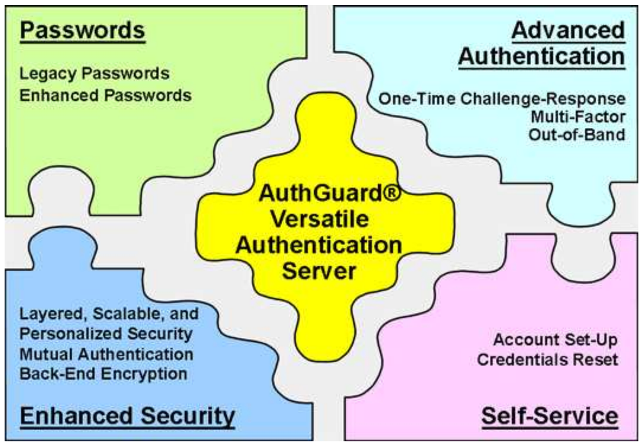
Authernative, Inc.

Authernative® Cryptographic Module

Software Version: 1.0.0



FIPS 140-2 Security Policy

Level 1 Validation

Document Version 1.1

Prepared for:

authernative®
THE AUTHENTICATION ALTERNATIVE

Authernative, Inc.

201 Redwood Shores Parkway, Suite 275
Redwood City, CA 94065
Phone: (650) 587-5263
Fax: (650) 587-5259
http://www.authernative.com

Prepared by:

Corsec

Corsec Security, Inc.

10340 Democracy Lane, Suite 201
Fairfax, VA 22030
Phone: (703) 267-6050
Fax: (703) 267-6810
http://www.corsec.com

Revision History

Version	Modification Date	Modified By	Description of Changes
0.1	2007-09-21	Xiaoyu Ruan	Initial draft
0.2	2008-01-10	Xiaoyu Ruan	Added ECBBlockCipher.class; removed DESEngine.class
0.3	2008-01-23	Xiaoyu Ruan	Added zeroize method; Put CAVP numbers
0.4	2008-01-25	Xiaoyu Ruan	Addressed Lab comments
0.5	2008-02-05	Xiaoyu Ruan	Addressed Lab comments
1.0	2008-05-01	Xiaoyu Ruan	Address CMVP comments
1.1	2008-05-09	Xiaoyu Ruan	Address CMVP comments

Table of Contents

Table of Figures

Table of Tables

1 Introduction

1.1 Purpose

This document is a non-proprietary Cryptographic Module Security Policy for the Authernative® Cryptographic Module from Authernative, Inc. This Security Policy describes how the Authernative® Cryptographic Module meets the security requirements of FIPS 140-2 and how to run the module in a secure FIPS 140-2 mode of operation. This policy was prepared as part of the Level 1 FIPS 140-2 validation of the Authernative® Cryptographic Module.

FIPS 140-2 (Federal Information Processing Standards Publication 140-2 – *Security Requirements for Cryptographic Modules*) 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) Cryptographic Module Validation Program (CMVP) website at: http://csrc.nist.gov/groups/STM/index.html.

In this document, the Authernative® Cryptographic Module is referred to as "the module". The application represents Authernative's software products, such as AuthGuard, linked with the cryptographic methods provided by the Authernative® Cryptographic Module.

1.2 References

This document deals only with the 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 Authernative website (http://www.authernative.com/) contains information on the full line of products from Authernative.
- The CMVP website (http://csrc.nist.gov/groups/STM/index.html) contains contact information for answers to 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
- Finite State Machine
- Other supporting documentation as additional references

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

2 AuthGuard and PassEnabler

Authernative, Inc. is a software company that develops, markets, and sells enterprise and consumer level security solutions. Authernative's granted and pending U.S. and International patents in the area of private and secure financial transactions, authentication algorithms, protocols, and encryption schemes are the foundation for the company technology and commercial product offerings. Authernative provides integrated security solutions for identity management, strong authentication to access network resources, and efficient authorization, administration and auditing control.

Authernative approaches security as a complex system having scientific, technological, engineering, marketing, and social components. The company believes that only a harmonized mixture of these components implemented in security products and backed with excellent services can bring long-lasting success and customer satisfaction.

Authernative currently sells two separate and complementary products: AuthGuard® and PassEnabler®. Both AuthGuard and PassEnabler are applications that use the Authernative Cryptographic Module. However, AuthGuard and PassEnabler are not being validated for FIPS compliance because all their security-relevant functions are provided by the Authernative Cryptographic Module.

2.1 Overview

AuthGuard is an authentication product. It provides an authentication server that supports and manages multiple authentication options. Those options allow AuthGuard to offer multifactor authentication, strong authentication, or layered authentication services. PassEnabler allows administrators to define what resources authorized users have access to and provides a secure authorization, administration, auditing, and web single-sign-on engine. PassEnabler is integrated with AuthGuard. PassEnabler enables corporate identity and access management using the authentication capabilities of AuthGuard. AuthGuard and PassEnabler can be used either separately or together as complementary tools within a tool suite.

The AuthGuard product is implemented using five components (as depicted in Figure 1):

- AuthGuard Server
- Administrative Utility
- Configuration Utility
- Licensing
- AuthGuard Client

The central component is the AuthGuard Server, which provides authentication services in a networked environment. Users attempting to access various systems are redirected to the AuthGuard Server. This provides them with a Graphical User Interface (GUI) to perform authentication. The GUI is provided by downloading the AuthGuard Client to a browser. The AuthGuard Client GUI changes depending on what forms of authentication are being performed, and communicates with the AuthGuard® Server.

