TRICX Cryptographic Library
FIPS 140-2 Non-Proprietary Security Policy

Trustonic

Version 1.1
August 2017

Contents

1 Introduction

TRICX is a software/firmware cryptographic module conformant to FIPS 140-2 Level 1 overall. As a software-only module, the FIPS 140-2 embodiment is designated a *multi-chip standalone* module, with the physical cryptographic boundary at the platform of the general-purpose operating environment and the logical boundary at the library API packaged as `libtricx.tee.lib`. The module is intended to be used within Trustonic's trusted operating system running in a Trusted Execution Environment (TEE)[16], using the hardware-level isolation provided by ARM TrustZone technology[2].

The TRICX library will be linked with a Trusted Application (TA) running on Trustonic's trusted operating system. The operating system logically isolates every TA from every other TA, as well as from processes running in the non-trusted environment. The module functions entirely within the process space of the calling application, and implicitly satisfies the FIPS 140-2 requirement for a single-user mode of operation.

For FIPS 140-2 validation, the module is tested by an accredited FIPS 140-2 testing laboratory on the operating environments shown in table 1.

Table 1: Tested operating environments

Module (version)	Operating system	Processor
TRICX 1.0	Trustonic Kinibi 400A	ARM Cortex-A53 with PAA[1]
TRICX 1.0	Trustonic Kinibi 400A	ARM Cortex-A53 without PAA[1]

[1] For this module, 'PAA' (see IG 1.21) refers to the use of ARMv8 crypto extensions for SHA-1, SHA-256, SHA-512 and AES, and NEON instructions for arithmetic operations.

As allowed by FIPS 140-2 IG G.5, the validation status of the module is maintained when operated in the following additional operating environments:

- ARMv8 with PAA
- ARMv8 without PAA
- ARMv7 with PAA
- ARMv7 without PAA

In these environments equipped with the same OS as on the tested operating environments, the module operates without any modification. The CMVP makes no statement as to the correct operation of the module on the operational environments for which operational testing was not performed.

Table 2 shows the FIPS 140-2 Security Levels for the module.

Table 2: Security Levels

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

The physical ports of the module are the same as the device on which it executes. The logical interfaces are those of the module API:

data input input parameters that are supplied to the API commands

data output output parameters that are returned by the API commands

control input API commands

status output return status provided by API commands

The module does not access physical ports. When the module is performing self-tests, or is in an error state, all output on the logical *data output* interface is inhibited. The module is single-threaded, and in error scenarios returns only an error value (no data output).

1.1 Modes of operation

The module is a static linked library that only supports a single FIPS-approved mode of operation. The library provides the cryptographic functions listed in section 2.

See section 8 for additional operational guidance.

2 Cryptographic functionality

This section describes the cryptographic primitives implemented and used by the module. Approved algorithms are listed in table 3; other allowed algorithms are listed in table 4.

Table 3: Approved algorithms

CAVP	Algorithm	Mode	Use	Parameters
#4468	AES[1]		key generation	key sizes: 128, 192,[3] 256 bits
		ECB[4] CBC[4] CBC-CS[10][1] CTR[4] XTS[9]	encryption, decryption	
		CMAC[8]	authentication	
		CCM[7] GCM[5]	authenticated encryption, authenticated decryption	
		KW[6][2]	key wrapping, key unwrapping	
#1450	DRBG[12][4]	Hash_DRBG	random bit generation	hash function: SHA-256[15]
#1090 #1180[5]	ECDSA[3][6]		key generation, signature, verification	curves: P-192,[7] P-224, P-256, P-384, P-521 (SHA-1,[8] SHA-224, SHA-256, SHA-384, SHA-512)
#2965	HMAC[17]		key generation, authentication	key sizes of 112 bits or more (multiples of 8 bits); hash functions SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
#119	KAS[11]	Ephemeral Unified[9]	key generation, key agreement	curves: P-224, P-256, P-384, P-521 (SHA-256, SHA-512)
#4468	KTS	KW[6][2]	key wrapping, key unwrapping	AE, AD, AES-128, AES-192, AES-256, FWD
#2443	RSA[3]		key generation	modulus lengths of 1024,[11] 2048, 3072, or 4096[12] bits; any supported digest
		PKCS#1 v1.5[14] PSS[14][10]	signature, verification	
#3680	SHS[15]		digest generation	hash functions: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
#2398	Triple-DES[13]	TECB[4] TCBC[4]	key generation, encryption, decryption	3-key

[1] Vendor-affirmed CBC-CS3. [2] Key-establishment methodology provides between 128 and 256 bits of encryption strength. [3] AES 192-bit keys are not supported for XTS mode. [4] The DRBG is instantiated through the API. There is no reseeding. [5] CVL certification for signature generation component only.

