XYPRO Technology Corporation

XYGATE /ESDK

Software Version: 3.3.2

FIPS 140-2 Non-Proprietary Security Policy

FIPS Security Level: 1
Document Version: 0.4

Prepared for:

XYPRO Technology Corporation
3325 Cochran Street, Suite 200
Simi Valley, CA 93063
USA

Phone: +1 (805) 583-2874
Email: info@xypro.com
http://www.xypro.com

Prepared by:

Corsec Security, Inc.
10340 Democracy Lane, Suite 201
Fairfax, VA 22030
USA

Phone: +1 (703) 267-6050
Email: info@corsec.com
http://www.corsec.com

Table of Contents

Table of Figures

List of Tables

1 Introduction

1.1 Purpose

This is a non-proprietary Cryptographic Module Security Policy for the XYGATE /ESDK from XYPRO Technology Corporation. This Security Policy describes how the XYGATE /ESDK meets the security requirements of Federal Information Processing Standards (FIPS) Publication 140-2, which details the U.S. and Canadian Government requirements for cryptographic modules. More information about the FIPS 140-2 standard and validation program is available on the National Institute of Standards and Technology (NIST) and the Communications Security Establishment Canada (CSEC) Cryptographic Module Validation Program (CMVP) website at http://csrc.nist.gov/groups/STM/cmvp.

This document also describes how to run the module in a secure FIPS-Approved mode of operation. This policy was prepared as part of the Level 1 FIPS 140-2 validation of the module. The XYGATE /ESDK is referred to in this document as the cryptographic module or the module.

1.2 References

This document deals only with operations and capabilities of the module in the technical terms of a FIPS 140-2 cryptographic module security policy. More information is available on the module from the following sources:

- The XYPRO website (http://www.xypro.com) contains information on the full line of products from XYPRO.
- The CMVP website (http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm) contains contact information for individuals to answer technical or sales-related questions for the module.

1.3 Document Organization

The Security Policy document is one document in a FIPS 140-2 Submission Package. In addition to this document, the Submission Package contains:

- Vendor Evidence document
- Finite State Model document
- Other supporting documentation as additional references

This Security Policy and the other validation submission documentation were produced by Corsec Security, Inc. under contract to XYPRO. With the exception of this Non-Proprietary Security Policy, the FIPS 140-2 Submission Package is proprietary to XYPRO and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact XYPRO.

2 XYGATE /ESDK

2.1 Overview

The XYPRO XYGATE /ESDK cryptographic module is packaged as a dynamically-linked library. The library provides the following basic functionalities:

- Symmetric key encryption including Advanced Encryption Standard (AES), Triple Digital Encryption Standard (Triple-DES), Skipjack, Digital Encryption Standard (DES) and others
- Hashing including Secure Hash Algorithm-1 (SHA-1), SHA-256, Message Digest 5 (MD5), Hashed Message Authentication Code (HMAC) and others
- Public key encryption using Rivest-Shamir-Adleman (RSA)
- Digital signature algorithms including RSA, Digital Signature Algorithm (DSA), ElGamal, and others
- Secure session protocols such as Secure Shell (SSH), Secure Sockets Layer (SSL), and Transport Layer Security (TLS)
- E-mail protocols such as Pretty-Good-Privacy (PGP) and Secure Multipurpose Internet Mail Extensions (S/MIME)

Section 2.7 below details which of the above-referenced algorithms are (and are not) approved for use in a FIPS-Approved mode of operation.

