Symbol Technologies

WTLS Cryptographic Module

Version 1.2, 1.3.2, and 1.3.3

by Columbitech

FIPS 140-2 Non-Proprietary Security Policy

Level 1 Validation
9/5/2006

1 Table of contents

2 Introduction

2.1 Purpose of the document

This document is a Security Policy Document describing the *WTLS Cryptographic Module* used in AirBEAM® Safe. It is written in order to show how the *crypto module* fulfils the requirements for certification according to Federal Information Processing Standards Publication 140-2.

2.2 Document conventions

Chapter 10 gives definitions of concepts used in this document. When the defined concepts occur in the text they are italicized.

2.3 References

[1] *FIPS PUB 140-2 Security Requirements for cryptographic modules,* May 25 2001
[2] *Derived Test Requirements for FIPS PUB 140-2, Security Requirements for Cryptographic Modules*, November 15, 2001, Draft
[3] *Wireless Transport Layer Security*, http://www.wapforum.org, version 06-Apr-2001
[4] *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)*
[5] *FIPS 197 Advanced Encryption Standard (AES)*, November 26, 2001
[6] *ANSI X9.52 – 1998 Triple Data Encryption Algorithms Modes of Operation*
[7] *FIPS 180-1 Secure Hash Standard*
[8] *IETF RFC 2104 HMAC: Keyed-Hashing for Message Authentication.* H. Krawczyk, M. Bellare, R. Canetti, February 1997, http://www.ietf.org/rfc
[9] *PKCS #1 RSA Cryptography Standard*, version 2.0, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html

2.4 Product description

Today, Federal governments, including Federal contractors, are required to use FIPS Certified products in all network security implementations. As FIPS is a recognized standard, also commercial buyers will often demand FIPS certification.

The *WTLS Crypto Module* is a software library that is that is used for establishing and maintaining WTLS sessions. The *WTLS Crypto Module* is encapsulated (embedded) in AirBEAM® Safe software components.

AirBEAM® Safe consists of software that provides convenient and secure access to the corporate LAN using different, wired or wireless, bearer networks. It consists of three components: AirBEAM® Safe Client, Gatekeeper and Enterprise Server.

Each AirBEAM® Safe component (Client, Gatekeeper and Enterprise server) uses an own standalone equal copy of the software *WTLS Cryptographic Module*. The WTLS Crypto module performs cryptographic operations for the Client, Gatekeeper,

Enterprise server software components that are outside of the defined software crypto boundary. From a system wide perspective, (outside the scope of FIPS 140-2) the only difference is that the WTLS Crypto Module will perform the services required of a client vs. gatekeeper vs. server based on the requests that it receives from a given AirBEAM® Safe component. AirBEAM® Safe components make specific calls (ie: services required of a client vs. a server) to the WTLS crypto module; it is the responsibility of the WTLS crypto module to respond to those requests and to perform the requested crypto operations.

When discussing the WTLS Crypto Module it is very important to note the following:

- **AirBEAM® Safe components themselves are not crypto modules.**
- The cryptographic functions offered by the WTLS Crypto Module are not callable by components other than the AirBEAM® Safe.
- The interfaces to the WTLS Crypto Module (data input, data output, control input, status output) are exactly the same regardless of the AirBEAM® Safe component the WTLS Crypto Module is contained within.

Supported clients: PPC 2002 and Windows 2000/XP
Supported servers: Windows 2000

Pocket PC 2002, or PPC 2002, must use a subset of Windows CE 3.0 as the operating system.

Additionally the cryptographic module has been ported to the following operating systems as per Implementation Guidance G.5:

- Windows CE 4.2
- Windows CE 5.0

The architecture of AirBEAM® Safe is illustrated below in Figure 1.

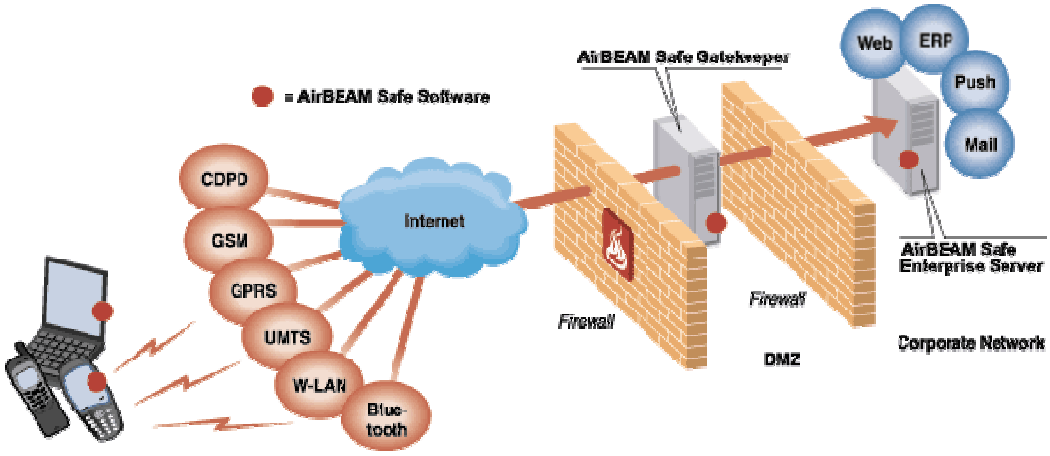

Figure 1: AirBEAM® Safe architecture

2.4.1 AirBeam Safe, a routable OSI layer 2 VPN

The AirBeam Safe client captures data packets going to and from the server applications in a Virtual NIC that is installed with the client. The virtual NIC works on layer 2 in the OSI model.
The data packets are then forwarded to the AirBeam Safe Client service that maintains the secure tunnel to the AirBeam Safe server on layer 4 in the OSI model. The benefits of this architecture is that AirBEAM safe is able to give the security of encrypting data packets on the data layer (layer 2), providing strong authentication and having a routable network protocol. This means that no information of internal network architectures can be deducted from the data packets even if the network protocol is routable.
AirBEAM safe is a routable layer 2 VPN that provides enhanced mobility features.

2.5 WTLS security layer

WTLS stands for Wireless Transport Layer Security (reference [3]) and is a part of the Wireless Application Protocol standard. WTLS is an adaptation of TLS for wireless environments. The commercially available WTLS layer is included in the software crypto boundary of the WTLS Crypto Module; the WTLS protocol provides a key establishment technique that has been implemented to meet the requirements of FIPS PUB 140-2 Annex D.

