Geotab Cryptographic Module

FIPS 140-3 Non-Proprietary Security Policy

Firmware Version 1.0

Document Version 1.0

GEOTAB®
management by measurement

Table of Contents

List of Tables

List of Figures

1. General

The following table lists the level of validation for each area in FIPS-140-3.

Table 1: Security Levels

ISO/IEC 24759 Section 6. [Number Below]	FIPS-140-3 Section Title	Security Level
1	General	1
2	Cryptographic module specification	1
3	Cryptographic module interfaces	1
4	Roles, services, and authentication	1
5	Software/Firmware security	1
6	Operational environment	1
7	Physical security	1
8	Non-invasive security	N/A
9	Sensitive security parameter management	1
10	Self-tests	1
11	Life-cycle assurance	1
12	Mitigation of other attacks	N/A
	Overall	1

Terms & Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CMVP	Cryptographic Module Validation Program
SSP	Sensitive Security Parameters
DRBG	Deterministic Random Bit Generator
FIPS	Federal Information Processing Standards
HMAC	Hashed Message Authentication Code
KAT	Known Answer Tests
NIST	National Institute of Standard And Technology
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
CKG	Cooperative Key Generation

2. Cryptographic Module Specification

Module Description

The Geotab Cryptographic Module, hereafter referred to as the module, is classified as **Level 1 Firmware Module** as per FIPS-140-3 guidelines operating within a limited operational environment. The module operates with a single-chip standalone module embodiment. The cryptographic boundary is a single-chip microcontroller which runs a limited operating environment such as bare-metal firmware image. The module is bundled with the firmware image used by the physical microcontrollers.

The module performs no communication other than with the calling application via well-defined APIs that invoke the Module.

Tested Configurations

The product has been tested and is intended to operate on the following Geotab Operating Environments. There are no Vendor affirmed operational environments.

Table 2: Tested Operational Environments

#	Operating System	Hardware Platform	Processor	PAA/Acceleration
1	N/A	NXP S32K148	ARM CORTEX M4F	N/A
2	N/A	ST STM32H7	ARM CORTEX M7	N/A

Approved Mode of Operation

When properly initialized as specified in Section 11 of this document, the module only operates in an approved mode of operation.

Approved Algorithms

Table 3: Approved Algorithms

CAVP Cert	Algorithm and Standard	Mode/Method	Description/Key Size(s)/Key Strength(s)	Use/Function
A4203	AES-CBC (SP800-38A)	CBC	256-bits	Encryption/Decryption
A4203	HMAC DRBG (SP 800-90Ar1)	HMAC_DRBG	HMAC-SHA2-256 with no PR	Random Number Generation
A4203	HMAC-SHA2-256 (FIPS 198-1)	SHA2-256	MAClen: 256-bits KeyLen: 256-bits	Calculate/Verify
A4203	RSA SigVer (FIPS 186-4)	Signature Verification	2048-bits PKCS #1 v1.5 SHA2-256	Signature verification
A4203	SHA2-256 (FIPS 180-4)	SHA2-256	N/A	Hash calculation

This module is compliant to IG C.F:

The module utilizes the approved modulus size 2048 bits for RSA signatures. This functionality has been CAVP tested as noted above. RSA SigVer is CAVP tested for the supported modulus size as noted above. The module does not perform FIPS 186-2 SigVer. All supported modulus sizes are CAVP testable and tested as noted above.

The module does not support “Vendor Affirmed Approved Algorithms”.

Non-Approved Security Functions

The module does not support any of the following:

"Non-Approved Algorithms Allowed in the Approved Mode of Operation",

"Non-Approved Algorithms Allowed in the Approved Mode of Operation with No Security Claimed",

"Non-Approved Algorithms Not Allowed in the Approved Mode of Operation".

Module Boundary

The following block diagram details modules' Cryptographic Boundary. The Tested Operational Environment's Physical Perimeter (TOEPP) is the physical perimeter of the Microcontroller/Special Purpose Computer.