[6] ECDSA may be used with any supported hash function, or none (in which case the input is assumed to be a message digest). [7] Curve P-192 is supported for verification only. [8] SHA-1 should only be used for signature verification. [9] ECDH implements the ECC CDH primitive from [11], applying the Single-Step KDF ([11] section 5.8.1.1 option 1) to the derived secret, using either SHA-256 or SHA-512 as the auxilliary function.

[10] For RSA-PSS operations the same hash function is used in the MGF as is used to hash the message. For verification, the salt length is inferred from the message. [11] 1024-bit modulus for signature verification only.

[12] RSA with 4096-bit modulus is vendor-affirmed for key generation and signature verification.

Table 4: Allowed (non-approved) algorithms

Algorithm	Use	Notes
RSA	key encapsulation, key decapsulation[1]	RSA-OAEP. Supported modulus lengths in bits: 2048, 3072, 4096.

[1] RSAES-OAEP. The same hash function is used in the MGF as is used to hash the label. The maximum label size is 512 bytes.

2.1 Critical security parameters and public keys

The critical security parameters used by the module are listed in tables 5 and 6 respectively.

Table 5: Critical security parameters

Name	Description and usage
AES-K-ENC	AES[1] encryption key (ECB[4], CBC[4], CTR[4], CTS[10], XTS[9], CCM[7], GCM[5]), KW[6] (128, 192 or 256 bits)
AES-K-DEC	AES[1] decryption key (ECB[4], CBC[4], CTR[4], CTS[10], XTS[9], CCM[7], GCM[5]), KW[6] (128, 192 or 256 bits)
AES-K-AUTH	AES[1] authentication key (CMAC[8]) (128, 192 or 256 bits)
DES-K-ENC	Triple DES[13] encryption key (ECB[4], CBC[4]) (192 bits)
DES-K-DEC	Triple DES[13] decryption key (ECB[4], CBC[4]) (192 bits)
DRBG-EI	Entropy input (between 256 and 2048 bits) used to instantiate the approved Hash_DRBG[12]
DRBG-STATE	Hash_DRBG[12] `V` and `C` values
EC-K-SIG	ECDSA[3] signature key (P-224, P-256, P-384, P-521)
EC-K-KA-PRI	ECDH[11] key-agreement private key (P-224, P-256, P-384, P-521)
HMAC-K-AUTH	HMAC[17] authentication key (minimum 112 bits)
RSA-K-SIG	RSA[3] signature key (PKCS#1 v1.5[14], PSS[14]) (2048, 3072 or 4096 bits)
RSA-K-DEC	RSA[3] decapsulation key (OAEP) (2048, 3072 or 4096 bits)

Table 6: Public keys

Name	Description and usage
EC-K-VER	ECDSA[3] verification key (P-192, P-224, P-256, P-384, P-521)
EC-K-KA-PUB	ECDH[11] key-agreement public key (P-224, P-256, P-384, P-521)
RSA-K-VER	RSA[3] verification key (PKCS#1 v1.5[14], PSS[14]) (1024, 2048, 3072 or 4096 bits)
RSA-K-ENC	RSA[3] encapsulation key (OAEP) (2048, 3072 or 4096 bits)

3 Roles, authentication and services

3.1 Roles

The module does not support a maintenance role or bypass capability. The module supports the Crypto Officer and User roles. The Crypto Officer installs and loads the module. The User uses the cryptographic services provided by the module.

Table 7 lists the available roles. The options for authentication type and data are common across roles.

Table 7: Roles

Name	Description
CO	cryptographic officer (module installer)
User	application or module user

3.2 Authentication

No authentication is required to use the module.

3.3 Services

Table 8 summarizes the services implemented by the module, with additional detail in table 9 for traceability of cryptographic functionality and access to CSPs and public keys by services.