Authernative has developed two utilities to manage the AuthGuard product. The first utility is the Administrative Utility, which provides an administrative console for management of the AuthGuard Server. The Administrative Utility provides a GUI to tweak roughly fifty options and features of the configuration of the AuthGuard Server, setting the user permissions and authentication. An administrator uses the Administrative Utility to initially configure the system. The second utility is the Configuration Utility, which is a desktop configuration tool that gives the administrator the ability to perform user account provisioning, manage roles, create users, and perform auditing. The Configuration Utility also allows auditing to be performed on users and administrator activities on the network from data in the AuthGuard Server's logs. The product allows a user to view network resources and to define resources that are placed under AuthGuard's authentication control.

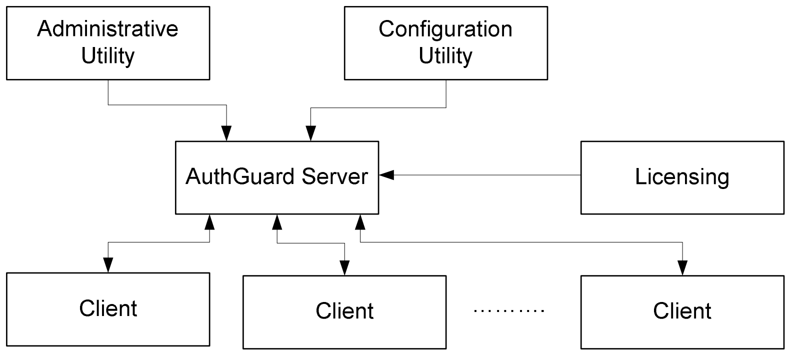


Figure 1 – Components of the AuthGuard Product

2.2 Client-Server Encryption and Authentication

Communications between the AuthGuard Server and the AuthGuard Client are encrypted using the Advanced Encryption Standard (AES) algorithm. The AuthGuard Server is implemented as a Java servlet within an Apache Tomcat container, and contains all required security functionality. The AuthGuard Client is distributed as a Java applet by the AuthGuard Server. The applet is loaded into a user's browser. The Client then provides the complete user GUI and performs the encryption operations enabling secure communications with the AuthGuard Server. Furthermore, the applet provides interfaces appropriate to the administrator-selected authentication methods and guides the user through authentication to the AuthGuard Server and access to resources.

Network users encounter the AuthGuard Server when they bring up a browser and request access to an authenticated resource. These requests are redirected by the resource to the AuthGuard Server if the request has not yet been authenticated. Optionally, users can point directly to an AuthGuard Server to begin authentication steps. Once contacted, the AuthGuard Server sends back the Client applet to the user along with a Session Random Key (SRK), which can be either an AES or a triple Data Encryption Standard (DES) key.

The SRKs are used to initialize secure sessions, and are created by the AuthGuard Server. When the servlet for the AuthGuard Server is initialized, it starts generating a new store of SRKs destined for future use. The SRKs are placed in an array that is constantly updated by the Server, and SRKs created by the Server are assigned a lifetime. After an SRK has expired, it will not be used to secure a new connection. Each SRK is associated with an array of Data Random Keys (DRKs), which is created for a particular session. The array of DRKs is erased if the SRK is erased. The Server can be configured to create a specific number of SRKs, and will then update them periodically. For an individual session, a single unused SRK is selected, and then sent to the client in the clear encoded as an array of bytes in a Java class. The SRK is then used by the Client to initiate the session between the Client and the Server. The Client first obtains a username from the GUI, and sends this to the server encrypted with the SRK. The Server receives this and decrypts the username.

After the exchange of a username and SRK, the Server selects a DRK from the array associated with the SRK, and sends it to the Client encrypted with the SRK. The encrypted bits are additionally byte-veiled, or bit-veiled as described in the next subsection. At this point, the Client retrieves the DRK, and displays a GUI to the user to collect password information. Meanwhile, the Client hashes the DRK, encrypts the hash with the DRK, and sends the result back to the Server to indicate that the DRK was successfully received and decrypted. The Server checks that this is correct by computing the same value.

At this point, the Server and Client have exchanged an SRK, DRK, and username but have not authenticated either side, or exchanged a key not subject to man-in-the-middle attacks. Now, the Server selects a second DRK (DRK2) from the DRK array. The server then retrieves the user's password information from its database. The Server then encrypts DRK2 with DRK and bit-veils, byte-veils, or both into a conversion array using values from a Random Number Generator (RNG) seeded with the user's password information. This is transmitted to the Client who can then use the same password information to reconstruct DRK2.

The Client then hashes DRK2, hides it in a conversion array using the password information, encrypts the conversion array with DRK2, and sends it back to the server to indicate he has DRK2. This step performs Client authentication based on possession of the user’s password information, and shares DRK2 with both sides. The same step is then performed by the Server to authenticate the Server to the Client using DRK2 and the Server password. The Server sends a hash of DRK2 in a conversion array using the Server password to seed the RNG for bit- or byte-veiling, and encrypting the array with DRK2. The Client already has the Server password and uses it to authenticate the Server. At this point client have performed mutual authentication, and share a session encryption key.

User password information can be a simple password, or can use Authernative’s passline (a chosen pattern in a grid), pass-step (an out-of-band challenge sent to email or phone to be entered), crossline (a challenge embedded in a grid), or passfield (image, colors, and a grid). Each of these processes allows the user to select secret password information, all or part of which can be provided in response to challenges.

The authentication step of exchanging a DRK using password information for the bit- and byte-veiling can be iterated as often as desired to provide a DRK3, DRK4, etc. Security can be layered to use multiple authentication steps, where different password information forms are employed. For example, a user could employ both a simple password and use passline. The password would be used for DRK2, and then passline would be used for DRK3, and that exchange would also depend upon DRK2. At this point, the DRK are not used by AuthGuard for secure data encryption, and are simply treated as a byproduct of the authentication. Other products may in the future use the DRKs for secure content exchange, but they are currently used only for authentication.