The XYPRO XYGATE /ESDK is validated at the following FIPS 140-2 Section levels:

Table 1 – Security Level Per FIPS 140-2 Section

Section	Section Title	Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services, and Authentication	1
4	Finite State Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC[1]	1
9	Self-tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A

2.2 Module Specification

The XYGATE /ESDK is a software module with a multi-chip standalone embodiment. The overall security level of the module is 1. The cryptographic module was tested and found compliant on the following platforms:

- Windows XP w/ SP3
- Hewlett-Packard (HP) Nonstop Server G06.31 (PIC[2] and non-PIC)

[1] EMI/EMC – Electromagnetic Interference / Electromagnetic Compatibility

- HP Nonstop H06.13
- HP Nonstop J06
- HP Nonstop OSS[3] Server G06.31 (PIC and non-PIC)
- HP Nonstop OSS H06.13
- HP Nonstop OSS J06
- HP-UX 10.2
- HP-UX B.11.11
- Solaris 10
- IBM AIX 5.2
- SuSE Linux 10
- Red Hat Enterprise Linux (RHEL) 5.1
- IBM z/OS 1.11

The following sections define the physical and logical boundary of the module.

2.2.1 Physical Cryptographic Boundary

As a software cryptographic module, there are no physical security mechanisms implemented. The module must rely on the physical characteristics of the general-purpose computer (GPC) on which it runs. The physical cryptographic boundary of the XYGATE /ESDK is defined by the hard metal enclosure around the host computer. Figure 1 below depicts the standard hardware platform with accompanying physical ports, the dotted line surrounding the module components represents the module's physical cryptographic boundary, while the ports and interfaces exist at the boundary and interface with the various controllers.

[2] PIC – Position-Independent Code
[3] OSS – Open System Services

.

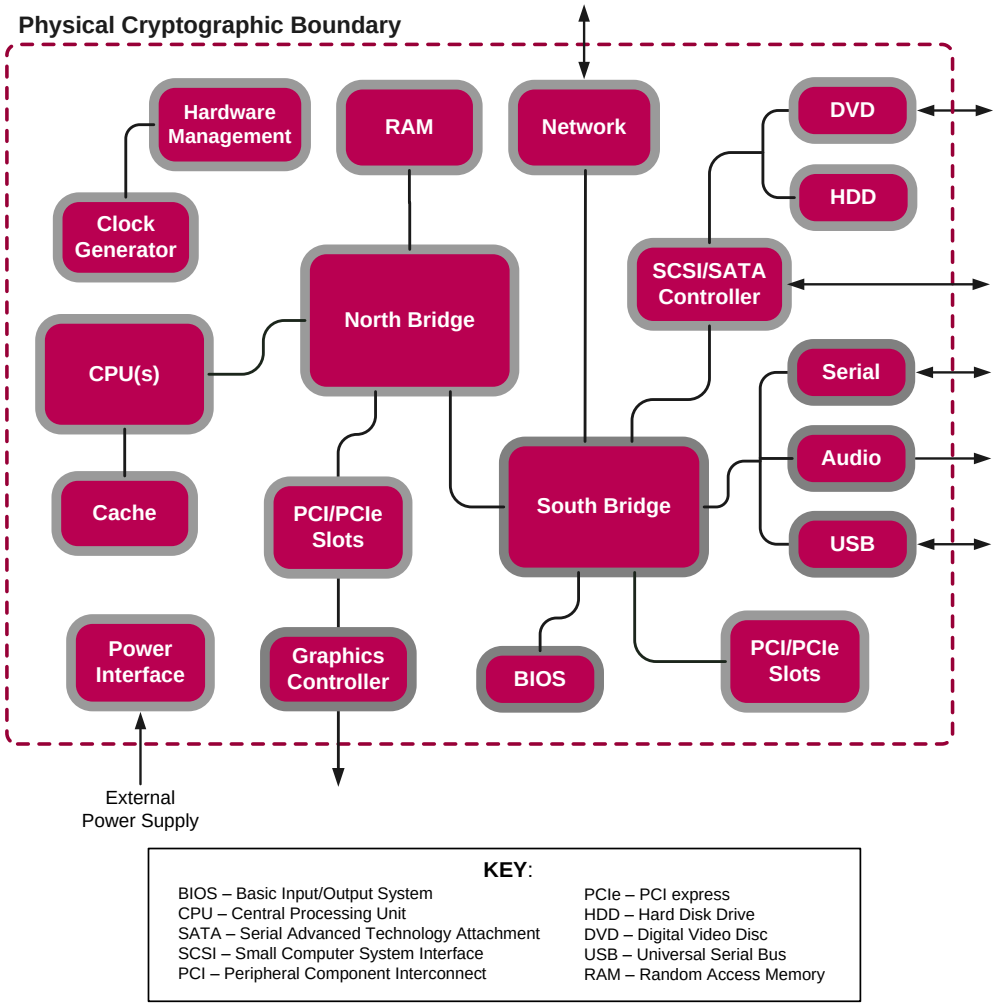

Figure 1 – Standard GPC Block Diagram

2.2.2 Logical Cryptographic Boundary

The logical cryptographic boundary of the XYGATE /ESDK is defined by the library and the signature file that contains the HMAC value of the library and the signature file. The name of the library is dependent on the host platform. The library names are:

- esdklib.dll (Windows)
- ESDKLIB (HP Nonstop, z/OS)
- esdklib.sl (HP-UX)
- esdklib.so (Solaris, AIX, Linux)

Figure 2 below shows a logical block diagram of the module. The module's logical cryptographic boundary encompasses all functionality contained within the library.

.

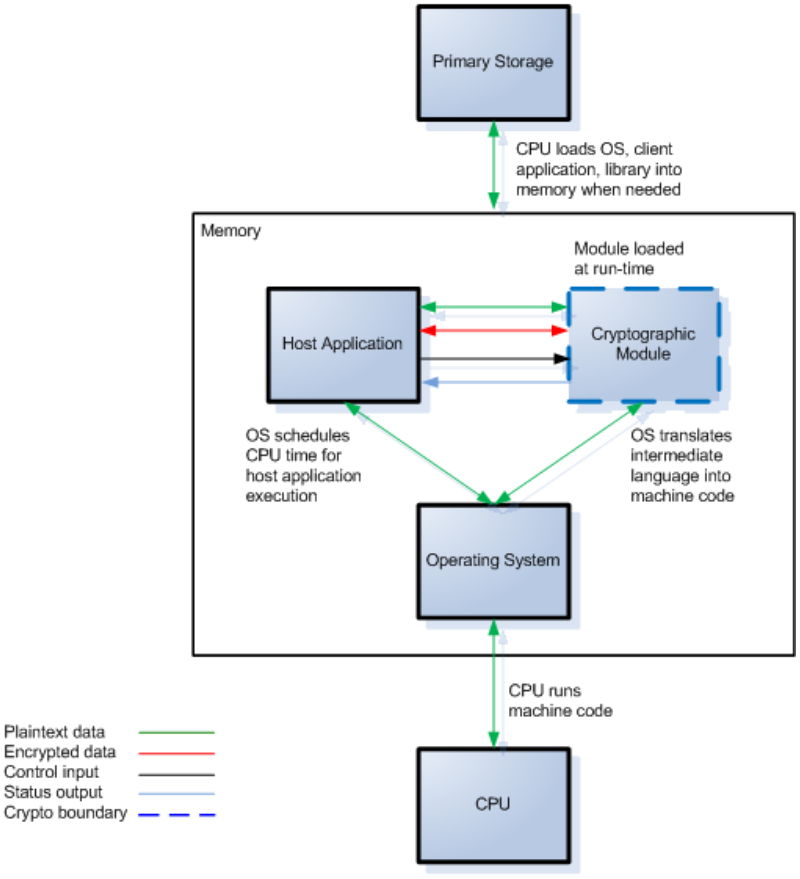

Figure 2 – Logical Block Diagram and Cryptographic Boundary

2.3 Module Interfaces

The module's logical interfaces exist at a low level in the software as an Application Programming Interface (API). The API interface is mapped to the following five logical interfaces:

- Data input
- Data output
- Control input
- Status output
- Power

As a software module, the module has no physical characteristics. Thus, the module's manual controls, physical indicators, and physical and electrical characteristics are those of the GPC, as depicted in Figure 1. The FIPS-defined interfaces map to their physical and logical counterparts as described in Table 2 below.

Table 2 – FIPS Interface Mappings

FIPS Logical Interface	Physical Interface Mapping (Standard GPC)	Logical Interface Mapping (Module)
Data Input Interface	Network ports, serial ports, USB, DVD drive	The API calls that accept input data for processing through their arguments

FIPS Logical Interface	Physical Interface Mapping (Standard GPC)	Logical Interface Mapping (Module)
Data Output Interface	Network ports, serial ports, USB, DVD drive	The API calls that return by means of their return codes or arguments generated or processed data back to the caller
Control Input Interface	Network ports, power button	The API calls that are used to initialize and control the operation of the module
Status Output Interface	LEDs, network ports, audio port, DVD drive	Return values for API calls
Power Interface	Power connector	Not Applicable

2.4 Roles and Services

The module supports role-based authentication. There are two roles in the module (as required by FIPS 140-2) that operators may assume: a Crypto Officer role and User role.

2.4.1 Crypto Officer Role

The Crypto Officer role has the ability to configure the module to run in a FIPS-Approved mode. The table below lists the individual functions contained within the module.

Table 3 – ESDK Cryptographic Library Functions

cryptAddCertExtension	*cryptFlushData*	*cryptCreateSession*	*cryptDestroyCert*
cryptDeviceOpen	*cryptCAGetItem*	*cryptImportCert*	*cryptQueryCapability*
cryptAddPrivateKey	*cryptGenerateKey*	*cryptCreateSignature*	*cryptDestroyContext*
cryptDeviceQueryCapability	*cryptCheckCert*	*cryptImportKey*	*cryptQueryObject*
cryptAddPublicKey	*cryptGenerateKeyAsync*	*cryptCreateSignatureEx*	*cryptDestroyEnvelope*
cryptEncrypt	*cryptCheckSignature*	*cryptInit*	*cryptSetAttribute*
cryptAddRandom	*cryptGetAttribute*	*cryptDecrypt*	*cryptDestroyObject*
cryptEnd	*cryptCheckSignatureEx*	*cryptKeysetClose*	*cryptSetAttributeString*
cryptAsyncCancel	*cryptGetAttributeString*	*cryptDeleteAttribute*	*cryptDestroySession*
cryptExportCert	*cryptCreateCert*	*cryptKeysetOpen*	*cryptSignCert*
cryptAsyncQuery	*cryptGetCertExtension*	*cryptDeleteCertExtension*	*cryptDeviceClose*
cryptExportKey	*cryptCreateContext*	*cryptPopData*	*cryptUIDisplayCert*
cryptCAAddItem	*cryptGetPrivateKey*	*cryptDeleteKey*	*cryptDeviceCreateContext*
cryptExportKeyEx	*cryptCreateEnvelope*	*cryptPushData*	*cryptUIGenerateKey*
cryptCACertManagement	*cryptGetPublicKey*		

The available functions are utilized to provide or perform the cryptographic services. The various services offered by the module are described below. Operators of the module implicitly assume a role based on the services of the module that they are using. Since all services offered by the module can only be used by either the Crypto Officer or the User (never both), the roles are mutually exclusive. The Critical Security Parameters (CSP) used by each service are listed below, followed in parenthesis by the type of access each service provides to the listed CSP.

Descriptions of the services available to the Crypto Officer role are provided in Table 4 below. Please note that the keys and CSPs listed in the table indicate the type of access required using the following notation:

- Read: The CSP is read.
- Write: The CSP is established, generated, modified, or zeroized.
- Execute: The CSP is used within an Approved or Allowed security function or authentication mechanism.

.