WTLS full-handshake
The WTLS Crypto Module uses the WTLS layer to establish a *secure tunnel* between the client and the server, which is performed using the WTLS handshake protocol. Below is a brief description of the handshake procedure:

1. The client sends a client hello message explaining which cryptographic algorithms it wants to use.
2. The server responds with a server *certificate* that contains a public *RSA* key followed by a client *certificate* request.
3. The client verifies the servers *certificate* by verifying it against a trusted CA *certificate*. If the server *certificate* is accepted, the client produces a *Pre master secret* of random data, which is encrypted with the server's *public key*. This is the client key exchange message. The client creates a client *certificate* verify message, which is a hash of all previous sent messages that is signed with the *private key* of the client *certificate*. The client sends the *certificate* verify message together with its client *certificate* and the client key exchange message to the server.
4. The server verifies the client *certificate* to ensure that it is an allowed user that has connected. The key exchange message is decrypted with the server's *private key*, which gives the result that both the client and the server has a shared secret, the master secret.
5. A *secure tunnel* is established between the client and the server. The master secret is used to create symmetric keys for *bulk data* transfer. Algorithms used for *symmetric encryption* are *AES* and *T-DES*.

WTLS abbreviated handshake
Abbreviated handshake (as specified in the WTLS protocol) is used when resuming a connection when for example the network connection is lost. Since both the client and server have the shared *master secret*, the key agreement procedure does not have to take place.

2.6 FIPS 140-2 Security Level

Table 1 specifies the security levels to which *WTLS Cryptographic Module* has been certified. This gives an overall level 1 certification.

Security Requirements Section	*FIPS* 140-2 Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services and Authentication	1
Finite State Model	1
Physical Security	1
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	3
Self-Tests	4
Design Assurance	2
Mitigation of Other Attacks	N/A

Table 1: Module Security Level Specification

3 Cryptographic Module

3.1 Cryptographic Boundary

WTLS Cryptographic Module is strictly a software module with the embodiment of multi-chip standalone. The *crypto module*'s cryptographic boundary can be divided into software cryptographic boundary and hardware cryptographic boundary:

- **Software cryptographic boundary**
  The software cryptographic boundary consists of the WTLS layer. Apart from the WTLS implementation it includes a cryptographic library and a *certificate* library. The software WTLS Crypto Module is identical on the AirBEAM® Safe Client, the Gatekeeper and the Enterprise server.

- **Physical cryptographic boundary**
  The physical cryptographic boundary is a standard PC hardware platform, and one of the operating systems listed under Section 2.4 above.

Each AirBEAM® Safe component (Client, Gatekeeper and Enterprise server) uses an own standalone equal copy of the software *WTLS Cryptographic Module*. The *crypto module* is encapsulated in the AirBEAM® Safe component that uses it.

3.2 Logical interfaces

The logical interfaces are divided into four distinguished parts (see Figure 2). The data input, data output, control input, and status output are all accessed via the WTLS Crypto Module API:

- **Data input interface**
  The password for the *private key* is input as data to the crypto module using the keyboard.
  Data that is input to the module to be encrypted or decrypted can be separated into two types of data:
  - Encrypted data coming from the *secure tunnel*, i.e. data coming from the public network.
  - Plaintext data coming from the local machine.

- **Data output interface**
  Data output can be separated into two types of data:
  - Encrypted data that is to be sent to the *secure tunnel*.
  - Decrypted data that is to be sent to the local machine.

- **Control input interface**
  Control inputs such as running *self-tests* on demand and algorithm configuration are done by the operator in the *FIPS-configuration interface.* The actual control input interface (as required by FIPS 140-2) is the WTLS Crypto Module API, and the *FIPS-configuration interface* is simply a GUI that allows an operator to send and receive data via the API

  Control input can also come from the other AirBEAM® Safe components via the API.

- **Status output interface**
  Error log messages, information log messages and *crypto module* status information. The status output interface is part of the WTLS Crypto Module API; status information is retrieved from the API and handled outside of the crypto boundary by the *FIPS-configuration interface (GUI)* and *FIPS-log interface (part of the OS)*.

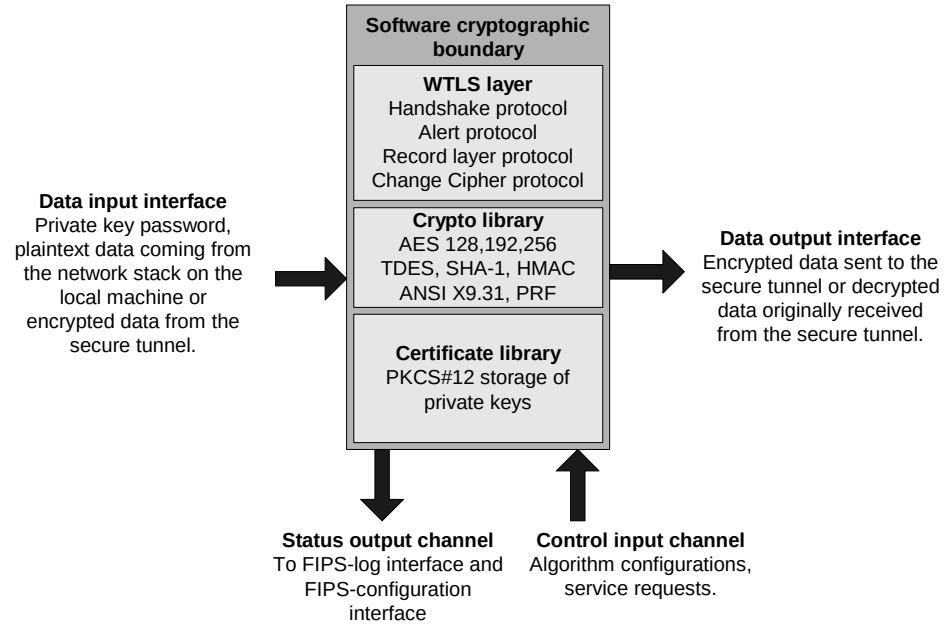

Figure 2: Software logical interfaces

3.3 Physical ports and mapping to logical interfaces

The physical boundary and its physical ports are shown in Figure 3. The following ports are used on a general PC:

- **Keyboard**
  The keyboard can be used to input data that is input to some application on the device and then sent to the module. The keyboard then maps to the data input channel. The keyboard can also be used to manipulate the *FIPS-configuration interface* and is then mapped to the control input channel.
- **Mouse**
  The mouse is used to edit the configuration in the *FIPS-configuration* interface. Maps to the control input channel.
- **Monitor**
  The monitor can be used to show the information from the *FIPS-configuration interface* and the *FIPS-log interface*. The monitor then maps to the status output channel. The monitor can also be used to show plaintext data that has been output from the crypto module to an application on the device. The monitor then maps to the data output.

- **Network connections**
  A network connection is a way to connect the device to other devices. Examples of network connections are network interface cards, modems and dial ups using cell phones via infrared sensors or bluetooth. The network connections are used for sending encrypted data to the *secure tunnel* and receiving encrypted data from the *secure tunnel*. Maps to the encrypted data part of the data input and output interface. No decrypted data is sent to the network connections. The TCP/IP stack on the local computer will redirect this traffic back to an application located inside the physical boundary. Control input can also be sent between AirBEAM Safe® components. The control input then comes into the device through a network connection and the network connection is then mapped to the control input interface.
- **Screen**
  On a Pocket PC, Windows CE 4.2 and Windows CE 5.0 the screen serves as data in, data out, control and status interface.

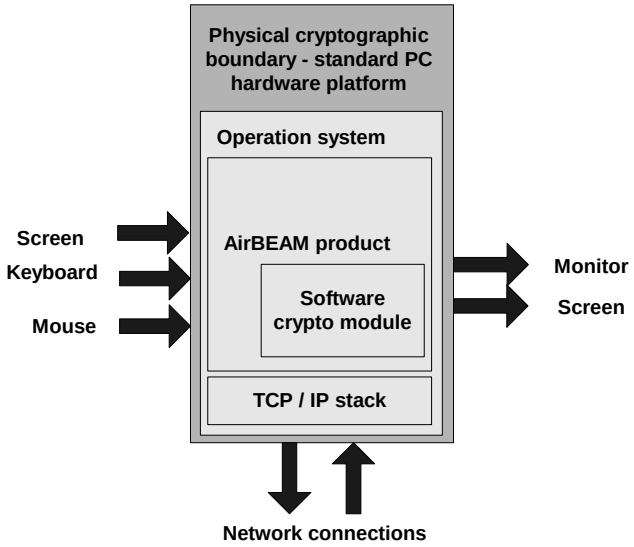

Figure 3: Physical boundary and physical ports

Apart from these interfaces there is also a power interface where the devices can be connected to external power sources. The possible external power sources are batteries and power cords. On Pocket PC, Windows CE 4.2 and Windows CE 5.0 there is also a built in battery that is recharged when the device is connected to an external power source and that is used when the device is not connected to an external power source.

4 Identification, Authentication and Access Control Policy

4.1 Cryptographic roles

A human *operator* of the *crypto module* can assume one role. This role is the crypto-officer role. An operator implicitly assumes this role when having physical access to the cryptographic module, and by authenticating to the OS.

There is also an internal software role called ABS-user, the User role required for FIPS 140-2. This role is assumed by the rest of the AirBEAM® Safe software. Assuming this role enables AirBEAM® Safe to request WTLS-session establishment, input data to the module and receive data that is output from the module. This enables AirBEAM® Safe to send and receive encrypted data (outside of the crypto boundary) over different communications networks and send and receive plaintext data to and from applications on the computer on which *WTLS Cryptographic Module* is run. This role is automatically assumed by AirBEAM® Safe when the *ABS-system* is started.

When the *ABS-system* is started the crypto officer only requests services from the *crypto module* indirectly via the other parts of the software and the direct requests to the *crypto module* are made by the ABS-user. There can only be one ABS-user for each *crypto module*. Therefore there can only be one concurrent user of the module.

4.2 Identification and authentication

4.2.1 Crypto officer authentication

In order for an *operator* to be able to operate the *crypto module* the operator must be logged into the operating system of the computer that the *crypto module* is running on. The security offered by this authentication depends on how the OS is configured. This authentication typically consists of stating a valid combination of username and password to the operating system. The WTLS Crypto Module relies on the OS to perform this authentication procedure; the OS login information is not passed to the crypto module itself. The operating system must be configured to require a password of at least six characters and it must be configured to work in a single user mode. It is up to the crypto officer to make sure that these requirements are met.

In order for the crypto officer to be able to read a private key from the hard drive so that it can be used when establishing a secure connection the crypto officer must also provide a password for the *private key*. This password is used to decrypt the private key, which is an additional authentication procedure supported by the module. This password must be at least six characters long.

4.2.2 ABS-user authentication

AirBEAM® Safe is used to enable secure remote access to a LAN. In order to maintain this security the different components must authenticate each other before sensitive data can be communicated. This is done by establishing secure, encrypted WTLS tunnels via the WTLS Crypto Module. In addition to the authentication performed when establishing the *secure tunnels* AirBEAM® Safe also supports other authentication mechanisms (outside of the crypto boundary).

4.3 Services

This section gives descriptions of the different services that the operators can request from *WTLS Cryptographic Module*.

4.3.1 Start ABS-system

Starting the *ABS-system* means starting the main process of the AirBEAM® Safe component. On the Windows based components this means starting a service. This can be done from the Windows Service Manager. On the clients the service can also be started from the AirBEAM® Safe Monitor which is a part of the interface for an AirBEAM® Safe client separate from the configuration interface. On Pocket PC 2002, Windows CE 4.2 and Windows CE 5.0 the *ABS-system* is a process that is started by running an executable file. When the *ABS-system* is started a message is sent to the *FIPS-log interface*.

4.3.2 Stop ABS-system

Stopping the *ABS-system* means stopping the main process of the AirBEAM® Safe component. On the Windows based components this means stopping the service. This can be done from the Windows Service Manager. On the client the service can also be stopped from the AirBEAM® Safe Monitor. On Pocket PC 2002, Windows CE 4.2 and Windows CE 5.0 the *ABS-system* is stopped by clicking “Exit” in the AirBEAM® Safe menu (see section 6.2). When the *ABS-system* is stopped a message is sent to the *FIPS-log interface*.

4.3.3 Initiate crypto module

The *crypto module* is initiated automatically upon powerup by seeding the *PRNG* and running the *self-tests*. This service returns an indicator as to whether the initialization was successful or not. If the *self-tests* fail the failed test is also logged to the *FIPS-logging interface*.

4.3.4 Delete private RSA key

Deletes a private *RSA* key from the hard drive (registry).

4.3.5 Import new key pair

An *RSA* key pair consists of a *private key* and a *public key* stored in a *certificate*. Importing a new key pair is done by importing a *certificate* from the certificate handling part of the configuration application for the component that uses the *crypto module*. The private key and the certificate are stored in different files. On Pocket PC 2002, Windows CE 4.2 and Windows CE 5.0 the files containing the key pair can be

placed anywhere on the device and the crypto officer can choose which key pair to import. On *crypto modules* running in FIPS mode on a computer using Windows the files must be placed on a floppy disk inserted in the A: drive of the computer, or via CD ROM. The desired file names are then typed in. If a *private key* and corresponding *certificate* already exist on the computer they must be deleted using the Delete private RSA key service before this service can be invoked. The file containing the *private key* will be deleted.

4.3.6 Establish secure connection

Establishes a secure connection to a *WTLS crypto module* on another AirBEAM® Safe component. This is done by exchanging the messages required to perform a WTLS-handshake (see section 2.4.1).

4.3.7 Close secure connection

Closes a secure connection, if one is established. The connection can either be immediately closed or messages can be sent so that the other party is notified that the connection should be closed. Which alternative is used is determined by input from the ABS-user.

4.3.8 Check if a secure connection is established

Gives status information indicating if the *crypto module* has an established secure connection with another *crypto module*.

4.3.9 Encrypt data

Encrypts plaintext data that is sent into the module and outputs the encrypted data.

4.3.10 Decrypt data

Decrypts encrypted data that is sent into the module outputs the decrypted plaintext data.

4.3.11 Get remote certificate

If a secure connection is established and if the other party in the connection sent a *certificate* during the establishment of the connection, the other party's *certificate* (and corresponding public key) can be obtained by requesting this service.

4.3.12 Get state

Gets the current state of the *crypto module*.

4.3.13 Run self-tests on demand

Performs the power-up *self-tests* on demand. What tests are performed is described in section 5.2. This service is requested from the *crypto module* by the crypto-officer by clicking the "Run self-tests" button in the *FIPS-configuration interface* (see section 6.2). The result of the *self-tests* is logged in the *FIPS-log interface*.

4.3.14 Configure crypto module

The crypto module is configured by making settings in the AirBEAM® Safe component's configuration applet, which accesses the control input interface via the crypto module's API. Most configuration changes aren't enforced until the next time the *ABS-system* is started (which is outside the scope of FIPS 140-2).

4.3.15 Read CA certificates from hard drive

The AirBEAM® Safe components store the CA *certificates* that it wishes to use on the hard drive (in the registry). When this service is requested the *crypto module* reads these certificates and stores them in the RAM.

4.3.16 Read client certificate and corresponding private key from hard drive

The AirBEAM® Safe clients store the client *certificate* and corresponding *private key* on the hard drive (in the registry). In order to be able to read the *private key* from the hard drive and make sense of it a password is required. This password is used to decrypt the private key, which is an additional authentication procedure supported by the module. The ABS-user asks the crypto-officer for a password and then the ABS-user requests this service from the *crypto module*. The *crypto module* returns an indicator as to whether the *certificate* and *private key* were successfully read or not. The *certificate* and *private key* are then stored in RAM.

4.3.17 Read server certificate and corresponding private key from hard drive

The Enterprise servers and Gatekeepers store the server certificates and corresponding *private keys* on the hard drive (in the registry). In order to be able to read the *private key* from the hard drive and make sense of it a password is required. When requesting this service from the crypto module the ABS-user asks the crypto-officer for the password needed for the *private key*. This password is then relayed into the module. The *certificate* and *private key* are then stored in RAM.

4.3.18 Read certificate revocation lists from hard drive

AirBEAM® Safe components store Certificate Revokation Lists (CRLs) on the hard drive (in the registry). Certificate revocation lists are used by a certificate authority to revoke *certificates* that for some reason (other then that their time of validity has expired) are no longer valid. When this service is requested the CRLs are read from the registry and stored in RAM. These can later be used when verifying a *certificate* to see that the *certificate* has not been revoked.

4.3.19 Generate random

The ABS-user can get data generated by the *crypto module*'s ANSI X9.31 compliant *PRNG*. The algorithm used is ANSI X9.31 (see reference [4]) with DES E-D-E two-key triple-encryption. When requesting this service the ABS-user specifies the desired number of bytes and the *crypto module* responds by generating the data and returning it to the ABS-user.

4.3.20 Check certificate against certificate revocation lists

This service allows the ABS-user to check a certificate against the revocation lists that have been read into the crypto module. The ABS-user inputs the certificate to the crypto module and the crypto module goes through the revocation lists it has to see if the certificate has been revoked and returns an indicator as to whether the certificate is revoked or not, according to the revocation lists.

4.3.21 View FIPS-log interface

All status information is sent from the WTLS Crypto Module via the defined status output interface. Once the status info leaves the crypto boundary it may be handled in various ways. Logging messages from the crypto module are sent to the *FIPS-logging interface (part of the OS)*. The crypto officer can then view these logged messages by either opening the Windows Eventlog, on Windows based crypto modules, or opening the Eventlog-file accessed from the AirBEAM® Safe menu (see section 6.2), on Pocket PC 2002, Windows CE 4.2 and Windows CE 5.0 based crypto modules.

4.3.22 Zeroize CPSs stored in RAM

By powering off (or rebooting) the device that the *crypto module* is run on the crypto-officer can make sure that all *CSPs* stored in RAM are zeroized. The *crypto module* does this by first writing zeroes over all the parts of the RAM where *CPSs* are stored and then deallocating these memory parts. The operator is then required to power off the device to ensure that all of the data stored in RAM is completely zeroized.

4.3.23 Zeroize crypto module

In order to completely remove everything related to the crypto module from the hard drive (including *private keys* stored on hard drive, *certificates*,software components, and all CSPs) the AirBEAM® Safe component must be uninstalled.

4.3.24 End crypto module

Releases all allocated resources necessary to maintain FIPS operating state.

4.4 CSPs

The *CSPs* are stored inside a security template class called CTSecBuf or inside a template class called SecBlock. These classes actively write zeros in the data buffer when the object of the class is destroyed. All these objects are destroyed when the *ABS-system* is stopped, which happens automatically if the computer that the crypto module runs on is powered off. The following *CSP*s are used inside the *crypto module*.

- ***Private key* password**
  Required for reading a *private key* from the hard drive. This password is used to decrypt the private key, which is an additional authentication procedure supported by the module. The password is entered by the crypto officer of the cryptographic module.
- **Private *RSA*-key**
  A *private key* corresponds to a *public key* given in a *certificate*. The *private key* is generated by an external *certificate* program that is not a part of the *crypto module*. When the crypto module is not running the private key is stored on the hard drive (in the registry). In order to make sure that only authorized operators get access to the private key a password is required for access to it. The password is expanded using SHA-1 hashing and the result is used to create an HMAC key and an AES encryption key. The private key is then HMACed and AES encrypted. When decrypting the private key the password is required and the HMAC is used to check that the decryption was successful. Decrypted private *RSA*-keys are only stored in RAM.
- ***Symmetric encryption* write key**
  Used for encrypting incoming *bulk data*. Is created by the *PRF* function within the WTLS protocol.
- ***Symmetric decryption* read key**
  Used for decrypting incoming *bulk data*. Is created by the *PRF* function within the WTLS protocol.
- **Write *IV***
  IV used in *CBC* mode when encrypting with the *symmetric encryption* algorithm. Is created by the *PRF* function within the WTLS protocol.
- **Read *IV***
  IV used in *CBC* mode when decrypting with the *symmetric decryption* algorithm. Is created by the *PRF* function within the WTLS protocol.
- ***MAC* write secret**
  Secret used in the *HMAC* function when sending encrypted data. Is created by the *PRF* function within the WTLS protocol.
- ***MAC* read secret**
  Secret used in the *HMAC* function when receiving encrypted data. Is created by the *PRF* function within the WTLS protocol.
- **Pre master secret**
  The pre master secret consists of random data generated by the client's *crypto module*. This secret is encrypted with the server's public key within the WTLS protocol. The pre master secret is created by the *PRNG*.
- **Master secret**
  The master secret is a shared secret between the client and the server. It is used

to generate the *Symmetric encryption* write key, the *Symmetric decryption* read key, the Write *IV*, the Read *IV*, the *MAC* write secret and the *MAC* read secret. The master secret is generated by the Pseudorandom Function (*PRF*) starting from the pre master secret within the WTLS protocol.

- **Seed and seed key to *PRNG***
  The seed consists of 64 bits of data and the seed key is a two-keyed *T-DES* key. They are used to initialize the *PRNG*. They are both generated from unpredictable data coming from a source outside the software *crypto module*.

4.5 Roles, services and authentication

Many of the services accessible to the ABS-user are requested on behalf of the person using the AirBEAM® Safe component, that is the crypto officer.

Role	Authorized Services
Crypto-officer	• Start *ABS-system* • Stop *ABS-system* • Delete private *RSA* key • Import new key pair • Run *self-tests* on demand • View *FIPS-log-interface* • Zeroize *CSPs* stored in RAM • Zeroize *crypto module*
ABS-user	• Initiate *crypto module* • End *crypto module* • Establish secure connection • Close secure connection • Check if a secure connection is established • Encrypt data • Decrypt data • Get remote *certificate* • Get state • Configure *crypto module* • Read CA *certificates* from hard drive • Read client *certificate* and corresponding *private key* from hard drive • Read server *certificate* and corresponding *private key* from hard drive • Read *Certificate* Revocation Lists from hard drive • Generate random • Check *certificate* against CRL

Table 2: Services Authorized for Roles

Service	Cryptographic Keys and *CSP*s	Type(s) of Access (R=read, W=write)
Initiate *crypto module*	• Seed to *PRNG*	W
Delete private *RSA* key	• Private *RSA* key (stored on hard drive)	W
Import new key pair	• Private *RSA* key (stored on hard drive)	W
Zeroize *CSPs* stored in RAM	• Private *RSA*-key (stored in RAM) • Master secret • *Symmetric encryption* write key • *Symmetric decryption* read key • Write *IV* • Read *IV* • *MAC* write secret • *MAC* read secret	W
Zeroize *Crypto Module*	• Private *RSA*-key (stored in RAM) • Private *RSA*-key (stored on hard drive) • Master secret • *Symmetric encryption* write key • *Symmetric decryption* read key • Write *IV* • Read *IV* • *MAC* write secret • *MAC* read secret	W
Read client *certificate*	• *Private key password* • Private *RSA*-key (stored on hard drive) • Private *RSA*-key (stored in RAM)	RW R W
Read server *certificate*	• *Private key password* • Private *RSA*-key (stored on hard drive) • Private *RSA*-key (stored in RAM)	RW R W

Establish secure connection	• Pre master secret • Master secret • *Symmetric encryption* write key • *Symmetric decryption* read key • Write *IV* • Read *IV* • *MAC* write secret • *MAC* read secret	RW
Encrypt data	• *Symmetric encryption* write key • Write *IV* • *MAC* write secret	RW
Decrypt data	• *Symmetric decryption* read key • Read *IV* • *MAC* read secret	RW

Table 3: Access Rights within Services

5 Self-tests

5.1 Self-test failure

If any of the *self-tests* fails, the *crypto module* will enter *FIPS-error-state*, with an error message sent to the *FIPS-log interface.* To be able to go back to *FIPS-mode-state*, the device must be rebooted. If the *self-tests* fail more than three times in a row, the software may be damaged and the customer *SHOULD* contact the manufacturer.

5.2 Power-up self-tests

The power-up *self-tests* are executed every time the *ABS-system* is started, without any *operator* input or activity. The operator can also activate the tests on demand. If all *self-tests* were successful, a message is sent to the *FIPS-log interface*. What happens if the tests fail is described in section 5.1. While the *crypto module* is performing the *self-tests* no data is allowed to pass the module.

The performed cryptographic algorithm tests are known answer tests since there is only one implemented version of each algorithm. The tests are executed in the following order:

.

- **Statistical PRNG tests**
  The following tests are performed, in accordance to reference [1], on 20000 bits of data generated by the *PRNG*:
  - Monobit Test
  - Poker Test
- **Digital signature algorithm test**
  In order to test that the *RSA digital signature* algorithm functions correctly an asymmetric key pair is compiled into the software. Using this key pair a digital signature is first generated for 20-byte of data generated by the PRNG. The signature is then compared to the signed data and if they are equal the test fails. After that a digital signature verification is performed. If the verification is successful the test succeeds, otherwise the test fails. This test is performed before the software integrity test.

- **Software Integrity Test**
  A digital signature (using 2048 bit *RSA*) of the binary executable file, which contains the *crypto module*, is saved in a file and distributed with the product. The power-up software integrity test verifies the signature using a public *RSA* key that is compiled into the product. If the signature cannot be verified the test fails.

- ***AES CBC* Known answer tests**
  The following test is performed with 128, 192 and 256 bit keys, using sequence number 0, 1, 5. A known key and IV is used in all tests. A new IV is calculated before encryption / decryption according to reference [3]. A known plaintext (1024 bytes) is encrypted resulting in a cipher text. The cipher text is compared with a known cipher text, the known answer. If they are not equal,

the test fails. The cipher text is decrypted using the known key. The decrypted plaintext is compared with the original plaintext. If the buffers are equal, the test for a specific key succeeds. If all the tests for 128, 192 and 256 bits keys succeed, the *AES CBC* known answer test succeeds, else it fails.

- **DES CBC Known-answer test**
  The DES CBC test uses sequence number 0. A known plaintext (1024 bytes) is encrypted resulting in a cipher text. The cipher text is compared with a known cipher text, the known answer. If they are not equal, the test fails. The cipher text is decrypted using the known key. The decrypted plaintext is compared with the original plaintext. If the buffers are equal, the test is successful, else it fails.

- ***T-DES CBC* 3-Keys Known-answer test**
  The T-DES CBC test uses sequence number 0. A known plaintext (1024 bytes) is encrypted resulting in a cipher text. The cipher text is compared with a known cipher text, the known answer. If they are not equal, the test fails. The cipher text is decrypted using the known key. The decrypted plaintext is compared with the original plaintext. If the buffers are equal, the test is successful, else it fails.

- ***SHA-1* Known-answer test**
  The *SHA-1* algorithm is tested with a stand-alone known-answer test. The known answer is a compiled in 20 bytes hash, which is created by hashing a compiled in text string. The test executes the hashing over the text string and compares the new hash with the compiled in hash. If the hashes are equal, the test is successful.

- ***HMAC* Known-answer test**
  The *HMAC* algorithm is implemented according to IETF RFC 2104 (see reference [8]), based on *SHA-1* as underlying hash algorithm. The algorithm is also defined on page 77 in reference [3]. The test algorithm uses a known compiled in *MAC*, created by a known compiled in *MAC*-secret and a known compiled in data string. The test then creates the *MAC* and compares it with the compiled in *MAC*. If the *MAC*s are equal the test is successful, else it fails.

- **RSA *Public key* encryption known answer test**
  A random generated data buffer of 20 bytes is used as known answer. The data is encrypted (RSAES-PKCS1-v1_5 described in PKCS#1 v2.0, see reference [9]) using a known *public key*. The encrypted buffer is compared with the plaintext data. If the buffers are equal, the test fails. If the buffers are not equal, the encrypted buffer is decrypted with the corresponding *private key*. If the decrypted data does not exactly match the known input data, the test fails.

- **Pseudorandom function known answer test**
  A known answer test that makes sure that the *PRF* works according to specification. Input data and the corresponding output are compiled into the software. The test is performed by feeding the input to the PRF and comparing the result to the compiled in output. If they are equal the test succeeds otherwise the test fails.

5.3 Conditional self-tests

The conditional tests are performed before the function to be tested is used. The following conditional *self-tests* are implemented. A failure of a conditional *self-test* is treated as described in section 5.1.

- **PRNG seed test**
  This test compares the PRNG seed to the PRNG seed key. If they are equal the test fails otherwise the test succeeds.
- **Continuous PRNG Test**
  This test is executed each time the *PRNG* is called to return random data. The test is implemented according to reference [2]. The generated data is divided into blocks of two bytes. Each such block is compared to the previously generated block. If they are equal the generated block is ignored and a new block is generated. The module has a counter that is incremented each time this test fails. If this happens three times in a row the *crypto module* enters the *FIPS-error state*. The first generated block is kept for comparison but not used as random data.

- ***RSA Public key* encryption, *private key* decryption pair-wise consistency test**
  This test is only performed on the Gatekeeper and the Enterprise server. After that the server received a ClientHello message (see reference [3]) and before the server *certificate* is sent to the client, a data buffer of 40 bytes of random plaintext data are encrypted with the *public key*. The encrypted buffer is compared with the plaintext data. If the buffers are equal, the test fails and the module is set to *FIPS-error state*. If the buffers are not equal, the encrypted buffer is decrypted with the corresponding *private key*. If the decrypted buffer is equal to the random plaintext data, the test is successful, and the *public key* is sent to the client. If the buffers are not equal, the test fails.

- ***RSA* Digital signature test**
  This test is only performed on the client side. After that the client verify message has been signed (see reference [3]) with the clients *private key*, the signed data is verified with the clients public key before it is sent to the server. If the verification was not successful or the signed data was equal to the original data, the *digital signature* test failed.

6 FIPS-mode

6.1 Introduction

AirBEAM® Safe can be operated in *FIPS-mode*. It is for this mode of operation that *WTLS Cryptographic Module* has been *FIPS*-validated. When in *FIPS-mode*, the crypto-module uses only *FIPS*-approved algorithms and other steps have been taken to further guarantee security.

Since the different components of AirBEAM® Safe (Client, Gatekeeper and Enterprise Server) contains separate *crypto modules* each component is individually configured to work in *FIPS-mode*.

6.2 FIPS-Configuration interface

The *FIPS-configuration interface* is a part of the configuration tool used by the AirBEAM® Safe components, and is not part of the defined crypto boundary. On Windows NT4, Windows 2000 and Windows XP the *FIPS-configuration interface* is a control panel applet. On Windows CE the *FIPS-configuration interface* is a part of the main AirBEAM® Safe Client application.

Figure 4 shows a screen shot from a Windows CE screen where the AirBEAM® Safe menu is opened. The *FIPS-configuration interface*, shown in figure 5, is launched by clicking the *FIPS* option in the menu.

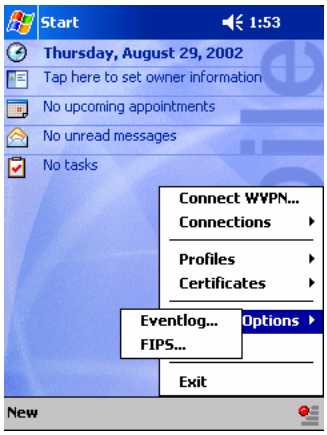

Figure 4: AirBEAM® Safe menu on Pocket PC

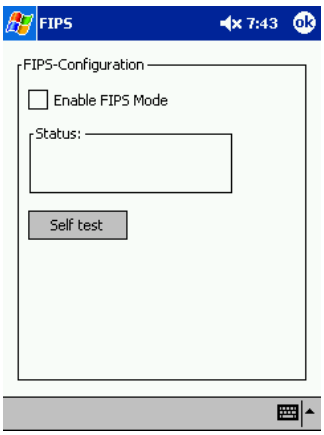

Figure 5: Windows CE *FIPS-Configuration interface*

Figure 6 shows the *FIPS-configuration interface* on the Windows Client, Windows Enterprise Server and Windows Gatekeeper. The interface can be activated by starting the control panel application for AirBEAM® Safe located in the Windows Control Panel and then choosing the *FIPS*-tab.

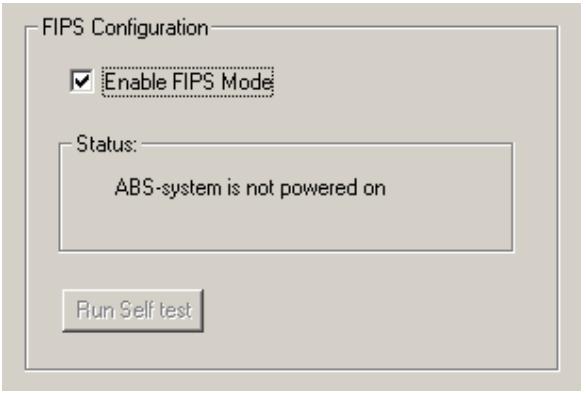

Figure 6: *FIPS-configuration interface* on Windows

The *FIPS-configuration interface* can be used both for entering control input and for viewing status output via the WTLS Crypto Module API. The check box labeled "Enable FIPS-mode" and the button labeled "Run Self test" represents control input that is entered into the *crypto module*. The Status field represents status output from the module.

Using the check box labeled "Enable FIPS Mode" the *operator* can choose whether *FIPS-mode* shall be employed or not. The module will not automatically be in *FIPS-mode* when this check box is checked. The *operator* will have to restart the *ABS-system* so that the module can make sure that all criteria for being in *FIPS-mode* are met.

The button labeled "Run Self test" makes it possible for the *operator* to run the *self-tests* on demand. To be able to run self-tests on demand, the status of the module must be "FIPS-mode".

The status field shows which state the *crypto module* is in. The possible states are:

- ABS-system is not powered on.
- Working but not in *FIPS-mode*
- *FIPS-mode*
- Running *self-tests*
- *FIPS-error state* (see section 6.4).

6.3 FIPS Mode Algorithms

When in *FIPS-mode* the following cryptographic algorithms are used:

- *RSA* with 1024 (or higher) bit keys for key-exchange and *digital signatures*.
- *T-DES 3-Key CBC* for encryption of *bulk data*.
- *T-DES 2-Key ECB* for random data generation.
- *AES* CBC using 128, 192 or 256 bits for encryption of *bulk data*.
- *SHA-1* for hashing.
- *HMAC* using *SHA-1* for computing message authentication codes.

Which algorithms are used within a specific WTLS-session is negotiated during the handshake. When in *FIPS-mode* the client only accepts algorithms that are chosen appropriately for FIPS 140-2. The operator of the Enterprise server and Gatekeeper determines which algorithms are actually used. This is done by configuration using the AirBEAM® Safe configuration applet. The configuration is done by choosing an acceptable *symmetric encryption* algorithm and an acceptable *RSA* key length of equal strength. The only hashing algorithm available in *FIPS-mode* is *SHA-1*. When in *FIPS-mode, WTLS Cryptographic Module* will only accept appropriate FIPS mode algorithm choices and no other combinations can be configured; non-FIPS mode configurations are restricted solely to non-FIPS mode, and are not accepted by the WTLS Cryptographic Module in FIPS mode. In this way the administrator of an AirBEAM® Safe installation determines which algorithms are used by configuring the Enterprise server and Gatekeeper. The *operator* of a client makes sure that only *FIPS*-compliant algorithm choices are used. The algorithms negotiated are also logged in the logging interface.

When in *FIPS-mode* the following combinations of algorithms for establishment of the symmetric keys, *symmetric encryption* and the hashing in *HMAC* calculations are allowed:

- Level 1 - *RSA* min 1024, *T-DES* 112, *SHA-1*
- Level 2 - *RSA* min 1024, *AES* 128, *SHA-1*
- Level 3 - *RSA* min 1024, *AES* 192, *SHA-1*
- Level 4 - *RSA* min 1024, *AES* 256, *SHA-1*

6.4 FIPS-error state

If the *self-tests* (see chapter 5) fail the *crypto module* enters a *FIPS-error state*. When the module is in the *FIPS-error state* all input and output of data to the cryptographic module is inhibited. Encryption or decryption operations will not be performed.

If the module enters the *FIPS-error state* the event that caused the error is logged in the *FIPS-log interface*. If the failed test was started from *FIPS-configuration interface* the status indication in the *FIPS-configuration interface* will indicate that the module is in *FIPS-error state*.

6.5 FIPS-log interface

Status messages from the *crypto module* are sent to the *FIPS-log interface*. The interface is a logging class that receives the information that should be logged. The information from the interface is presented somewhat differently depending on the OS that the *crypto module* is running on but the logged information is identical on all platforms.

- Windows 2000 and Windows XP
  On these platforms the information from the *FIPS-log interface* is presented in the event-log that is built into the OS.
- Pocket PC 2002, Windows CE 4.2 and Windows CE 5.0
  On this platform the information from the *FIPS-log interface* is presented as a log file that can be opened by clicking the "Eventlog" option in the AirBEAM® Safe menu (see section 6.2).

The error messages produced by events that forced the *crypto module* into *FIPS-error state* and other status messages are listed in Appendix A.

7 Physical Security Policy

WTLS Cryptographic Module is purely software based. Therefore no hardware security can be built in to the *crypto module* itself. Instead the module falls back on the measures taken to physically secure the computer on which it is run. It will typically be run on a general PC or Pocket PC, Windows CE 4.2 and Windows CE 5.0. Both general PCs and CE devices are made up of production-grade components.

Besides the physical security aspects of the construction of the computers on which the *crypto module* is run, there are also physical security aspects regarding the manner in which the computers are handled. For maximum security only authorized *operators* should be granted physical access to computers on which the *crypto module* is installed.

8 Mitigate attacks

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

9 Non FIPS approved mode

* NOTE THAT THIS SECTION DESCRIBES NON-FIPS MODE.

The use of *FIPS* approved algorithms in the non-*FIPS* mode provides no claims to *FIPS* validated protection.

The *crypto module* can be set up in a non *FIPS* approved mode. This is done by making sure that the check box labeled “Enable FIPS Mode” in the *FIPS-configuration interface* (see section 6.2) is not checked and then (re)starting the *ABS-system*.

In the non *FIPS* approved mode the following algorithms can be used:

- *RSA* with 512-15360 bit keys for key-exchange and *digital signatures*.
- *T-DES 3-Key CBC* for encryption of *bulk data*.
- *T-DES 2-Key ECB* for random data generation.
- *AES CBC* using 128, 192 or 256 bits for encryption of *bulk data*.
- *DES 56-bits* for encryption of *bulk data*.
- *DES 40-bits* for encryption of *bulk data*.
- SHA 160 (*SHA-1*), 256, 384 and 512 bits for hashing.
- 128-bits MD5 for hashing.
- *HMAC* using SHA for computing message authentication codes.
- *HMAC* using *MD5* for computing message authentication codes.

Which algorithms are used is configured the same way as in *FIPS-mode* (see section 6.3). That is, the administrator of the installation chooses the used algorithms by configuring the Enterprise Servers and Gatekeepers and the client can see in the logging interface which algorithms were negotiated during the WTLS-handshake. The difference here is that algorithm choices that are inappropriate for FIPS mode can be made. It is therefore up to the administrator to make sure that a sufficient security level is maintained.

10 Definitions and acronyms

ABS-system
The *ABS-system* is the main part of the AirBEAM® Safe component in which the *crypto module* is used. On Windows 2000 and Windows XP the *ABS-system* is a service. On Pocket PC 2002, Windows CE 4.2 and Windows CE 5.0 the *ABS-system* is a process that is started by running an executable file.