2.3 BitVU, ByteVU, and BBVU

Authernative has secured three patents on the processes described above, with claims in the patents that cover the use of a conversion array, key generation, and bit- and byte-veiling. The process of “Bit-Veil-Unveil (BitVU), Byte-Veil-Unveil (ByteVU), and Byte-Bit-Veil-Unveil (BBVU)” mentioned above are the subject of the patents, and are integral to the authentication process. The BitVU and ByteVU processes take an array of random data and effectively hide or intersperse message data within the array. The array of random data with the interspersed messages is referred to as a conversion array, and may be further encrypted before transmission within the AuthGuard schemes described. The locations of the message data within the conversion array are determined by a deterministic RNG seeded with a secret value. Two parties that share this secret value can both use the same RNG to compute the locations of the data within the conversion array. The process of ByteVU involves generating a conversion array, and “veiling” individual bytes of the message data by sparsely distributing them through the conversion array. The process of BitVU does the same, but on a bit-wise basis.

3 Authernative® Cryptographic Module

3.1 Overview

The module was developed and tested on Microsoft Windows XP (Service Package 2) with Sun Java Runtime Environment (JRE) 1.5. The module can run on any Java Virtual Machine (JVM) regardless of operating system (OS) and computer architecture. The minimum version of the JRE supported by the module is 1.5.

Logically the module is a single Java ARchival (JAR), *AuthCryptoApi.jar*. Table 1 shows the OS and name of the binary file.

Table 1 – Binary Form of the Module

When	Operating System	Binary File Name
Development	Windows XP with Sun JRE 1.5	AuthCryptoApi.jar
Runtime	Any JVM with JRE 1.5 or later regardless of OS and computer architecture	AuthCryptoApi.jar

The module is stored on the hard disk and is loaded in memory when a client application calls cryptographic services exported by the module. As of this writing, the client application is AuthGuard. However, Authernative may develop more applications making use of the module in the future.

When operating in the Approved mode of operation, the Authernative® Cryptographic Module is validated at FIPS 140-2 section levels shown in Table 1. Note that in Table 2, EMI and EMC mean Electromagnetic Interference and Electromagnetic Compatibility, respectively, and N/A indicates “Not Applicable”.

Table 2 – 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
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A

3.2 Module Interfaces

The module, *AuthCryptoApi.jar*, provides client applications with a set of cryptographic services in the form of Application Programming Interface (API) calls. Figure 2 shows the logical cryptographic boundary for the module. The module is a JAR file that consists of 42 java classes. Out of the 42 classes, 29 are Bouncy Castle classes that implement underlying cryptographic algorithms. Bouncy Castle is an open-source Java library available at http://www.bouncycastle.org/. The Bouncy Castle classes do not have public methods. The other 13 classes, developed by Authernative, implement public methods of the module. The JAR file manifest, *MANIFEST.MF*, contains the signature of the JAR (used in the power-up integrity test).

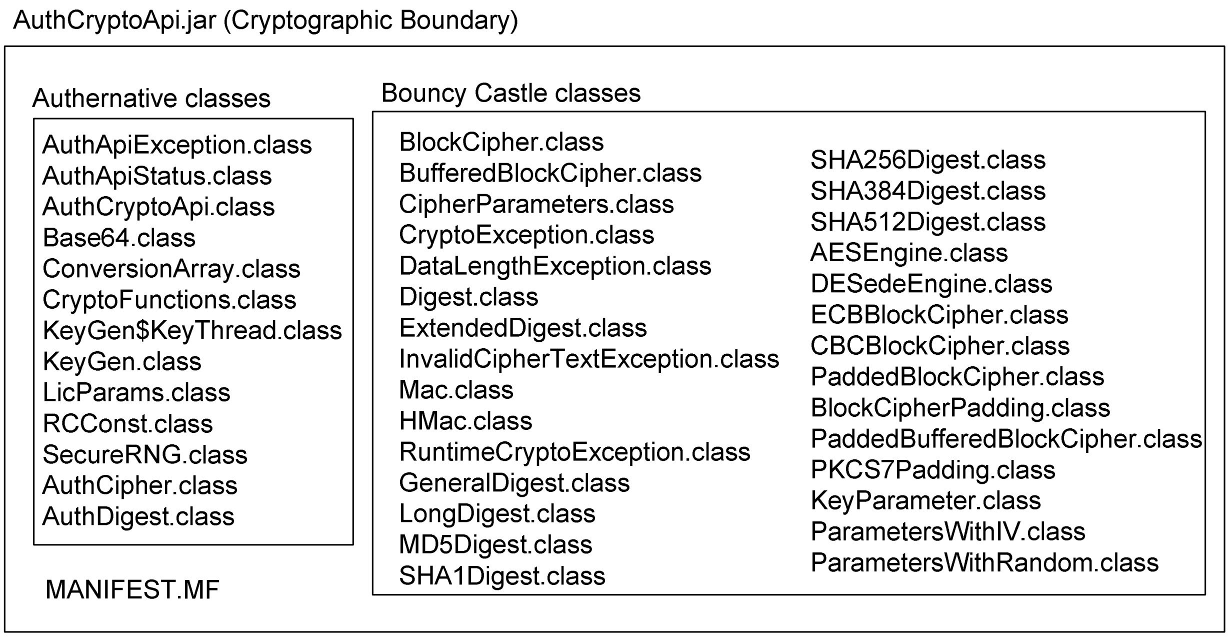


Figure 2 – Logical Cryptographic Boundary

The descriptions of the Authernative classes are described in Table 3 – Authernative Classes in *AuthCryptoApi.jar*. A complete list of exported methods is available in the module's API reference manual.

