FIPS 140-2 Security Policy for INTEGRITY Security Services High Assurance Embedded Cryptographic Toolkit

Module Version 2.0

Document Version 2.0.4

19415 Deerfield Ave Suite #204
Lansdowne, VA 20176
USA
Tel: 206-310-6795
Fax: 978-383-0530
www.ghs.com

DISCLAIMER

ISS ECT FIPS 140-2 Security Policy
PubID: ISS ECT-SP-004
Version: 2.0.4
October 2014

Table of Contents

1 Introduction 5

1.1 Audience 5

1.2 Document Purpose 5

1.3 Additional Information 5

1.4 ISS HA-ECT API 6

2 Module Specification 7

2.1 The ISS HA-ECT FIPS Object Module 9

2.1.1 Integrity Validation Data 9

2.1.2 Exclusivity of Integrity Tests 10

2.1.3 Run-time Integrity Validation 10

2.2 Ports and Interfaces 11

2.3 Approved and Allowed Cryptographic Algorithms 12

2.4 Non-Approved Cryptographic Algorithms 13

2.5 Approved Mode of Operation 14

2.5.1 Rules of Operation 16

2.6 Test Environment 17

3 Roles, Services, and Authentication 18

3.1 Roles and Services 18

3.2 Authentication 18

3.3 Authorized Services 19

3.4 Other Non-Approved Services 22

4 Operational Environment 23

4.1 Compatible Platforms 23

4.2 Software Security 24

4.3 Self-Tests 25

4.3.1 Power-up Self-Tests 26

4.3.2 Conditional Self-Tests 27

4.3.3 Critical Function Tests 27
4.3.4 Physical Security 27
4.3.5 Mitigation of Other Attacks 27
5 Design Assurance 28
5.1 Source Code Control 28
5.2 Application Management of Keys and Critical Security Parameters (CSPs) 28
5.2.1 Keys and Critical Security Parameters (CSPs) 28
5.2.2 Identifying Keys and CSPs 29
5.2.3 Key Generation 29
5.2.4 Storage of Keys and CSPs 30
5.2.5 Destruction of Keys and CSPs 30
6 Glossary 31
7 References 32
Appendix A Installation, Validation, and Initialization 29

Chapter 1

1 Introduction

This document is the Non-Proprietary FIPS 140-2 security policy for the Green Hills Software INTEGRITY Security Services High Assurance Embedded Cryptographic Toolkit v2.0 to meet FIPS 140-2 level 1 requirements. This Security Policy details the secure operation of the Green Hills Software INTEGRITY Security Services High Assurance Embedded Cryptographic Toolkit module, libect.o, as required in Federal Information Processing Standards Publication 140-2 (FIPS 140-2) as published by the National Institute of Standards and Technology (NIST) of the United States Department of Commerce.

In this document, the Green Hills Software INTEGRITY Security Services High Assurance Embedded Cryptographic Toolkit module is referred to as the *module*, or *ISS HA-ECT* or *ISS HA-ECT FIPS Object Module*.

1.1 Audience

This document is required as a part of the FIPS 140-2 validation process. It describes the ISS HA-ECT FIPS Object Module in relation to FIPS 140-2 requirements. The companion document HA-ECT API Guide is a technical reference for operators using, and system administrators installing, the ISS HA-ECT FIPS Object Module, for use in risk assessment reviews by security auditors, and as a summary and overview for program managers.

1.2 Document Purpose

This Security Policy document is available in Green Hills Software ISS HA-ECT distributions.

This document outlines the functionality provided by the module and gives high level details on the means by which the module satisfies FIPS 140-2 requirements.

1.3 Additional Information

For more information on the ISS HA-ECT please contact support-iss@ghs.com. For more information on NIST and the cryptographic module validation program, please visit http://csrc.nist.gov/cryptval/.

1.4 ISS HA-ECT API

The ISS HA-ECT FIPS Object Module, libect.o, is designed as a complete standalone crypto library. Applications can use the FIPS validated cryptographic functions of libect.o by linking the library into applications as needed and calling the appropriate ISS HA-ECT API functions directly.

Chapter 2

2 Module Specification

For the purposes of FIPS 140-2 validation, the ISS HA-ECT FIPS Object Module v2.0 is defined as a specific discrete unit of binary object code generated from a specific source configuration owned and managed by Green Hills Software. The module object code is created by Green Hills Software and placed into a library format appropriate for numerous operating systems. Each binary version of the module is suitable for a specific CPU architecture family. Each binary version can be reproduced on demand by Green Hills Software by using its configuration management system (CM) which contains specific tags to track each validated module. The module provides a cryptographic API (Application Programming Interface) to external applications.

The term Module elsewhere in this document refers to this ISS HA-ECT FIPS Object Module. For FIPS 140-2 purposes, the Module is classified as a multiple-chip standalone module.

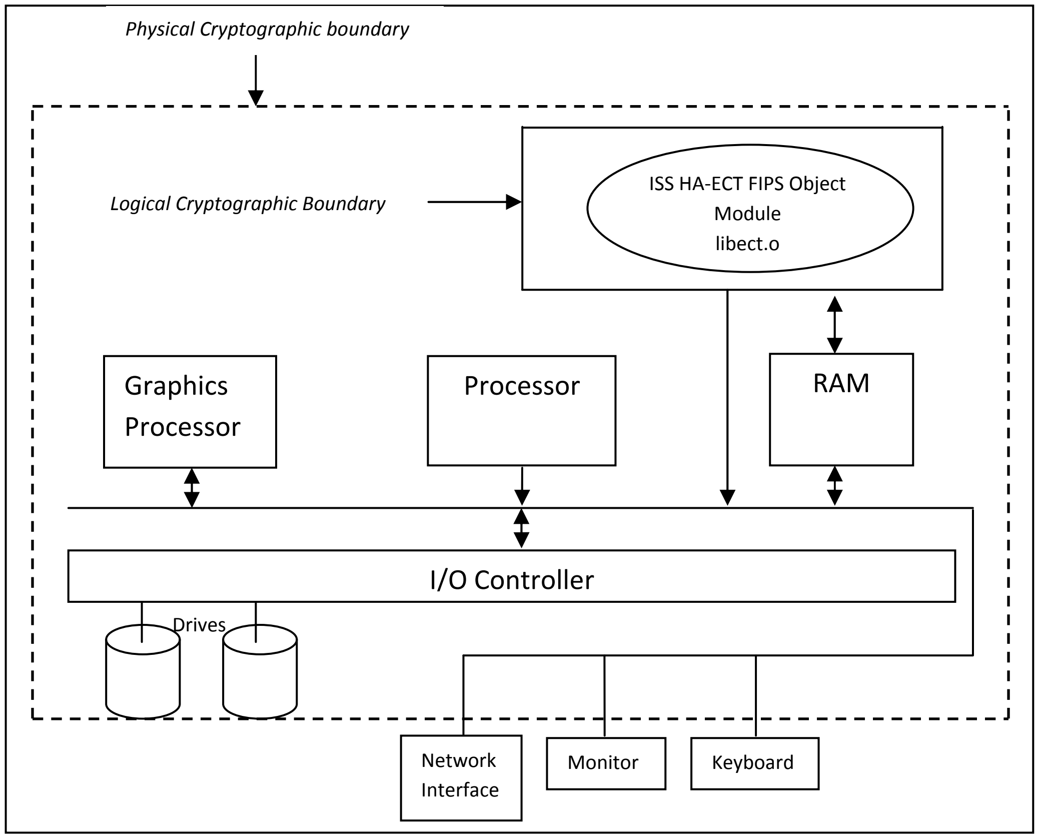
Physical Cryptographic boundary
Logical Cryptographic Boundary
ISS HA-ECT FIPS Object Module libect.o
Graphics Processor
Processor
RAM
I/O Controller
Drives
Network Interface
Monitor
Keyboard