Table 8: Module services

Service	Description	Role
Authentication	Calculate or valdidate an authentication tag on data (CMAC, HMAC, AES-CCM, AES-GCM)	User
Crypto extensions	Improve performance of hashing and AES operations through the use of ARMv8 crypto extensions if available	User
Decryption	Decrypt data (AES, DES)	User
Encryption	Encrypt data (AES, DES)	User
Hashing	Calculate a message digest (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)	User
Initialization	Initialize the module by loading the application. Run the integrity check and suite of self-tests required by FIPS 140-2	CO
Key agreement	Calculate a shared secret key from a private and a public key (ECDH)	User
Key decapsulation	Decrypt a secret key using an encapsulation key (RSA-OAEP)	User
Key deletion	Delete and zeroize a key (AES, DES, HMAC, RSA, EC)	User
Key encapsulation	Encrypt a secret key using an encapsulation key (RSA-OAEP)	User
Key export	Export a key (RSA, EC)	User
Key generation	Generate a key (AES, DES, HMAC, RSA, EC)	User
Key import	Import a key (AES, DES, HMAC, RSA, EC)	User
Key unwrapping	Decrypt a secret key using a wrapping key (AES-KW)	User
Key wrapping	Encrypt a secret key using a wrapping key (AES-KW)	User
Public key export	Export a public key (RSA, EC)	User
Public key import	Import a public key (RSA, EC)	User
RNG initialization	Instantiate the module's DRBG, supplying entropy[1]	User
RNG use	Generate random byte strings, IVs and keys from the module's DRBG	User
RNG zeroization	Zeroize the internal state of the module's DRBG	User
Selftest	Run the module's self-tests and integrity check	User
Signature	Calculate a digital signature using an asymmetric private key (RSA, ECDSA)	User
Status enquiry	Read the current state of the module	User
Verification	Verify a digital signature using an asymmetric public key (RSA, ECDSA)	User
Version enquiry	Read the version number of the module	User
Zeroization	Zeroize all CSPs by unloading the application	CO

[1] The application using the module is running in a trusted execution environment, through which it has access to a PRNG (which uses the Hash_DRBG algorithm from SP 800-90A) seeded with 256 bits of entropy from a hardware-based NDRBG. The application is expected (but not required) to obtain its entropy seed from this source.

Table 9 defines the relationship between access to CSPs and the different module services. The modes of access shown in the table are defined as:

G (generate) The module generates the CSP.

R (read) The module reads the CSP. This is typically performed before the module uses the CSP.

E (execute) The module executes using the CSP.

W (write) The module writes the CSP. This is typically performed after a CSP is imported into the module, or when the module generates a CSP, or when the module overwrites an existing CSP.

Z (zeroize) The module zeroizes the CSP.

Table 9: CSP access rights within services

Service	AES-K-ENC	AES-K-DEC	AES-K-AUTH	DES-K-ENC	DES-K-DEC	DRBG-EI	DRBG-STATE	EC-K-SIG	EC-K-KA-PRI	HMAC-K-AUTH	RSA-K-SIG	RSA-K-DEC
Authentication	RE		RE							RE		
Crypto extensions												
Decryption		RE			RE							
Encryption	RE			RE								
Hashing												
Initialization							Z					
Key agreement									RE			
Key decapsulation												RE
Key deletion	Z	Z	Z	Z	Z			Z	Z	Z	Z	Z
Key encapsulation												
Key export								R	R		R	R
Key generation	GW	GW	GW	GW	GW		REW	GW	GW	GW	GW	GW
Key import	W	W	W	W	W			W	W	W	W	W
Key unwrapping		RE										
Key wrapping	RE											
Public key export												
Public key import												
RNG initialization						RE	W					
RNG use							REW					
RNG zeroization							Z					
Selftest												
Signature								RE			RE	
Status enquiry												
Verification												
Version enquiry												
Zeroization	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z

The module is loaded and initialized by the loading of the application with which it is statically linked. The module will automatically execute its self-tests before perfoming any user-requested operation. To enable the self-tests, the DRBG is first instantiated with a fixed seed; on completion of the self-tests the DRBG is reset to uninstantiated state.

The *Status enquiry* service allows the user to determine the current state of the module: whether it is in an error state (for example because a self-test failed), a no-entropy state (before RNG initialization, when only operations that do not require an RNG may be performed), or fully operational. This service does not access any CSPs.