Table 4 – Crypto Officer Services

Service	Description	Input	Output	CSP and Type of Access
Configuring in FIPS mode	The Crypto Officer has to follow the steps outlined in the Secure Operation section to enable FIPS mode.	Command	Result of FIPS mode set	None

2.4.2 User Role

The User role has the ability to perform cryptographic operations using the module. Descriptions of the services available to the User role are provided in Table 5 below.

Table 5 – User Services

Service	Description	Input	Output	CSP and Type of Access
Initialize and un-initialize cryptographic operations	All users can change the state of the module by initializing and un-initializing the module.	Command	Initialized Module	Integrity Check Key (read)
Perform encryption and decryption	Every user has the ability to encrypt and decrypt data.	Command, optional Key, Input Data	Status Output, Output Data	Symmetric Keys and Public/Private Keypairs (execute)
Perform hashing and HMAC	Every user has the ability to hash and use HMAC functions.	Command, optional Key, Input Data	Status Output, Output Data	HMAC Key (execute)
Create and manage certificates	Every user can both create and manage certificates. The Crypto Officer can manage any certificate, while the Users can only manage their own.	Command	Status Output, Certificate	Certificate (execute)
Generate random keys and numbers	All users can generate random keys and numbers. This random generation uses a FIPS-Approved Pseudo Random Number Generator (RNG).	Command, seed, key	Status output, random number	Seed, entropy source (read)
Create and manage secure sessions	All users can both create and manage secure sessions such as TLS and SSH.	Command	Status Output, Secure Session Information	Symmetric and Public Keys (execute)
Create and manage secure e-mail protocols	Every user can both create and manage secure e-mail protocols such as PGP and S/MIME.	Command	Status Output, Output Data	Symmetric and Public Keys (execute)

.

2.5 Physical Security

The cryptographic module is a software module and does not include physical security mechanisms.

2.6 Operational Environment

The XYGATE /ESDK module was evaluated and tested on a standard PC running each of the platforms listed in Section 2.2 of this document. All of the platforms are configured for single operator mode as per NIST guidance. As such, all keys, intermediate values, and other CSPs remain only in the process space of the operator using the module. The operating systems ensure that outside processes cannot access the process space and memory used by the module.

2.7 Cryptographic Key Management

The module implements the FIPS-Approved algorithms listed in Table 6 below.

Table 6 – FIPS-Approved Algorithm Implementations

Algorithm	Certificate Number
AES – ECB, CBC, CFB128, OFB mode (128-, 192-, 256-bit key sizes)	1571
Triple-DES – ECB, CBC, CFB-64, OFB mode (112-, 168-bit key sizes)	1028
RSA signature generation/verification (1024-, 2048-, 4096-bit modulus)	764
DSA signature generation/verification (1024-bit modulus)	482
SHA-1, SHA-256	1391
HMAC using SHA-1, SHA-256	918
ANSI X9.31 Appendix A.4.2 Random Number Generator	845

The module utilizes the following FIPS-Allowed algorithm implementations when running in a FIPS-Approved mode of operation:

- Diffie-Hellman – for key agreement
- RSA – for key transport

Additionally, the module utilizes the following non-FIPS-Approved algorithm implementations:

- Hardware RNG – for seeding the FIPS-approved deterministic RNG
- Symmetric Key Algorithms – DES, Skipjack (non-compliant), Blowfish, CAST-128, RC2, RC4, RC5, IDEA
- Hashing Algorithms – MD2, MD4, MD5, RIPE-MD
- MAC Algorithms – HMAC-MD5, HMAC-RIPE-MD

.

The module supports the critical security parameters (CSPs) listed below in Table 7.