Figure 1 – Physical/Logical Cryptographic Boundary

The *physical* cryptographic boundary of the Module is the enclosure of the general-purpose computing system on which it is executing with the *logical* cryptographic boundary of the Module is the ISS HA-ECT FIPS Object Module (Figure 1). The Module performs no communications other than with the process that calls it. It makes no network or interprocess connections and creates no files.

One instance of the Module was tested by the FIPS 140-2 Cryptographic Module Testing (CMT) laboratory for the specific platform:

Green Hills Software INTEGRITY Multivisor v4 for ARM

Development tools used by CMT to build for the target platform were:

INTEGRITY Multivisor
Green Hills Software MULTI v6.1.4 Compiler v2012.5.4 for ARM

The ISS HA-ECT FIPS Object Module, when generated from the identical unmodified source code, are "Vendor Affirmed" to be FIPS 140-2 compliant when running on other supported computer systems provided the conditions described in "Operational Environment", are met. On any platform, the Module is not considered FIPS 140-2 validated if the binary's integrity is not verified to match the known validated values listed in this Security Policy document.

The Module was designed and implemented to meet FIPS 140-2 requirements. In order to ensure FIPS 140-2 validated behavior, after linking, loading, and initializing the ISS HA-ECT FIPS Object Module within a runtime executable application , it is required that all rules and procedures listed in this Security Policy be followed by the operator at all times while the module operates in FIPS-mode.

The process of generating the runtime application from ISS HA-ECT FIPS Object Module is the same for all platforms and is documented in the ISS HA-ECT API Guide. The Module provides confidentiality, integrity, and message digest services. It natively supports the following algorithms: AES-GCM, AES-ECB, AES-CBC, AES-CMAC, AES-CCM, AES-XTS, AES-CTR DRBG, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, RSA, ECDSA, HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, as well as a selection of non-approved algorithms.

2.1 The ISS HA-ECT FIPS Object Module

Since the Module is distributed as prebuilt binary code, the integrity of the Module and executables which incorporate the Module is accomplished with a simple HMAC-SHA-1 digest verification.

2.1.1 Integrity Validation Data

The ISS HA-ECT FIPS Object Module file is protected by a HMAC-SHA-1 digest. This digest protects only object code belonging to the Module. The ISS HA-ECT FIPS Object Module is built from a set of object files linked into a single relocatable object file. This monolithic object file may be incorporated into a shared library or runtime executable application files, but in any event must be incorporated intact and in its entirety in order to operate in a FIPS 140-2 compliant mode.

The ISS HA-ECT FIPS Object Module's HMAC-SHA-1 digest is documented for each supported platform. Upon delivery, the system operator runs a Green Hills supplied utility program- ghash, which, using a FIPS validated crypto library (ie this library), computes and displays the HMAC-SHA-1 digests for the Module. The operator compares these digests with the known good documented values to gain assurance that the Module is indeed the FIPS-validated Module.

When the ISS HA-ECT FIPS Object Module is linked to the application by the operator, the Module is relocated, causing its contents (addressing) to change. The operator then runs the ghash utility program to re-compute the HMAC-SHA-1 digests, which must be stored into the final executable. These digests are verified at run-time to match the ghash-inserted values. The operator must provide assurance that the FIPS Object is not tampered with during the development process.

For operators of resource constrained devices who are forced to use a subset of the FIPS 140-2 validated module are able to do so by using only a subset of object files used to create the validated monolithic module; however, this method of usage is not considered FIPS 140-2 validated. Nevertheless, the operator can gain assurance that the objects come from the validated Object Module by performing the same HMAC-SHA-1 digest verification described above. Furthermore, the computation of the HMAC-SHA-1 digests inserted into the executable will still be run-time validated when the Crypto library is initialized.

The ISS HA-ECT FIPS Object Module is carefully isolated from all other application object code. This isolation is accomplished by collecting all of the ISS HA-ECT FIPS Object Module code and data into reserved, specially named program sections. These two sections and their contents are as follows:

- .ecttext: ISS HA-ECT FIPS Object Module executable code
- .ectrodata: ISS HA-ECT FIPS Object Module constant/read-only data

The ISS HA-ECT FIPS Object Module contains only the contents of these two fixed sections. The integrity of the ISS HA-ECT FIPS Object Module file is protected by two HMAC-SHA-1 digests, one for each of these contiguous program sections. The HMAC-SHA-1 digests are computed using the ghash utility and stored in another special read-only data section, .ecthash. This section also contains the reference points (address range) corresponding to the hash values. These reference points ensure that the run-time digest is computed over exactly the same memory that is used at build-time.

2.1.2 Exclusivity of Integrity Tests

Standard object file dump utilities can be used to verify that the ISS HA-ECT FIPS Object Module contains no other code/data other than what is found in the specially named sections. User application code is always placed into sections (e.g. .text, .data) that are not these special reserved sections. The operator can easily verify that the size of the special named FIPS sections are the same in the FIPS validated re-locatable Module as in the final executable. Therefore, no other memory is included in the HMAC_SHA1 validation.