The *RNG initialization* service allows the user to inject entropy into the module and instantiate the DRBG.

The *Zeroization* service zeroizes all secret state in the module. The Crypto Officer can perform this by unloading the application; all data residing in the module's memory is zeroized before it can be read by any other process. In addition, the User can zeroize individual CSPs (both DRBG internal state and keys) by calling API functions.

4 Self-tests

Each time the module is loaded, it tests that the cryptographic algorithms operate correctly. The same suite of self-tests may also be initated by the user at any time through an API function. If any test fails, the module enters an error state and no further cryptographic functionality is available.

Tables 10 and 11 list the self-tests performed by the module. The module design utilizes built-in initialization code to call the tests in table 10 without operator intervention prior to control being passed to the calling application. The tests must be completed successfully prior to any other use of cryptography by the module. If any self-test fails, the module transitions to the error state (and a call to `tricx_get_state` will return `tricx_state_error`) until it is unloaded.

Table 10: Power-up self-tests

Target	Description
Software integrity	HMAC with SHA-256, with fixed 128-bit key over the various contiguous segments of the library; tag compared against reference calculated at (executable image) link time.
AES (#4468)	Separate encryption and decryption KATs using a 128-bit key and 128-bit message in ECB mode.
DRBG (#1450)	SP 800-90A[12] health tests for the Hash_DRBG including KAT using a NIST test vector.
ECDH (#119)	KAT for scalar multiplication in P-256.
ECDSA (#1090 #1180)	Round-trip signature and verification of a 256-bit digest using a P-256 key.
HMAC (#2965)	HMAC generation KAT using SHA-256, a 128-bit key, 128-bit message and 256-bit tag.
RSA (#2443)	Separate KATs of 2048-bit signature generation and verification using PKCS1 v1.5 padding and a 256-bit message.
SHS (#3680)	Separate KATs of SHA-1, SHA-256 and SHA-512 using a 256-bit message.
Triple-DES (#2398)	Separate encryption and decryption KATs using a 64-bit message in ECB mode.

Table 11: Conditional self-tests

Target	Description
ECDH	Pairwise consistency test performed on each key pair generation.
ECDSA	Pairwise consistency test performed on each key pair generation.
RSA	Pairwise consistency test performed on each key pair generation.

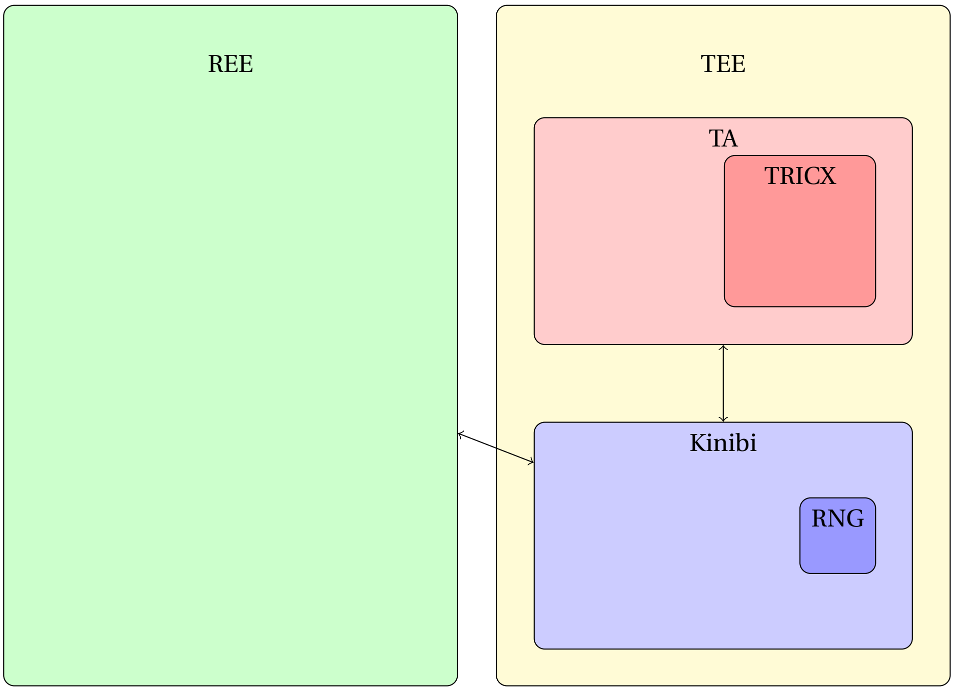
5 Physical security policy

Physical security is not applicable to software-only modules.

