destroyed.
6. All host system components that can contain sensitive cryptographic data (main memory, system bus, disk storage) must be located in a secure environment.
7. The referencing application accessing the Module runs in a separate virtual address space with a separate copy of the executable code.
8. The unauthorized reading, writing, or modification of the address space of the Module is prohibited.
9. The writable memory areas of the Module (data and stack segments) are accessible only by a single application so that the Module is in "single user" mode, i.e. only the one application has access to that instance of the Module.
10. The operating system is responsible for multitasking operations so that other processes cannot access the address space of the process containing the Module.

3.2. Secure Operation Initialization Rules

Because FIPS 140-2 prohibits the use of non-FIPS approved algorithms while operating in a FIPS compliant manner, the TCM should be initialized to ensure FIPS level 1 compliance.

1. Start a Teamcenter application that uses the TCM
2. When the TCM enters the Uninitialized state, the application should initialize the TCM using `TcCrypto_System_Initialize(TcCrypto_SystemType_Fips)`.
3. The application should check the return code to ensure the application initialization was successful.

When initialized in this fashion, the TCM will only use FIPS-approved algorithms. Note that the state of an TCM can be determined at any time by calling the `TcCrypto_System_GetType()` function, which will return `TcCrypto_SystemType_Fips` if the TCM is operating in FIPS mode.

Note that when configuring the random number generator, that the seed and seek key should not be the same value.

3.3. Operating Systems

The TCM has been officially validated on the following platforms:

TCM Version 2.0

- Windows 7 SP1 (x86 / x64)
 - Visual Studio 2012 (11)
- Linux SuSE 11.2 (x64)
 - Compiler – g++ 4.3.4
- Mac OSX 10.8 (x64)
 - Compiler - clang LLVM 4.2

TCM Version 1.1.1

- Windows XP (32-bit)
 - Visual Studio 2003 (7.1)
- Solaris 8 (64-bit)
 - Sun WorkShop 6 update 2 C/C++ 5.3
- Solaris 10 (64-bit)
 - Sun Studio 10 C++ 5.7

In addition to the validation, the TCM has been tested by SIEMENS PLM on the following platforms:

TCM Version 2.0

- Windows
 - Windows 8 (x64) – Visual Studio 2012
 - Windows 7 SP1 (x64/x86) – Visual Studio 2010
- Linux
 - RedHat 6.x (x64/x86) – g++ 4.4.4
 - SUSE 10.x (x64) – g++ 4.1.2
- Solaris
 - Solaris 10 (32-bit /64-bit) – Solaris Studio 12.3 C++ 5.12
- Mac OS X
 - OSX 10.8 (x86) – clang LLVM 4.2
- AIX
 - AIX 5.3 (32-bit) – xlc 8.0

- AIX 5.3 (64-bit) – xLC 8.0
- AIX 6.0 (32-bit) – xLC 11.1
- AIX 6.0 (64-bit) – xLC 11.1

TCM Version 1.1.1

- Windows
 - Windows XP (32-bit) – Visual Studio 2003 (7.1)
 - Windows XP (32-bit) – Visual Studio 2005 (8.0)
 - Windows XP (64-bit) – Visual Studio 2005 (8.0)
- Linux
 - SUSE 9 (32-bit: i386) – gcc 3.3.3
 - SUSE 9 (64-bit: x86_64) – gcc 3.3.3
- HP-UX
 - HP-UX 11.11 (32-bit: PA-RISC) – aCC 03.57
 - HP-UX 11.11 (64-bit: PA-RISC) – aCC 03.57
 - HP-UX 11.23 (32-bit: Itanium) – aC++/C A.06.05
 - HP-UX 11.23 (64-bit: Itanium) – aC++/C A.06.05
- Solaris
 - Solaris 8 (32-bit) – Sun WorkShop 6 update 2 C++ 5.3
 - Solaris 8 (64-bit) – Sun WorkShop 6 update 2 C++ 5.3
- Mac OS X
 - OSX 10.4.6 (32-bit: ppc) – gcc 4.0.1
 - OSX 10.4.6 (32-bit: i386) – gcc 4.0.1
 - OSX 10.4.6 (64-bit: ppc64) – gcc 4.0.1
 - OSX 10.4.6 (32-bit: universal pcc/i386) – gcc 4.0.1
- AIX
 - AIX 5.1 (32-bit) – 6.0.0.11 C++ compiler, 6.0.0.10 C compiler
 - AIX 5.1 (64-bit) – 6.0.0.11 C++ compiler, 6.0.0.10 C compiler
 -
- IRIX
 - IRIX 6.5.22m (32-bit) – c/c++ compiler 7.4.2m
 - IRIX 6.5.22m (64-bit) – c/c++ compiler 7.4.2m

Definition of SRDIs Modes of Access

This section specifies the TCM's Security Relevant Data Items as well as the access control policy enforced by the TCM.

3.4. Cryptographic Keys, CSPs, and SRDIs

While operating in a level 1 FIPS-compliant manner, the TCM stores no security relevant data items. Any security relevant data, like cryptographic keys, cipher state, etc, are fully contained in memory provided by the calling application, and thus not under control of the TCM. All such memory is under control (stored by) higher-level applications. No actual cryptographic items are stored in the TCM, although the TCM does provide mechanisms to zero out memory once the calling application is finished using the memory (`OPENSSL_cleanse()`) and calls the appropriate shutdown methods of the API (`TcCrypto_System_Shutdown()`).

There are no cryptographic keys provided with the TCM. The operator must generate or otherwise provide any keys to be used during operation.

3.5. Access Control Policy

Access control is assumed to be handled by higher-level applications and the operating system, since the TCM has no mechanisms to restrict or limit calls to the APIs. The only access control is protection around the internal FIPS-state variable, which ensures that once the application is switched into FIPS mode, it cannot be switched out of FIPS mode without first going through a shutdown operation. The application can then be re-initialized in a non-FIPS mode.

3.6. Self-tests

The following list shows all the self-tests implemented in the cryptographic module.

```
FIPS_selftest_rng()
FIPS_selftest_sha1() //supported only in non-FIPS mode
FIPS_selftest_sha224()
FIPS_selftest_sha256()
FIPS_selftest_sha384()
FIPS_selftest_sha512()
FIPS_selftest_hmac()
FIPS_selftest_aes()
FIPS_selftest_des() // includes DES, 2-key 3DES, and 3-key 3DES tests
FIPS_selftest_rsa()
FIPS_selftest_dsa()
```

In addition to these self-tests, the TCM also contains an embedded watermark that will be verified at runtime to ensure that the library has not been corrupted or modified.

Also, the library performs continuous Random Number Generator tests on the output of the Approved RNG to ensure that it is not “stuck”.

Mitigation of Other Attacks

This section is not applicable.