2.1.3 Run-time Integrity Validation

When the ISS HA-ECT FIPS Object Module is initialized at run-time, the HMAC_SHA1 digests are computed dynamically for all three sections and then compared against the constant values stored in the .ecthash section. The HA-ECT FIPS Module self-tests can also be executed during normal program operation. This can be useful for periodic health tests to ensure that the ISS HA-ECT FIPS Object Module executable and read-only data (e.g. algorithm constants in tables) have not been corrupted relative to their build-time values. Note that the writable data section (.ectdata) can only be verified at module initialization time since the data contents are modified at run-time.

Note this integrity testing technique is conceptually similar to the case of firmware integrity testing: not only is external code omitted from the digest, but also non-executing ancillary data specific to the particular executable format. Only the actual machine code byte sequence directly executed by the CPU and the actual data referred to in the course of execution are included in the digest; i.e. only what really matters to the general purpose computer at runtime, pure code as with firmware.

2.2 Ports and Interfaces

For the purposes of this FIPS 140-2 validation, the Module is considered to be a multiple-chip standalone module. Although the Module is software, the physical embodiment is a general purpose computing platform that consists of multiple components considered to be a multichip standalone module by FIPS 140-2.

The logical cryptographic boundary for the Module is the discrete contiguous block of object code (the ISS HA-ECT FIPS Object Module) containing the machine instructions and data generated from the ISS HA-ECT FIPS source, as used by the calling application. The physical cryptographic boundary contains the general purpose computing hardware of the system executing the application. This system hardware includes the central processing unit(s), cache and main memory (RAM), system bus, and peripherals including disk drives and other permanent mass storage devices, network interface cards, keyboard and console and any terminal devices.

The Module provides a logical interface via an Application Programming Interface (API). This logical interface exposes services that applications may utilize directly or extend to add support for new data sources or protocols. The API provides functions that may be called by the referencing application.

The API interface provided by the Module is mapped onto the FIPS 140-2 logical interfaces: data input, data output, control input, and status output. Each of the FIPS 140-2 logical interfaces relates to the Module's callable interface, as follows:

- Data input: input parameters to all API functions that accept input from Crypto-Officer or User entities
- Data output: output parameters from all API functions that return data as arguments or return values from Crypto-Officer or User entities
- Control input: all API function input into the module by the Crypto-Officer and User entities
- Status output: API information returned via return/exit codes to Crypto-Officer or User entities

The API function specifications are included in the ISS HA-ECT project documentation which covers both FIPS-approved, and non-FIPS-approved functions.

2.3 Approved and Allowed Cryptographic Algorithms

The Module supports the following FIPS-approved cryptographic algorithms:

- Advanced Encryption Standard (FIPS 197)
- Secure Hashing Algorithm (SHA-1, SHA-224, SHA-256,384,512: FIPS 180-3)
- Rivest Shamir Adleman (RSA) digital signatures
- Keyed-Hash Message Authentication Code (HMAC: FIPS 198)
- Random Number Generator (DRBG: NIST SP 800-90A)
- Elliptic Curve Digital Signature Algorithm (ECDSA FIPS 186-4)

Algorithm Type	**Algorithm**	**Validation Certificate**	**Use**
Asymmetric keys	FIPS 186-4 RSA: SigGen: 2048-bit; SHA-224, SHA-256, SHA-384, SHA-512 SigVer: 1024 and 2048-bit; SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	1441	Sign and Verify operations
	FIPS 186-4 ECDSA using P-256, P-384 and P-521 curves	482	Sign, Verify, key pair generation operations
	FIPS 186-4 ECDSA Signature Generation Component using P-256, P-384 and P-521 curves	185 (CVL)	
Symmetric keys	AES with modes CBC, ECB, CTR - each with 128, 192, or 256-bit keys	2745	Encrypt/Decrypt operations
	CCM with 128, 192, or 256-bit keys	2748	
	AES XTS with 128 or 256-bit keys	2750	
	AES GCM with 128, 192, or 256-bit keys	2753	
	AES CMAC with 192 or 256-bit keys	2749	Generate operations
HMAC	HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	1724	Software and Message Integrity and Authenticity
Hashing	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	2319	Hashing

Algorithm Type	Algorithm	Validation Certificate	Use
RNG	SP 800-90A CTR DRBG using AES 128, 192, 256-bit	464	Random Number Generations
Key Exchange for System Level Key Agreement	Diffie-Hellman	Non-Approved. Allowed in FIPS-mode	Key Agreement (only primitives; not used to establish keys in the module)
	Elliptic-Curve Diffie-Hellman	Non-Approved. Allowed in FIPS-mode	Key Agreement (only primitives; not used to establish keys in the module)
Encrypt/Decrypt for key transport only	RSA Encrypt/Decrypt	Non-Approved. Allowed in FIPS-mode	Key Transport; not used to establish keys in the module
Key Wrap	AES Key Wrap using 128, 192, or 256-bit keys	(Underlying AES Cert. #2745.) Non-Approved. Allowed in FIPS-mode	Key Wrap; not used to establish keys in the module (only used for system level key establishment, which is beyond the scope of this module)
KDF	PBKDF2 per SP 800-132	Vendor affirmed	Returns the value of the derived key; not used to establish keys in the module

2.4 Non-Approved Cryptographic Algorithms

The following Non-FIPS-Approved algorithms are included in the Module. These Non-Approved cryptographic algorithms MUST NOT be used while operating the module in the FIPS-Approved mode.

Algorithm Type	Algorithm	Validation Certificate	Use
Asymmetric keys	FIPS 186-4 RSA: SigGen 1024-bit (all SHA sizes), 2048-bit (SHA-1)[1]	1441	Signature Generation operations
Symmetric Keys	Triple-DES (non-compliant)	N/A	Encrypt/Decrypt Operations
Hashing	MD5	N/A	Hashing

[1] These algorithm options are Disallowed as of January 1, 2014 per the NIST SP 800-131A algorithm transitions. Algorithms providing less than 112 bits of security strength are not allowed in the FIPS Approved mode of operation for use by Federal agencies.

HMAC	HMAC-MD5	N/A	Software Integrity and Message Integrity

2.5 Approved Mode of Operation

Any call to an approved cryptographic algorithm will automatically try and initialize the Module for operation in the FIPS 140-2 Approved mode, referred to herein as "FIPS mode", as depicted in Figure 2.

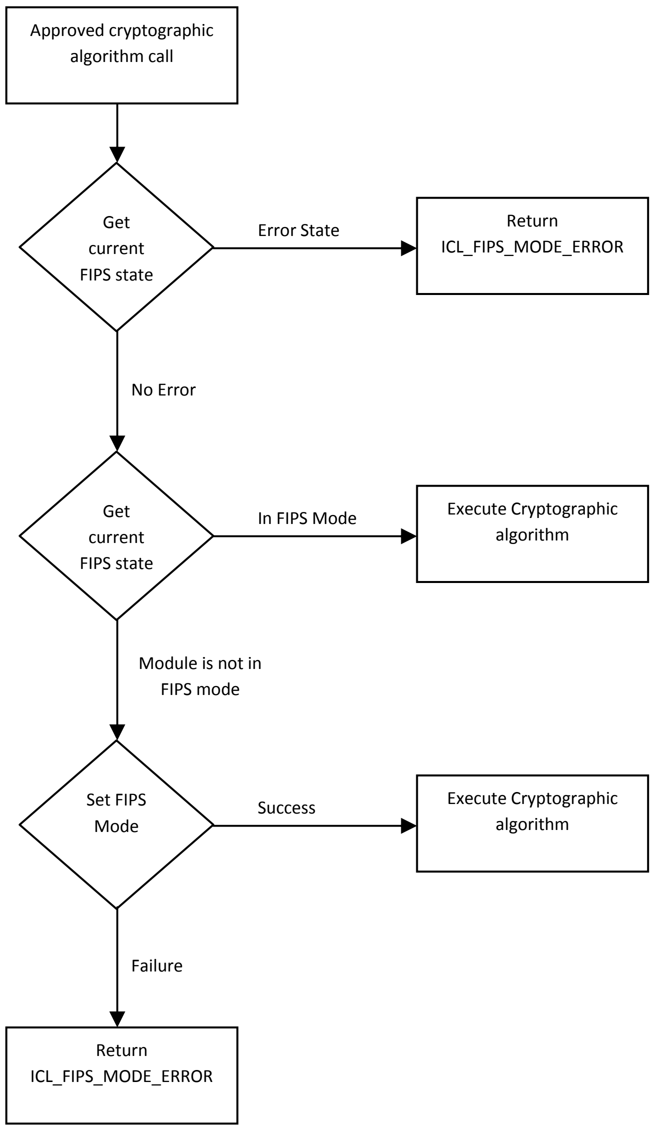
Approved cryptographic algorithm call
Get current FIPS state
Error State
Return ICL_FIPS_MODE_ERROR
No Error
Get current FIPS state
In FIPS Mode
Execute Cryptographic algorithm
Module is not in FIPS mode
Set FIPS Mode
Success
Execute Cryptographic algorithm
Failure
Return ICL_FIPS_MODE_ERROR

Figure 2 – Algorithm FIPS Mode initialization

All approved cryptographic algorithms when executed, will first call ect_GetState() to determine what state the Module is in. If the Module is in an error state, the algorithm will return

ICL_FIPS_MODE_ERROR indicating the Module had failed a previous attempt to enter FIPS mode. If the state is “FIPS mode”, the crypto algorithm will execute; if the Module is not in “FIPS mode”, a single initialization call to ect_Set_FIPS_Mode() will be made to initialize the Module for operation in “FIPS mode”. When the Module is in FIPS mode, all applicable power-up and conditional self-tests have been executed by the module. Use of the ect_Set_FIPS_Mode() function call is described in the API Guide. The Module is not in FIPS mode until this function call is made.

Prior to this invocation, the Module is uninitialized with the internal Module state set to ECT_STATE_NON_FIPS and internal Module error status set to ECT_ERR_UNINITIALIZED-indicating Non-FIPS mode by default. When executed, the ect_Set_FIPS_Mode() function call verifies the integrity of the runtime executable using HMAC_SHA-1 digests computed at build time. If the computed HMAC_SHA-1 digests match the pre-computed stored digest, then the power-up self-tests, consisting of the algorithm specific Sign/Verify and Known Answer tests, are performed.

If the integrity check or any component of the power-up self-test fails, ect_Set_FIPS_Mode() returns ECT_ERROR and the internal Module state is set to ECT_STATE_ERROR, indicating an internal failure. When in this error state, subsequent invocation of any cryptographic function calls will fail. Any self-test failure can only be cleared by a successful FIPS-mode invocation. Additional information regarding the failure can be obtained from a call to ect_Get_ErrorStatus(), documented in the HA_ECT API manual.

If all components of the power-up self-tests are successful, ect_Set_FIPS_Mode() returns ECT_SUCCESS, the Module state is set to ECT_STATE_FIPS_MODE, and the Module is now considered to be in FIPS mode. When in FIPS mode, the module operators should only use the algorithms and key sizes outlined in section ‘Approved Cryptographic Algorithms’ of this document.

If the Module is in “FIPS mode”, and a subsequent call is made to ect_Set_FIPS_Mode(), then ect_Set_FIPS_Mode() will not execute any self-tests; it will return ECT_ERROR_IN_FIPS_MODE, and the Module state will remain in ECT_STATE_FIPS_MODE.

At any time, the current Module state and Module error status can be obtained with calls to ect_GetState() and ect_GetErrorStatus() which will return ECT_STATE_FIPS_MODE and ECT_SUCCESS respectively, when the module is successfully placed in the FIPS-mode. Additional information for these routines is documented in the HA_ECT API manual.

Please note: The module operators MUST follow the instructions in Appendix A for a proper installation and initialization before the FIPS-enabling procedure can be followed. It is important that the Crypto-Officer ensures the correct libect.o file is used before proceeding with the FIPS-mode enabling procedure.

2.5.1 Rules of Operation

- The operator shall not make any changes to the Module's code while building an application.
- Only the HMAC-SHA-1 verified libect.o should be used without any modifications for a FIPS 140-2 compliant operation.
- The Module is initialized in the FIPS mode of operation (any call to an approve algorithm will automatically start initialization).
- Once in FIPS mode, users should only use algorithms and key sizes outlined in section "Approved and Allowed Cryptographic Algorithms" of this document.
- The operator/application must use appropriate entropy sources for generating the seeding material for the FIPS-Approved RNGs of the module. The entropy in the seeding material input to the module must be at least as much as the entropy in the key generated by the Approved RNG.
  For example, when generating a 256-bit AES key, the entropy in the seed key provided to the module by the user must be at least 256-bits.
- The replacement or modification of the Module by unauthorized intruders is prohibited. The Operating System enforces authentication method(s) to prevent unauthorized access to Module services.
- All host system components that can contain sensitive cryptographic data (main memory, system bus, disk storage) must be located in a secure environment.
- The referencing application accessing the Module runs in a separate virtual address space with a separate copy of the executable code.
- The unauthorized reading, writing, or modification of the address space of the Module is prohibited.
- 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.
- The operating system is responsible for multitasking operations so that other processes cannot access the address space of the process containing the Module.

2.6 Test Environment

The Module was tested by the FIPS 140-2 CMT laboratory on the following computer systems:

> **Samsung Galaxy Note II (ARM Cortex A9)**
> Green Hills Software INTEGRITY Multivisor v4 for ARM

Development tools used by CMT to build for target platforms were:

> **INTEGRITY Multivisor**
> Green Hills Software MULTI v6.1.4 Compiler v2012.5.4 for ARM

The ISS HA-ECT FIPS binary Object Module is “Vendor Affirmed” to be FIPS 140-2 compliant when running on other supported computer systems provided the conditions described in “Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program”, are met.

Chapter 3

3 Roles, Services, and Authentication

3.1 Roles and Services

The Module meets the FIPS 140-2 level 1 requirements for Roles and Services for User role and Crypto-Officer role.

The User and Crypto Officer roles are implicitly assumed by the entity accessing services implemented in the Module. The Crypto Officer role can install, initialize and remove the Module; this role is implicitly assumed while installing the Module or initializing the FIPS-mode on the module.

Role	**Authorized Services**
User Role	Access to all services (loading Module, calling API's) except installation and FIPS-mode initialization
Crypto Officer role	Module install and FIPS-mode initialization Uninstalling or removing the Module

3.2 Authentication

As allowed by FIPS 140-2, the Module does not support user identification or authentication for those roles. Only one role may be active at a time and the Module does not allow concurrent operators. The Module does not provide identification or authentication mechanisms that would distinguish between the two supported roles. These roles are implicitly assumed by the services that are accessed, and can be differentiated by assigning module installation and configuration services to the Crypto Officer.

3.3 Authorized Services[2]

The services provided by the Module are listed in the following table. All services may be performed in User role with the exception of Module installation and Initialization services which are the only services performed in the Crypto Officer role:

* Services impacted by the SP 800-131A algorithm transitions. It is the responsibility of the module operator to ensure that algorithms, modes, and key sizes Disallowed per NIST SP 800-131A are not used.

Service	Role	Critical Security Parameters	Algorithm	API Functions	Access
Symmetric Encryption/ Decryption	User	Symmetric Key	AES,	IclAES_Init, IclAES_ProcessBytes, IclAES_Finish, IclAESCMAC_Auth, IclAESCCM_Init, IclAESCCM_AuthEncrypt, IclAESCCM_AuthDecrypt, IclAESCCM_Finish, IclAESGCM_Init, IclAESGCM_AuthEncrypt, IclAESGCM_AuthDecrypt, IclAESGCM_Finish	Read Write execute
Digital Signature*	User	Asymmetric private key	RSA, ECDSA,	IclRSA_Decrypt, IclRSA_Encrypt, IclRSA_GenKeys, IclRSA_GenRandKeys, IclRSA_Sign, IclRSA_Verify, IclPKCS_Encode, IclPKCS_Sign, IclPKCS_Verify, IclECC_CalcPublicKey, IclECC_GenerateKey, IclECC_InitPoint, IclECC_IsValidKey, IclECDSA_Sign, IclECDSA_SignFromHash, IclECDSA_Verify, IclECDSA_VerifyFromHash	Read, Write, Execute
Message Digest	User	none HMAC key	SHA-1 SHA-256 SHA-384 SHA-512 HMAC	IclHash_Finish, IclSHA1_Finish, IclSHA224_Finish, IclSHA256_Finish, IclSHA384_Finish, IclSHA512_Finish, IclHash_HashBytes, IclSHA1_HashBytes,	Read, Write, Execute

[2] Note that only services using the algorithms listed in section "Approved Cryptographic Algorithms" are allowed for use in the FIPS-approved mode.

Service	Role	Critical Security Parameters	Algorithm	API Functions	Access
				IclSHA224_HashBytes, IclSHA256_HashBytes, IclSHA384_HashBytes, IclSHA512_HashBytes, IclHash_Init, IclHash_Update, IclHMAC_Auth, IclHMAC_Finish, IclHMAC_Init, IclHMAC_SHA1_Auth, IclHMAC_SHA224_Auth, IclHMAC_SHA256_Auth, IclHMAC_SHA384_Auth, IclHMAC_SHA512_Auth, IclHMAC_Update	
Random Number Generation	User	Seed key	SP800-90A	IclRand, IclReseed	Read Write Execute
Show Status	User	None	N/A	ect_GetState, ect_GetErrorStatus	None
Module Install and Initialization	Crypto Officer	HMAC-SHA-1 Integrity Key	N/A	ect_Set_FIPS_Mode	Read Execute
Uninstalling or removing the module	Crypto-Officer	None	None	None	None
Perform Power-up Self-Tests	User	HMAC-SHA-1 Integrity Key	N/A	ect_SelfTest	Read Execute
Key Establishment [3]	User	Asymmetric public and private keys	ECDH, DH RSA	IclDH_CalcSharedSecret, IclDH_GenParams, IclDH_GetPublicKey, IclDH_Init, IclDH_InitRand, IclDH_SetParams IclECDH_CalcSharedSecret, IclECDH_GetPublicKey, IclECDH_Init	Read Write Execute
Zeroization	User	Asymmetric and symmetric keys	ECC RSA AES HMAC	IclRSA_DestroyKeys, IclECC_DestroyKeys, IclHMAC_Finish, IclAES_Finish, IclFinish	Read Zeroize
Big Number Operations	User			IclBigNum_Add, IclBigNum_AddUint, IclBigNum_Assign, IclBigNum_Cmp, IclBigNum_CmpUint,	Read, Write, Execute

[3] This service does not establish a key in the module; rather, it is used by the calling application as part of a system level key establishment scheme.

Service	Role	Critical Security Parameters	Algorithm	API Functions	Access
				IclBigNum_DivMod, IclBigNum_ModMulInv, IclBigNum_Mul, IclBigNum_Pack, IclBigNum_Sub, IclBigNum_SubUint, IclBigNum_Unpack, IclBigNum_ZeroExtend, IclBigNumMont_ExpMod, IclBigNumMont_Init	

3.4 Other Non-Approved Services[4]

Algorithm	Module's APIs
Triple-DES,	IclTDES_Finish, IclTDES_Init, IclTDES_ProcessBytes
MD5	IclMD5_HashBytes IclMD5_Finish
HMAC-MD5	IclHMAC_MD5_Auth

[4] These services MUST not be used in the FIPS-approved mode.

Chapter 4

4 Operational Environment

The ISS HA-ECT FIPS Object Module is generated from source code available for use on a wide variety of computer hardware and operating system platforms. Applications referencing the ISS HA-ECT FIPS Object Module run as processes under the control of the host system operating system. Modern operating systems segregate running processes into virtual memory areas that are logically separated from all other processes by the operating system and CPU. NOTE: in some simple computer systems (sometimes running on microprocessors without memory protection hardware), the ISS HA-ECT FIPS Object Module necessarily is linked together in the same memory space as the other application code. However, the ISS HA-ECT FIPS Object Module was primarily designed for modern computer systems that provide process protection.

The ISS HA-ECT FIPS Object Module functions completely within the process space of the process which loads it. The Module does not communicate with any processes other than the one that loads it, and satisfies the FIPS 140-2 requirement for a single user mode of operation.

The ISS HA-ECT FIPS Object Module was tested on specific hardware/software environments. As stated in "Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program", the binary Module maintains FIPS 140-2 validation on other hardware and operating systems which were not included as part of the validation testing and without retesting the cryptographic module on the new OS(s) and/or GPC(s). However, the CMVP makes no statement as to the correct operation of the module when executed on an OS(s) and/or GPC(s) not listed on the validation certificate. The module validated by the CMVP and which assurance is provided is based as caveated on the validation certificate and operated on the reference operating systems annotated on the certificate.

4.1 Compatible Platforms

The Module is designed to run on a very wide range of hardware and software platforms as long as the conditions in FIPS 140-2 Implementation Guidance G.5 are met. Any such computing platform that meets the conditions listed above can be used to host a FIPS 140-2 validated Module generated in accordance with this Security Policy. Such use will be considered "vendor-affirmed" for platforms other than those used for FIPS 140-2 testing as specified in this Security Policy.

If any platform specific errors occur that can only be corrected by modification of the Module source files, then the Module will not be validated for that platform. Note also that future releases of ISS HA-ECT may add support for additional platforms requiring new platform specific source replacing parts of the source corresponding to the current validated Module, in which case the modified Module will require revalidation under FIPS/NIST guidelines.

Note there is a possibility that the introduction of a new platform may be incompatible with the design of the integrity test, preventing a valid verification. The implementation of the integrity test is designed to fail for any such unrecognized platforms.

4.2 Software Security

Multiple integrity checks are performed in the process of generating and running an application using the Module:

- The integrity of the binary ISS HA-ECT FIPS Object Module file is checked before generating the runtime executable application.

When the ISS HA-ECT FIPS Object Module file has been built and validated, a set of HMAC-SHA-1 digests for the FIPS 140-2 validated Module are documented in this Security Policy for each validated platform. When the operator is about to incorporate the ISS HA-ECT FIPS Object Module into the application, the ghash utility is used to compute and display the HMAC-SHA-1 digests corresponding to the Module's code and data. These digests must match the known good digests documented in this Security Policy in order for the module to be considered FIP 140-2 validated and the operator to proceed.

- The integrity of the FIPS Module within the application is checked at run-time.

After the application is generated by linking with the validated Module, the ghash utility is to be used to re-compute the SHA1 digests of the ISS HA-ECT library. These digests should be embedded within a special read-only data section in the application executable. The ISS HA-ECT library's FIPS initialization function, ect_Set_FIPS_Mode(), will dynamically compute the SHA-1 digests and compare them against these build-time stored values to make sure they match. This run-time check ensures that the ISS HA-ECT library has not been corrupted between application build-time and system run-time.

This chain of integrity checks assures that applications using ISS HA-ECT will use FIPS 140-2 validated cryptography when built using the validated ISS HA-ECT FIPS Object Module.

4.3 Self-Tests

The Module performs a number of power-up and conditional self-tests to ensure proper operation of the Module. Power-up self-tests include cryptographic algorithm known answer tests and integrity tests. Input, output, and cryptographic functions cannot be performed while the Module is in a self-test or error state as the module is single threaded and will not return to the calling application until the power-up self tests are complete. If the power-up self tests fail, subsequent calls to the module will fail and thus no further cryptographic operations are possible.

Any call to an approved cryptographic algorithm will automatically try and initialize the Module for operation in the FIPS 140-2 Approved mode, provided the Module is NOT in FIPS mode, and is NOT in an error state (i.e. prior FIPS mode initialization failed). Within the approved cryptographic algorithm call, a single call to ect_Set_FIPS_mode(), will initialize the Module for operation in the FIPS 140-2 Approved mode. It returns 0 (ECT_SUCCESS) on success. When the Module is in FIPS mode, all security functions and cryptographic algorithms are performed in Approved mode. Prior to this invocation, the Module is uninitialized in the non-FIPS mode by default.

The integrity tests called from the ect_Set_FIPS_mode() function are performed using HMAC-SHA-1 digests calculated over the object code in the ISS HA-ECT FIPS Object Module. The HMAC-SHA1 key, which is used only for the power-up integrity test, is a key value hard-coded in the module's code. Since this key is used solely for the integrity test, it is not required to be zeroized. The module also stores the integrity hashes, which get integrated into the module upon computation during the library build process. If the computed HMAC-SHA-1 digest matches the stored known digest, then the power-up self-tests consisting of the algorithm specific Sign/Verify and Known Answer tests, are performed. The failure of any power-up self-test or conditional test causes the Module to enter the ECT_STATE_ERROR state with all cryptographic operations disabled until the Module is reinitialized with a successful ect_Set_FIPS_Mode() call. Any function that invokes the conditional test that fails will return -1 (ECT_FAILURE). If all power-up self-tests are successful, ect_Set_FIPS_mode() returns

Power-up tests are run automatically when the Module is initialized and do not require any inputs or actions from the module operator. Additionally, power-up tests may be executed at any time by calling the ect_SelfTest() API function and verifying it returns 0 (ECT_SUCCESS) to indicate success. No FIPS-mode cryptographic functionality will be available until after successful execution of all power-up tests. No authentication is required to perform self-tests either automatically or upon demand.

4.3.1 Power-up Self-Tests

Known Answer Tests (KATs) are tests where a cryptographic value is calculated and compared with a stored previously determined answer. The following power-up self-tests are implemented by the Module:

Algorithm	Power-up Self-test
AES-CBC-128	encryption and decryption KATs with 128-bit key
AES-CBC-192	encryption and decryption KATs with 192-bit key
AES-CBC-256	encryption and decryption KATs with 256-bit key
AES-GCM-128	authenticated encryption and decryption KATs with 128-bit key
AES-GCM-192	authenticated encryption and decryption KATs with 192-bit key
AES-GCM-256	authenticated encryption and decryption KATs with 256-bit key
AES-ECB-128	encryption and decryption KATs with 128-bit key
AES-ECB-192	encryption and decryption KATs with 192-bit key
AES-ECB-256	encryption and decryption KATs with 256-bit key
AES CCM-128	authenticated encryption and decryption KATs with 128-bit key
AES CCM-192	authenticated encryption and decryption KATs with 192-bit key
AES CCM-256	authenticated encryption and decryption KATs with 256-bit key
XTS AES	encryption and decryption KATs with 128 bit key
AES Key wrap	encryption and decryption KATs with 128 bit key
RSA-2048 PKCS#1	sign and verify KATs with 2048-bit key
RSA-2048	Encrypt and decrypt KATs with 2048-bit key
SHA-1	KAT for SHA-1
SHA-256	KAT for SHA-256
SHA-384	KAT for SHA-384
SHA-512	KAT for SHA-512
HMAC-SHA-1	KAT for HMAC-SHA-1
HMAC-SHA-256	KAT for HMAC-SHA-256
HMAC-SHA-384	KAT for HMAC-SHA-384
HMAC-SHA-512	KAT for HMAC-SHA-512
SP 800-90A DRBG	KAT for FIPS SP 800-90A
ECDSA nist256p	sign/verify pairwise consistency test with 256 bit key
ECDSA nist384p	sign/verify pairwise consistency test with 384 bit key
ECDSA nist521p	sign/verify pairwise consistency test with 521 bit key
PBKDF2	KAT for PBKDF2 with SHA-256
EC Diffie-Hellman	KAT: Shared secret calculation with P-256
Diffie-Hellman	KAT: Shared secret calculation with 1024-bit key

4.3.2 Conditional Self-Tests

Algorithm	Conditional Test
RSA	Pair-wise consistency sign/verify test
ECDSA	Pair-wise consistency sign/verify test
DRBG	Continuous random number generator test

4.3.2.1 Sign/Verify Test

A sign/verify test is performed when RSA key pairs or ECDSA key pairs are generated by signing data with a private key and verifying that signature using the associated public key.

4.3.2.2 Software/Firmware Load Test

Not applicable; the Module does not allow the loading of any external software or firmware.

4.3.2.3 Manual Key Entry Test

Not applicable; keys are not manually entered into the Module.

4.3.2.4 Bypass Test

Not applicable; the Module does not implement a bypass capability.

4.3.3 Critical Function Tests

The Module does not implement any critical function tests.

4.3.4 Physical Security

The Module does not claim to enforce any physical security as it is implemented entirely in software.

4.3.5 Mitigation of Other Attacks

The Module does not mitigate against any specific attacks.

Chapter 5

5 Design Assurance

The Module is managed in accordance with the established configuration management and source version control procedures of the Green Hills Software end user products group. These plans and procedures form part of a comprehensive quality management and improvement system used for all of Green Hills Software's critical products, including operating systems and compilers.

In addition, the ISS HA-ECT Toolkit Module is managed with additional mechanisms to assure the integrity of binary code as delivered and used to create applications.

5.1 Source Code Control

Software development functions for ISS HA-ECT software (configuration management, version control, change control, software defect tracking) are managed by the ISS group. The source code revisions are maintained in an SVN repository within Green Hills Software's private development team. Individually packaged revisions can be released periodically in electronic archive or portable media form. The integrity of the Module is based on HMAC-SHA-1.

5.2 Application Management of Keys and Critical Security Parameters (CSPs)

5.2.1 Keys and Critical Security Parameters (CSPs)

A Key and Critical Security Parameter (CSP) is information, such as symmetric keys, asymmetric private keys, etc., that must be protected from unauthorized access. Since the Module is accessed via an API from a referencing application, the Module does not manage Keys and CSPs.
The application designer and the end operator share a responsibility to ensure that Keys and CSPs are always protected from unauthorized access. This protection will generally make use of the security features of the host hardware and software which is outside of the cryptographic boundary defined for this Module.

5.2.2 Identifying Keys and CSPs

All Keys and CSPs must be created, stored, and destroyed in an approved manner as described by the FIPS PUB 140-2, Security Requirements for Cryptographic Modules. Keys and CSPs are those items of information which must be protected from disclosure, modification, and substitution, such as symmetric keys, asymmetric private keys, etc. Note that the application designer and operator/system administrator/Crypto Officer share a responsibility for protection of Keys and CSPs; the former to include appropriate technical protections and the latter to install and configure the application correctly. Technical protections include checks to require that files storing Keys and CSPs have appropriate permissions (not group writable or world readable, for example). Administrative protections include installation of the runtime software (executables and configuration files) in protected locations. End users have a responsibility to refrain from compromising CSPs (as by sending a password in clear text or copying an encryption key to an unprotected location).

The module includes the following keys and CSPs:

- AES Keys
- RSA Private Key*
- ECDSA Private Key
- HMAC-SHA Key
- Diffie-Hellman Private Key
- Elliptic-Curve Diffie-Hellman Private Key
- DRBG State

The module includes the following public keys:

- RSA Public Key*
- ECDSA Public Key
- EC DH Public Key
- DH Public Key

*CSPs/public keys impacted by the SP 800-131A algorithm transitions. It is the responsibility of the module operator to ensure that algorithms, modes, and key sizes Disallowed per NIST SP 800-131A are not used.

5.2.3 Key Generation

The Module API provides cryptographic functions used for asymmetric key generation for RSA, ECDSA and for generation of public and private parameters for DH and ECDH . For example, a call to the IclRSA_GenRandKeys ()API function would be used to generate RSA keys. The control input that drives the invocation of the Module API functions comes from the calling application.

5.2.4 Storage of Keys and CSPs

The Module only stores the HMAC-SHA-1 key used for the integrity self-test within its binary file. If this key value is modified, the HMAC-SHA-1 value of the module's binary libect.o before linking with the application, will be different than the values published in this Security Policy, Appendix A, and will not be considered FIPS-compliant. The Module does not store any other Keys or CSPs in persistent media; while the Module is initialized, all Keys and CSPs reside temporarily in RAM and are destroyed at the end of the session.