AES
Advanced Encryption Standard (see reference [5]). A *symmetric encryption* and *decryption* algorithm that can use 128, 192 or 256 bit keys.

Asymmetric encryption
Encryption using public or *private keys*

Asymmetric decryption
Decryption using public or *private keys*

Bulk data
Bulk data is data that the *operator* either wishes to encrypt and then send or receive and then decrypt.

CA
Certificate Authority

CBC
Cipher-Block-Chaining. A mode used in *symmetric encryption* or decryption.

Certificate
A *public key certificate* is a data structure in which some trusted third party guarantees the authenticity of an entity's *public key*. The crypto module supports two certificate formats: x.509 and WTLS.

Crypto module/ WTLS Cryptographic Module
The *crypto module* is the part of an AirBEAM® Safe component that handles all the cryptographic operations.

CSP
Critical Security Parameter. Security related information such as secret and *private cryptographic keys* and authentication data such as passwords.

Digital signature
A digital signature for a message (or some other array of data) is generated using a hashing function and a *private key*. The signature is then verified using the same hashing function and the *public key* corresponding to the *private key* used to generate the signature. The verification makes sure that the message has not been altered since the signature was generated and that it was generated using a specific *private key*.

FIPS
Federal Information Processing Standards

FIPS-configuration interface
Defined in section 6.2.

FIPS-error state
Defined in section 6.4.

FIPS-log interface
Defined in section 6.5.

FIPS-mode
Defined in chapter 6.

HMAC
Hashed Message Authentication Code. Ensures that data is not modified or deleted.

IV
Initialization Vector. Input to *symmetric encryption* algorithms.

LAN
Local Area Network

MAC
Message Authentication Code

NIST
National Institute of Standards and Technology.

Operator
An individual accessing a *cryptographic module* or a process (subject) operating on behalf of the individual, regardless of the assumed role.

PRF
WTLS specification of the pseudorandom function. The function is used for getting keys from the master secret and not for generating random data. This is used only within the commercially available WTLS protocol. See Reference [3] section 11.3.2 for more details.

Private key
The private part of an asymmetric key-pair. Used for decrypting public-key encrypted data and generating *digital signatures*.

Private key password
Password that provides an additional authentication mechanism, and allows the use of the RSA private key.

PRNG
The *crypto modules PRNG* is a pseudo-random number generator. The algorithm used is ANSI X9.31 (see reference [4]) with DES E-D-E two-key triple-encryption.

Public key
The public part of an asymmetric key-pair. Used for public key encryption and verifying *digital signatures*.

RSA
RSA is an asymmetric cryptographic algorithm used for key exchange, *digital signatures* and authentication (through the use of *certificates*). It is implemented in accordance with the encryption scheme RSAES-PKCS1-v1_5 described in PKCS#1 v2.0 (see reference [9]).

Secure Tunnel
A session established between the server and the client in the WTLS layer. All data passed through the secure tunnel is encrypted with a *symmetric encryption* algorithm.

Self-tests
Tests executed to ensure that the *crypto module's* cryptographic functions work.

SHA-1
Secure hash algorithm (see reference [7]). Produces a 20 bytes data sequence from a data buffer with arbitrary length.

Symmetric encryption
Encryption of data using a key where the encrypted data can be decrypted with the same *symmetric decryption* algorithm. Examples of algorithms are *AES* and *T-DES*.

Symmetric decryption
The opposite to *symmetric encryption*.

T-DES
Triple Digital Encryption Standard (see reference [6]). A *symmetric encryption* algorithm that uses three keys. The crypto module uses T-DES in two-keyed ECB mode and 3-keyed CBC mode.

TCP/IP
Transmission Control Protocol/Internet Protocol

WTLS

Wireless Transport Layer Security (see reference [3]).

11 Appendix A – Status messages

Id	Message
1	Failed to read the certificate revokation list %s.
2	Could not find a CA certificate for the certificate revokation list %s.
3	Invalid server certificate
4	Failed to get server certificate key. If the certificate was previously installed in FIPS mode, the server certificate must be reinstalled to the server.
5	Server certificate ignored. Reason: RSA key size mismatches security configuration
6	Unable to find a client certificate
7	Unable to find a client certificate in certificate map
8	A key in the registry used to retrieve the client certificates is missing. \n Due to this no client certificate could be found.
9	Certificates private key length mismatches allowed setting.\n Key length is %s and max allowed key length is %s and min allowed key length is %s
10	CA certificate does not exist, or the installed CA certificate does not match the other peers certificate.
11	Found corrupt CA certificate
12	No valid CA certificate is available
13	Verify the other peers certificate, verify subject failed (Subject(%s) CN(%s))
14	Verify the other peers certificate, certificate expired (Subject(%s) CN(%s))
15	Verify the other peers certificate, certificate revoked (Subject(%s) CN(%s))
16	Verify the other peers certificate, verify signature failed (Subject(%s) CN(%s))
17	The server certificate has expired or is not yet valid.
18	The client certificate has expired or is not yet valid.
19	No server certificate available.
20	No client certificate available.
21	The private key could not be read and decrypted.
22	The private key was successfully read and decrypted.
23	Selected Hash algorithm:%s Selected Cipher algorithm:%s
24	FIPS power-up test failed. Software integrity test failed.
25	FIPS power-up test failed. The monobit test failed.
26	FIPS power-up test failed. The poker test failed.
27	FIPS power-up test failed. Known-answer test for AES encryption has failed.
28	FIPS power-up test failed. Known-answer test for DES encryption has failed.
29	FIPS power-up test failed. Known-answer test for 3DES 3-key encryption has failed.
30	FIPS power-up test failed. Digital signature test failed.
31	FIPS power-up test failed. Known-answer test for SHA-1 has failed.
32	FIPS power-up test failed. Known-answer test for HMAC, SHA-1 has failed.
33	FIPS power-up test failed. Known-answer test for RSA public key encryption

	has failed.
34	FIPS power-up test failed. Known-answer test for the PRF function has failed.
35	FIPS 140-2 Continuous Random Number Generator Test failed. Please reboot the system to continue.
36	FIPS 140-2 Continuous Seed Test failed. Please reboot the system to continue.
37	Failed to decrypt the private key three times. The wrong password has been entered or there are no server certificate installer or the server certificates private key is not protected by a password. Please enter the correct password or install the server certificate again.
38	Entered FIPS error state. Please reboot the system.
39	Crypto Module is now running in FIPS-mode.
40	The FIPS self tests were executed successfully.
41	Continuous RSA public key encryption test failed.
42	Continuous RSA digital signature test failed.