Cryptographic Boundary

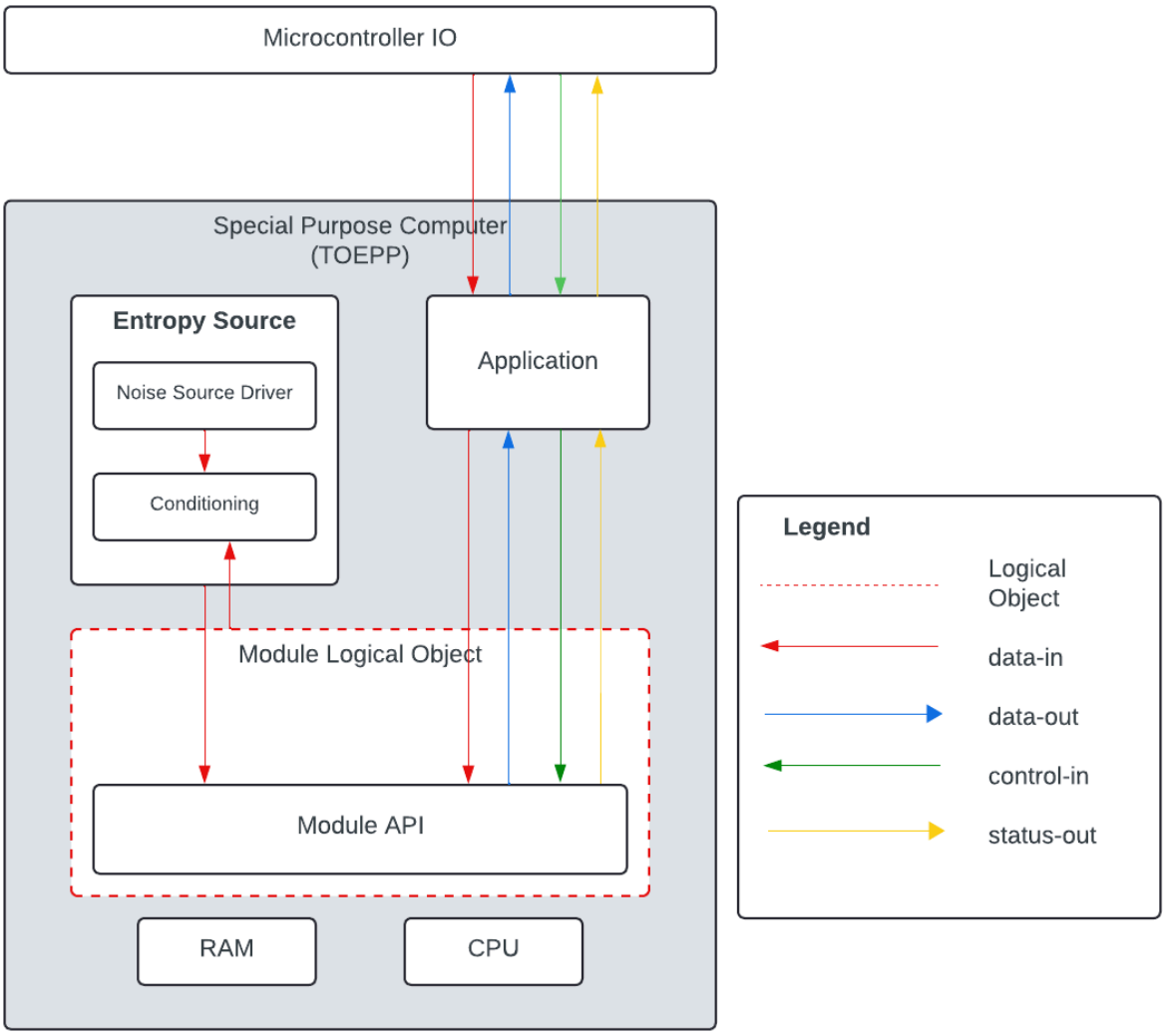

Figure 1: Cryptographic Boundary

TOEPP

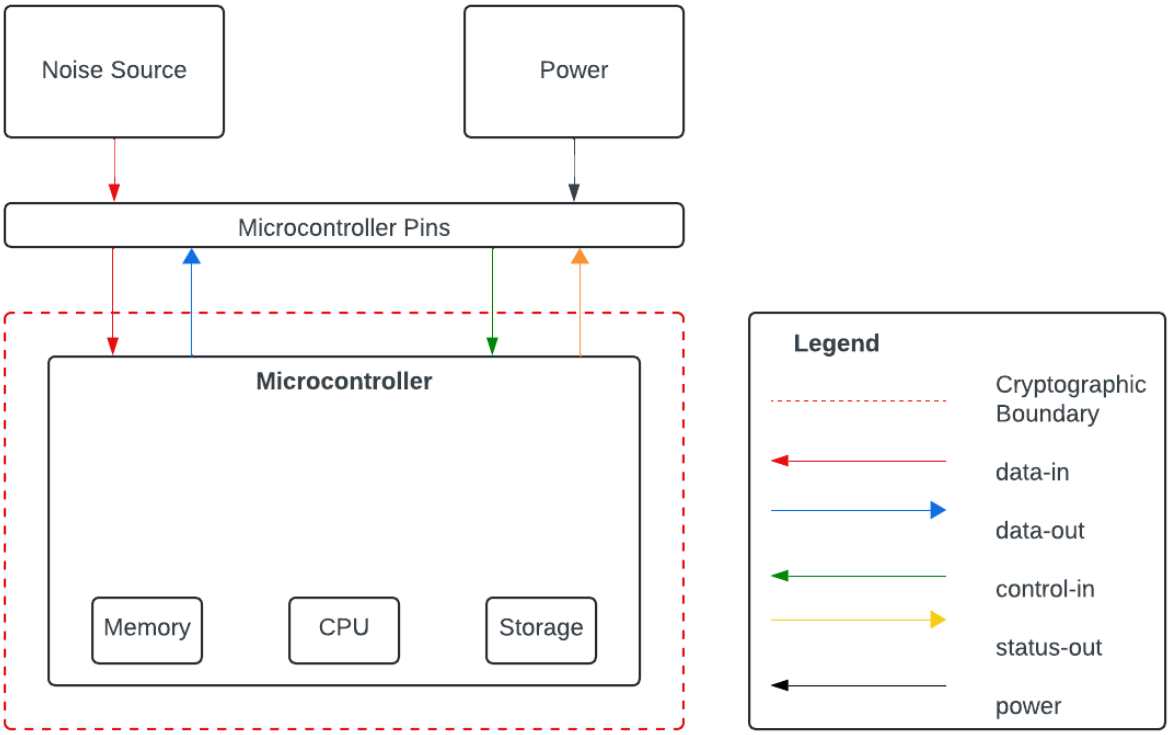

Figure 2: TOEPP

3. Cryptographic Module Interfaces

The physical ports of the module are the same as the special purpose single chip microcontroller system on which it is executing. The logical interface is an application programming interface (API), the mapping of logical interface type is explained below.

Table 4: Ports and Interfaces

Physical Port	Logical Interface	Data that passes over port/interface
N/A	Data Input	API entry point data input parameters
N/A	Data Output	API entry point data output parameters
N/A	Control Input	API entry point and corresponding parameters
N/A	Status Output	API entry point return values

Since the module is a firmware module, control of the physical ports is outside the module scope. When the module is in self-test state or error state, all output on the logical data output interface is prohibited. In the error state the module will only return an error value (no data output is returned).

4. Roles, Services and Authentication

Roles

The module supports following roles:

- **User role**: performs cryptographic services, gets module status.
- **Crypto officer role**: Performs module initialization ONLY.

The module does not support operator authentication and does not allow concurrent operators. The user and Crypto Officer roles are implicitly assumed by the application accessing services implemented by the module.