5.2.5 Destruction of Keys and CSPs

When no longer needed, Keys and CSPs contained within the application must be deleted by overwriting in a way that will make them unrecoverable (zeroized). The HA-ECT accomplishes this by the user calling algorithm **_Finish() API's or when power is removed from the hardware. For asymmetric keys only, Icl*_DestroyKeys() make keys unrecoverable by zeroization.

Chapter 6

6 Glossary

Term	Description
AES	Advanced Encryption Standard
API	Application Programming Interface
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CMT	Cryptographic Module Testing
CMVP	Cryptographic Module Validation Program
CO	Crypto Officer
CSP	Critical Security Parameter
CTR	Counter
DH	Diffie-Hellman
ECB	Electronic Codebook
FIPS	Federal Information Processing Standard
FSM	Finite State Machine
GCM	Galois Counter Mode
HA-ECT	High Assurance Embedded Crypto Toolkit (ISS Crypto library)
IV	Initialization Vector
KAT	Known Answer Test
NIST	National Institute of Standards and Technology (USA)
OFB	Output Feedback
OS	Operating System
RSA	Rivest, Shamir and Adleman
SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	Secure Hash Algorithm (FIPS 180-3)
HMAC -SHA-1 -SHA-224 -SHA-256 -SHA-385 -SHA-512	Keyed-Hash Message Authentication Code (FIPS 198-1)

Chapter 7

7 References

- FIPS PUB 140-2, Security Requirements for Cryptographic Modules, May 2001, National Institute of Standards and Technology
- FIPS PUB 197, Advanced Encryption Standard (AES), 26 November 2001, National Institute of Standards and Technology
- FIPS 186-3, The Digital Signature Standard, June 2009, National Institute of Standards and Technology
- The Advanced Encryption Standard Algorithm Validation Suite (AESAVS) , 15 November 2002, National Institute of Standards and Technology
- Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program, Initial Release: March 28, 2003, Last Update: December 23, 2010, National Institute of Standards and Technology
- FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC), July 2008, National Institute of Standards and Technology
- FIPS 180-3, Secure Hash Standard (SHS), October 2008, National Institute of Standards and Technology
- Derived Test Requirements for FIPS PUB 140-2, Security Requirements for Cryptographic Modules, January 04, 20011 (draft), National Institute of Standards and Technology
- Green Hills Software ISS HA-ECT API Guide

Appendix A – Installation, Validation and Initialization

These instructions assume the following:

- The reader has the basic knowledge of how to unpack the Module's binary distribution.
- Target environment is as specified in section 2 of this policy
- The reader has loaded Green Hills Software INTEGRITY Multivisor Software Development Kit (SDK) containing version 6.1.4, Compiler v2012.5.4 of Green Hills Software MULTI development environment for ARM, Green Hills Software INTEGRITY Multivisor v4 for ARM in order to execute the instructions below.
- The reader has experience using Green Hills Software Tools and the INTEGRITY Operating system.

A.1 Module Validation

The Green Hills Software, ISS ISS HA-ECT FIPS Object Module v2.0 consists of the FIPS Object Module (the *libect.o* contiguous unit of binary object code) generated from the specific source files found in the specific Green Hills distribution, *ISS_HA_ECT.tar.gz.* The set of files specified in this tar file constitutes the complete set of source files of this module. There shall be no additions, deletions, or alterations of this set as used during module build.

The Green Hills Software build environment for the tested platforms consists of ***Compiler v2012.5.4 for ARM.*** Green Hills Software's ***INTEGRITY Multivisor v4*** was the targeted ARM Operating systems.

After receiving the libect.o library from Green Hills Software, generate and print the HMAC-SHA-1 digests for the installed Module using the Green Hills ghash utility (provided as part of the Module distribution). The following command can be used to display the digests of the Module:

ghash –p libect.o

The Crypto Officer should compare these digests with the below published values for your applicable platform to confirm that the Module is authentic and unmodified.

INTEGRITY Multivisor (ARM) HMAC-SHA-1 digests:

```
.ectrodata HMAC-SHA-1 Hash: C3CD6B57611104EE7638F32812AD39B365AF504B
.ecttext   HMAC-SHA-1 Hash: B81BEC1A033063AA32BA6423E314228706CBCA19
```

A.2 Installing and Protecting the ISS HA-ECT FIPS Object Module

The Module should be installed within the operator's configuration management system by the Crypto officer such that it is protected from unauthorized modification. In particular, operators using the object Module to build an application should only have read access to the object Module.

A.3 Linking the Runtime Executable Application

Once the Crypto Officer has confirmed the digests match, ensuring the correct/valid crypto library is being used, the library can be linked into the application by adding libect.o to each corresponding program ".gpj" file. An include path pointing to the crypto toolkit "include" directory should also be added to the program ".gpj" providing application access to crypto routines.

After the user application has been compiled, and the Module has been linked with the application, the ghash utility must be executed (prior to integration for INTEGRITY targets) to compute and embed the application-specific HMAC-SHA-1 digests that protects the Module's integrity. The following command is used:

ghash <application_name>

The application is now considered Module-enabled.

The Crypto officer should install the Module-enabled application in a location protected by the host operating system security features. These protections should allow write access only to Crypto Officers and read access only to authorized users. Note that applications code interfacing with the ISS HA-ECT FIPS Object Module are outside of the cryptographic boundary, although the entire application must be write-protected since the Module is now linked with the application.

Failure to embed the digest in the executable object will prevent initialization of FIPS mode. At runtime, the ect_Set_FIPS_Mode() function compares the HMAC-SHA-1 digests from the module's binary with the digests generated from the Module-enabled application. These digests are the final link in the chain of validation from the original Module binary to the runtime executable application file.

A.4 FIPS Mode Initialization

Somewhere very early in the execution of the application, FIPS mode must be enabled. This should be done by invocation of the ect_Set_FIPS_Mode() function call as in this example:

```
if (ect_Set_FIPS_Mode() != ECT_SUCCESS) ||

   (ect_Set_FIPS_Mode() != ECT_ERROR_IN_FIPS_MODE)

    fprintf(stderr,”***FIPS SELF TEST FAILURE***\n”);

else

    fprintf(stderr,"*** IN FIPS MODE ***\n");
```

Note that it is permitted to _not_ enable FIPS mode, in which case, the ISS HA-ECT FIPS Object Module used in conjunction with the ISS HA-ECT API shall function in a non-FIPS 140-2-validated mode.