Table 3 – Authernative Classes in *AuthCryptoApi.jar*

Class	Description
AuthApiException.class	The class implements the exception thrown when and if there is an error state in the API.
AuthApiStatus.class	The class implements methods that report configurations and status of the API.
AuthCryptoApi.class	This is the core API class and contains all the public methods. This class simply collects the interfaces into a single object. Most of the functions of the module are implemented by the other classes.
Base64.class	The class implements the base64 encoding and decoding methods.
ConversionArray.class	The class implements Authernative's patented BitVU, ByteVU, and BBVU technology. See Section 2.3 of this document for a description of this technique.
CryptoFunctions.class	The class contains all the cryptographic functions realized by the module.
KeyGen$KeyThread.class	The class is a subclass of the *KeyGen* class. This class implements the mechanism of generating a new key every 60 seconds.
KeyGen.class	The class implements key generation methods.
LicParams.class	The class stores the licensing information of the module.
RCConst.class	The class contains all the return codes for the API errors for use with the *AuthApiException* class.

Class	Description
SecureRNG.class	The class implements the American National Standards Institute (ANSI) X9.31 Appendix A.2.4 RNG.
AuthCipher.class	This is an Authernative wrapper class to enhance usability for all of the Bouncy Castle cipher functionality.
AuthDigest.class	This is an Authernative wrapper class to enhance usability for all of the Bouncy Castle digest functionality.

The module's interactions with surrounding components, including Central Processing Unit (CPU), hard disk, memory, client application, and the OS are demonstrated in Figure 3.

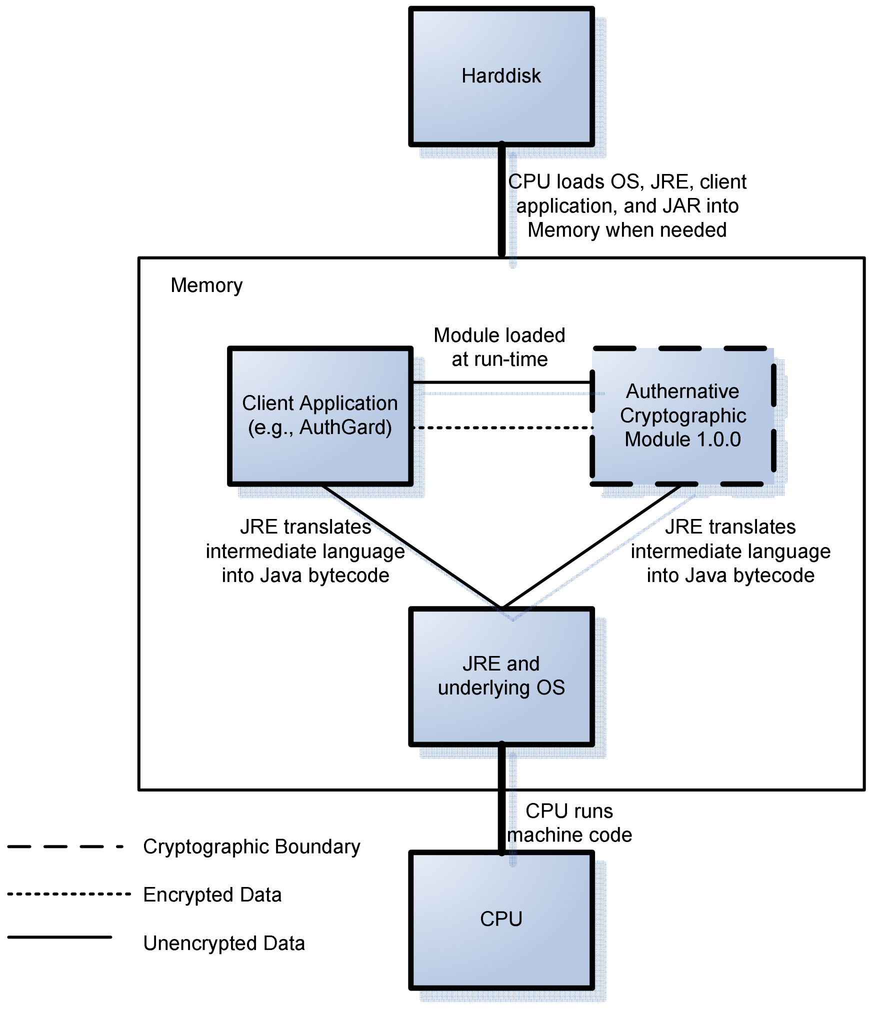

Harddisk
CPU loads OS, JRE, client application, and JAR into Memory when needed
Memory
Module loaded at run-time
Client Application (e.g., AuthGard)
Authernative Cryptographic Module 1.0.0
JRE translates intermediate language into Java bytecode
JRE translates intermediate language into Java bytecode
JRE and underlying OS
CPU runs machine code
Cryptographic Boundary
Encrypted Data
Unencrypted Data
CPU

Figure 3 – Logical Cryptographic Boundary and Interactions with Surrounding Components

The module is validated for use on the platforms listed in the second column of Table 1. In addition to the binaries, the physical device consists of the integrated circuits of the motherboard, the CPU, Random Access Memory (RAM), Read-Only Memory (ROM), computer case, keyboard, mouse, video interfaces, expansion cards, and other hardware components included in the computer such as hard disk, floppy disk, Compact Disc ROM (CD-ROM) drive, power supply, and fans. The physical cryptographic boundary of the module is the opaque hard metal and plastic enclosure of the server running the module. The block diagram for a standard general-purpose computer

(GPC) is shown in Figure 4. Note that in this figure, I/O means Input/Output, BIOS stands for Basic Input/Output System, PCI stands for Peripheral Component Interconnect, ISA stands for Instruction Set Architecture, and IDE represents Integrated Drive Electronics.

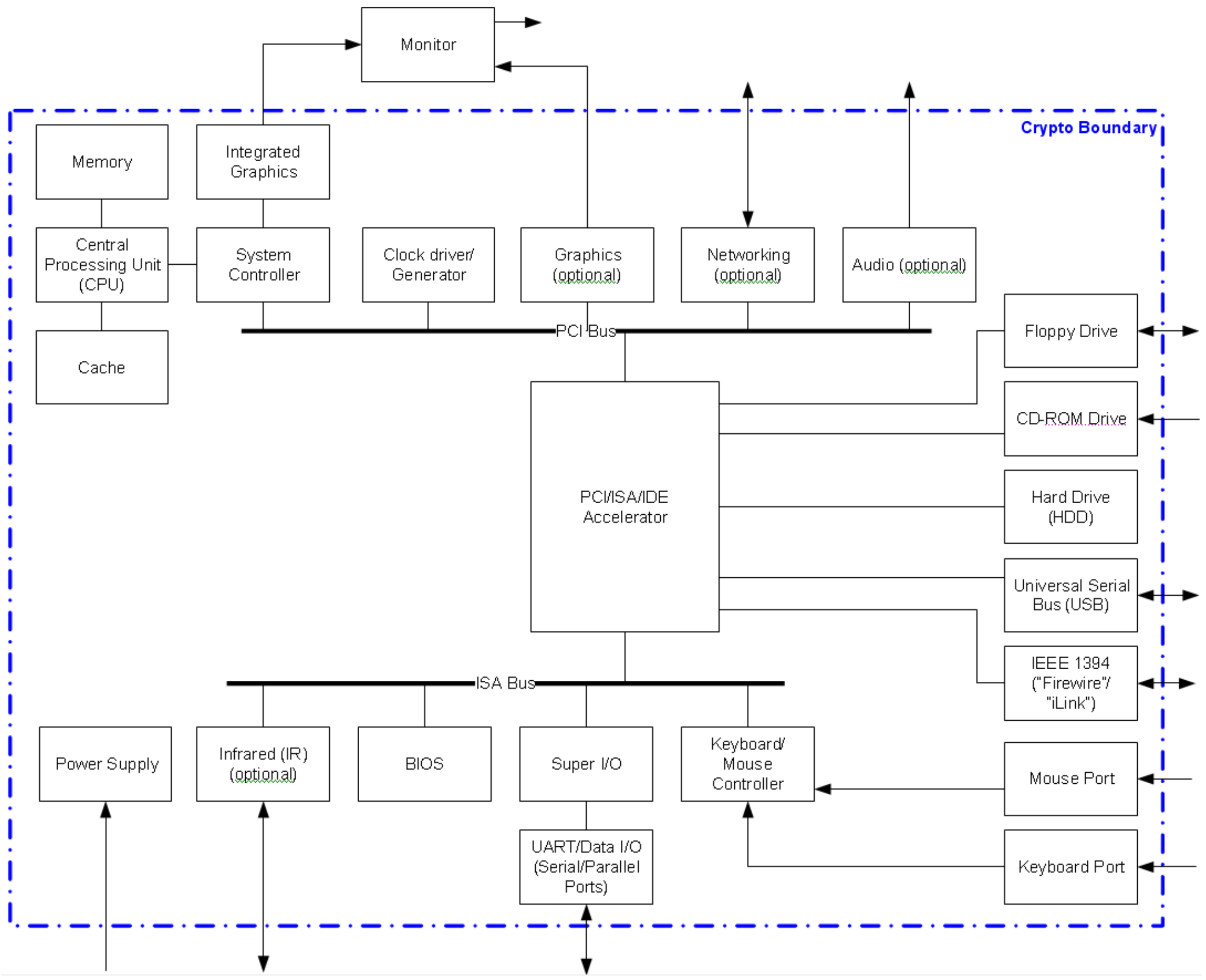


Figure 4 – Physical Block Diagram of a Standard GPC

All of these physical ports are separated into logical interfaces defined by FIPS 140-2, as described in Table 3.

Table 4 – Logical, Physical, and Module Interface Mapping

Logical Interface	Physical Port Mapping	Module Mapping
Data Input	Keyboard, mouse, CD-ROM, floppy disk, and serial/USB/parallel/network ports	Arguments for API calls that contain data to be used or processed by the module
Data Output	Hard Disk, floppy disk, monitor, and serial/USB/parallel/network ports	Arguments for API calls that contain module response data to be used or processed by the caller
Control Input	Keyboard, CD-ROM, floppy disk, mouse, and serial/USB/parallel/network port	API calls
Status Output	Hard disk, floppy disk, monitor, and serial/USB/parallel/network ports	Arguments for API calls, return value, error message

3.3 Roles and Services

The operators of the module can assume two roles as required by FIPS 140-2: a Crypto Officer role and a User role. The operator of the module assumes either of the roles based on the operations performed. The operator is not required to authenticate to the module before accessing services.

The module provides an API for client applications. Table 5 – Crypto Officer Services shows the public methods that are run by the Crypto Officer role. The method name is shown in the first column ("Service"). Its function is described in the second column ("Description"). Each method exported by the module is an individual Crypto Officer service. User services (see Table 6 – User Services) are also available to the Crypto Officer role.

Table 6 – User Services shows the public methods that are run by the User role. Similar to Table 5 – Crypto Officer Services, the method name is shown in the first column ("Service"). Its function is described in the second column ("Description"). Each method exported by the module is an individual User service. User services are also available to the Crypto Officer role.

The Critical Security Parameters (CSPs) mentioned in the rightmost columns correspond to the ones listed in Table 7 – List of Cryptographic Keys, Cryptographic Key Components, and CSPs.

