Dell AppAssure Crypto Library
Version 1.0
Dell, Inc.

FIPS 140-2 Non-Proprietary Security Policy
Revision 1.0.2

Revision Date: March 26, 2014

Table of Contents

Module Overview

The Dell AppAssure Crypto Library (further designated as the Module) provides data encryption functionality to applications. The Module is a software component designed for use as a part of other software products to encrypt and decrypt data on general-purpose x64 PCs with Microsoft Windows OS. The Module is not intended for direct interaction with a human operator; the word *operator* stands in this document for a software product employing the Module. Keys used to encrypt data are provided by the calling application, the Module does not have built-in encryption keys. The module is a program code library that invokes AES CBC mode functions from Intel Integrated Performance Primitives (IPP) library.

The Module's certification levels are shown in the table 1.

Table 1: Certification levels

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
	Overall Level	**1**

Cryptographic Boundary

The cryptographic boundary contains two files: crypto.dll (DLL that implements the Module's functionality) and crypto.rsa (digital signature used for integrity self-test). On hardware side, the cryptographic boundary includes an x64 CPU on which the software is executed (Intel Xeon CPU family or compatible) and RAM attached to it that stores the Module's executable code and data used and produced by the Module.

Module's ports are library's entry points, i.e. callable functions exposed by the library. Data coming into the boundary are cryptographic keys, data to be encrypted or decrypted and initialization vectors. Data going out from the boundary are crypto contexts (data structures identifying encryption keys for future use), results of encryption or decryption, as well as the Module's current status. Data flows are straightforward: each encryption function consumes its arguments and returns values. All data processing is performed inside the library.

Relationships between the Module, the operator (i.e. the calling application), OS and hardware are depicted on Fig 1.

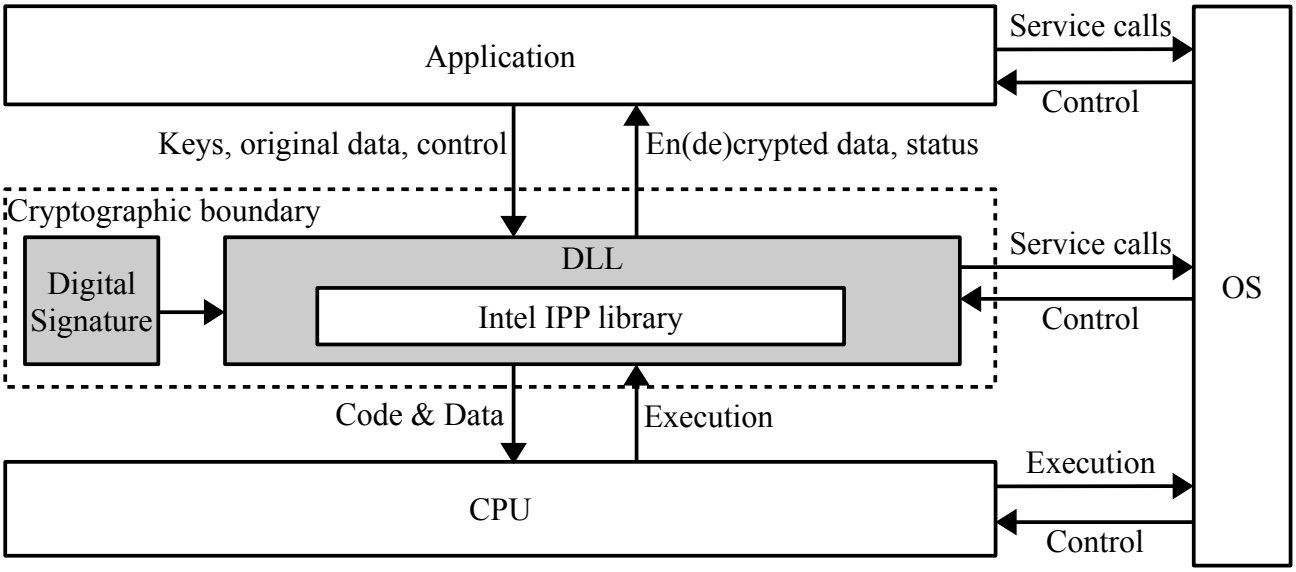

Fig 1: Logical Diagram

Modes of Operation

The Module supports only Approved mode.

Interfaces and Ports

Since the Module is a software component, its interfaces are defined in terms of the API functions it provides. Data Input Interface corresponds to passing parameters to user-mode API functions. Data output interface consists of return value of a function that creates a cryptographic context and output parameters of encryption/decryption functions. Control Input interface is a function call mechanism itself. Finally, Status Output Interface consists the status of the Module returned by `get_status` and `self_test` functions and return values of encryption/decryption functions, see table 2.

Table 2: Interface definition

FIPS 140-2 Interfaces	Module Logical Interfaces
Data Input Interface	Input parameters passed to user-mode API functions
Data Output Interface	Return value of a function that creates a cryptographic context; output parameters of encryption/decryption functions
Control Input Interface	Function call mechanism
Status Output Interface	Self-testing function; A function that returns the state of the Module; Return values of encryption/decryption functions; Null or non-null value of a function creating a cryptographic context

Roles and Services

The Module implements a User role and a Crypto-Officer role (see table 3). The Module does not support identification or authentication for these roles, an operator takes one of these two roles by calling respective functions. The operator can take only one role at any particular moment of time; concurrent invocation of functions meets the following rules:

- While at least one user-mode function is being executed by the Module, any call to a crypto-officer function is waiting;
- While a crypto-officer function is being executed, any call to a user-mode function is waiting.

Table 3: Roles

Role	Services
User	All encryption and decryption services; get status
Crypto-Officer	Run self-tests

An overview of API functions is presented in table 4. None of the functions use any cryptographic service providers – algorithms from Intel IPP library are used.

Table 4: API overview

Name	Purpose	Role	CSP* usage
`aes256_cbc_init`	Initialize a cryptographic context	User	Read
`aes256_cbc_rekey`	Assign a new key to an existing context	User	Write
`aes256_cbc_encrypt`	Encrypt a block of data	User	Execute
`aes256_cbc_decrypt`	Decrypt a block of data	User	Execute
`aes256_cbc_close`	Zeroize and dispose a cryptographic context	User	None
`get_status`	Get the Module's current status	User	None
`self_test`	Perform self-tests and set the Module's state	Crypto-officer	None

(*) CSP, if any, is an AES CBC key.

Operational Environment

The module is intended to be used in Windows applications on general-purpose x64 PCs (that are multi-chip standalone devices). There are no special requirements regarding additional components installed on the system, specific hardware or custom settings needed to operate the Module.

The Module only works in single operator mode; it is enforced by Windows DLL infrastructure: if multiple processes use the same DLL, each process maps it to its private address space and therefore its interaction with the DLL cannot be influenced by other processes. In other words, multiple operators of a DLL act as if each of them were a single operator of a separate copy of the DLL.

The Module has been tested in hardware and OS environments listed in table 5:

Table 5: Environments used for testing

Machine	CPU	AES-NI	OS
Dell PowerEdge T610	Xeon	On	Windows 2008 R2 64-bit
Dell PowerEdge T610	Xeon	Off	Windows 2008 R2 64-bit
Dell PowerEdge R720	Xeon	On	Windows 2012 64-bit
Dell PowerEdge R720	Xeon	Off	Windows 2012 64-bit

Physical Security

Because of purely software nature of the Module, physical security requirements are not applicable. Windows security mechanisms protect DLL's executable code and data in the calling process' address space from unauthorized access.

Cryptographic Key Management

The Module does not generate, store in a persistent storage, output cryptographic keys or other security-relevant data. Cryptographic keys are provided to user-mode encryption functions by the calling application, and it is the caller's responsibility to store cryptographic keys securely. Given a key, the Module creates a so called *context* (an internal data structure used for further encryption and decryption operations) and returns its pointer to the caller. Windows memory management mechanism guarantees that a Module, since it is a DLL, works in a protected address space of the calling process that is not accessible to other processes. Furthermore, a context is presented to the caller as an abstract pointer (void*) that hides the context's details. Before disposing a context, the Module fills its memory location with zeros.

Table 6: Keys used in the Module

	User-mode key	Software integrity test key
Key Type	AES CBC	RSA 1024 bit public key
Generation/ Input	From the calling application	Generated at design time, hard-coded
Output	None	None
Storage	None	Hard-coded
Zeroization	On demand	None
Use	Encrypt or decrypt caller's data	RSA with SHA-512 signature verification

Except symmetric cryptographic keys in user-mode functions, the Module uses an asymmetric key for integrity self-tests (see the respective section for more details). The key consisting of private and public parts has been generated once as a part of development process. Its private part is hard-coded in a signing tool and is not known to the Module, and the public part is hard-coded in the Module.

The summary of keys used in the Module is shown in table 6.

Approved algorithms are showh in table 7.

Table 7: Summary of cryptographic algorithms

Algorithm	CAVP validation No.
AES (Rijndael) CBC	2601
RSA Signature Verification	1329
SHA-512	2185

Self Tests

The module has the following power-up self tests, which also can be called on demand:

- Signature verification (RSA + SHA-512) known-answer self-test;

- Software integrity self-test;
- AES Encryption known-answer test;
- AES Decryption known-answer test.

A sequence of all tests (in the order shown above) is performed automatically on start-up and can be initiated at any time by the crypto-officer. If any test fails, the remaining tests in the sequence are skipped, and the Module switches to an error state that disables all cryptographic functions (this means, in particular, disabling the Data Output interface). The module indicates the error state with a non-zero return value for the `get_status` service. In the error state, only three functions are available: get current state, re-run self-tests and dispose a cryptographic context (that might be created before the error occurred). While a self-test is performed, all cryptographic functions (together with the Data Output interface) are inhibited.

The Module does not perform any operations that require conditional self-tests (in particular, there is no public/private key pair generation and persisting data).

Signature Verification Known-Answer Test

Before checking software integrity by verifying DLL's digital signature, the signature verification algorithm itself should be verified. For this purpose, the DLL contains a hard-coded piece of text and its signature generated once at design time with the same private key that is used to sign the DLL. The test uses a hard-coded public key to verify the known signature of the known text.

Software Integrity Test

The DLL file is shipped together with a file that contains its digital signature (RSA scheme with the SHA-512, as defined in version 1.5 of the PKCS#1 standard) that is produced during build process. After the DLL file is built, a build step invokes a signing tool that generates a signature and writes it to a file. Software integrity test verifies contents of the DLL file against the signature (the public part of the key is hard-coded in the DLL). This test fails if either the signature does not match or when the signature file could not be found.

Encryption and Decryption Known-Answer Tests

The Module contains a hard-coded piece of text. Encryption test encrypts the text with a predefined key and compares the result with a known answer that is hard-coded in the Module. Then the encrypted text is decrypted with the same key and the result is compared to the original. Each of these two tests fails if the actual result of encryption or decryption does not match the expected result.

Design Assurance

The Module is designed and managed in accordance with established procedures of Dell AppAssure. The source code and documentation is maintained in a Git repository on GitHub, with restricted access. Each change is formally reviewed using an online review capability of GitHub, and must be approved before commit. The Module is built on a build server when it detects source code updates. Signing the Module is a part of build process and is performed automatically.

Mitigation of Other Attacks

The Module is not intended to mitigate attacks not addressed by the security requirements of FIPS 140-2.