Table 7 – List of Cryptographic Keys, Cryptographic Key Components, and CSPs

Key	Key Type	Generation / Input	Output	Storage	Zeroization	Use
AES key	128, 196, 256-bit encryption key	Generated internally or input by the calling application in ciphertext	Never exits the module	Stored as plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	Using the cryptDestroyContext() API	Encryption and decryption of messages
Triple-DES key	112, 168-bit encryption key	Generated internally or input by the calling application in ciphertext	Never exits the module	Stored as plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	When session is over	Encryption and decryption of messages
DH public key	Public key	Generated internally or input by the calling application in ciphertext	Exits the module in plaintext	Stored as plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	At the users discretion	Key agreement
DH private key	Private key	Generated internally or input by the calling application in ciphertext	Never exits the module	Stored as plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	When session is over	Key agreement
RSA public key	Public key	Generated internally or input by the calling application in ciphertext	Exits the module in plaintext	Stored as plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	At the users discretion	Signature generation and verification, and key exchange
RSA private key	Private key	Generated internally or input by the calling application in ciphertext	Never exits the module	Stored as plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	When session is over	Signature generation and verification, and key exchange
DSA public key	Public key	Generated internally or input by the calling application in ciphertext	Exits the module in plaintext	Stored in plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	At the users discretion	Signature generation and verification

.

Key	Key Type	Generation / Input	Output	Storage	Zeroization	Use
DSA private key	Private key	Generated internally or input by the calling application in ciphertext	Never exits the module	Stored as plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	When session is over	Signature generation and verification
HMAC-SHA key	128-bit HMAC key	Generated internally or input by the calling application in ciphertext	Never exits the module	Stored in plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	When the session is over	Hashing messages
RNG seed	8-byte seed value	Generated by non-approved RNG	Never exits the module	Plaintext in volatile memory	At power-cycle	Seeding an Approved RNG
RNG seed key	16-byte seed key value	Generated by non-approved RNG	Never exits module	Plaintext in volatile memory	At power-cycle	Seeding an approved RNG
Skipjack key (non-compliant)	80-bit encryption key	Generated internally or input by the calling application in ciphertext	Never exits the module	Stored in plaintext within cryptographic contexts, which are resident in page-locked memory (if available)	When session is over	Encryption and decryption of messages

2.8 EMI/EMC

The module is a software module and depends on the GPC for its physical characteristics. However, the GPC must have been tested for, and meet, applicable Federal Communications Commission (FCC) EMI and EMC requirements for business use as defined in Subpart B, Class A of FCC 47 Code of Federal Regulations Part 15. All systems sold in the United States must meet the applicable FCC requirements.

2.9 Self-Tests

2.9.1 Power-Up Self-Tests

The XYGATE /ESDK performs the following self-tests at power-up:

- Software integrity check: Verifying the integrity of the module using a Message Authentication Code produced from a 128-bit keyed hash of the module (HMAC-SHA-1)
- Approved algorithm tests:
    - AES Known Answer Test (KAT): Verifies the correct operation of the AES algorithm implementation
    - Triple-DES KAT: Verifies the correct operation of the Triple-DES algorithm implementation
    - Skipjack KAT: Verifies the correct operation of the Skipjack algorithm implementation
    - RSA KAT: Verifies the correct operation of the RSA algorithm implementation
    - DSA KAT: Verifies the correct operation of the DSA algorithm implementation
    - SHA KAT: Verifies the correct operation of the SHA-1 and SHA-256 algorithm implementations
    - HMAC SHA-1 KAT: Verifies the correct operation of the HMAC SHA-1 algorithm implementation
    - ANSI X9.31 RNG KAT: Verifies the correct operation of the RNG implementation
- Non-Approved algorithm tests:
    - KATs for Blowfish, CAST-128, DES, IDEA, RC2, RC4, RC5, MD2, MD4, MD5, RIPE-MD, HMAC-MD5, and HMAC-RIPE-MD

Because the module is a software library, the power-up self-tests are run when a host application performs the call to the cryptInit() function. Linking the module invokes certain API function calls, which triggers the self tests. If these self tests fail, then the API calls fail and the API will not be functional.