6 Operational environment

The module operates in a modifiable operational environment under the FIPS 140-2 definitions.

The module is a static library intended to be linked with a third-party ‘trusted application’ (TA) on top of Trustonic’s proprietary trusted operating system (Kinibi), running within a trusted execution environment (TEE). It is isolated from the rich execution environment (REE) outside the TEE by the hardware protection mechanisms afforded by ARM TrustZone. Isolation between different TAs is enforced by Kinibi.



For FIPS 140-2 validation, the module has been tested by an accredited FIPS 140-2 testing laboratory on the following operating environments:

- Kinibi 400A on ARMv8 with ARM crypto extensions
- Kinibi 400A on ARMv8 without ARM crypto extensions

7 Mitigation of other attacks

The module has not been designed to mitigate attacks outside the scope of FIPS 140-2.

8 Security rules and guidance

The module design corresponds to the module security rules. As the module is statically linked to a TA it does not provide authentication. It implements and enforces the following security rules:

1. The module inhibits data output during power-up self-tests and error states.
2. Status information does not contain CSPs or sensitive data.
3. As there is no non-Approved mode of operation, all CSPs exist only in the Approved mode.
4. Zeroization of all CSPs is achieved by unloading the TA that includes the module. The Kinibi operating system will ensure that all TA-owned memory is zeroized before it is exposed to any other task. The memory is released on task termination, and a garbage-collection mechanism zeros out the memory before marking it as available for reuse.
5. All random number generation performed by the module uses an approved DRBG. In particular, IVs generated using the API function `tricx_rng_generate` conform to FIPS 140-2 requirements.
6. The length of the data unit for any instance of an implementation of XTS-AES shall not exceed 2^{20} blocks. The module explicitly checks that the two halves of the XTS key are not equal.

The following security rules must be adhered to for operation in the FIPS 140-2 Approved mode:

1. The module must be linked to a TA running on Kinibi. This ensures isolation of all CSPs from all other TAs and zeroization of memory after unloading.

2. The entropy string used to seed the RNG must come from a secure source and provide at least the equivalent of 256 bits of entropy. For example, it may be obtained from Kinibi via one of the API functions that Kinibi provides for this purpose.

3. The IV supplied to any GCM operation must comprise at least 96 bits generated using the module's DRBG (i.e. using `tricx_rng_generate`). The IV must never be reused with the same key for two different encryption operations.

4. XTS-AES was designed for the cryptographic protection of data on storage devices and shall only be used for that purpose. An XTS-AES key shall not be associated with more than one key scope.

5. A Triple-DES key shall not be used for processing (either encrypting or decrypting) more than 2^{28} 8-byte data blocks.

6. Signature operations using ECDSA and RSA algorithms must be invoked with one of the following digest algorithms: SHA-224, SHA-256, SHA-384, or SHA-512.

All TAs are authenticated by Kinibi before being loaded. The trust anchor for this authentication depends on the type of TA, but is always either:

- an image or public-key hash contained within the Kinibi image (the image itself being signed by a licenced OEM, with the signature verified early in the device boot chain); or

- stored in a container residing on the device, encrypted using a device-unique key accessible only from Kinibi, and established via a chain of trust extending back to another device-unique key generated in the factory where the device was manufactured.

References

[1] Advanced Encryption Standard. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

[2] ARM TrustZone. `https://www.arm.com/products/security-on-arm/trustzone`

[3] Digital Signature Standard. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`

[4] Recommendations for Block Cipher Modes of Operation. `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`

[5] Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf`

[6] Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping. `http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf`

[7] Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf`

[8] Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38b.pdf`

[9] Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdf`

[10] Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode. `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a-add.pdf`

[11] Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

[12] Recommendation for Random Number Generation Using Deterministic Random Bit Generators. http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

[13] Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-67r1.pdf

[14] RSA Cryptography Specifications Version 2.1. https://tools.ietf.org/html/rfc3447

[15] Secure Hash Standard. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[16] TEE System Architecture v1.0. Available from: http://www.globalplatform.org/specificationsdevice.asp

[17] The Keyed-Hash Message Authentication Code (HMAC). http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf