

# **FIPS 140-2 Security Policy**

## **UGS Teamcenter Cryptographic Module**

---

UGS Corp  
5800 Granite Parkway, Suite 600  
Plano, TX 75024  
USA

May 18, 2007

Version 1.3

containing OpenSSL library source code

This product includes cryptographic software written by

Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com))  
Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com))



# FIPS 140-2 Security Policy

## UGS Corp Teamcenter Cryptographic Module

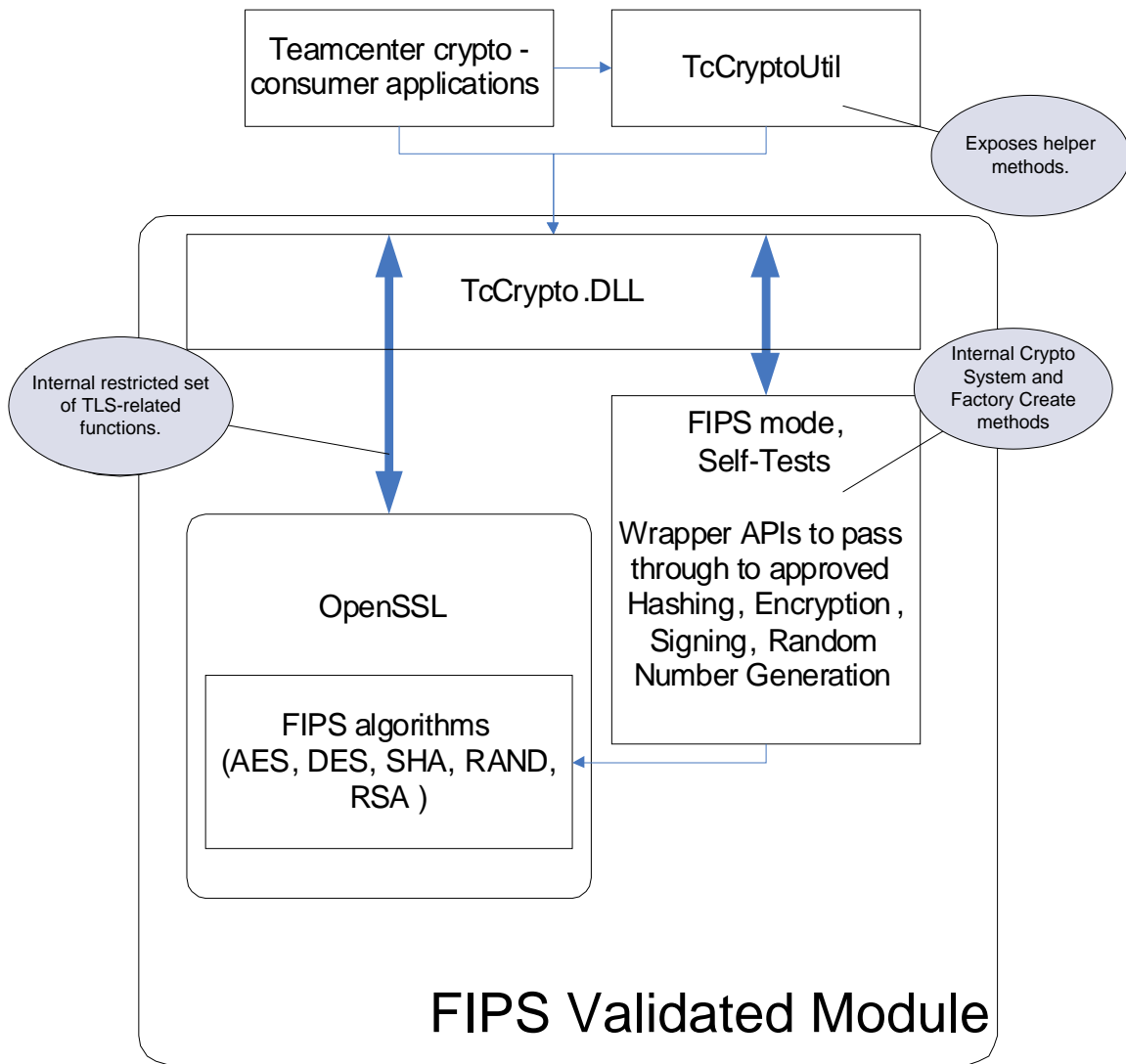
### 1. Introduction

The following describes the security policy for the UGS Corp 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.

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.

While the UGS Teamcenter Cryptographic Module incorporates source code from OpenSSL, the TCM does not use OpenSSL's FIPS 140-2 validation. Rather, the TCM represents a separate, independent validation that does not depend upon any other validation.

# 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	<a href="http://www.openssl.org/">http://www.openssl.org/</a>

This document may be reproduced only in its original entirety (without revision).

### 1.3. Glossary

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

## 2. Roles, Services, and Authentication

The TCM provides two roles and a set of services (common to both roles). 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

- Triple-DES (Cert. #443)
- AES (Cert. #410)
- SHS (Cert. #477, all sizes)
- HMAC (Cert. #183, all SHS sizes)
- RNG (Cert. #204)
- DSA (Cert. #170)
- RSA (Cert. #150)

#### 2.3.2. Non-Approved

- DES (single key TDES, non-compliant)
- MD5 (allowed as part of TLS, but non-FIPS mode only otherwise)
- Diffie-Hellman (key agreement; key establishment methodology provides between 80 and 150 bits of encryption strength)
- RSA (key wrapping; key establishment methodology provides between 80 and 150 bits of encryption strength)

### 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).




This document may be reproduced only in its original entirety (without revision).

- 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 are not required.
  - 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

 <a href="#">TcCrypto_Asymmetric_GetContext</a>	Called to initialize a new <a href="#">TcCryptoContext</a> for use in asymmetric (public / private key) encryption.
 <a href="#">TcCrypto_Asymmetric_GetOutputSize</a>	Can be called after <a href="#">TcCrypto_Asymmetric_SetPublicKey</a> to determine the maximum buffer size needed to hold encrypted data.
 <a href="#">TcCrypto_Asymmetric_PrivateKey_Decrypt</a>	Performs decryption of a value encrypted with <a href="#">TcCrypto_Asymmetric_PublicKey_Encrypt</a> . <b>This should only be used to decrypt crypto keys, not data.</b>
 <a href="#">TcCrypto_Asymmetric_PublicKey_Encrypt</a>	Performs asymmetric encryption, using the public key. Only the recipient of the private key will be able to decrypt the resulting message with with <a href="#">TcCrypto_Asymmetric_PrivateKey_Decrypt</a> . <b>This should only be used to encrypt crypto keys, not data.</b>
 <a href="#">TcCrypto_Asymmetric_SetPrivateKey</a>	Loads the <a href="#">TcCryptoContext</a> object with a PKCS#8 private key, in preparation for decrypting data encrypted via a call to <a href="#">TcCrypto_Asymmetric_PublicKey_Encrypt</a>  FIPS 140-2 considers this private key to be in plaintext form, as FIPS does not allow password-derived keys to be used for data encryption.
 <a href="#">TcCrypto_Asymmetric_SetPublicKey</a>	Loads the <a href="#">TcCryptoContext</a> object with a public key, in preparation for data encryption.
 <a href="#">TcCrypto_Cipher_Final</a>	Finishes the encryption or decryption operation. This effectively flushes any buffered data from a partial final block into the output buffer, and

This document may be reproduced only in its original entirety (without revision).




	shuts down the cipher.
 <a href="#">TcCrypto_Cipher_GetBlockLength</a>	Returns the cipher block length, which is 8 for DES or TDES, and 16 for AES.
 <a href="#">TcCrypto_Cipher_GetContext</a>	Creates a <a href="#">TcCryptoContext</a> 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).
 <a href="#">TcCrypto_Cipher_Init</a>	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.
 <a href="#">TcCrypto_Cipher_SetPadding</a>	Allows cipher padding to be turned on or off.
 <a href="#">TcCrypto_Cipher_Update</a>	Called to encrypt or decrypt data (based on how the cipher was configured in <a href="#">TcCrypto_Cipher_Init</a> ). Can be called one or more times to encrypt data into the output buffer.
 <a href="#">TcCrypto_Digest_Final</a>	Called after all data to hash has been passed through <a href="#">TcCrypto_Digest_Update</a> . This produces a hash, which is a unique bit pattern based on the bits of data passed through <a href="#">TcCrypto_Digest_Update</a> . If even a single bit of input changes, the entire output value will change in approximately 50% of the bits.
 <a href="#">TcCrypto_Digest_GetContext</a>	Called to initialize a new <a href="#">TcCryptoContext</a> for use in hashing. The context contains hash-related state.
 <a href="#">TcCrypto_Digest_Init</a>	Initializes the digest state to prepare it for computing a new hash value via <a href="#">TcCrypto_Digest_Update</a> .
 <a href="#">TcCrypto_Digest_Update</a>	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.
 <a href="#">TcCrypto_Hmac_Final</a>	Called after all data to hash has been passed through <a href="#">TcCrypto_Hmac_Update</a> . This produces an HMAC hash, which is a unique bit pattern based on the bits of data passed through <a href="#">TcCrypto_Hmac_Update</a> , using the secret HMAC key supplied in <a href="#">TcCrypto_Hmac_Init</a> .

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



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

This document may be reproduced only in its original entirety (without revision).

	system that is currently allocated.
 <a href="#">TcCrypto_System_Initialize</a>	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.
 <a href="#">TcCrypto_System_SetLastError</a>	Sets the thread's last error state, retrievable by <a href="#">TcCrypto_System_GetLastError</a> .
 <a href="#">TcCrypto_System_Shutdown</a>	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.
TcCrypto_Watermark_Generate TcCrypto_Watermark_GetData TcCrypto_Watermark_IsValid	These functions are not intended to be called by any consumers of this library – they are used internally and exported for use by the UGS watermarking utility, which is beyond the scope of this document.

### **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 specified in FIPS PUB 186-2 Appendix 3.1, 3.3 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.

This document may be reproduced only in its original entirety (without revision).

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 seed key shall not be the same value.

### **3.3. Operating Systems**

The TCM has been officially validated on the following platforms:

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

In addition the validation, the TCM has been tested by UGS on the following platforms:

- 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

This document may be reproduced only in its original entirety (without revision).

## 4. 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.

### 4.1. 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 or generated with the TCM. The operator must generate or otherwise provide any keys to be used during operation.

### 4.2. 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.

### 4.3. Self-tests

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

```
FIPS_selftest_rng()
FIPS_selftest_sha1()
FIPS_selftest_sha224()
FIPS_selftest_sha256()
FIPS_selftest_sha384()
FIPS_selftest_sha512()
FIPS_selftest_hmac()
FIPS_selftest_aes()
FIPS_selftest_des() // includes 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 HMAC-SHA-512 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 both the Approved RNG and the seed-smoother to ensure that neither becomes "stuck".

This document may be reproduced only in its original entirety (without revision).

## ***Mitigation of Other Attacks***

This section is not applicable.