2.9.2 Conditional Self-Tests

The XYGATE /ESDK performs the following conditional self-tests:

- Continuous RNG Test: Verifies that the Approved and non-Approved RNGs do not repeatedly generate a constant value
- RSA Pair-wise Consistency Test: Verifies the correct operation of the RSA algorithm implementation
- DSA Pair-wise Consistency Test: Verifies the correct operation of the DSA algorithm implementation
- DH Pair-wise Consistency Test: Verifies the correct operation of the DH algorithm implementation

If any error occurs during the tests, the cryptographic context that requested the keys or random numbers will be deactivated until a proper key or random number is generated again and passes the self test, or until the context is destroyed and a new one created and another request made.

.

2.10 Design Assurance

XYPRO stores their source code within Microsoft Visual SourceSafe. Code must be checked out to edit and then checked in to become part of the final source tree for the ESDK. Corsec Security, Inc. also stores FIPS validation documents in Visual SourceSafe to ensure that documents are not tampered with and are only edited by those who have the proper permissions.

This Security Policy describes the secure operation of the XYGATE /ESDK module, specifies the procedures for secure installation, initialization, startup, and operation of the module, and provides guidance for use by Crypto Officers and Users.

2.11 Mitigation of Other Attacks

This section is not applicable. The modules do not claim to mitigate any attacks beyond the FIPS 140-2 Level 1 requirements for this validation.

3 Secure Operation

The XYGATE /ESDK meets Level 1 requirements for FIPS 140-2. The sections below describe how to place and keep the module in FIPS-approved mode of operation.

3.1 Initial Setup

The module has to be installed on one of the tested platforms, which are listed in Section 2.2 of the Security Policy. The platform must be setup for single-user operation mode. Installing the module is simply ensuring that the ESDK dynamic library is in the same directory as the accompanying application.

3.2 Secure Management

This section should provide guidance which ensures that the module is always operated I a secure/FIPS compliant configuration. It will generally include services and activities allotted to the Crypto-Officer. An example is provided below.

3.2.1 Initialization

The Crypto Officer is required to ensure that two functions are called before any other API calls are made. The first is a call to cryptolib_fips_set(1), which will place the module in the FIPS-Approved mode. Next, the Crypto Officer will make a call to cryptInit( ), which will perform the integrity test and the power-up self tests.

3.2.2 Management

The module can run in two different modes: FIPS-Approved and non-Approved. While in a FIPS-Approved mode, only FIPS-Approved and Allowed algorithms may be used.

3.2.3 Zeroization

All keys are zeroized when the associated cryptographic context is deleted. Contexts can be deleted either explicitly using the cryptDestroyContext function, or implicitly when the cryptEnd() function is called.

3.3 User Guidance

When using key establishment protocols (RSA and DH) in FIPS approved mode, the user is responsible for selecting a key size that provides the appropriate level of key strength for the key being transported.

.

4 Acronyms

This section defines the acronyms used throughout this document.

Table 8 – Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
DES	Digital Encryption Standard
DH	Diffie-Hellman
DSA	Digital Signature Algorithm
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
ESDK	Encryption Software Development Kit
FIPS	Federal Information Processing Standard
GPC	General Purpose Computer
HMAC	(Keyed-) Hash Message Authentication Code
HP	Hewlett-Packard
IBM	International Business Machines Corporation
KAT	Known Answer Test
MAC	Message Authentication Code
MD	Message Digest
NIST	National Institute of Standards and Technology
OS	Operating System
PGP	Pretty-Good-Privacy
RHEL	Red Hat Enterprise Linux
RNG	Random Number Generator
RSA	Rivest Shamir and Adleman
SHA	Secure Hash Algorithm
S/MIME	Secure Multipurpose Internet Mail Extensions
SP	Service Pack
SSH	Secure Shell
SSL	Secure Socket Layer

.

Acronym	Definition
TLS	Transport Layer Security
VSS	Visual SourceSafe

Prepared by:
Corsec Security, Inc.

10340 Democracy Lane, Suite 201
Fairfax, VA 22030
USA

Phone: +1 (703) 267-6050
Email: info@corsec.com
http://www.corsec.com