Table 5 – Crypto Officer Services

Service	Description	Input	Output	CSP and Type of Access
Installation	To install the module	Command	Status	None
Uninstallation	To uninstall the module	Command	Status	All CSPs – overwrite
AuthCryptoApi	The API's only constructor. The instance of the API will be defined by the parameters that are passed in	Crypto type, hash type, crypto mode, key size, padding scheme	Status	None
getInstance	This method is provided for singleton use of the API	Crypto type, hash type, crypto mode, key size, padding scheme	Status, the instance of *AutghCryptoApi*	None
printByteArray	Prints out a byte array in hexadecimal notation	Text string, byte array	Status, the printout	None
printByteArray	Prints out a byte array in hexadecimal notation	Byte array	Status, the printout	None
hexStrToByteArray	Converts a hexadecimal string into a byte array	Hexadecimal string	Status, byte array	None
checkLicense	Checks the license	License string from application, client information	Status	None
getStatus	Gets information and configuration about the API	None	Status, API object information and configuration	None
setSeed	Sets the seed, date/time (DT) value, and Triple DES key to random numbers (generated by the non-Approved RNG) for the ANSI X9.31 RNG	None	Status	ANSI X9.31 RNG seed for key generation methods – write, overwrite ANSI X9.31 RNG DT value for key generation methods – write, overwrite ANSI X9.31 RNG Triple DES key for key generation methods – write, overwrite
setSeed	Sets the Triple DES key to specified values for the ANSI X9.31 RNG	Triple DES key	Status	ANSI X9.31 RNG Triple DES key for key generation methods – write, overwrite

Service	Description	Input	Output	CSP and Type of Access
setSeed	Sets the seed, DT value, and Triple DES key to specified values for the ANSI X9.31 RNG	Seed, Triple DES key, DT value	Status	ANSI X9.31 RNG seed for key generation methods – write, overwrite ANSI X9.31 RNG DT value for key generation methods – write, overwrite ANSI X9.31 RNG Triple DES key for key generation methods – write, overwrite
nextInt	Generates a random number	None	Status, random number	ANSI X9.31 RNG seed for key generation methods – read ANSI X9.31 RNG DT value for key generation methods – read ANSI X9.31 RNG Triple DES key for key generation methods – read
nextInt	Generates a random number between zero and the specified integer	An integer (range of the random number)	Status, random number	ANSI X9.31 RNG seed for key generation methods – read ANSI X9.31 RNG DT value for key generation methods – read ANSI X9.31 RNG Triple DES key for key generation methods – read
nextBytes	Generates a random number array	Pointer to a byte array	Status, random number array	ANSI X9.31 RNG seed for key generation methods – read ANSI X9.31 RNG DT value for key generation methods – read ANSI X9.31 RNG Triple DES key for key generation methods – read
zeroize	Zeroizes CSPs	None	Status	All CSPs in HashMap and filesystem – overwrite

Table 6 – User Services

Service	Description	Input	Output	CSP and Type of Access
setNumberOfKeys	Sets the maximum number of keys that the key generator will create before restarting at zero	Number of keys	Status	None
setPersistence	Sets the way the keys will be saved for the key generator	Mode (save in keys in file system or memory)	Status	None
setPath	Sets the location that the keys will be saved to the file system	Path of the file system	Status	None

Service	Description	Input	Output	CSP and Type of Access
getSecretKey	Creates and returns a Java secret key (*javax.crypto.SecretKey*)	None	Status, a secret key (*javax.crypto.SecretKey*)	AES key or Triple DES key for caller use – write, read
getRawKey	Creates and returns a Java secret key (byte array)	None	Status, a secret key (byte array)	AES key or Triple DES key for caller use – write, read
startKeyGen	Starts a thread that will perform key generation and save the keys. Keys will be generated every 60 seconds	None	Status	Triple DES key for veiling and unveiling methods – write
stopKeyGen	Stops the key generation	None	Status	Triple DES key for veiling and unveiling methods – overwrite
getSecretKeyFromRepos	Gets a key (*javax.crypto.SecretKey*) from the repository that is created by the *startKeyGen* method call	Index to the repository	Status, a secret key (*javax.crypto.SecretKey*)	Triple DES key for veiling and unveiling methods – read
getRawKeyFromRepos	Gets a key (byte array) from the repository that is created by the *startKeyGen* method call	Index to the repository	Status, a secret key (byte array)	Triple DES key for veiling and unveiling methods – read
setSecretKey	Sets the secret key (byte array) to be used in crypto operations	Secret key	Status	AES key or Triple DES key for encryption and decryption methods – write, overwrite
setSecretKey	Sets the secret key (*javax.crypto.SecretKey*) to be used in crypto operations	Secret key	Status	AES key or Triple DES key for encryption and decryption methods – write, overwrite
setIV	Sets the initialization vector if crypto uses CBC mode	Initialization vector	Status	None
updateHash	Updates the current message for hashing	Byte array added to the message	Status	None
hashValue	Performs the final hashing for message	Byte array added to the message before the final hashing is done	Status, hash value	None
updateEncrypted	Updates the current plaintext for encryption	Byte array added to the plaintext to be encrypted	Status	None