Table 5: Roles, Service Commands, Input and Output

Role	Service	Input	Output
Crypto officer	gc_Init	Module HMAC	Init status
User	gc_aes_CbcEncrypt	Plaintext	Ciphertext
User	gc_aes_CbcDecrypt	Ciphertext	Plaintext
User	gc_aes_CbcSetKey	Key	None
User	gc_aes_CbcResetEncryption	Injection Vector	None
User	gc_aes_CbcResetDecryption	Injection Vector	None
User	gc_sha_Reset	None	None
User	gc_sha_DataAdd	Data	None
User	gc_sha_Calculate	None	SHA 256 hash
User	gc_rsa_Init	None	None
User	gc_rsa_IsSignatureValid	RSA N, RSA E, calculated hash, signed hash	Validation status
User	gc_hmac_Reset	Key	None
User	gc_hmac_Update	Data	None
User	gc_hmac_Finish	None	HMAC

User	gc_drbg_Init	None	None
User	gc_drbg_AlgInit	Entropy, additional input	None
User	gc_drbg_GetNextRandom	Requested number of bytes	DRBG output
User	gc_AddRawEntropy	Raw entropy bytes	None
User	gc_IsConditionedEntropyReady	None	Status
User	gc_GetModuleState	None	State
User	gc_GetModuleVersion	None	Version
User	Power Cycle	None	SSPs Zeroization

Approved Services

Access rights column key:
G = Generate: The module generates or derives the SSP.
R = Read: The SSP is read from the module (e.g. the SSP is output).
W = Write: The SSP is updated, imported, or written to the module.
X = Execute: The module uses the SSP in performing a cryptographic operation.
Z = Zeroise: The module zeroises the SSP.

Table 6: Approved Services

Service	Description	Approved Security Functions	Keys and/or SSPs	Roles	Access rights to Keys and/or SSPs	Indicator
gc_Init	Initialize the cryptographic module instance and verify its integrity. Run Pre-operational self tests. Proceed into operational state if initiation is successful or into error state if it is not. This is typically run on the power-on of the module	HMAC-SHA2-256	N/A	Crypto officer	WX	Successful Completion of Service
gc_aes_CbcEncrypt	Perform AES 256 CBC encryption on provided plaintext	AES-CBC	AES-256 key	User	X	Successful Completion of Service
gc_aes_CbcDecrypt	Perform AES 256 CBC decryption on provided plaintext	AES-CBC	AES-256 key	User	X	Successful Completion of Service

gc_aes_CbcSetKey	Update the AES key in the specified AES context. This also resets both encryption and decryption CBC chains.	AES-CBC	AES-256 key	User	X	Successful Completion of Service
gc_aes_CbcResetEncryption	Reset the CBC chain for encryption in the specified AES context	AES-CBC	N/A	User	N/A	Successful completion of service
gc_aes_CbcResetDecryption	Reset the CBC chain for decryption in the specified AES context	AES-CBC	N/A	User	N/A	Successful completion of service
gc_sha_Reset	Reset the SHA calculation for the specified SHA context	SHA2-256	N/A	User	N/A	Successful completion of service
gc_sha_DataAdd	Add the data input to the SHA calculation for the specified context	SHA2-256	N/A	User	N/A	Successful completion of service
gc_sha_Calculate	Finalize the SHA calculation and provide the calculated hash	SHA2-256	N/A	User	N/A	Successful completion of service
gc_rsa_Init	Initialize RSA context	RSA SigVer	N/A	User	N/A	Successful Completion of Service
gc_rsa_IsSignatureValid	Using the provided modulus and exponent (public key), verify if the signed has was signed with the matching private key	RSA SigVer	RSA 2048 signature verification key	User	WX	Successful Completion of Service
gc_hmac_Reset	Reset the HMAC calculation for the specified HMAC context	HMAC-SHA2-256	HMAC Key	User	WX	Successful Completion of Service
gc_hmac_Update	Add the data input to the HMAC calculation for the specified context	HMAC-SHA2-256	HMAC Key	User	X	Successful Completion of Service
gc_hmac_Finish	Finalize the HMAC calculation and provide the calculated hash	HMAC-SHA2-256	HMAC Key	User	X	Successful Completion of Service
gc_drbg_Init	Initialize the DRBG context and use the module defined entropy function to retrieve entropy for both entropy input and additional seed input.	HMAC DRBG	DRBG seed, DRBG Entropy Input String, DRBG HMAC Key, DRBG	User	WX	Successful Completion of Service

			HMAC V			
gc_drbg_AlgInit	Initialize the DRBG algorithm context and provide it with its entropy acquisition function. This is only used for CAVP testing of DRBG.	HMAC DRBG	DRBG seed, DRBG Entropy Input String, DRBG HMAC Key, DRBG HMAC V	User	WX	Successful Completion of Service
gc_drbg_GetNextRandom	Use DRBG to derive the required number of bytes	HMAC DRBG	DRBG seed, DRBG Entropy Input String, DRBG HMAC Key, DRBG HMAC V	User	X	Successful Completion of Service
gc_AddRawEntropy	Add raw entropy bytes to be conditioned	N/A	N/A	User	N/A	Successful completion of service
gc_IsConditionedEntropyReady	Indicates whether a sufficient number of raw entropy bytes has been added and processed by the module to begin performing RBG	N/A	N/A	User	N/A	Successful Completion of Service
gc_GetModuleState	Indicates what state the module is in. (Init, Error or Operational)	N/A	N/A	User	N/A	Successful Completion of Service
gc_GetModuleVersion	Indicates what the current module version is	N/A	N/A	User	N/A	Successful Completion of Service
Power Cycle	Cuts power to the device and zeroizes all SSPs stored in volatile memory.	N/A	All SSPs	User	Z	Successful completion of service

The indicator field of the above table specification of “successful completion of service” means services return success status; non-zero value or the approved security function finishes successfully.

The module implements services corresponding to all mandatory services from FIPS 140-3, AS04.12 as denoted below:

- *Show module's versioning information* - gc_GetModuleVersion
- *Show status* - gc_GetModuleState
- *Perform self-tests* - gc_Init
- *Perform zeroisation* - Power Cycle
- *Perform approved security functions* - All other services.

Non-Approved Services

The module does not support any non-approved services.

5. Software/Firmware Security

The firmware module is in the form of a compiled binary, one for each supported operating environment. The module is closed source and does not require compilation by the user.

After a module is powered on, the module's integrity check is performed during module instantiation. The operator is able to initiate the integrity test on demand by re-initializing the module. Upon initialization the module performs an integrity test of the module on itself using HMAC-SHA2-256(Cert Num A4203). If the Integrity check fails, it will cause the module to enter the error state.

Additionally the module performs listed self-tests (Specified in Section 10) and if successful the module will enter the Operational State.

6. Operational Environment

The Module tested operating environments are listed below. The module implements a limited operating environment. The module functions entirely within the Operating environment provided space for the calling application.

- NXP S32K148 with S32K14X series processor
- ST STM32H7 with the STM32 Series Processor

System Description

The system described by this security policy is a single-chip firmware module, running in a limited operational environment which loads a single binary, consisting of both the calling application and the validated module, that requires a power-on reset to install. The operating environment does not have an operating system and contains a single CPU with one core. The environment is single threaded and parallel execution is not possible. Program instructions of the module and volatile SSPs are stored within the cryptographic boundary. The module does not provide non-volatile key storage. Keys are provided externally by the application layer and are not stored by the module in non-volatile memory.

7. Physical Security

The module is a single-chip cryptographic module and composed of production grade components. The module obtains its physical security from a single chip with standard passivation coating which protects against any environmental or physical damage. As the module consists of production grade components with standard passivation it satisfies the security requirements of Security Level 1.

8. Non-Invasive Security

This cryptographic module does not claim any Non-Invasive security features.

9. Sensitive Security Parameters Management

Sensitive Security Parameters

The module does not persistently store any sensitive parameters within the module logical object. The module receives the SSPs through API calls to the module. The SSPs are not manually entered into or output from a module. All non-DRBG related sensitive security parameters are provided externally to the module and handled by the application using the module.
The module supports the following SSP's listed below.

Table 7: SSPs

Key/SSP/ Name/ Type	Strength	Security Function and Cert. Number	Gener-ation	Import/ Export	Establish-ment	Storage	Zero-isation	Use & Related keys
AES-256 Key	256-bits	AES-CBC (A4203)	external	Imported plaintext	N/A	Volatile memory plaintext	Power cycle	Used as an input to AES cipher operation
HMAC Key	256-bits	HMAC-SHA2-256 (A4203)	external	Imported plaintext	N/A	Volatile memory plaintext	Power cycle	Used as an input to HMAC operation
RSA 2048 signature verification key	112-bits	RSA-SigVer (A4203)	external	Imported plaintext	N/A	Volatile Memory plaintext	Power cycle	Used as an input for the RSA verify operation
DRBG Entropy Input String	512-bits	HMAC DRBG (A4203)	Generated from ESV	N/A	N/A	Volatile Memory plaintext	Power cycle	Used as an input to initialize DRBG
DRBG seed	256-bits	HMAC DRBG (A4203)	Derived per SP800-90Ar1	N/A	N/A	Volatile memory plaintext	Power cycle	Used as an input to initialize DRBG
DRBG HMAC Key	256-bits	HMAC DRBG (A4203)	Derived per SP800-90Ar1	N/A	N/A	Volatile memory plaintext	Power cycle	Used by HMAC during DRBG
DRBG HMAC V	256-bits	HMAC DRBG (A4203)	Derived per SP800-90Ar1	N/A	N/A	Volatile memory plaintext	Power cycle	Used by HMAC during DRBG

RBG Entropy Sources

The entropy source resides outside the module's logical object but within the module's physical perimeter.

Table 8: Non-Deterministic Random Number Generation Specification

Entropy sources	Minimum number of bits of entropy	Details
Accelerometer Based Entropy Source	The entropy input string is 512 bits, and the entropy source produces full entropy. Minimum number of bits of entropy should be 512 bits.	Produces 256-bits of entropy per 256-bit output, and the DRBG requests 512 bits from the entropy source. (Refer to ESV Cert #E86)

SSP Zeroisation

All SSP values are stored in volatile memory and are zeroized. The zeroization service for the SSP values consists of powering off the module, which resets the state of volatile memory and effectively sets all instances of SSPs to 0 values, making them non-retrievable.

10. Self-Tests

A power cycle is required in order to run power on self-tests on demand.

Pre-Operational Self-Tests

Table 9: Pre-Operational Self Tests

Test	Description
Pre-operational firmware integrity test	Integrity test performed on the module image. Integrity verified using HMAC-SHA2-256.

The Cryptographic algorithm test used to perform the approved integrity technique i.e. HMAC-SHA2-256 is passed before the Pre-operational software/firmware integrity test starts.

The module does not support bypass.
The module does not implement any pre-operational critical function tests.

Conditional Self Tests

Conditional Cryptographic Algorithm Tests

Table 10: Conditional Cryptographic Algorithm Tests

Test	Description
AES-CBC-256 Encryption KAT	Known answer test for AES-CBC-256, confirms that a known plaintext encrypts to a known ciphertext.
AES-CBC-256 Decryption KAT	Known answer test for AES-CBC-256, confirms that a known ciphertext decrypts to a known plaintext
RSA-2048-Verify PKCS #1 v1.5 SHA2-256 KAT	Known answer test for RSA-2048 confirms that a known signed message is successfully verified by a known key
DRBG HMAC-SHA2-256 with no PR KAT (Instantiate & Generate)	Known answer test for DRBG confirms that a known seed & input string input generates the expected entropy value
HMAC-SHA2-256 KAT	Known answer test for HMAC confirms that a Known key and message produce a known digest
SP 800-90B RCT Health Test	Continuous test performed whenever a new seed is requested by the DRBG from the conditioned entropy source (ESV#E86). confirms that the conditioned entropy source is not stuck.
SP 800-90B APT Health Test	Continuous test performed whenever a new seed is requested by the DRBG from the conditioned entropy source (ESV#E86). Confirms no loss of entropy occurs as a result of some physical failure or environmental change affecting the noise source.
DRBG Continuous test	Continuous test performed whenever a new random value is requested from DRBG. Confirms that the DRBG output is not stuck.
DRBG Test Instantiate	Known answer test to confirm that the instantiation completes as expected. The known answer is compared against the Generate output which is always called after.
DRBG Test Generate	Known answer test to confirm that generation completes as expected.
DRBG Test Un-instantiate	Health Test to ensure that error handling is performed correctly, and the internal state has been zeroized

The DRBG reseed function is unavailable as the module is never re-seeded during a power-on cycle.

Conditional Pair-Wise Consistency Test

- N/A Public/Private keys are not generated.

Conditional software/firmware load test

- N/A. The module does not support a firmware load service.

Conditional manual entry test

- N/A. No manual entry of SSP's or key components.

Conditional bypass test

- N/A. No Bypass Capability.

Conditional critical functions test

- N/A.

Error State

If any self-test fails, the module enters the error state. In the error state, only the gc_GetModuleState and gc_GetModuleVersion API's are available. All other API's will return '1' to indicate the error state. To attempt to clear the error, power off the module then power on and attempt to initialize the module again.

11. Life-Cycle Assurance

Configuration Management

The code repositories are managed through git based configuration management and version control systems.

Table 11: Compilers

Platform	Hardware	Compiler	Configuration
N/A	NXP S32K148	gcc-arm-none-eabi 4.9	-MMD -MP
N/A	ST STM32H7	arm-none-eabi-gcc-9.3.1	-O0 -ffunction-sections -fdata-sections -Wall -fstack-usage -MMD -MP --specs=nano.specs -mfpu=fpv5-d16 -mfloat-abi=hard -mthumb

Table 12: Linkers

Platform	Hardware	Linker	Configuration
N/A	NXP S32K148	gcc-arm-none-eabi 4.9	N/A
N/A	ST STM32H7	arm-none-eabi-gcc-9.3.1	N/A

Vendor Testing

The cryptographic module undergoes continuous unit tests and integration tests during the life-time of the module. On top of this, the cryptographic module undergoes continuous scanning through automatic diagnostic tools. Next section details automatic diagnostic tools in detail:

Security Tools

Cryptographic modules are scanned for vulnerabilities using automatic diagnostic tools. Some of the rules enabled in this process are;

- Tainted array index detection
- Buffer overflow detection
- Integer overflow/underflow
- Command injection scanning
- Environment injection
- File injection

Through this Cryptographic module development provides necessary assurance for secure code development and deployment.

Delivery & Operation

Initialization procedure shall be invoked as per to section Crypto Officer Services presented in this document. Crypto Officer has responsibility to ensure that all conditions of initialization are passed as recommended before the actual usage of the module begins.

End of Life

The module's life is defined by the application invoking the APIs associated with the module. The module end of life is reached once the application relieves the module. The module does not persistently store any SSPs nor does it store information on non reconfigurable memory. The procedure for secure sanitization of the module is to power it off, which is the action of zeroization of the SSPs. The sanitization via power-off, results in removal of SSPs from the module.

Finite State Model

Figure 3: Finite State Model

States which are dotted are not present in the state machine.

The Finite State Model consists of following states:

- Power Off : The module embodiment (hardware) is powered off.
- Power On: The module embodiment (hardware) is powered on.
- Initialization: Module is initialized by calling Init routine.
- Integrity Check : the module checksum itself and verifies that it hasn't been maliciously altered.
- Self Test : The module performs its KAT tests.
- Operational: Module is in its normal operating state, All APIs (gc_*) can be used at this moment and selected continuous testing is in force.
- Continuous Test: After the module is in the operational state, selected continuous testing is performed, and if returns error goes to error state otherwise back to operational state.
- Error: The module has entered error state, All APIs except gc_GetModuleState and gc_GetModuleVersion will return an error when used.

Transition Number	Initial State	Resulting State	Description
1	Power Off	Power On	The module embodiment (hardware) is powered off.
2	Power On	Initialization	The module embodiment (hardware) is powered on.
3	Initialization	Integrity Check	Module is initialized by calling init routine.
4	Integrity Check	Self Test	The module performs its KAT tests.
5	Self Test	Operational	Module is in its normal operating state.
6	Operational	Continuous Test	After the module is in operational state, selected continuous testing is performed, and if returns error, goes to error state, otherwise back to operational state.
7	Continuous Test	Operational	After the module is in the operational state, selected continuous testing is performed, and if returns error, goes to error state otherwise back to operational state.
8	Integrity Check / Self Test /	Error	The module has entered error state. All APIs except gc_GetModuleState and

	Continuous Test		gc_GetModuleVersion will return an error (‘1’) when used.
9	Operational / Error	Power Off	The module embodiment (hardware) is powered off.

Figure 4: State Transition Table

Description

When Device is powered on, the module goes to initialization state. The module goes into Integrity check then performs self-tests. The module is not operational until the self tests are completed successfully. Upon successful completion of the self tests, the module enters the Operational State. While the module is in the Operational State, it performs selected continuous testing. The module will return an error state output if the integrity check fails or the self tests are not yet completed. If no more operations are performed and hardware is powered off, the module will go to power off state.

Crypto Officer Guidance

The crypto officer is responsible for initializing the crypto module within its operational boundary. Refer Crypto Officer Services segment for available APIs to perform the task at hand. The officer shall ensure that the module has proceeded to the Operational State upon successful initialization. Crypto officers shall ensure that the best practices of the Module usage is followed throughout the application operational sphere.

Crypto User Guidance

The crypto user is able to access the module using the API defined in Crypto User Services segment. The crypto user shall ensure the best practices outlined in the Module APIs usage are followed with the guidance of Crypto officers.

Approved Mode of Operation

The conditions for using the module in the Approved Mode of Operation are:

1. The module is a cryptographic library and is intended to be used with a calling application.
2. The calling application must invoke the init routine to put the module into the Approved Mode of Operation. All data output will be inhibited otherwise.

The following procedure will properly initialize the module:

- Run gc_init command as CO

Non-Compliant State

Failure to follow the directions in the rules noted in this section will result in the module operating in a non-compliant state, which is considered out of scope of this validation.

12. Mitigation of Other Attacks

The module does not claim to mitigate any additional attacks.