Service	Description	Input	Output	CSP and Type of Access
encryptValue	Performs the final encryption for the plaintext	Byte array added to the plaintext before the final encryption is done	Status, ciphertext	AES key or Triple DES key for encryption and decryption methods – read
decryptValue	Decrypts ciphertext	Plaintext	Status, plaintext	AES key or Triple DES key for encryption and decryption methods – read
encryptValue	Encrypts plaintext with specified secret key (*javax.crypto.SecretKey*)	Plaintext, secret key (*javax.crypto.SecretKey*)	Status, ciphertext	AES key or Triple DES key for encryption and decryption methods – read
decryptValue	Decrypts ciphertext with specified secret key (*javax.crypto.SecretKey*)	Ciphertext, secret key (*javax.crypto.SecretKey*)	Status, plaintext	AES key or Triple DES key for encryption and decryption methods – read
encryptValue	Encrypts plaintext with specified secret key (byte array)	Plaintext, secret key (byte array)	Status, ciphertext	AES key or Triple DES key for encryption and decryption methods – read
decryptValue	Decrypts ciphertext with specified secret key (byte array)	Ciphertext, secret key (byte array)	Status, plaintext	AES key or Triple DES key for encryption and decryption methods – read
encode	Performs Base64 encoding on bytes	Bytes to be encoded	Encoded bytes	None
encode	Performs Base64 encoding on strings	Strings to be encoded	Encoded string	None
decode	Performs Base64 decoding on bytes	Bytes to be decoded	Decoded bytes	None
decode	Performs Base64 decoding on strings	Strings to be decoded	Decoded string	None
veilData	Hides bits, bytes, or bits and bytes in a larger array	Mode (bit, byte, or bit and byte), byte array to be hidden, Triple DES key for the ANSI X9.31 RNG	Conversion array with hidden byte array	Triple DES key for veiling and unveiling methods – write, read
unveilData	Extracts the data from conversion array	Mode (bit, byte, or bit and byte), conversion array, Triple DES key for the ANSI X9.31 RNG	Original byte array	Triple DES key for veiling and unveiling methods – write, read

3.4 Physical Security

The Authernative® Cryptographic Module is a multi-chip standalone module. The physical security requirements do not apply to this module, since it is purely a software module and does not implement any physical security mechanisms.

3.5 Operational Environment

The module was tested and validated on general-purpose Microsoft Windows XP with Service Package 2 with Sun JRE 1.5. The module can run on any JVM regardless of OS and computer architecture. The minimum version of the JRE supported by the module is 1.5. The module must be configured in single user mode as per the instructions provided in Section 4.1 of this document. Recommended configuration changes for the supported OS can also be found in Section 4.1.

3.6 Cryptographic Key Management

The module implements the following FIPS-approved algorithms in the Approved mode of operation.

- SHA-1, SHA-256, SHA-384, SHA-512 (certificate #725). SHA means Secure Hash Algorithm.
- HMAC-SHA-1 (certificate #375). HMAC means Keyed-Hash Message Authentication Code.
- Triple DES: 112 and 168 bits, in ECB and CBC modes (certificate #629). ECB and CBC mean Electronic Codebook and Cipher Block Chaining, respectively.
- AES: 128, 192, and 256 bits, in ECB and CBC modes (certificate #697)
- ANSI X9.31 Appendix A.2.4 RNG with 2-key Triple DES (certificate #408)

In the Approved mode of operation, the module uses a non-Approved RNG to seed the ANSI X9.31 RNG. This non-Approved RNG is the *SecureRandom* class provided by the JRE and is not implemented by the module itself. The non-Approved RNG is outside the cryptographic boundary of the module and is used by the module only for seeding the ANSI X9.31 RNG. In the non-Approved mode of operation, the module supports MD5.

The module supports the following CSPs in the Approved mode of operation:

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

Key	Key Type	Generation / Input	Output	Storage	Zeroization	Use
Triple DES key for caller use	Triple DES symmetric keys	Generated by ANSI X9.31 RNG	In plaintext	1. Plaintext in volatile memory; 2. Plaintext in filesystem	Zeroized when the *zeroize* method is called	Use is at the discretion of the caller
AES key for caller use	AES symmetric key	Generated by ANSI X9.31 RNG	In plaintext	1. Plaintext in volatile memory; 2. Plaintext in filesystem	Zeroized when the *zeroize* method is called	Use is at the discretion of the caller
Triple DES key for encryption and decryption methods	Triple DES symmetric keys	Input by caller in plaintext	Never	Plaintext in volatile memory	Zeroized after encryption or decryption is done	Encrypt plaintext or decrypt ciphertext
AES key for encryption and decryption methods	AES symmetric key	Input by caller in plaintext	Never	Plaintext in volatile memory	Zeroized after encryption or decryption is done	Encrypt plaintext or decrypt ciphertext

Key	Key Type	Generation / Input	Output	Storage	Zeroization	Use
Triple DES key for veiling and unveiling methods	Triple DES symmetric keys	Input by caller in plaintext	Never	Plaintext in volatile memory	Zeroized after veiling or unveiling is done	Veil or unveil data
ANSI X9.31 RNG DT value for key generation methods	Date/time variable	1. Generated internally by retrieving system date/time value 2. Input by caller in plaintext	Never	Plaintext in volatile memory	Zeroized when new DT value is generated	Generate keys
ANSI X9.31 RNG Triple DES key for key generation methods	Triple DES symmetric keys	1. Generated using the non-Approved RNG 2. Input by caller in plaintext	Never	Plaintext in volatile memory	Zeroized when new Triple DES key is generated	Generate keys
ANSI X9.31 RNG seed for key generation methods	Seed	1. Generated using the non-Approved RNG 2. Input by caller in plaintext	Never	Plaintext in volatile memory	Zeroized when new seed is generated	Generate keys
Software integrity test key	512-bit HMAC-SHA-1 key	Hardcoded	Never	Plaintext in nonvolatile memory	Zeroized when the module is uninstalled	Used in software integrity test

3.6.1 Key Generation

The module uses an ANSI X9.31 RNG with 2-key Triple DES to generate cryptographic keys. This RNG is a FIPS-Approved RNG as specified in Annex C to FIPS 140-2.

3.6.2 Key Input/Output

Symmetric keys are input to and output from the module in plaintext. The module does not use asymmetric-key cryptography.

3.6.3 Key Storage and Protection

Keys and other CSPs are stored in volatile memory or file system in plaintext. All key data resides in internally allocated data structures and can only be output using the module's defined API. The OS and JRE protect memory and process space from unauthorized access.

3.6.4 Key Zeroization

Generally speaking, CSPs resides in internal data structures that are cleaned up by JVM's garbage collector. Java handles memory in unpredictable ways that are transparent to the user. The Crypto Officer may manually invoke the zeroization of keys stored in HashMap and filesystem by calling the *zeroize* method.

3.7 EMI/EMC

Although the module consists entirely of software, the FIPS 140-2 platform is a server that has been tested for and meets applicable Federal Communications Commission (FCC) EMI and EMC requirements for business use as defined in Subpart B of FCC Part 15.

3.8 Self-Tests

The power-up self-tests are triggered by instantiation of an object of the *AuthCryptoApi* class. The Authernative® Cryptographic Module performs the following power-up self-tests:

- Software integrity test using HMAC-SHA-1
- Known Answer Test (KAT) on 2-key Triple DES in ECB mode
- KAT on 128-bit AES in ECB mode
- KATs on SHA-1, SHA-256, SHA-384, and SHA-512
- KAT on ANSI X9.31 RNG

The module implements the following conditional self-tests.

- Continuous test for the ANSI X9.31 RNG
- Continuous test for the non-Approved RNG

If the self-tests fail, an exception will be thrown on the failure. The application is then alerted that the self-tests failed, and the module will not load and will enter an error state. When in the error state, execution of the module is halted and data output from the module is inhibited.

3.9 Mitigation of Other Attacks

This section is not applicable. No claim is made that the module mitigates against any attacks beyond the FIPS 140-2 level 1 requirements for this validation.

4 Secure Operation

The Authernative® Cryptographic Module meets Level 1 requirements for FIPS 140-2. The subsections below describe how to place and keep the module in the Approved mode of operation.

4.1 Operating System Configuration

The user of the module is a software application. FIPS 140-2 mandates that a cryptographic module be limited to a single user at a time. A single instantiation of the Authernative® Cryptographic Module shall only be accessed by one client application, which is the User of this instantiation of the Authernative® Cryptographic Module.

For enhanced security, it is recommended that the Crypto Officer configure the OS to disallow remote login.

To configure Windows XP to disallow remote login, the Crypto Officer should ensure that all remote guest accounts are disabled in order to ensure that only one human operator can log into Windows XP at a time. The services that need to be turned off for Windows XP are

- Fast-user switching (irrelevant if server is a domain member)
- Terminal services
- Remote registry service
- Secondary logon service
- Telnet service
- Remote desktop and remote assistance service

Once Windows XP has been configured to disable remote login, the Crypto Officer can use the system "Administrator" account to install software, uninstall software, and administer the module.

A CMVP public document, *Frequently Asked Questions for the Cryptographic Module Validation Program*[1], gives instructions in Section 5.3 for configuring various Unix-based operating systems for single user mode.

4.2 Approved Mode Configuration

The Authernative® Cryptographic Module itself is not an end-user product. It is provided to the end-users as part of the application (e.g., AuthGuard). The module is installed during installation of the application. The installation procedure is described in the installation manual for the application.

In order to access functions of the module, the application has to execute the constructor of class *AuthCryptoApi* by instantiating an object of class *AuthCryptoApi*. The constructor of class *AuthCryptoApi* is:

public AuthSecurityApi(int crpytoType, int hashType, int codeBook, int keySize, int padding)

If the value passed in to the argument *int hashType* is SHA (integer value 1, 2, 3, or 4), then the module is operating in the Approved mode of operation. If the value passed in to the argument *int hashType* is MD5 (integer value 0), then the module is operating in the non-Approved mode of operation.

The constructor of class *AuthCryptoApi* performs all required power-up self-tests. If all power-up self-tests are passed, then an internal flag will be set to true. All other public methods of the module check this internal flag and ensure it is true before performing any other functions.

[1] Available at http://csrc.nist.gov/groups/STM/cmvp/documents/CMVPFAQ.pdf.

Notice that the Approved mode configuration described above is transparent to an operator. The configuration is performed by the client application.

4.3 CSP Zeroization

The Crypto Officer should zeroize CSPs when they are no longer needed. See Section 3.6.4 of this document for details on CSP zeroization.

4.4 Status Monitoring

The module's cryptographic functionality and security services are provided via the application. The module is not meant to be used without an associated application. End-user instructions and guidance are provided in the user manual and technical support documents of the application software. Although end-users do not have privileges to modify configurations of the module, they should make sure that the Approved mode of operation is enforced in the application, thereby ensuring that the proper cryptographic protection is provided.

5 Acronyms

Table 8 – Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
BBVU	Byte-Bit-Veil-Unveil
BIOS	Basic Input/Output System
BitVU	Bit-Veil-Unveil
ByteVU	Byte-Veil-Unveil
CBC	Cipher Block Chaining
CD-ROM	Compact Disc Read-Only Memory
CMVP	Cryptographic Module Validation Program
CPU	Central Processing Unit
CSP	Critical Security Parameter
DES	Data Encryption Standard
DRK	Data Random Key
DT	Date/Time
ECB	Electronic Codebook
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standard
GPC	General-Purpose Computer
GUI	Graphical User Interface
HDD	Hard Drive
HMAC	Keyed-Hash Message Authentication Code
IDE	Integrated Drive Electronics
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IR	Infrared
ISA	Instruction Set Architecture
JAR	Java ARchival
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KAT	Known Answer Test

Acronym	Definition
MAC	Message Authentication Code
N/A	Not Applicable
OS	Operating System
PCI	Peripheral Component Interconnect
RAM	Random Access Memory
RNG	Random Number Generator
ROM	Read Only Memory
SHA	Secure Hash Algorithm
SRK	Session Random Key
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus