User Guide

TMP_DIS_1302_001 v0.5

EASY PLUG - VAULTIC ELIB FOR 4XX

Distribution

1. Approved Customers only (and internal release).
2. Valid Non Disclosure Agreement required.
3. All copies must be registered
   (with Name and Serial Number on each page).

driving trust™ inside SECURE

Table of Contents

1. Glossary

Table 1-1.

Term	Detailed Description
API	Application Programming Interface
SPI	Synchronous Physical Interface
TWI	Two Wire Interface
USB	Universal Serial Bus
DLL	Dynamic Link Library
SO	Shared Object library (typically seen in BSD Linux/Unix/MAC OS systems)
XOR	Bitwise Exclusive Or
DES	Data Encryption Standard
TDES	Triple DES
AES	Advanced Encryption Standard
CRC	Cyclic Redundancy Check
CCITT	Comit Consultatif International Tiphonique et Tigraphique
VaultIC Security Module	The VaultIC device is a hardware cryptographic module.
VaultIC eLib	The programmatic interface to allow communication with the VaultIC Device
Base API	The methods within the VaultIC API which provide a one to one mapping with the VaultIC Device commands
IDE	Integrated Development Environment

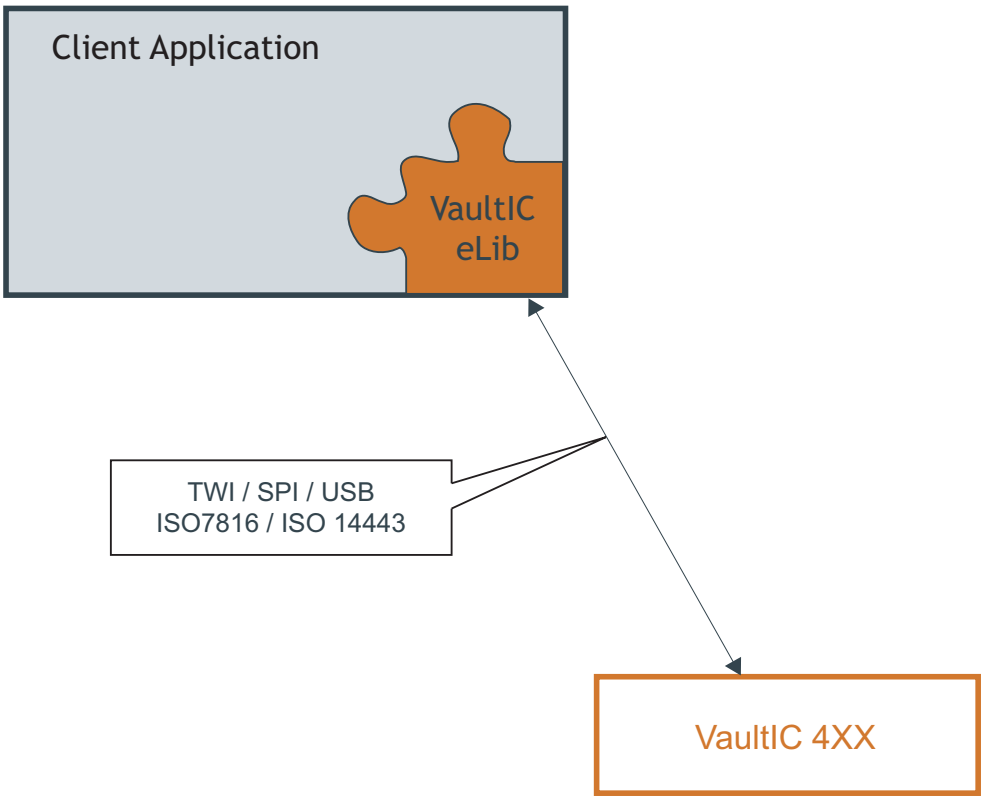
2. Introduction

The VaultIC Security Module product family offers a comprehensive security solution to a wide variety of secure applications. It features a set of standard public domain cryptographic algorithms, as well as other supporting services. These features are accessible through a well defined command set.

The aim of the VaultIC eLib is to provide a programmatic interface to the command set supported by the VaultIC Security Module product family. As the VaultIC Security Module supports a wide range of communication interfaces, the primary function of the VaultIC eLib is to serialise and de-serialise the specified command set in a manner that is easy to use and manage by the end customer application.

The VaultIC eLib provides a unique and easy to use interface that allows the customer to readily integrate VaultIC Security Module functionality in their own applications. This interface attempts to shield the customer application from the complexity of the underlying VaultIC Security Module command set and the effect of possible changes.

Figure 2-1.



The is delivered in various formats to suit the end customer application operating system and individual target constraints. These formats are:

- Dynamic Link Library (DLL) – for Windows based systems.
- Shared Object libraries (.so) – for Linux/MAC OS
- Source Code – for Embedded Targets

3. Reading this manual

It is strongly recommended that you read this manual in conjunction with the VaultIC 4XX Technical Datasheet [R1] in order to become familiar with the VaultIC Security Module capabilities, limitations and general usage information. Please contact your Inside Secure representative to obtain a copy.

This structure of this document should allow the reader to concentrate on the sections of interest. This will minimise the time taken to integrate the VaultIC eLib.

VaultIC eLib Architecture

The general architecture of the VaultIC eLib. This is particularly useful for customers that intend to integrate the VaultIC eLib into their embedded target applications.

VaultIC eLib Integration

This section provides the reader with step by step instructions on how to integrate the VaultIC eLib in the Client Application. However, this section is further divided down to the individual target environment that the client application will execute in, e.g. Windows, Linux, Mac OS, embedded targets.

Configuration & Customisation

This section describes how to configure the VaultIC eLib. Information is provided on how to configure the VaultIC eLib to match the end client application requirements e.g. which services are particularly useful for different application types.

VaultIC eLib Methods

The list of methods supported by the VaultIC eLib and its interfaces as well as a detailed description of each method, the expected behaviour, limitations and general usage information.

Client Stub Interfaces

This section is exclusively of interest to customers that intend to integrate the VaultIC eLib in their embedded target applications. It specifies which interfaces need to be implemented to cater for their system's specific requirements.

Troubleshooting

This section outlines basic trouble shooting scenarios.

Support & Contact Us

This section provides information on how to contact us with your questions, comments and suggestions.

4. VaultIC eLib Architecture

The VaultIC eLib has been designed to allow it to be deployed on a variety of platforms. The majority of the source code is common to all platforms. Changes to the supplied source code should only be required if the VaultIC eLib is being targeted at an embedded platform. The architecture of the VaultIC eLib can be seen in Figure 4-1.

Figure 4-1.

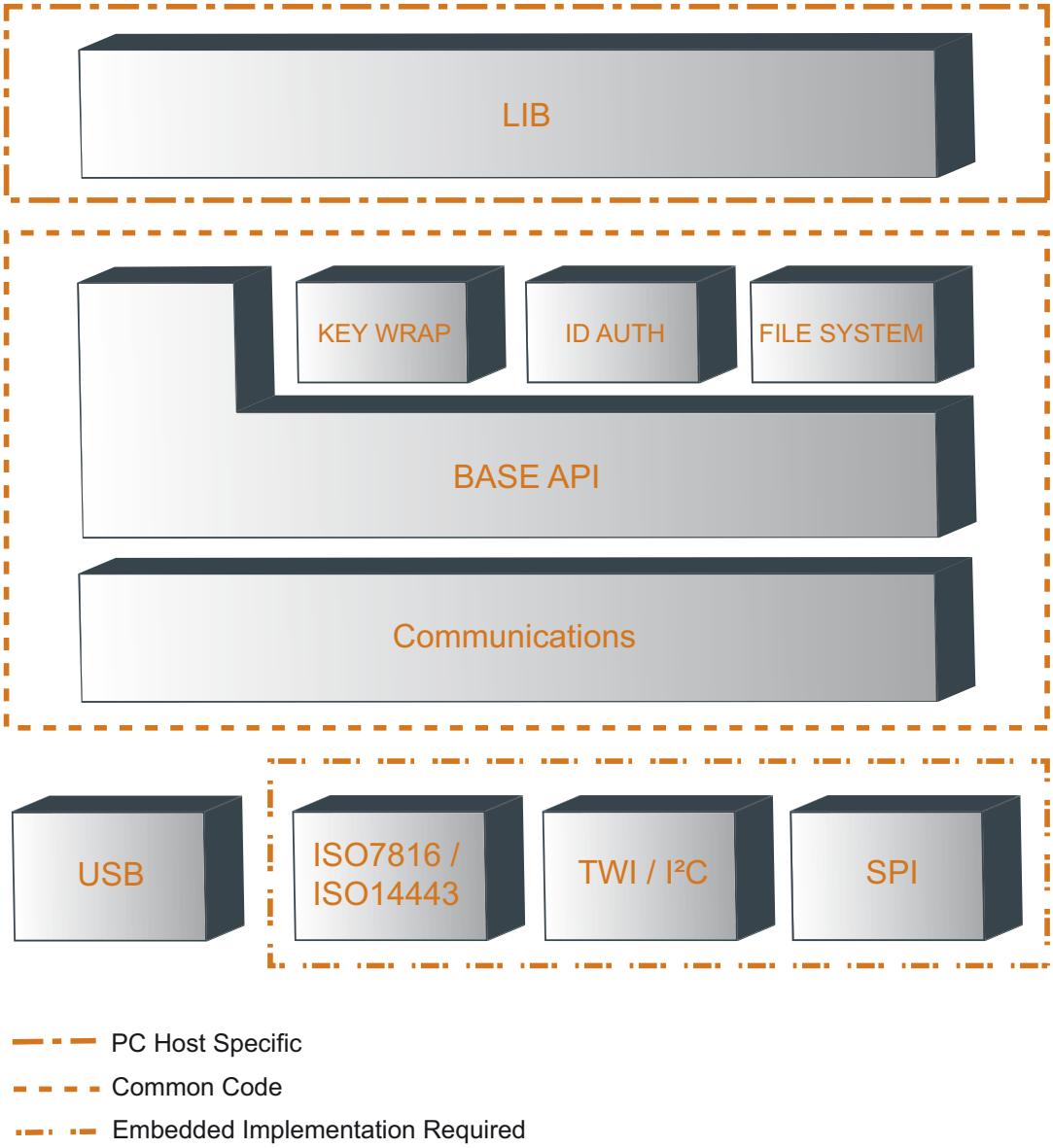
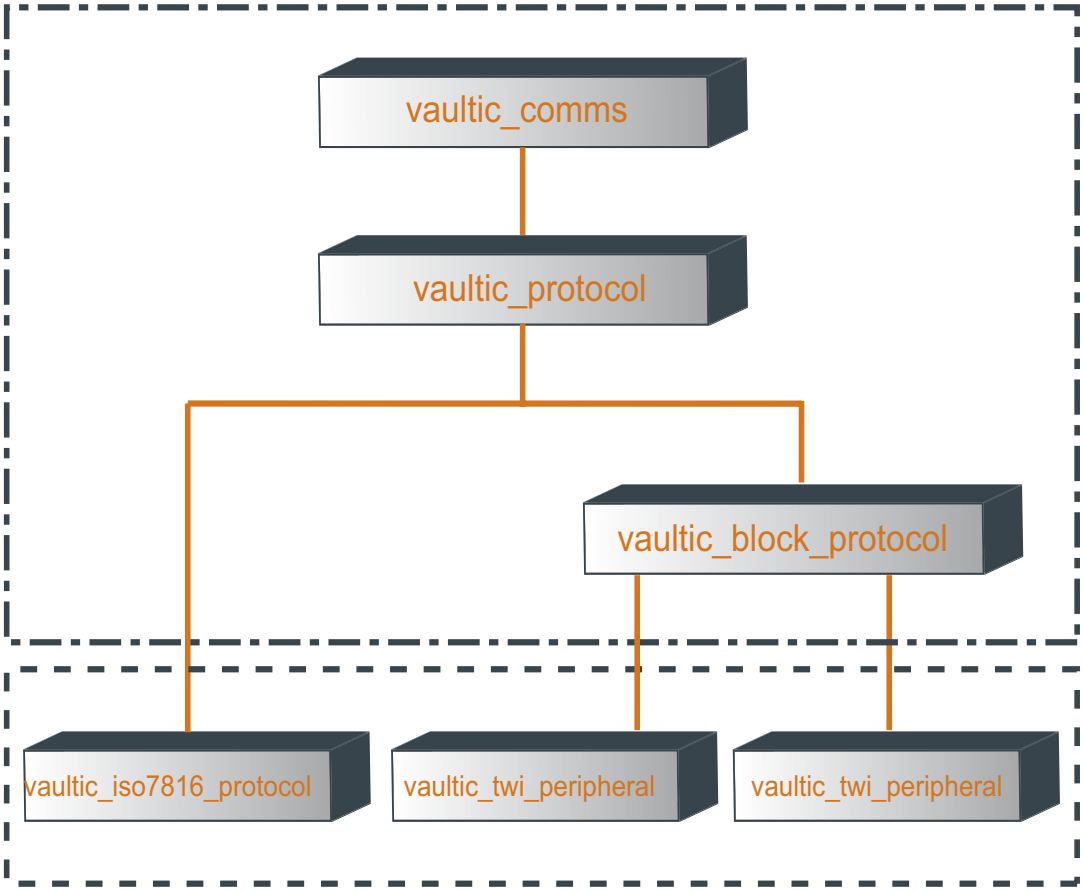


Figure 4-1 has links to the relevant sections of the document to better describe the specific areas of the VaultIC eLib. The “Common Code” can be configured to only include functionality that is required by the client application. Instructions on how to achieve this can be found in the Configuration & Customisation section of the document.

5. VaultIC eLib Communications Layer

This section gives a very brief overview of the structure of the communications layers of the VaultIC eLib. The structure of the communications layer can be seen in Figure 5-1.

Figure 5-1.



Common Code

Embedded Implementation Required

The module within the "Embedded Implementation Required" i.e. `vaultic_iso7816_protocol`, `vaultic_spi_peripheral`, `vaultic_twi_peripheral` map to the client stub interfaces detailed in the Client Stub Interfacessection.

The modules within the "Common Code" such as such as `vaultic_comms`, `vaultic_protocol` and `vaultic_block_protocol` section should not need to be modified. They contain comms logic code that is common to peripherals as shown in the diagram. The application code relies heavily on this structure and logic provided by those modules to ensure successful communications.

6. Getting Started

Use of the API is best understood by referring to the code samples and demonstration provided as part of the API package. These samples are described in this section.

6.1 Demonstration Code

There are demonstration code projects available for Microsoft Visual Studio 2005 & 2012, and Linux Netbeans. The code demonstrates how to load the VaultIC eLib library and obtain access to the base API, and service methods. The main purpose of the code is to illustrate how to personalize the device, authenticate using the strong authentication service and perform some simple cryptographic operations.

The following samples are available:

Table 6-1.

Sample Name	Purpose
Xxx_KeyDerivation	Demonstrates how to perform a key derivation
Xxx_KeyWrapping	Demonstrates how to perform key wrapping
Xxx_KeyWrapping_RSA	Demonstrates how to perform key wrapping using a RSA key
Xxx_Signature	Demonstrates how to perform a signature.
Xxx_SampleCode	Gives examples of calling all the base API functions and services
Xxx_Reinitialization	Demonstrates how to re-initialize a VaultIC device.

6.2 Deployment Options

The API is provided in ANSI 'C' compliant source code format. For ease of integration into OS based applications, the API is also provided in dynamically loadable library format.

- DLL - for Windows platforms
- SO - for Linux platforms
- dylib - for MacOS platform

6.2.1 Library Linkage

Application code links to the API library using the explicit linkage method (i.e. using LoadLibrary and GetProcAddress or its platform specific equivalent). There are library entry point functions which serve access to the API function by returning structures of function pointers as a means of implementing an interface-based development model.

6.2.2 Direct Source Code "Embedded" Linkage

As an alternative to library linkage, the API source code can be included directly in the build of the user application. The API functions can then be called directly.

7. VaultIC eLib Integration

This section describes the integration options for the VaultIC eLib. The source code is written in ANSI C. It is possible to integrate the *VaultIC eLib* for execution on multiple platforms. The platforms directly supported are: Windows, Linux, Mac OS, and embedded targets. The Windows, Linux and Mac OS libraries provided have additional dependencies to allow communication with the *VaultIC Security Module*. These dependencies are shown in Table 7-1.

Table 7-1.

	ISO 7816	SPI	TWI
Windows	Appropriate reader driver (For a windows platform application the USB CCID kernel driver must be installed)	Total Phase Aardvark or Total Phase Cheetah and appropriate dll/so	Total Phase Aardvark and appropriate dll/so
Linux	pcsc-lite library and CCID driver		
Mac OS			

Drivers and libraries for the Total Phase Aardvark and Cheetah Host Adapters can be found in:

http://www.totalphase.com/products/aardvark_i2cspi/

http://www.totalphase.com/products/cheetah_spi/

The PCSC lite library for Linux can be found in http://pcsclite.alioth.debian.org/

The *Smart Card Services software development kit* (SDK) should be included in your MAC OSX at **/System/Library/Frameworks/PCSC.framework**.

More information for the MACOS X can be found here.

7.1 PC Library Integration (All Platforms)

The libraries on Windows, Linux and MacOS export the following functions as the library entry points:

- VltInitLibrary
- VltCloseLibrary
- VltGetLibraryInfo
- VltGetApi
- VltGetAuth
- VltGetKeyWrapping
- VltGetFileSystem
- VltGetKeyWrapping
- VltGetCrc16

Caution

`VltGetLibraryInfo`, `VltGetApi`, `VltGetAuth`, `VltGetKeyWrapping`, and `VltGetFileSystem` depend upon a successful call of `VltInitLibrary`

The methods exposed by the library do not provide direct access to the VaultIC eLib, Base API methods, or service methods. The methods exposed by the library, provide the ability to config-

ure the VaultIC eLib, establish communications, and get the structures of function pointers for the Base API, and service methods.

7.2 Windows Integration

Windows integration for client applications is handled by loading a dynamic linked library -DLL.

The VLT_PLATFORM section describes how to configure the VaultIC eLib for a windows platform. The integrated development environment used for this library was Microsoft Visual Studio 2005 & 2012. The express edition of the IDE can be downloaded from :

http://www.microsoft.com/downloads/en/default.aspx

7.3 Linux Integration

Linux integration for client applications is handled by loading a dynamically linked shared object library – so.

The VLT_PLATFORM section describes how to configure the VaultIC eLib for a Linux platform. The integrated development environment used for this library was Netbeans, which is freely available from http://netbeans.org/downloads/index.html with its C/C++ plug-in.

7.4 Mac OS Integration

Mac OS integration for client applications is handled by loading a dynamically linked library –dylib.

The VLT_PLATFORM section describes how to configure the VaultIC eLib for a Mac OS platform. The integrated development environment used for this library was Netbeans, which is freely available from http://netbeans.org/downloads/index.html with its C/C++ plug-in.

7.5 Embedded Integration

Embedded integration for client applications is handled by incorporating the VaultIC eLib source code directly into the client embedded code project. The library is configured by calling the following method:

- VltApiInit

The VaultIC eLib source code does not provide a complete embedded solution. Some software development will be required to write target specific memory manipulation, timer, and communications routines. The Client Stub Interfaces section of this document provides a complete list of all of the interfaces that can have a target specific implementation. Not all of the interfaces are required to utilise the Base API methods.

Target specific implementation for the following interfaces is required as a minimum to achieve TWI communication end to end:

- vaultic_mem
- vaultic_timer_delay
- vaultic_twi_peripheral
- vaultic_spi_peripheral

The Configuration & Customisation section describes how to configure the VaultIC eLib for an embedded platform.

8. Configuration & Customisation

This section describes the configuration options for the VaultIC eLib.

The VaultIC eLib must be correctly configured to allow it to be built for the target platform. To do this certain configuration features must be correctly setup.

In addition to setting up the VaultIC eLib to be built for a specific target, certain modules may not be applicable to the deployment of the target. To reduce the memory footprint, the VaultIC eLib allows certain modules to be excluded from the build.

8.1 Configurable Features

To use the VaultIC eLib, the *vaultic_config.h* file must be correctly configured for the host target. The values that can be given to the configurable values can be found within the *vaultic_options.h* file.

8.1.1 VLT_ENDIANNESS

The `VLT_ENDIANNESS` specifies the endianess of the target hardware. The definition must be given one of two values:

1. `VLT_BIG_ENDIAN`
2. `VLT_LITTLE_ENDIAN`

8.1.2 VLT_PLATFORM

The `VLT_PLATFORM` specifies the target platform on which the VaultIC eLib is being deployed. The definition must be given one of four values:

1. `VLT_WINDOWS`
2. `VLT_LINUX`
3. `VLT_MAC_OS`
4. `VLT_EMBEDDED`

The value given should match the host target. If the host target is not running Windows, Linux or Mac OS, then `VLT_EMBEDDED` should be specified.

8.1.3 VAULT_IC_VERSION

The `VAULT_IC_VERSION` specifies the version the VaultIC Security Module firmware. The version specified must match the version on the supplied VaultIC Security Module or correct behaviour cannot be guaranteed. The definition must be given one of the following values :

1. `VAULTIC_VERSION_1_0_X`
2. `VAULTIC_VERSION_1_2_0`
3. `VAULTIC_VERSION_1_2_1`

8.2 Build Customisation

In order to ensure that the VaultIC Security Module uses the least amount of memory possible, certain modules and data structures can be excluded from the build. The *vaultic_config.h* file contains a list of definitions for the modules that can be excluded. These features are listed in Table 8-1.

Table 8-1.

Feature	#define	Further Information
Secret Keys	VLT_ENABLE_KEY_SECRET	Key Types
HOTP Keys	VLT_ENABLE_KEY_HOTP	
TOTP Keys	VLT_ENABLE_KEY_TOTP	
RSA Keys	VLT_ENABLE_KEY_RSA	
DSA Keys	VLT_ENABLE_KEY_DSA	
ECDSA Keys	VLT_ENABLE_KEY_ECDSA	
Host/Device ID	VLT_ENABLE_KEY_HOST_DEVICE_ID	
Secure Channel 02	VLT_ENABLE_SCP02	Secure Channel 02
Secure Channel 03	VLT_ENABLE_SCP03	Secure Channel 03
Microsoft Smart card Minidriver	VLT_ENABLE_MS_AUTH	Microsoft Smart Card Minidriver
Cipher DES	VLT_ENABLE_CIPHER_DES	Cipher DES
Cipher TDES	VLT_ENABLE_CIPHER_TDES	Cipher TDES
Cipher AES	VLT_ENABLE_CIPHER_AES	Cipher AES
Fast CRC16 CCITT	VLT_ENABLE_FAST_CRC16CCIT	Fast CRC Calculations
ISO 7816 Protocol	VLT_ENABLE_ISO7816	ISO7816 Protocol
Block Protocol	VLT_ENABLE_BLOCK_PROTOCOL	Block Protocol
TWI Peripheral	VLT_ENABLE_TWI	TWI Peripheral
SPI Peripheral	VLT_ENABLE_SPI	SPI Peripheral
Key Wrapping	VLT_ENABLE_KEY_WRAPPING	Key Wrapping
Identity Authentication	VLT_ENABLE_IDENTITY_AUTH	Identity Authentication
File System	VLT_ENABLE_FILE_SYSTEM	File System
Aardvark Suppress Error	VLT_ENABLE_AARDVK_SPPRSS_ERR	Aardvark Suppress Error
Aardvark adapter	VLT_ENABLE_AARDVARK	Aardvark adpater
Cheetah adapter	VLT_ENABLE_CHEETAH	Cheetah adapter

To include the feature within the build the feature should be defined with VLT_ENABLE, and to disable the feature it should be defined as VLT_DISABLE. If the feature is disabled, any method calls within the feature will return EMETHODNOTSUPPORTED.

8.2.1 Key Types

The data structures and code associated with the various types of keys can be removed from the build by correctly configuring the *vaultic_config.h* to only build in the key types appropriate to the client application.

Dependencies: None

8.2.2 Secure Channel 02

The Secure Channel 02 module should be removed from the build if authentication using Secure Channel 02 is not required. This module depends on the Cipher DES and Cipher TDES modules being included in the build.

Dependencies: Cipher DES, Cipher TDES

8.2.3 Secure Channel 03

The Secure Channel 03 module should be removed from the build if authentication using Secure Channel 03 is not required. This module depends on the Cipher AES module being included in the build.

Dependencies: Cipher AES

8.2.4 Microsoft Smart Card Minidriver

The Microsoft Smart card minidriver module should be removed from the build if authentication using Microsoft Smart card minidriver is not required. This module depends on the Cipher DES and Cipher TDES modules being included in the build.

Dependencies: Cipher DES, Cipher TDES

8.2.5 Cipher DES

The implementation of DES given is intended as an example only. The code has not been optimised. It is intended to allow a working solution to be put in place with minimal effort. If the host device has DES hardware the module should be updated to make use of that rather than rely on the software implementation.

Dependencies: None

8.2.6 Cipher TDES

The implementation of TDES given is intended as an example only. The code has not been optimised. It is intended to allow a working solution to be put in place with minimal effort. If the host device has DES hardware the module should be updated to make use of that rather than rely on the software implementation.

Dependencies: None

8.2.7 Cipher AES

The implementation of AES given is intended as an example only. The code has not been optimised. It is intended to allow a working solution to be put in place with minimal effort. If the host device has AES hardware the module should be updated to make use of that rather than rely on the software implementation.

Dependencies: None

8.2.8 Fast CRC Calculations

CRC CCITT calculations are used within the VaultIC Security Module. A faster implementation of this is achieved by using a lookup table. As a consequence this requires data space. If this is an issue then s slightly slower, but smaller code implementation can be used by setting this definition as `VLT_DISABLE`.

Dependencies: None

8.2.9 ISO7816 Protocol

The implementation to allow communications with the VaultIC Security Module must be provided by the host application. This is only applicable if the host intends to communicate with the VaultIC Security Module via ISO 7816. If not then the `VLT_ENABLE_ISO7816` define should be set to `VLT_DISABLE`.

Dependencies: None

8.2.10 Block Protocol

The block protocol sits above both the SPI and TWI communications peripherals. If either of these methods of communication are to be used the `VLT_ENABLE_BLOCK_PROTOCOL` define must be set to `VLT_ENABLE`.

Dependencies: TWI Peripheral, SPI Peripheral

8.2.11 TWI Peripheral

The implementation to allow communications with the VaultIC Security Module must be provided by the host application. This is only applicable if the host intends to communicate with the VaultIC Security Module via TWI. If not then the `VLT_ENABLE_TWI` define should be set to `VLT_DISABLE`.

Dependencies: Block Protocol

8.2.12 SPI Peripheral

The implementation to allow communications with the VaultIC Security Module must be provided by the host application. This is only applicable if the host intends to communicate with the VaultIC Security Module via SPI. If not then the `VLT_ENABLE_SPI` define should be set to `VLT_DISABLE`.

Dependencies: Block Protocol

8.2.13 Key Wrapping

The key wrapping service provides a means of securely writing and reading keys to and from the VaultIC Security Module. This service is an alternative to using Secure Channel 02 or Secure Channel 03, and cannot be used if either of these authentication methods are being used. To disable the service the `VLT_ENABLE_KEY_WRAPPING` define should be set to `VLT_DISABLE`.

Dependencies: Cipher DES, Cipher TDES, Cipher AES

8.2.14 Identity Authentication

The identity authentication service provides a simple way to authenticate with the VaultIC Security Module using Secure Channel 02, Secure Channel 03 or Microsoft Smart Card Minidriver. The method of choice must be included in the build using the appropriate define. If none of these authentication methods are to be used the `VLT_ENABLE_IDENTITY_AUTH` define should be set to `VLT_DISABLE`.

Dependencies: Secure Channel 02, Secure Channel 03, Microsoft Smart Card Minidriver

8.2.15 File System

The file system service is intended to provide a straightforward means of accessing the file system on the VaultIC Security Module. The Base API methods can be used as an alternative, but this is not recommended. To disable the file system service the `VLT_ENABLE_FILE_SYSTEM` define should be set to `VLT_DISABLE`.

Dependencies: None

8.2.16 Aardvark Suppress Error

The Total Phase Aardvark adapter occasionally returns an error when attempting to send data (`AA_SPI_WRITE_ERROR` returned from aa_spi_write). The data has actually been sent and communications can continue. If the `VLT_ENABLE_AARDVK_SPPRSS_ERR` is set to `VLT_ENABLE` this error will be ignored and a `VLT_OK` status will be reported. If it is set to `VLT_DISABLE` the specific error message `EAARSNDFAIL` will be reported.

Dependencies: Aardvark adpater

8.2.17 Aardvark adpater

The Total Phase Aardvark adapter module should be removed from the build if Aardvark adpater is not required. **Dependencies**: None

8.2.18 Cheetah adapter

The Total Phase Cheetah adapter module should be removed from the build if Cheetah adpater is not required.

Dependencies: None

8.3 Communication mode

These three parameters allow to select which communication mode will be supported by the library.

```
#define VLT_ENABLE_ISO7816          VLT_ENABLE
#define VLT_ENABLE_TWI              VLT_DISABLE
#define VLT_ENABLE_SPI              VLT_DISABLE
```

8.4 Multi-slot support

Some applications using PC/SC offer possibilities to use more than one card simultaneously. To be compatible with this feature, the following declaration must be set to VLT_ENABLE.

```
#define VLT_ENABLE_MULTI_SLOT       VLT_ENABLE
```

9. VaultIC eLib Methods

This section provides information on the methods supported by the VaultIC eLib.

9.1 Overview

The API functions are organised into the following functional groups:

Library Methods

These functions are the methods exported from the dynamically loadable library implementation.

Base API

This section describes the Base API methods which supply a one to one mapping with the commands exposed by the VaultIC Security Module. As an example the VltSubmitPassword method provides the interface to the Submit Password command documented in the VaultIC Security Module datasheet [R1].

9.1.1 Error & Status Code

All functions return a status code of type `VLT_STS`. A return value of `VLT_OK` indicates successful completion. Any other return value indicates an execution error. Status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib library while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

9.2 Library Methods

These are the methods exported from the library implementation (DLL, dylib or SO) of the API. These methods are the access points for all API functions and services when linking an application to the library implementation. The sample code demonstrates use of these methods.

9.2.1 VltInitLibrary

The *VltInitLibrary* method is responsible for initialising the VaultIC eLib library

Syntax

```
VLT_STS VltInitLibrary(
  _in VLT_INIT_COMMS_PARAMS* pInitCommsParams
);
```

Parameters

`pInitCommsParams [in]`

The comms init parameters structure

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib library while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The *VltInitLibrary* method is used to initialise the host PC library, by establishing communications with the VaultIC Security Module.

Caution

There are different means of communication with the VaultIC Security Module, ISO7816 T0, ISO7816 T1, ISO7816 USB, SPI and TWI. Not all communication methods will be available on the entire VaultIC Security Module product line. Ensure your selected VaultIC Security Module product supports the desired method of communication.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.2.2 VltCloseLibrary

The *VltCloseLibrary* method is responsible for closing the VaultIC eLib library

Syntax

```
VLT_STS VltCloseLibrary( void );
```

Parameters

None

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib library while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

The *VltCloseLibrary* method is used to release handles to any external reader drivers. It is vital this method is called when exiting the host application.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.2.3 VltGetLibraryInfo

The *VltGetLibraryInfo* method populates a provided `VLT_LIBRARY_INFO` structure with information about the configuration of the library.

Syntax

```
VLT_STS VltGetLibraryInfo(
  _out VLT_LIBRARY_INFO* pLibraryInfo
);
```

Parameters

`pLibraryInfo [out]`

The library info structure

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib library while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

The *VltGetLibraryInfo* method can only be used once the VltInitLibrary method has been successfully called.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.2.4 VltGetApi

The *VltGetApi* method is responsible for providing a means of calling the VaultIC eLib methods. It does this by providing a pointer to a `VAULTIC_API` structure.

Syntax

```
VAULTIC_API* VltGetApi( void );
```

Parameters

None

Return Value

Upon successful completion a `VAULTIC_API` structure pointer will be returned, check the pointer is valid. The `VAULTIC_API` structure contains function pointers.

Note

The *VltGetApi* method is used to obtain the structure of Base API function pointers. Use the returned structure to call the Base API methods.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.2.5 VltGetAuth

The *VltGetAuth* method is responsible for providing a means of calling the VaultIC eLib authentication service methods. It does this by providing a pointer to a `VAULTIC_AUTH` structure.

Syntax

```
VAULTIC_AUTH* VltGetAuth( void );
```

Parameters

None

Return Value

Upon successful completion a `VAULTIC_AUTH` structure pointer will be returned, check the pointer is valid. The `VAULTIC_AUTH` structure contains function pointers for the authentication service interface methods.

Note

The *VltGetAuth* method is used to obtain the structure of authentication service function pointers. Use the returned structure to call the authentication service methods.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.2.6 VltGetKeyWrapping

The *VltGetKeyWrapping* method is responsible for providing a means of calling the VaultIC eLib key wrapping service methods. It does this by providing a pointer to a `VAULTIC_KEY_WRAPPING` structure.

Syntax

```
VAULTIC_KEY_WRAPPING* VltGetKeyWrapping( void );
```

Parameters

None

Return Value

Upon successful completion a `VAULTIC_KEY_WRAPPING` structure pointer will be returned, check the pointer is valid. The `VAULTIC_KEY_WRAPPING` structure contains function pointers for the key wrapping service interface methods.

Note

The *VltGetKeyWrapping* method is used to obtain the structure of key wrapping service function pointers. Use the returned structure to call the key wrapping service methods.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.2.7 VltGetFileSystem

The *VltGetFileSystem* method is responsible for providing a means of calling the VaultIC eLib file system service methods. It does this by providing a pointer to a `VAULTIC_FILE_SYSTEM` structure.

Syntax

```
VAULTIC_FILE_SYSTEM* VltGetFileSystem( void );
```

Parameters

None

Return Value

Upon successful completion a `VAULTIC_FILE_SYSTEM` structure pointer will be returned, check the pointer is valid. The `VAULTIC_FILE_SYSTEM` structure contains function pointers for the key wrapping service interface methods.

Note

The *VltGetFileSystem* method is used to obtain the structure of key wrapping service function pointers. Use the returned structure to call the file system service methods.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.2.8 VltFindDevices

The *VltFindDevices* method provides the caller an XML based list of all supported reader hardware – devices currently connected to the system. The caller can then parse the data in order to use when calling the VltApiInit method.

Syntax

```
VLT_STS VltFindDevices(
  _in VLT_PU32 pSize,
  _out VLT_PU8 pXmlReaderString
);
```

Parameters

`pSize [in,out]`

In: the size of the xml string buffer.

Out: the actual size of the xml string.

`pXmlReaderString [out]`

Pointer to a character buffer.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that haveoriginated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The *VltFindDevices* method is a helper function that can be called before calling VltInit, the xml string will contain a list of all supported reader devices connected to the host system. The peripheral string can be used to obtain a serial number, or reader name, which are elements of the `VLT_INIT_COMMS_PARAMS` structure passed to VltApiInit method.

Caution

This method is not supported in the embedded environments. Also this method depends on having the Total Phase aardvark.dll/so binary located in the same path as your executable, or in the standard operating system library search path.

Example string

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<devices>
     <interface type="pcsc">
          <peripheral idx="00">Gemplus USB Smart Card Reader
          0</peripheral>
     </interface>
     <interface type="aardvark">
          <peripheral idx="00">2237366715</peripheral>
     </interface>
     <interface type="cheetah">
          <peripheral idx="00">1363924836</peripheral>
     </interface>
</devices>
```

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.2.9 VltGetCrc16

The VltGetCrc16 method provides the caller the ability to generate a CRC16CCITT value from a stream of bytes.

Syntax

```
VLT_STS VltCrc16(
  VLT_U16 u16Crc,
  const VLT_U8 *pu8Block,
  VLT_U16 u16Length
);
```

Parameters

`u16Crc [in]`

The initial value of the CRC:

– VLT_CRC16_CCITT_INIT_Fs

– VLT_CRC16_CCITT_INIT_0s

`pu8Block [out]`

Pointer to the byte stream to CRC.

`u16Length [in]`

The number of bytes to CRC.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The VltCrc16 method can be used to calculate CRC16 CCITT values. CRC16 CCITT values are required to use the Key Wrapping Service methods *VltWrapKey*, *VltUnwrapKey*, the Base Api methods *VltPutKey*, and *VltReadKey*.

Example

```
VAULTIC_CRC16* pTheCrc16;
if( NULL == ( pTheCrc16 = VltGetCrc16() ) )
{
return( VLT_FAIL );
}
VLT_U16 Crc16 = VLT_CRC16_CCITT_INIT_Fs;
VLT_U8 pBinaryBlock[32] =
{
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
```

```
};
// Calculate the crc16 of the first 10 bytes of the binary block
// array.
Crc16 = pTheCrc16->VltCrc16( &Crc16, &pBinaryBlock[0], 10 );
// Calculate the crc16 of the remaining 22 bytes of the binary block
// array. On return from the call the Crc16 variable will hold the
// CRC16 CCITT calculation of the entire 32 byte binary block.
Crc16 = VLT_CRC16_CCITT_INIT_Fs;
Crc16 = pTheCrc16->VltCrc16( &Crc16, &pBinaryBlock[10], 22 );
return( VLT_OK );
```

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.3 Embedded Host Configuration Methods

9.3.1 VltApiInit

The *VltApiInit* method provides an initialisation entry point to the entire VaultIC eLib Upon successful completion a number of internal system resources would be allocated and used by subsequent calls, these resources will remain in use until a call to the VltApiClose method is made. If the call to the *VltApiInit* is unsuccessful, calls to the rest of the VaultIC eLib methods will produced undefined results.

Syntax

```
VLT_STS VltApiInit(
  _in VLT_INIT_COMMS_PARAMS* pInitCommsParams
);
```

Parameters

`pInitCommsParams [in]`

The comms init parameters structure

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The *VltApiInit* method is a helper function that must be called to setup and initialize communications with the VaultIC eLib. Appendix H in the VaultIC datasheet [R1] lists the required wait times for other device modes.

Example

```
// The values used in this example are not the recommended settings,
// contact your field applications engineer or applications for assistance.
#define SPI_PARAMS(x) \
x.Params.VltBlockProtocolParams.VltPeripheralParams.PeriphParams.VltS
piParams
#define BLOCK_PARAMS (x) x.Params.VltBlockProtocolParams
```

```
VLT_INIT_COMMS_PARAMS commsParams;
VLT_STS status = VLT_FAIL;
//
// Setup the comms init structure for SPI.
//
commsParams.u8CommsProtocol = VLT_SPI_COMMS;
BLOCK_PARAMS(commsParams).u16BitRate = 125;
BLOCK_PARAMS(commsParams).u8CheckSumMode = BLK_PTCL_CHECKSUM_SUM8;
SPI_PARAMS(commsParams).u32FstPollByteDelay = 1000;
SPI_PARAMS(commsParams).u32IntPollByteDelay = 10000;
SPI_PARAMS(commsParams).u32IntByteDelay = 1;
SPI_PARAMS(commsParams).u32PollMaxRetries = 0x0FFF;
BLOCK_PARAMS(commsParams).u16msSelfTestDelay = 2000;
BLOCK_PARAMS(commsParams).u32AfterHdrDelay = 1000;
BLOCK_PARAMS(commsParams).u32InterBlkDelay = 1000;
if( VLT_OK != ( status = VltApiInit( &commsParams ) ) )
{
return( -2 );
}
```

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltApiClose

9.3.2 VltApiClose

The *VltApiClose* method provides a finalisation entry point to the entire VaultIC eLib library.

Syntax

```
VLT_STS VltApiClose( void );
```

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltApiInit

9.3.3 VltCardEvent

The *VltCardEvent* method provides provides the current status of a smart card in a reader.

Syntax

```
VLT_STS VltCardEvent(
  _in VLT_PU8 pu8ReaderName,
```

```
  _inDWORD dwTimeout,
  _out PDWORD pdwEventState
);
```

Parameters

`pu8ReaderName [in]`

Buffer which contains the reader name to be monitored.

`dwTimeout [in]`

The maximum amount of time, in milliseconds, to wait for an action. A value of zero causes the function to return immediately.

A value of INFINITE(`0xFFFFFFFF`) causes this function never to time out.

`pdwEventState [out]`

Current state of the smart card in the reader.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution This method is blocking until a card event happens or that the timeout expires.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.3.4 VltSelectCard

The *VltSelectCard* method allow to select a specific card .

Syntax

```
VLT_STS VltSelectCard(
  _inout SCARDHANDLE hScard,
  _inout SCARDCONTEXT hCtx,
  _in DWORD dwProtocol
);
```

Parameters

`hScard [in,out]`

Smart card handle value

`hCtx [in,out]`

Smart card context value

`dwProtocol [in]`

Protocol to use value

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4 Base API

This section describes the Base API methods which supply a one to one mapping with the commands exposed by the VaultIC Security Module. As an example the *VltSubmitPassword* method provides the interface to the Submit Password command documented in the VaultIC Security Module datasheet [R1].

9.4.1 VltSubmitPassword

The *VltSubmitPassword* method authenticates a user using a password.

Syntax

```
VLT_STS VltSubmitPassword(
  _in VLT_U8 u8UserID,
  _in VLT_U8 u8RoleID,
  _in VLT_U8 u8PasswordLength,
  _in const VLT_U8 *pu8Password
);
```

Parameters

`u8UserID [in]`

Operator ID. Possible values are:

- VLT_USER0
- VLT_USER1
- VLT_USER2
- VLT_USER3
- VLT_USER4
- VLT_USER5
- VLT_USER6
- VLT_USER7

`u8RoleID [in]`

Role ID. Possible values are:

- VLT_APPROVED_USER
- VLT_NON_APPROVED_USER
- VLT_MANUFACTURER
- VLT_ADMINISTRATOR
- VLT_EVERYONE

`u8PasswordLength [in]`

Password Length. (0..32)

`pu8Password [in]`

The password value.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

The `u8UserID` must already exist within the VaultIC Security Module, or an error will be returned.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltCancelAuthentication

9.4.2 VltInitializeUpdate

The *VltInitializeUpdate* method performs Secure Channel initiation, or Microsoft Smart Card Minidriver authentication.

Syntax

```
VLT_STS VltInitializeUpdate(
  _in VLT_U8 u8UserID,
  _in VLT_U8 u8RoleID,
  _in VLT_U8 u8HostChallengeLength,
  _in const VLT_U8 *pu8HostChallenge,
  _out VLT_INIT_UPDATE *pRespData
);
```

Parameters

`u8UserID [in]`

Operator ID. Possible values are:

– VLT_USER0
– VLT_USER1
– VLT_USER2
– VLT_USER3
– VLT_USER4
– VLT_USER5
– VLT_USER6
– VLT_USER7

`u8RoleID [in]`

Role ID. Possible values are:

– VLT_APPROVED_USER
– VLT_NON_APPROVED_USER

– VLT_MANUFACTURER
– VLT_ADMINISTRATOR
– VLT_EVERYONE

`u8HostChallengeLength [in]`

The length of the host challenge. This should be set to 0 for Microsoft Smart Card Minidriver Authentication.

`pu8HostChallenge [in]`

The host challenge. This should be NULL for Microsoft Smart Card Minidriver Authentication.The length of the host challenge.

`pRespData` `[out]`

`VLT_INIT_UPDATE` structure to receive SCP02, SCP03 response data, or Microsoft Smart Card Minidriver device challenge.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command is used in conjunction with the VltExternalAuthenticate command to initiate a Microsoft Smart Card Minidriver authentication. The VaultIC Security Module supports using this command to initiate secure channel authentication. This is not recommended using the Base Api.

It is possible to establish a secure channel using this method. The mechanics of he active secure channel will not be processed correctly for subsequent Base API commands, and result in the channel being closed. To overcome this issue, the Identity Authentication Service has been provided for secure channel, and Microsoft Smart Card Minidriver authentication. Using the service will result in Base API commands being sent over the secure channel.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltExternalAuthenticate

9.4.3 VltExternalAuthenticate

The *VltExternalAuthenticate* method completes a Microsoft Smart Card Minidriver or secure channel authentication.

Syntax

```
VLT_STS VltExternalAuthenticate(
  _in VLT_U8 u8AuthLevel,
  _in VLT_U8 u8ChannelLevel,
  _in VLT_U8 u8CryptogramLength,
  _in const VLT_U8 *pu8Cryptogram
);
```

Parameters

`u8AuthLevel [in]`

The authentication level, possible values are :

– VLT_LOGIN_SCP02

– VLT_LOGIN_SCP03

– VLT_LOGIN_MS

`u8ChannelLevel [in]`

Security level, possible values are:

– VLT_NO_CHANNEL

– VLT_CMAC

– VLT_CMAC_CENC

– VLT_CMAC_RMAC

– VLT_CMAC_CENC_RMAC

– VLT_CMAC_CENC_RMAC_RENC

`u8CryptogramLength [in]`

The length of the host cryptogram.

`pu8Cryptogram [in]`

The host cryptogram.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command is used in conjunction with the VltInitializeUpdate command to complete a Microsoft Card Minidriver authentication. The VaultIC Security Module supports using this command to complete a secure channel authentication. This is not recommended using the Base API.

It is possible to establish a secure channel using this method. The mechanics of the active secure channel will not be processed correctly for subsequent Base API commands, and result in the channel being closed. To overcome this issue, the **Identity Authentication Service** has been provided for secure channel, and Microsoft Smart Card Minidriver authentication. Using the service will result in Base API commands being sent over the secure channel.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltInitializeUpdate

9.4.4 VltManageAuthenticationData

The *VltManageAuthenticationData* method updates a user account for the VaultIC Security Module.

Syntax

```
VLT_STS VltManageAuthenticationData(
  _in const VLT_MANAGE_AUTH_DATA *pAuthSetup
);
```

Parameters

`pAuthSetup [in]`

Manage Authentication Data data structure

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Refer to the VaultIC Security Module Generic Data Sheet for a description of the limitations of when this method can be used and by whom.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.5 VltGetAuthenticationInfo

The *VltGetAuthenticationInfo* method gets the authentication information about an operator.

Syntax

```
VLT_STS VltGetAuthenticationInfo(
  _in VLT_U8 u8UserID,
  _out VLT_AUTH_INFO pRespData
);
```

Parameters

`u8UserID [in]`

Operator ID. Possible values are:

- VLT_USER0
- VLT_USER1
- VLT_USER2
- VLT_USER3
- VLT_USER4
- VLT_USER5
- VLT_USER6
- VLT_USER7

`pRespData [out]`

Authentication Information data structure

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.6 VltCancelAuthentication

The *VltCancelAuthentication* method resets the VaultIC Security Module authentication state, and closes any established secure channel.

Syntax

```
VLT_STS VltCancelAuthetication( void );
```

Parameters

None

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Logs out the currently authenticated user, and resets the state of an established secure channel.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.7 VltGetChallenge

The *VltGetChallenge* method initiates a generic Host Unilateral Authentication to the VaultIC Security Module.

Syntax

```
VLT_STS VltGetChallenge(
  _in const VLT_GENERIC_AUTH_SETUP_DATA *pAuthParameters,
  _out VLT_PU8 pu8DeviceChallenge
);
```

Parameters

`pAuthParameters [in]`

Generic Strong Authentication parameters

`pAuthParameters [out]`

Device Challenge (Cd)

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.8 VltGenericInternalAuthenticate

The *VltGenericInternalAuthenticate* method perform a generic Device Unilateral Authentication of the VaultIC Security Module, or generic Mutual authentication with the host.

Syntax

```
VLT_STS VltGenericInternalAuthenticate(
  _in const VLT_GENERIC_AUTH_SETUP_DATA *pAuthParameters,
  _in const VLT_U8 *pu8HostChallenge,
  _out VLT_PU8 pu8DeviceChallenge,
  _inout VLT_PU16 pu16SignatureLength,
  _out VLT_PU8 pu8Signature
);
```

Parameters

`pAuthParameters [in]`

Generic Strong Authentication parameters

`pu8HostChallenge [in]`

Host Challenge (Ch)

`pu8DeviceChallenge [out]`

Buffer to receive VaultIC Security Module Device Challenge (Cd)

`pu16SignatureLength [in, out]`

On entry this holds the maximum size of the signature buffer. On exit it is set to the amount of signature buffer used.

`pu8Signature [out]`

Buffer to receive VaultIC Security Module signature SIGNk.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltInitializeAlgorithm

VltGenericExternalAuthenticate

9.4.9 VltGenericExternalAuthenticate

The *VltGenericExternalAuthenticate* method is used after VltGenericInternalAuthenticate to complete a generic Mutual Authentication.

Syntax

```
VLT_STS VltGenericExternalAuthenticate(
  _in VLT_U8 u8HostChallengeLength,
  _in const VLT_U8 *pu8HostChallenge,
  _in VLT_U16 u16HostSignatureLength,
  _in const VLT_U8 *pu8HostSignature
);
```

Parameters

`u8HostChallengeLength [in]`

Host Challenge (Ch) length

`pu8HostChallenge [in]`

Host Challenge (Ch)

`u16HostSignatureLength [in]`

Host signature (SIGNk) length

`pu8HostSignature [in]`

Host signature (SIGNk)

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command may be used after a VltGenericInternalAuthenticate to complete a generic Mutual Authentication protocol, or can be sent after a VltGetChallenge command to complete a generic Host Unilateral Authentication to the VaultIC Security Module. This command gets a signature from a host and a host challenge Ch, and attempts verification. It returns the result of this verification. The applicable security parameters object and key index are `VLT_SELECT` in a prior call.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltGenericInternalAuthenticate

VltGetChallenge

9.4.10 VltInitializeAlgorithm

The *VltInitializeAlgorithm* method initializes a cryptographic algorithm, a cryptographic key and conditionally some specific algorithm parameters for subsequent cryptographic services.

Syntax

```
VLT_STS VltInitializeAlgorithm(
  _in VLT_U8 u8KeyGroup,
  _in VLT_U8 u8KeyIndex,
  _in VLT_U8 u8Mode,
  _out VLT_ALGO_PARAMS *pAlgorithm
);
```

Parameters

`u8KeyGroup [in]`

Key Group index. Shall be zero for digest initialization.

`u8KeyIndex [in]`

Key index. Shall be zero for digest initialization.

`u8Mode [in]`

Mode of operation for subsequent commands. Possible values:

– VLT_ENCRYPT_MODE
– VLT_DECRYPT_MODE
– VLT_GENERIC_STRONG_AUTH_MODE
– VLT_DIGEST_MODE
– VLT_WRAP_KEY_MODE
– VLT_UNWRAP_KEY_MODE
– VLT_SIGN_MODE
– VLT_VERIFY_MODE

`pAlgorithm [out]`

Algorithm parameters.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The key is identified by the key group index and the key index given by the u8KeyGroup and u8KeyIndex parameters. The key is fetched from the Internal Key Ring. The command data field optionally carries specific algorithm parameters. They define all algorithm parameters to be applied to any subsequent cryptographic operations.

Caution

The logged-in user must have the Execute privilege on the involved key file. The *VltInitializeAlgorithm* shall be sent before the following cryptographic services:

- VltGenerateSignature
- VltVerifySignature
- VltComputeMessageDigest
- VltEncrypt
- VltDecrypt
- VltGetChallenge
- VltGenericInternalAuthenticate

The *VltInitializeAlgorithm* command **may** be sent before the following cryptographic services:

– `VltUninitializeAlgorithm` for key unwrapping (key sent encrypted to the device)
– `VltReadKey` for key wrapping (key sent encrypted from the device)

Note

Any other command will discard the algorithm, wipe its parameters and unload the key. Command Chaining is allowed for the underlying cryptographic service. As soon as the service type changes, the algorithm being initialized is discarded. Algorithm parameters cannot be shared between different cryptographic operations.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltEncrypt

VltDecrypt

VltGenerateSignature

VltVerifySignature

VltComputeMessageDigest

VltGetChallenge

VltGenericInternalAuthenticate

VltUninitializeAlgorithm

VltReadKey

9.4.11 VltUninitializeAlgorithm

The *VltInitializeAlgorithm* method initializes a cryptographic algorithm, a cryptographic key and conditionally some specific algorithm parameters for subsequent cryptographic services.

Syntax

```
VLT_STS VltUninitializeAlgorithm( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltInitializeAlgorithm

9.4.12 VltPutKey

The *VltPutKey* method imports a key into the internal key ring.

Syntax

```
VLT_STS VltPutKey(
  _in VLT_U8 u8KeyGroup,
  _in VLT_U8 u8KeyIndex,
  _in const VLT_FILE_PRIVILEGES *pKeyFilePrivileges,
  _in const VLT_KEY_OBJECT *pKeyObj
);
```

Parameters

`u8KeyGroup [in]`

Key Group index.

`u8KeyIndex [in]`

Key index.

`pKeyFilePrivileges [in]`

Key file Access Conditions. The logged-in user must grant its own user ID write permission on the key file.

`pKeyObj [in]`

Key object

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The key is identified by the key group index and they key index given by u8KeyGroup and u8KeyIndex. The key file holding the imported key is automatically created and owned by the currently logged-in user.

Note

The key can either be transported in plaintext or encrypted with a cipher available. If key unwrapping is required, the VltInitializeAlgorithm command shall be previously sent with the algorithm identifier set with a valid Key Transport Scheme identifier.

Note

The put key command shall be sent two times to download a key pair.

Caution

In Approved Mode of operation, secret or private keys cannot be downloaded in plaintext. A secure channel with encrypted command data field C-ENC level shall be opened or a key wrapping mechanism shall be used.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltReadKey

VltDeleteKey

VltInitializeAlgorithm

9.4.13 VltReadKey

The *VltReadKey* method exports a key from the internal key ring.

Syntax

```
VLT_STS VltReadKey(
  _in VLT_U8 u8KeyGroup,
  _in VLT_U8 u8KeyIndex,
  _in const VLT_KEY_OBJECT *pKeyObj
);
```

Parameters

`u8KeyGroup [in]`

Key Group index.

`u8KeyIndex [in]`

Key index.

`pKeyObj [in]`

Key object

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The key is identified by the key group index and the key index given by u8KeyGroup and u8KeyIndex.

Note

The key can either be transported in plaintext or encrypted by any cipher vailable. If key wrapping is required, the VltInitializeAlgorithm command shall be previously sent with the algorithm identifier set with a valid Key Transport Scheme identifier.

Note

The logged-in user must have the read privilege on the involved key file.

Caution

In Approved Mode of operation, secret or private keys cannot be extracted in plaintext. A secure channel with encrypted response data field R-ENC level shall be opened or a key unwrapping mechanism shall be used.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltUninitializeAlgorithm

VltDeleteKey

VltInitializeAlgorithm

9.4.14 VltDeleteKey

The *VltDeleteKey* method deletes a key from the internal key ring.

Syntax

```
VLT_STS VltDeleteKey(
  _in VLT_U8 u8KeyGroup,
  _in VLT_U8 u8KeyIndex,
);
```

Parameters

`u8KeyGroup [in]`

Key Group index.

`u8KeyIndex [in]`

Key index.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The key is identified by the key group index and the key index given by u8KeyGroup and u8KeyIndex.

Caution

The logged-in user must have the read privilege on the involved key file.

See Also

VltUninitializeAlgorithm

VltReadKey

9.4.15 VltEncrypt

The *VltEncrypt* method encrypts the supplied message and outputs the result.

Syntax

```
VLT_STS VltEncrypt(
  _in VLT_U32 u32PlainTextLength,
  _in const VLT_U8 *pu8PlainText,
  _inout VLT_PU32 pu32CipherTextLength,
```

```
  _out pu8CipherText
);
```

Parameters

`u32PlainTextLength [in]`

Length of the plaintext.

`pu8PlainText [in]`

Plaintext.

`pu32CipherTextLength [in, out]`

On entry this holds the maximum size of the ciphertext buffer. On exit it is set to the amount of ciphertext buffer used

`pu8CipherText [out]`

Buffer to receive ciphertext.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

The VltInitializeAlgorithm command shall be previously sent with the algorithm identifier set with a valid Cipher Identifier. The cipher key is fetched at this time and the cipher engine is initialized with specific algorithm parameters.

See Also

VltInitializeAlgorithm

VltDecrypt

9.4.16 VltDecrypt

The *VltDecrypt* method decrypts the provided message and outputs the result.

Syntax

```
VLT_STS VltDecrypt(
  _in VLT_U32 u32CipherTextLength,
  _in pu8CipherText
  _inout VLT_PU32 pu32PlainTextLength,
  _out const VLT_U8 *pu8PlainText,
);
```

Parameters

`u32CipherTextLength [in]`

Ciphertext length.

`pu8CipherText [in]`

Ciphertext.

`pu32PlainTextLength [in, out]`

On entry this holds the maximum size of the plaintext buffer. On exit it is set to the amount of plaintext buffer used.

`pu8PlainText [out]`

Plaintext.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

The VltInitializeAlgorithm command shall be previously sent with the algorithm identifier set with a valid Cipher Identifier. The cipher key is fetched at this time and the cipher engine is initialized with specific algorithm parameters.

See Also

VltInitializeAlgorithm

VltEncrypt

9.4.17 VltGenerateAssurance

The *VltGenerateAssurance* method generates an assurance message for private key possession assurance.

Syntax

```
VLT_STS VltGenerateAssurance(
  _in VLT_U8 u8KeyGroup,
  _in VLT_U8 u8KeyIndex,
  _inout VLT_PU8 pu8SignerIdLength,
  _inout VLT_PU8 pu8SignerID,
  _out VLT_ASSURANCE_MESSAGE* pAssuranceMsg,
);
```

Parameters

`u8KeyGroup [in]`

The public key group that will be used in the verify operation

`u8KeyIndex [in]`

The public key index that will be used in the verify operation.

`pu8SignerIdLength [in, out]`

The SignerID length must equal `VLT_GA_SIGNER_ID_LENGTH`.

`pu8SignerID [in, out]`

The SignerID, the VaultIC returns the actual assigned SignerID, check the values match.

`pAssuranceMsg [out]`

Returned assurance method structure.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

This command can only be made by an approved user role.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.18 VltGenerateSignature

The *VltGenerateSignature* method generates a signature from a message.

Syntax

```
VLT_STS VltGenerateSignature(
  _in VLT_U32 u32MessageLength,
  _in const VLT_U8 *pu8Message,
  _inout VLT_PU16 pu16SignatureLength,
  _in VLT_PU8 pu8Signature
);
```

Parameters

`u32MessageLength [in]`

Message length.

`pu8Message [in]`

Message buffer.

`pu16SignatureLength [in, out]`

On entry this holds the maximum size of the signature buffer. On exit it is set to the amount of signature buffer used.

`pu8Signature [out]`

Signature buffer.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command gets a raw message or a hashed message and returns its signature.

Note

This command is also used for One Time Password Generation.

Caution

The VltInitializeAlgorithmcommand shall be previously sent with the algorithm identifier set with a valid Signer Identifier. The signer key is fetched at this time and the signer engine is initialized with specific algorithm parameters.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltInitializeAlgorithm

VltVerifySignature

9.4.19 VltUpdateSignature

The *VltUpdateSignature* method continues a multiple-part signature operation, processing another data part.

Syntax

```
VLT_STS VltUpdateSignature(
  _in VLT_U32 u32MessagePartLength,
  _in const VLT_U8 *pu8MessagePart
);
```

Parameters

`u32MessagePartLength [in]`

Message part length.

`pu8MessagePart [in]`

Message part buffer.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

Before the first call to this command, the VltInitializeAlgorithm command shall be previously sent with the algorithm identifier set with a valid Signer Identifier. The signer key is fetched at this time and the signer engine is initialized with specific algorithm parameters.

Caution

To get the signature, a call to VltComputeSignatureFinal must be sent when all message part have been processed with*VltUpdateSignature*.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltComputeSignatureFinal

VltUpdateVerify

9.4.20 VltComputeSignatureFinal

The *VltComputeSignatureFinal* method finishes a multiple-part signature operation, returning the signature.

Syntax

```
VLT_STS VltComputeSignatureFinal(
  _inout VLT_PU16 pu16SignatureLength,
  _out VLT_PU8 pu8Signature
);
```

Parameters

`pu16SignatureLength [in, out]`

On entry this holds the maximum size of the signature buffer. On exit it is set to the amount of signature buffer used.

`pu8Signature [out]`

Signature buffer.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltUpdateSignature

9.4.21 VltVerifySignature

The *VltVerifySignature* method verifies the signature of a message.

Syntax

```
VLT_STS VltVerifySignature(
  _in VLT_U32 u32MessageLength,
  _in const VLT_PU8 pu8Message,
  _in VLT_U16 u16SignatureLength,
  _in const VLT_PU8 pu8Signature
);
```

Parameters

`u32MessageLength [in]`

Message length.

`pu8Message [in]`

Message buffer.

`u16SignatureLength [in]`

On entry this holds the maximum size of the signature buffer. On exit it is set to the amount of signature buffer used.

`pu8Signature [in]`

Signature buffer.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command gets a raw message or a hashed message and a signature and verifies the signature.

Note

This command is also used for One Time Password Generation.

Caution

The VltInitializeAlgorithmcommand shall be previously sent with the algorithm identifier set with a valid Signer Identifier. The signer key is fetched at this time and the signer engine is initialized with specific algorithm parameters.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltInitializeAlgorithm

VltGenerateSignature

9.4.22 VltUpdateVerify

The *VltUpdateVerify* method continue a multiple-part verification operation, processing another data part.

Syntax

```
VLT_STS VltUpdateVerify(
  _in VLT_U32 u32MessagePartLength,
  _in const VLT_PU8 pu8MessagePart
);
```

Parameters

`u32MessagePartLength [in]`

Message part length

`pu8MessagePart [in]`

Message part buffer

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

Before the first call to this command, the VltInitializeAlgorithmcommand shall be previously sent with the algorithm identifier set with a valid Signer Identifier. The signer key is fetched at this time and the signer engine is initialized with specific algorithm parameters.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltComputeVerifyFinal

VltUpdateSignature

9.4.23 VltComputeVerifyFinal

The *VltComputeVerifyFinal* method finishes a multiple-part verification operation, checking the signature.

Syntax

```
VLT_STS VltComputeVerifyFinal(
  _in VLT_U32 u32SignatureLength,
  _in const VLT_U8 *pu8Signature
);
```

Parameters

`u32SignatureLength [in]`

It is set to the signature buffer lenth.

`pu8Signature [in]`

Signature buffer

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltUpdateVerify

9.4.24 VltComputeMessageDigest

The *VltComputeMessageDigest* method computes the digest of a message.

Syntax

```
VLT_STS VltComputeMessageDigest(
  _in VLT_U32 u32MessageLength,
  _in const VLT_U8 *pu8Message,
```

```
  _inout VLT_PU8 pu8DigestLength,
  _in VLT_PU8 pu8Digest
);
```

Parameters

`u32MessageLength [in]`

Message data length.

`pu8Message [in]`

Message data buffer.

`pu8DigestLength [in, out]`

On entry this holds the maximum size of the digest buffer. On exit it is set to the amount of digest buffer used.

`pu8Digest [out]`

Message digest buffer.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

The VltInitializeAlgorithmcommand shall be previously sent with the algorithm identifier set with a valid Digest Identifier.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltInitializeAlgorithm

9.4.25 VltUpdateMessageDigest

The *VltUpdateMessageDigest* method continue a multiple-part message digest operation, processing another data part.

Syntax

```
VLT_STS VltUpdateMessageDigest(
  _in VLT_U32 u32MessagePartLength,
  _in const VLT_U8 *pu8PartMessage
);
```

Parameters

`u32MessagePartLength [in]`

Message part length.

`pu8MessagePart [in]`

Message part buffer.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

Before the first call to this command, the VltInitializeAlgorithmcommand shall be previously sent with the algorithm identifier set with a valid Signer Identifier. The signer key is fetched at this time and the signer engine is initialized with specific algorithm parameters.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltComputeMessageDigestFinal

9.4.26 VltComputeMessageDigestFinal

The *VltComputeMessageDigestFinal* method finishes a multiple-part message digest operation, returning the message digest.

Syntax

```
VLT_STS VltUpdateMessageDigest(
  _inout VLT_PU8 pu8DigestLength,
  _out VLT_PU8 pu8Digest
);
```

Parameters

`pu8DigestLength [in, out]`

On entry this holds the maximum size of the digest buffer. On exit it is set to the amount of digest buffer used.

`pu8Digest [out]`

Digest buffer.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltUpdateMessageDigest

9.4.27 VltGenerateRandom

The *VltGenerateRandom* method generates random bytes.

Syntax

```
VLT_STS VltGenerateRandom(
```

```
  _in VLT_U8 u8NumberOfCharacters,
  _out VLT_PU8 pu8RandomCharacters,
);
```

Parameters

`u8NumberOfCharacters [in]`

Number of random characters to generate, (1..255)

`pu8RandomCharacters [out]`

Buffer to receive random characters.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command fills the supplied buffer with the requested number of random characters.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.28 VltGenerateKeyPair

The *VltGenerateKeyPair* method generates a public and private key pair, and stores the keys in the key ring.

Syntax

```
VLT_STS VltGenerateKeyPair(
  _in VLT_U8 u8PublicKeyGroup,
  _in VLT_U8 u8PublicKeyIndex,
  _in const VLT_FILE_PRIVILEGES *pPublicKeyFilePrivileges,
  _in VLT_U8 u8PrivateKeyGroup,
  _in VLT_U8 u8PrivateKeyIndex,
  _in const VLT_FILE_PRIVILEGES *pPrivateKeyFilePrivileges,
  _in const VLT_KEY_GEN_DATA *pKeyGenData
);
```

Parameters

`u8PublicKeyGroup [in]`

Public key group index.

`u8PublicKeyIndex [in]`

Public key index.

`pPublicKeyFilePrivileges [in]`

Public key file Access Conditions. The logged-in operator must grant its own user ID write permission on the key file.

`u8PrivateKeyGroup [in]`

Private key group index.

`u8PrivateKeyIndex [in]`

Private key index.

`pPrivateKeyFilePrivileges [in]`

Private key file Access Conditions. The logged-in operator must grant its own user ID write permission on the key file.

`pKeyGenData [in]`

Algorithm Parameters Object.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Caution

Generates a public key pair and stores it in the key ring. This command can generate DSA, ECDSA and RSA keys. The key files holding the generated key pair are automatically created and owned by the currently logged-in user.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.29 VltConstructDHAgreement

The *VltConstructDHAgreement* method construct a DH agreement.

Syntax

```
VLT_STS VltConstructDHAgreement(
  _in VLT_U8 u8resultKeyGroup,
  _in VLT_U8 u8resultKeyIndex,
  _in const VLT_FILE_PRIVILEGES *pKeyFilePrivileges,
  _in const VLT_KEY_MATERIAL *pKeyMaterial
);
```

Parameters

`u8resultKeyGroup [in]`

Group index for DH Agreement.

`u8resultKeyIndex [in]`

Index for DH Agreement.

`pKeyFilePrivileges [in]`

Privileges for DH Agreement.

`pKeyMaterial [in]`

Key material for DH agreement construction.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib while status values smaller than VLT_OK are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltDeriveKey

9.4.30 VltDeriveKey

The *VltDeriveKey* method derive a key using a DH agreement.

Syntax

```
VLT_STS VltDeriveKey(
  _in VLT_U8 u8keyGroup,
  _in VLT_U8 u8keyIndex,
  _in const VLT_FILE_PRIVILEGES *pKeyFilePrivileges,
  _in VLT_U8 u8DerivatedKeyType,
  _in VLT_U16 u16WDerivatedKeyLen,
  _in const VLT_KEY_DERIVATION *pKeyDerivation
);
```

Parameters

`u8keyGroup [in]`

Group index for derived key.

`u8keyIndex [in]`

Index for derived key.

`pKeyFilePrivileges [in]`

Privileges for derived key.

`u8DerivatedKeyType [in]`

Derived key type.

`u16WDerivatedKeyLen [in]`

Derived key length.

`pKeyDerivation [in]`

Key derivation parameters.

Return Value

Upon successful completion a VLT_OK status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than VLT_OK are errors that have originated in the VaultIC eLib while status values smaller than VLT_OK are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltConstructDHAgreement

9.4.31 VltBeginTransaction

The *VltBeginTransaction* method starts the transaction method on file system updates.

Syntax

```
VLT_STS VltBeginTransaction( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

VltBeginTransaction command shall be invoked prior to start sensitive file update operations. All the following file system updates will be protected against power-loss event. File system integrity and data consistency will be guaranteed.

If a tear event occurs while a transaction is in progress, any updates to files content and file system structure are discarded. VaultIC Security Module anti-tearing engine will restore the file system as it was when *VltBeginTransaction* command has been received. The transaction is committed by VltEndTransaction command.

Caution

When a transaction is started, the currently selected file or folder is unselected. A VltSelectFileOrDirectory command shall be sent inside the transaction.

Caution

The content of the file system which is update outside a secure transaction is not predictable following a tear or reset during the update.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltSelectFileOrDirectory

VltEndTransaction

9.4.32 VltEndTransaction

The *VltEndTransaction* method starts the transaction method on file system updates.

Syntax

```
VLT_STS VltBeginTransaction( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Any file system updates during a transaction are only done conditionally. All these conditional updates shall be committed at the very end of the transaction sending *VltEndTransaction* command.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltBeginTransaction

9.4.33 VltSelectFileOrDirectory

The *VltSelectFileOrDirectory* method selects a file.

Syntax

```
VLT_STS VltSelectFileOrDirectory (
  _in const VLT_U8 *pu8Path,
  _in VLT_U8 u8PathLength,
  _out VLT_SELECT *pRespData
);
```

Parameters

`pu8Path [in]`

Path of the file to select

`u8PathLength [in]`

Path length, including the NULL terminator

`pRespData [out]`

Returned file size, access conditions and attributes.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Selects a file and retrieves the file size and the access conditions. The Current File Position is initialized to the first byte of the file.

Note

Secure transaction is supported.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltBeginTransaction

9.4.34 VltListFiles

The *VltListFiles* method returns a list of files in the currently selected directory.

Syntax

```
VLT_STS VltListFiles(
  _inout VLT_PU16 pu16ListRespLength,
  _out VLT_PU8 pu8RespData
);
```

Parameters

`pu16ListRespLength [in,out]`

On entry this holds the maximum size of the buffer. On exit it is set to the amount of buffer used.

`pu8RespData [out]`

Buffer to receive multi-string.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Returns a multi-string containing the names of files and directories located in the currently selected directory. Files or directories with the hidden attribute are skipped out the listing.

Note

The order in which files are reported may be different before and after a secure transaction performed on the selected directory.

Note

A single NULL byte is returned if the current directory does not contain any files or subdirectories.

Caution

The logged-in operator must have the List privilege on the selected directory.

Caution

An error code is returned if the buffer passed in is not large enough to accommodate all of the data returned by the VaultIC Security Module. The `pu16ListRespLength` parameter is updated to the size of the buffer required to return the full directory listing. A partial directory listing will be contained within the buffer, but this should be ignored. A buffer of the size reported back by the updated `pu16ListRespLength` should be constructed and passed as the input parameters to *VltListFiles*.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltSelectFileOrDirectory

9.4.35 VltCreateFile

The *VltCreateFile* method creates a new file.

Syntax

```
VLT_STS VltCreateFile(
  _in VLT_U8 u8UserID,
  _in VLT_U32 u32FileSize,
  _in const VLT_FILE_PRIVILEGES *pFilePriv,
  _in VLT_U8 u8FileAttribute,
  _in VLT_U16 u16FileNameLength,
  _in const VLT_U8 *pu8FileName
);
```

Parameters

`u8UserID [in]`

Operator ID (0..7). Possible values are:

– VLT_USER0
– VLT_USER1
– VLT_USER2
– VLT_USER3
– VLT_USER4
– VLT_USER5
– VLT_USER6
– VLT_USER7

`u32FileSize [in]`

Initial file size in bytes. Maximum file size is 65535 bytes.

`u8FileAttribute [in]`

Attributes.

`pFilePriv [in]`

Access conditions.

`u16FileNameLength [in]`

Length of file name, including the NULL terminator (2..9).

`pu8FileName [in]`

Filename.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Creates a new file in the currently selected directory. The newly created file becomes the currently selected file.

Caution

The logged-in operator must have the Create privilege on the selected directory.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltCreateFolder

9.4.36 VltCreateFolder

The *VltCreateFolder* method creates a new folder.

Syntax

```
VLT_STS VltCreateFolder(
  _in VLT_U8 u8UserID,
  _in const VLT_FILE_PRIVILEGES *pFilePriv,
  _in VLT_U8 u8FolderAttribute,
  _in VLT_U16 u16FolderNameLength,
  _in const VLT_U8 *pu8FolderName
);
```

Parameters

`u8UserID [in]`

Operator ID (0..7). Possible values are:

– VLT_USER0
– VLT_USER1
– VLT_USER2
– VLT_USER3
– VLT_USER4
– VLT_USER5
– VLT_USER6
– VLT_USER7

`pFilePriv [in]`
Access conditions

`u8FolderAttribute [in]`
Attributes

`u16FolderNameLength [in]`
Length of folder name, including the NULL terminator (2..9)

`pu8FolderName [in]`
Folder name

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Creates new sub directory in the currently selected directory.

Caution

The logged-in operator must have the Create privilege on the selected directory.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltDeleteFolder

9.4.37 VltDeleteFile

The *VltDeleteFile* method deletes the current file.

Syntax

```
VLT_STS VltDeleteFile( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Once the file is deleted the parent directory is still the current directory but no file is selected.

Note

Read-only files are protected against deletion.

Caution

The logged-in operator must have the Delete privilege on the selected directory.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltCreateFile

9.4.38 VltDeleteFolder

The *VltDeleteFolder* method deletes the current directory.

Syntax

```
VLT_STS VltDeleteFile(
  _in VLT_U8 u8Recurssion
);
```

Parameters

`u8Recurssion [in]`

Specifies whether or not to delete all contained files and folders.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Once the folder is deleted, the root directory becomes the current directory.

Caution

The logged-in operator must have the Delete privilege on the selected directory.

Caution

If recursion is required, Delete privilege must be granted on all files and directories located under the selected directory.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltCreateFolder

9.4.39 VltWriteFile

The *VltWriteFile* method writes data to the currently selected file.

Syntax

```
VLT_STS VltWriteFile(
  _in const VLT_U8 *pu8Data,
  _in VLT_U8 u8DataLength,
  _in VLT_U8 u8ReclaimSpace
);
```

Parameters

`pu8Data [in]`

Data to write

`u8DataLength [in]`

Length of the data (1..255)

`u8ReclaimSpace [in]`

Specfies whether data beyond the completion of the write should be reclaimed by the file system. Possible values are:

– VLT_NO_RECLAIM_SPACE
– VLT_RECLAIM_SPACE

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Updates part of the contents of the current file, from the current file position. Write operation is not allowed on read-only files.

If the provided data go beyond the End Of File, the file will grow automatically provided there is enough space in the file system. It is also possible to shrink the file and discard previous data that were beyond the new End Of File. File system space is allocated/reclaimed accordingly.

The current file position is set at the end of the written data.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltSeekFile

9.4.40 VltReadFile

The *VltReadFile* method reads data from the currently selected file.

Syntax

```
VLT_STS VltReadFile(
  _inout VLT_PU16 pu8ReadLength,
  _out VLT_PU8 pu8RespData
);
```

Parameters

`pu8ReadLength [in, out]`

On entry this holds the maximum size of the buffer. On exit it is set to the amount of buffer used.

`pu8RespData [out]`

Buffer to read data to

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Reads part of the contents of the current file from the current file position.

If the requested length goes beyond the End of File, only the available data are returned and a specific status is returned.

The current file position is set at the end of the read data.

Caution

If `pu8ReadLength` specifies a value larger than the VaultIC can return, an error will be returned and `pu8ReadLength` will be set to the maximum number of bytes that can be read.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltSeekFile

9.4.41 VltSeekFile

The *VltSeekFile* method sets the current file position.

Syntax

```
VLT_STS VltSeekFile(
  _in VLT_U32 u32SeekLength
);
```

Parameters

`u32SeekLength [in]`

The new file position relative to the beginning of the file.

Return Value

Upon successful completion a VLT_OK status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than VLT_OK are errors that have originated in the VaultIC eLib while status values smaller than VLT_OK are the APDU status words that are returned by the VaultIC Security Module.

Note

If the provided offset is beyond the End Of File, a specific status is returned, and the current file position is set just after the last byte of the file (so that a write operation can be performed to append data to the file).

Caution

The logged-in operator must have read or write privilege on the selected file.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltDeleteFolder

VltReadFile

9.4.42 VltSetPrivileges

The *VltSetPrivileges* method updates Access Conditions of the currently selected file or folder.

Syntax

```
VLT_STS VltSetPrivileges(
  _in const VLT_FILE_PRIVILEGES *pFilePriv
);
```

Parameters

pFilePriv [in]

File or folder Access Conditions.

Return Value

Upon successful completion a VLT_OK status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than VLT_OK are errors that have originated in the VaultIC eLib while status values smaller than VLT_OK are the APDU status words that are returned by the VaultIC Security Module.

Note

If the provided offset is beyond the End Of File, a specific status is returned, and the current file position is set just after the last byte of the file (so that a write operation can be performed to append data to the file).

Note

This operation is protected against tear event without the need to initiate a transaction.

Caution

The logged-in operator must have read or write privilege on the selected file.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltDeleteFolder

VltReadFile

9.4.43 VltSetAttributes

The *VltSetAttributes* method updates the attributes of the currently selected file or folder.

Syntax

```
VLT_STS VltSetAttributes(
  _in VLT_U8 u8Attributes
);
```

Parameters

`u8Attributes [in]`

Attributes.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This operation is protected against tear event without the need to initiate a transaction.

Caution

The logged-in operator must have read or write privilege on the selected file.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltDeleteFolder

VltReadFile

9.4.44 VltGetInfo

The *VltGetInfo* method returns information about the security module.

Syntax

```
VLT_STS VltGetInfo(
  _out VLT_TARGET_INFO pRespData
);
```

Parameters

`pRespData [out]`

Structure to receive information.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command returns some information about the security module: chip identifiers, unique serial number, life cycle state and available file system space.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.45 VltSelfTest

The *VltSelfTest* method initiates and runs self testing.

Syntax

```
VLT_STS VltSelfTest( void );
```

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Self-tests sequence ensures that VaultIC is working properly and is automatically performed at each power-up or reset. The self-tests command allows any user, authenticated or not, to initiate the same test flow on-demand for periodic test of the VaultIC.

Note

During self-tests operation, all approved cryptographic algorithms are tested using knownanswer and firmware integrity is checked using CRC-16 CCITT.

Caution

If self-tests failed, any authentication is cancelled, any secure channel is closed and the VaultIC is automatically switched to TERMINATED state.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.46 VltSetStatus

The *VltSetStatus* method changes the life cycle state of the VaultIC.

Syntax

```
VLT_STS VltSetStatus(
  _in VLT_U8 u8State
);
```

Parameters

u8State [in]

New state value. Possible values are:

- VLT_CREATION
- VLT_OPERATIONAL_ACTIVE
- VLT_OPERATIONAL_DEACTIVE
- VLT_TERMINATED

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.47 VltSetConfig

The *VltSetConfig* method sets VaultIC configuration parameters.

Syntax

```
VLT_STS VltSetConfig(
  _in VLT_U8 u8ConfigItem,
  _in VLT_U8 u8DataLength,
  _in VLT_PU8 pu8ConfigData
);
```

Parameters

`u8ConfigItem [in]`

VLT_USB_*, VLT_ADMIN_*

`u8DataLength [in]`

Length of data buffer

`pu8ConfigData [in]`

Data buffer containing configuration value

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Defines internal parameters of the VaultIC chip. These settings are applied using the antitearing mechanism and permanently stored into the internal memory.

Caution

Setting wrong values may damage the VaultIC chip. Use with caution.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

9.4.48 VltSetGpioDirection

The *VltSetGpioDirection* method manages the VaultIC GPIO lines.

Syntax

```
VLT_STS VltSetGpioDirection(
  _in VLT_U8 u8GpioDirMask,
  _in VLT_U8 u8GpioMode,
);
```

Parameters

`u8GpioDirMask [in]`

Bitfield representing the GPIO direction. Each bit corresponds to a physical line: Di: 0 <= i <= 7 direction for GPIO(i) where: 0 = Input Direction 1 = Output Direction.

`u8GpioMode [in]`

Bitfield representing the GPIO output mode. Each bit corresponds to a physical line: Di: 0 <= i <= 7 direction for GPIO(i) where: 0 = Open Drain Mode 1 = CMOS Mode

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Changes the direction of the GPIO lines between input and output directions. Optionally activates CMOS mode for output lines.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltWriteGpio

VltReadGpio

9.4.49 VltWriteGpio

The *VltWriteGpio* method writes to the VaultIC GPIO lines.

Syntax

```
VLT_STS VltWriteGpio(
  _in VLT_U8 u8GpioValue
);
```

Parameters

`u8GpioValue [in]`

Bitfield representing the GPIO output mode. Each bit corresponds to a physical line: Di: 0 <= i <= 7 output for GPIO(i) where: 0 = Sets to logic low 1 = Sets to logic high

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note Sets the value of the current GPIO outputs.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltSetGpioDirection

VltReadGpio

9.4.50 VltReadGpio

The *VltReadGpio* method reads value of the VaultIC GPIO lines.

Syntax

```
VLT_STS VltReadGpio(
  _out VLT_PU8 pu8GpioValue
);
```

Parameters

`pu8GpioValue [out]`

Bitfield representing the port physical value. Each bit corresponds to a physical line: Di: 0 <= i <= 7 output for GPIO(i) where: 0 = Reads as logic low 1 = Reads as logic high

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Reads the physical value of the current GPIO outputs.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltSetGpioDirection

VltWriteGpio

9.4.51 VltTestCase1

The *VltTestCase1* method is a dummy APDU case 1 command for integration testing.

Syntax

```
VLT_STS VltTestCase1 ( void );
```

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command is a dummy APDU case 1 command for integration testing purposes.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltTestCase2

VltTestCase3

VltTestCase4

9.4.52 VltTestCase2

The *VltTestCase2* method is a dummy APDU case 2 command for integration testing.

Syntax

```
VLT_STS VltTestCase2(
  _in VLT_U8 u8RequestedDataLength,
  _out VLT_U8 u8RespData
);
```

Parameters

`u8RequestedDataLength [in]`

Number of output bytes (1..255)

`pu8RespData [out]`

Buffer to receive output bytes

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command is a dummy APDU case 2 command for integration testing purposes.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltTestCase1

VltTestCase3

VltTestCase4

9.4.53 VltTestCase3

The *VltTestCase3* method is a dummy APDU case 3 command for integration testing.

Syntax

```
VLT_STS VltTestCase3(
  _in VLT_U8 u8DataLength,
  _in VLT_U8 pu8DataIn
);
```

Parameters

`u8DataLength [in]`

Number of output bytes (1..255)

`pu8DataIn [in]`

Input bytes

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command is a dummy APDU case 2 command for integration testing purposes.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltTestCase1

VltTestCase2

VltTestCase4

9.4.54 VltTestCase4

The *VltTestCase4* method is a dummy APDU case 4 command for integration testing.

Syntax

```
VLT_STS VltTestCase4(
  _in VLT_U8 u8DataLength,
  _in const VLT_U8 *pu8Data,
  _in VLT_U8 u8RequestedDataLength,
  _out VLT_PU8 pu8RespData
);
```

Parameters

`u8DataLength [in]`
Number of output bytes (1..255)

`pu8Data [in]`
Input bytes

`u8RequestedDataLength [in]`
Number of output bytes (1..255)

`pu8RespData [out]`
Buffer to receive output bytes

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This command is a dummy APDU case 4 command for integration testing purposes.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_api.h

See Also

VltTestCase1

VltTestCase2

VltTestCase3

9.5 Key Wrapping Service

The Key Wrapping Service provides a means of wrapping and unwrapping keys to and from the VaultIC.

9.5.1 VltKeyWrappingInit

The *VltKeyWrappingInit* method initializes the Key Wrapping Service.

Syntax

```
VLT_STS VltKeyWrappingInit(
  _in VLT_U8 u8KTSKeyGroup,
  _in VLT_U8 u8KTSKeyIndex,
  _in WRAP_PARAMS* pWrapParams,
  _in VLT_KEY_OBJECT* pKTSKey
);
```

Parameters

`u8KTSKeyGroup [in]`

Key group index of the Key Transport Scheme.

`u8KTSKeyIndex [in]`

Key index of the Key Transport Scheme.

`pWrapParams [in]`

The parameters used to wrap/unwrap the key.

`pKTSKey [in]`

The KTS key used to encrypt/decrypt the key.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This method is used to initialise the key wrapping/unwrapping service. The `WRAP_PARAMS` structure passed in should be populated with the appropriate values to select the algorithm used to wrap/unwrap the key.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_key_wrapping.h

See Also

VltKeyWrappingClose

9.5.2 VltUnwrapKey

The *VltUnwrapKey* method wraps the key data to be sent to the VaultIC.

Syntax

```
VLT_STS VltUnwrapKey(
  _in VLT_U8 u8KeyGroup,
  _in VLT_U8 u8KeyIndex,
```

```
  _in const VLT_FILE_PRIVILEGES *pKeyFilePrivileges,
  _in const VLT_KEY_OBJ_RAW* pKeyObj
);
```

Parameters

`u8KeyGroup [in]`
Key Group index.

`u8KeyIndex [in]`
Key index.

`pKeyFilePrivileges [in]`
Pointer to the privileges for the key being put down to the VaultIC.

`pKeyObj [in]`
Pointer to the key object to be put down to the VaultIC.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_key_wrapping.h

See Also

VltWrapKey

9.5.3 VltWrapKey

The *VltWrapKey* method unwraps the key data received from the VaultIC.

Syntax

```
VLT_STS VltWrapKey(
  _in VLT_U8 u8KeyGroup,
  _in VLT_U8 u8KeyIndex,
  _out VLT_KEY_OBJ_RAW* pKeyObj
);
```

Parameters

`u8KeyGroup [in]`
Key Group index.

`u8KeyIndex [in]`
Key index.

`pKeyObj [out]`
Pointer to a key object to be filled with the key being read from the VaultIC.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_key_wrapping.h

See Also

VltUnwrapKey

9.5.4 VltKeyWrappingClose

The *VltKeyWrappingClose* method closes the Key Wrapping Service.

Syntax

```
VLT_STS VltKeyWrappingClose( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_key_wrapping.h

See Also

VltKeyWrappingInit

9.6 Identity Authentication Service

The Identity Authentication Service provides a simple means to authenticate a user. The following methods of authentication are supported:

- Secure Channel 02
- Secure Channel 03
- Microsoft Smart Card Minidriver

To authenticate a user using a password, the Base API method VltSubmitPassword should be used. An authentication established using a password must be cancelled using the VltCancelAuthentication method.

9.6.1 Conditions of Use

If the Identity Authentication Service is used the following rules must be adhered to:

If the VltAuthInit method is successfully called, the VltAuthClose method must be called to cancel the authentication.

The following Base API methods must not be called when a user has been authenticated using the Identity Authentication Service:

- VltSubmitPassword
- VltCancelAuthentication

– VltInitializeUpdate
– VltExternalAuthenticate

Once authenticated, if any further method call returns an error code, the VltAuthGetState method should be called to check whether the authentication has been cancelled.

9.6.2 VltAuthInit

The *VltAuthInit* method is used to initialise the Identity Authentication Service.

Syntax

```
VLT_STS VltAuthInit(
  _in VLT_U8 u8AuthMethod,
  _in VLT_U8 u8UserID,
  _in VLT_U8 u8RoleID,
  _in VLT_U8 u8ChannelLevel,
  _in KEY_BLOB_ARRAY keys
);
```

Parameters

`u8AuthMethod [in]`

Authentication Method, possible values are:

– VLT_LOGIN_SCP02
– VLT_LOGIN_SCP03
– VLT_LOGIN_MS

`u8UserID [in]`

Operator ID (0..7). Possible values are:

– VLT_USER0
– VLT_USER1
– VLT_USER2
– VLT_USER3
– VLT_USER4
– VLT_USER5
– VLT_USER6
– VLT_USER7

`u8RoleID [in]`

Role ID. Possible values are

– VLT_APPROVED_USER
– VLT_NON_APPROVED_USER
– VLT_MANUFACTURER
– VLT_ADMINISTRATOR
– VLT_EVERYONE

`u8ChannelLevel [in]`

Secure Channel Level (valid for SCP02 and SCP03) possible values are:

– VLT_NO_CHANNEL
– VLT_CMAC

– VLT_CMAC_CENC
– VLT_CMAC_RMAC
– VLT_CMAC_CENC_RMAC
– VLT_CMAC_CENC_RMAC_RENC

`keys [in]`

A structure containing a the number of keys and a pointer to an array of type KEY_BLOB.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This service shall be used exclusively to authenticated users who have one of the following authentication methods: Secure Channel 02. Secure Channel 03. Microsoft Smart Card Minidriver.

On success, the user will be authenticated (logged into) the VaultIC Security Module, providing access to most methods.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_identity_authentication.h

See Also

VltAuthClose

9.6.3 VltAuthClose

The *VltAuthClose* method closes the identity authentication service.

Syntax

```
VLT_STS VltAuthClose ( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Logs the current user out of the VaultIC Security Module and closes an open secure channel.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_identity_authentication.h

See Also

VltAuthInit

9.6.4 VltAuthGetState

The *VltAuthGetState* method returns the state of the identity authentication service.

Syntax

```
VLT_STS VltAuthGetState ( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Provides the host with the authentication (login) state of the current user.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_identity_authentication.h

See Also

VltAuthInit

9.7 File System Service

The File System Service provides a convenient means to update the contents of the VaultIC Security Module file system.

9.7.1 VltFsOpenFile

The *VltFsOpenFile* method opens a file and makes it ready for reading or writing.

Syntax

```
VLT_STS VltFsOpenFile(
  _in VLT_U16 u16FileNameLength,
  _in const VLT_PU8 pu8FileName,
  _in VLT_U8 u8TransactionMode,
  _out VLT_FS_ENTRY_PARAMS *pFsFileParams
);
```

Parameters

`u16FileNameLength [in]`

The length of the file name

`pu8FileName [in]`

The file name

`u8TransactionMode [in]`

Specifies whether further operations should be carried out within a transaction to protect against power loss. Possible values:

– VLT_TRANSACTION_DISABLED

– VLT_TRANSACTION_ENABLED

`pFsFileParams [out]`

The File Entry parameters for the file opened

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

The specified file is opened ready for reading/writing. If the `u8TransactionMode` specifies that a transaction should be initiated, this is done. The `pFsFileParams` structure is populated with the parameters for the file.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

See Also

VltFsCloseFile

9.7.2 VltFsCloseFile

The *VltFsCloseFile* method closes a file after it has been updated.

Syntax

```
VLT_STS VltFsCloseFile ( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This method is used to close a file following read/write operations. This method must be called if VltFsOpenFile was called with the `u8TransactionMode` argument set as `VLT_TRANSACTION_ENABLED` to ensure that the transaction is ended and that the updates to the file are committed to the file system.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

See Also

VltFsOpenFile

9.7.3 VltFsCreate

The *VltFsCreate* method creates files or folders within the file system.

Syntax

```
VLT_STS VltFsCreate(
  _in VLT_U16 u16EntryNameLen,
  _in const VLT_PU8 pu8EntryName,
  _in const VLT_FS_ENTRY_PARAMS *pFsEntryParams,
  _in VLT_U8 u8UserId
);
```

Parameters

`u16EntryNameLen [in]`

The length of the file/folder path being passed in.

`pu8EntryName [in]`

The full name of the file/folder to be created.

`pFsEntryParams [in]`

The parameters needed to create the file/folder.

`u8UserId [in]`

The user id of the owner of the file system entry.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

See Also

VltFsDelete

9.7.4 VltFsDelete

The *VltFsDelete* method deletes a file or folder within the file system.

Syntax

```
VLT_STS VltFsDelete(
  _in VLT_U16 u16EntryNameLen,
  _in const VLT_PU8 pu8EntryName,
  _in VLT_U8 u8Recursion
);
```

Parameters

`u16EntryNameLen [in]`

The length of the file/folder path being passed in

`pu8EntryName [in]`

The full name of the file/folder to be created

`u8Recursion [in]`

If the entry to be deleted is a folder, specifies whether a recursive delete of files and

folders should take place. Possible values:

– VLT_NON_RECURSIVE_DELETE

– VLT_RECURSIVE_DELETE

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

See Also

VltFsCreate

9.7.5 VltFsReadFile

The VltFsReadFile method is used to read data from an open file.

Syntax

```
VLT_STS VltFsReadFile(
  _in VLT_U32 u32Offset,
  _out VLT_PU8 pu8DataOut,
  _inout VLT_PU32 pu32DataLength
);
```

Parameters

`u32Offset [in]`

The position within the file to start reading from. Don't use `VLT_SEEK_TO_END`, an error of `EFSRDSEEKFAILED` will be returned.

`pu8DataOut [in, out]`

The buffer to place the data read from the VaultIC Security Module.

`pu32DataLength [in, out]`

The length of the data to be read.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This method is used to read data from a file previously opened with VltFsOpenFile

Note

If the requested length goes beyond the End of File, only the available data are returned and a specific status (`VLT_EOF`) is returned.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

See Also

VltFsOpenFile

9.7.6 VltFsWriteFile

The *VltFsWriteFile* method is used to write data to an open file.

Syntax

```
VLT_STS VltFsWriteFile(
  _in VLT_U32 u32Offset,
  _in VLT_PU8 pu8DataIn,
  _in VLT_PU32 pu32DataLength,
  _in VLT_U8 u8ReclaimSpace
);
```

Parameters

`u32Offset [in]`

The position within the file to start writing from. Use constants `VLT_SEEK_FROM_START`, `VLT_SEEK_TO_END`, or any value less than or equal to the open file size.

`pu8DataIn [in]`

The buffer containing the data to be written to the VaultIC Security Module.

`pu32DataLength [in]`

The length of the data to be written

`u8ReclaimSpace [in]`

Specifies whether any existing data preceding the written data should be deleted or not. Possible values:

– VLT_NO_RECLAIM_SPACE
– VLT_RECLAIM_SPACE

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

This method is used to write data from a file previously opened with VltFsOpenFile.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

See Also

VltFsOpenFile

9.7.7 VltFsListFiles

The *VltFsListFiles* method lists the files and folders contained within the specified folder.

Syntax

```
VLT_STS VltFsListFiles
  _in VLT_U16 u16FolderNameLength,
  _in const VLT_PU8 pu8FolderName,
  _inout VLT_PU16 pu16ListRespLength,
  _in VLT_PU8 pu8RespData
);
```

Parameters

`u16FolderNameLength [in]`

The length of the folder name.

`pu8FolderName [in]`

The folder name to perform the listing on.

`pu16ListRespLength [in, out]`

On entry this holds the maximum size of the buffer. On exit it is set to the amount of buffer used.

`pu8RespData [in]`

Buffer to receive multi-string.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than `VLT_OK` are errors that

have originated in the VaultIC eLib while status values smaller than `VLT_OK` are the APDU status words that are returned by the VaultIC Security Module.

Note

Returns a multi-string containing the names of files and directories located in the currentlyselected directory.

Files or directories with the hidden attribute are skipped out the listing.

A multi-string is a sequence of NULL-terminated strings. The sequence is terminated by a NULL byte, therefore the multi-string ends up by two consecutive bytes.

The order in which files are reported may be different before and after a secure transaction performed on the selected directory.

The logged-in operator must have the List privilege on the selected directory.

A single NULL byte is returned if the current directory does not contain any files or subdirectories.

An error code is returned if the buffer passed in is not large enough to accommodate all of the data returned by the VaultIC Security Module. The pu16ListRespLength parameter is updated to the size of the buffer required to return the full directory listing. A partial directory listing will be contained within the buffer, but this should be ignored. A buffer of the size reported back by the updated pu16ListRespLength should be constructed and passed as the input parameters to *VltFsListFiles*.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

See Also

VltFsOpenFile

9.7.8 VltFsSetPrivileges

The *VltFsSetPrivileges* method changes the privileges on a file system entry.

Syntax

```
VLT_STS VltFsSetPrivileges
  _in VLT_U16 u16EntryNameLength,
  _in const VLT_PU8 pu8EntryName,
  _in const VLT_FS_ENTRY_PRIVILEGES* pFsEntryPrivileges
);
```

Parameters

`u16EntryNameLength [in]`

The length of the file/folder path being passed in

`pu8EntryName [in]`

The full name of the file/folder to be modified.

`pFsEntryPrivileges [in]`

The updated access privileges to set on the file system entry.

Return Value

Upon successful completion a VLT_OK status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than VLT_OK are errors that have originated in the VaultIC eLib while status values smaller than VLT_OK are the APDU status words that are returned by the VaultIC Security Module.

Note

This method is used to change the access privileges on a file or folder.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

9.7.9 VltFsSetAttributes

The *VltFsSetAttributes* method changes the attributes on a file system entry.

Syntax

```
VLT_STS VltFsSetAttributes
  _in VLT_U16 u16EntryNameLength,
  _in const VLT_PU8 pu8EntryName,
  _in VLT_FS_ENTRY_ATTRIBS* pFsEntryAttributes
);
```

Parameters

`u16EntryNameLength [in]`

The length of the file/folder path being passed in.

`pu8EntryName [in]`

The full name of the file/folder to be modified.

`pFsEntryAttributes [in]`

The updated attribute to set on the file system entry.

Return Value

Upon successful completion a VLT_OK status will be returned otherwise the appropriate error code will be returned. Please note, status values larger than VLT_OK are errors that have originated in the VaultIC eLib while status values smaller than VLT_OK are the APDU status words that are returned by the VaultIC Security Module.

Note

This method is used to change the attributes on a file or folder.

Requirements

Header: Library - vaultic_lib.h, Embedded – vaultic_file_system.h

10. Client Stub Interfaces

This section details the client stub interfaces which require an implementation to be added. Not all of the interfaces will require an implementation to be added. This is dependant on which communication protocol(s) the embedded target wishes to utilise.

10.1 vaultic_mem

The *vaultic_mem* interface provides a means of manipulating the contents of memory. The supplied code provides an implementation that should work on any target as it uses standard ANSI C methods. The methods contained within this module are called throughout the VaultIC eLib code.

10.1.1 host_memcpy

The **`host_memcpy`** method copies `len` bytes from `src` to `dest`.

Syntax

```
void host_memcpy(
  _out VLT_PU8 dest,
  _in const VLT_U8 *src,
  _in VLT_U32 len
);
```

Parameters

`dest [out]`

The memory address of the destination buffer.

`src [in]`

The memory address of the source buffer.

`len [in]`

The length, in bytes, to be copied from src to dest.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

The purpose of this method is to copy the contents of one memory range (`src`) to another (`dest`). It mimics the behaviour of the ANSI C **`memcpy`** method, which is what the supplied example code calls to implement the behaviour. The size of the memory ranges are specified in bytes by the value `len`.

10.1.2 host_memset

The **`host_memset`** method sets `len` bytes from `dest` to the value specified in value.

Syntax

```
void host_memset(
  _out VLT_PU8 dest,
  _in VLT_U8 value,
  _in VLT_U32 len
);
```

Parameters

`dest [out]`

The memory address of the destination buffer.

`value [in]`

The value to be set

`len [in]`

The number of bytes to set the value.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

The purpose of this method is to set the contents of a memory range (`dest`)to a specified `value`. It mimics the behaviour of the ANSI C `memset` method, which is what the supplied example code calls to implement the behaviour. The size of the memory ranges are specified in bytes by the value `len`.

10.1.3 host_memcmp

The **`host_memcmp`** method compares len bytes from src1 and src2.

Syntax

```
VLT_STS host_memcmp(
  _in const VLT_U8* src1,
  _in const VLT_U8* src2,
  _in VLT_U32 len
);
```

Parameters

`src1 [in]`

The memory address of the first buffer for comparison.

`src2 [in]`

The memory address of the second buffer for comparison.

`len [in]`

The number of bytes to be compared.

Return Value

If the buffers are equal 0 should be returned. A non-zero value should be returned if the buffers are not equal.

Note

The purpose of this method is to compare the contents of one memory range (`src1`) to another (`src2`). It mimics the behaviour of the ANSI C `memcmp` method, which is what the supplied example code calls to implement the behaviour. The size of the memory ranges are specified in bytes by the value `len`.

10.1.4 host_memxor

The **`host_memxor`** method performs a XOR of the `src` and `dest` values for `len` bytes, and places the result in `dest`.

Syntax

```
VLT_STS host_memxor(
```

```
  _out VLT_PU8 dest,
  _in const VLT_PU8 src,
  _in VLT_U32 len
);
```

Parameters

`dest [out]`

The memory address of the first buffer for XOR and the destination of the result.

`src [in]`

The memory address of the second buffer for XOR.

`len [in]`

The number of bytes to be XOR'd.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

The purpose of this method is to XOR the contents of one memory range (`dest`) with another (`src`). The result of this operation overwrites the contents of the `dest` memory range. The size of the memory ranges are specified in bytes by the value `len`.

10.1.5 host_memcpyxor

The **`host_memcpyxor`** method copies a memory range (`src`) to another memory range (`dest`) whilst also performing a XOR of a specified `mask` value.

Syntax

```
VLT_STS host_memcpyxor(
  _out VLT_PU8 dest,
  _in const VLT_PU8 src,
  _in VLT_U32 len
  _in VLT_U8 mask
);
```

Parameters

`dest [out]`

The memory address of the destination buffer.

`src [in]`

The memory address of the source buffer.

`len [in]`

The number of bytes to copied.

`mask [in]`

The mask value to XOR the destination buffer with.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

The purpose of this method is to copy the contents of one memory range (`src`) to another memory range (`dest`) whilst also performing an XOR with a `mask` value. The size of the memory ranges are specified in bytes by the value `len`.

10.1.6 host_lshift

The **`host_lshift`** method left bit-shifts the contents of the `arrayIn` by `bitsToShift` bits.

Syntax

```
VLT_STS host_lshift(
  _inout VLT_PU8 arrayIn,
  _in VLT_U32 arrayInLen,
  _in VLT_U8 bitsToShift
);
```

Parameters

`arrayIn [in, out]`

The memory address of the array.

`arrayInLen [in]`

The length of the array in bytes.

`bitsToShift [in]`

The number of bits to left shift the array.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

The purpose of this method is to left shift the contents of an array `arrayIn` by `bitsToShift` bits. On completion the contents of `arrayIn` will be the updated array.

10.2 vaultic_timer_delay

The vaultic_timer_delay interface provides a means of delaying code execution. The communications layer of the API makes use of the method.

10.2.1 VltSleep

The **`VltSleep`** method should delay code execution on the host for the specified period of microseconds.

Syntax

```
VLT_STS VltSleep(
  _in VLT_U32 uSecDelay
);
```

Parameters

`uSecDelay [in]`

The number of microseconds to delay code execution on the host side.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

The purpose of this method is to delay code execution for a period of `uSecDelay` microseconds. Depending on the host device this level of granularity may not be possible.

10.3 vaultic_iso7816_protocol

The vaultic_iso7816_protocol interface provides the methods required to communicate with the VaultIC Security Module using the ISO 7816 standard.

10.3.1 VltIso7816PtclInit

The VltIso7816PtclInit method initialises the host's ISO 7816 hardware to allow communications with the VaultIC Security Module.

Syntax

```
VLT_STS VltIso7816PtclInit(
  _in VLT_INIT_COMMS_PARAMS* pInitCommsParams,
  _in VLT_MEM_BLOB *pOutData,
  _in VLT_MEM_BLOB *pInData,
  _out VLT_PU16 pu16MaxSendSize,
  _out VLT_PU16 pu16MaxReceiveSize
);
```

Parameters

`pInitCommsParams [in]`

The communications initialisation parameters.

`pOutData [in]`

This parameter is not used.

`pInData [in]`

This parameter is not used.

`pu16MaxSendSize [out]`

This specifies the maximum number of APDU Data In bytes that the host is capable of sending.

`pu16MaxReceiveSize [out]`

This specifies the maximum number of APDU Data Out bytes that the host is capable of receiving.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should initialise the host systems ISO 7816 hardware to allow communications with the VaultIC Security Module to take place. The pInitCommsParams argument provides the parameters to be used. The `pu16MaxSendSize` argument should be populated with the maximum number of APDU Data In bytes that the host can send. The `pu16MaxReceiveSize` argument should be populated with the maximum number of APDU Data Out bytes that the host can receive.

Note

The `pOutData and pInData` arguments are not used by this method. They are present because this method adheres to the interface defined for *VltPtclInit*.

10.3.2 VltPtclClose

The *VltPtclClose* method is responsible for closing down the host's ISO 7816 hardware.

Syntax

```
VLT_STS VltPtclClose( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should correctly close down the host's ISO 7816 hardware.

10.3.3 VltIso7816PtclSendReceiveData

The *VltIso7816PtclSendReceiveData* method is responsible for sending and receiving commands and responses from the VaultIC Security Module.

Syntax

```
VLT_STS VltIso7816PtclSendReceiveData(
  _in const VLT_MEM_BLOB *pOutData,
  _out VLT_MEM_BLOB *pInData
);
```

Parameters

`pOutData [in]`

The buffer in which the command data to be sent is present.

`pInData [in]`

The buffer into which, the response data will be placed.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method is responsible for send and receiving commands and responses from the VaultIC Security Module. It should send the data provided in the `pOutData` argument, and populate the `pInData` with the received data.

10.4 vaultic_spi_peripheral

The vaultic_spi_peripheral interface provides the methods required to communicate with the VaultIC Security Module using the SPI bus.

10.4.1 VltSpiPeripheralInit

The *VltSpiPeripheralInit* method is responsible for initislisng the host's SPI hardware to allow communications with the VaultIC.

Syntax

```
VLT_STS VltSpiPeripheralInit (
  _in VLT_INIT_COMMS_PARAMS* pInitCommsParams
);
```

Parameters

`pInitCommsParams [in]`

The communication initialisation parameters.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should initialise the host systems SPI hardware to allow communications with the VaultIC Security Module to take place. The `pInitCommsParams` argument provides the parameters to be used.

10.4.2 VltSpiPeripheralClose

The *VltSpiPeripheralClose* method is responsible for closing down the host's SPI hardware.

Syntax

```
VLT_STS VltSpiPeripheralClose( void );
```

Parameters

None.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should correctly close down the host's SPI hardware.

10.4.3 VltSpiPeripheralIoctl

The *VltSpiPeripheralIoctl* method is repsonsible for dealing with I/O control commands.

Syntax

```
VLT_STS VltSpiPeripheralIoctl(
  _in VLT_U32 u32Id,
  _in void* pConfigData
);
```

Parameters

`u32Id [in]`

The ID of the command to service. Possible values:

– VLT_AWAIT_DATA

– VLT_RESET_PROTOCOL

– VLT_UPDATE_BITRATE

`pConfigData [in]`

The data needed to service the command. This should be cast to a void*.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should provide the functionality required to service the I/O control commands. The second parameter is passed as a `void*` to allow flexibility when passing data. The supported `u32Id` values are published within the *vaultic_peripheral.h* file. This method is called from the *vaultic_block_protocol* module, so any additional `u32Id` values required by the client hardware should result in calls to this method being added there. Of the three values used at present, only the `VLT_UPDATE_BITRATE` value passes data via the `pConfigData` parameter. This is a `VLT_U16*` which specifies the bitrate, in kHz, to be used.

10.4.4 VltSpiPeripheralSendData

The *VltSpiPeripheralSendData* method is responsible for sending data from the host to the VaultIC Security Module.

Syntax

```
VLT_STS VltSpiPeripheralSendData (
  _in VLT_MEM_BLOB *pOutData,
  _in VLT_U8 u8DelayArraySz,
  _in VLT_DELAY_PAIRING* pDelayPairing
);
```

Parameters

`pOutData [in]`

The data to be sent to the VaultIC Security Module.

`u8DelayArraySz [in]`

The number of entries in the pDelayPairing array.

`pDelayPairing [in]`

The array of delay pairings that specify specific delays that should be inserted before the byte position specified.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should send the supplied data to the VaultIC Security Module. If a specific delay between certain bytes is need an array of `VLT_DELAY_PAIRING` structures should be passed in to specify the bytes and delays required.

10.5 vaultic_twi_peripheral

The vaultic_twi_peripheral interface provides the methods required to communicate with the VaultIC Security Module using the TWI bus.

10.5.1 VltTwiPeripheralInit

The *VltTwiPeripheralInit* method is responsible for initislisng the host's TWI hardware to allow communications with the VaultIC Security Module.

Syntax

```
VLT_STS VltTwiPeripheralInit (
  _in VLT_INIT_COMMS_PARAMS* pInitCommsParams
);
```

Parameters

`pInitCommsParams [in]`

The communication initialisation parameters.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Remarks

This method should initialise the host systems TWI hardware to allow communications with the VaultIC Security Module to take place. The `pInitCommsParams` argument provides the parameters to be used.

10.5.2 VltTwiPeripheralClose

The *VltTwiPeripheralClose* method is responsible for closing down the host's TWI hardware.

Syntax

```
VLT_STS VltTwiPeripheralClose( void );
```

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should correctly close down the host's TWI hardware.

10.5.3 VltTwiPeripheralIoctl

The *VltTwiPeripheralIoctl* method is repsonsible for dealing with I/O control commands.

Syntax

```
VLT_STS VltTwiPeripheralIoctl(
  _in VLT_U32 u32Id,
  _in void* pConfigData
);
```

Parameters

`u32Id [in]`

The ID of the command to service. Possible values:

- VLT_AWAIT_DATA
- VLT_RESET_PROTOCOL
- VLT_UPDATE_BITRATE

`pConfigData [in]`

The data needed to service the command. This should be cast to a `void*`.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should provide the functionality required to service the I/O control commands. The second parameter is passed as a `void*` to allow flexibility when passing data. The supported `u32Id` values are published within the *vaultic_peripheral.h* file. This method is called from the *vaultic_block_protocol* module, so any additional `u32Id` values required by the client hardware should result in calls to this method being added there. Of the three values used at present, only the `VLT_UPDATE_BITRATE` value passes data via the `pConfigData` parameter. This is a `VLT_U16*` which specifies the bitrate, in kHz, to be used.

10.5.4 VltTwiPeripheralSendData

The *VltTwiPeripheralSendData* method is responsible for sending data from the host to the VaultIC Security Module.

Syntax

```
VLT_STS VltTwiPeripheralSendData (
  _in VLT_MEM_BLOB *pOutData,
  _in VLT_U8 u8DelayArraySz,
  _in VLT_DELAY_PAIRING* pDelayPairing
);
```

Parameters

`pOutData [in]`

The data to be sent to the VaultIC Security Module.

`u8DelayArraySz [in]`

The number of entries in the `pDelayPairing` array.

`pDelayPairing [in]`

The array of delay pairings that specify specific delays that should be inserted before the byte position specified.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

Note

This method should send the supplied data to the VaultIC Security Module. If a specific delay between certain bytes is need an array of `VLT_DELAY_PAIRING` structures should be passed in to specify the bytes and delays required.

10.5.5 VltTwiPeripheralReceiveData

The *VltTwiPeripheralReceiveData* method is responsible for receiving data from the the VaultIC Security Module.

Syntax

```
VLT_STS VltTwiPeripheralReceiveData (
  _out VLT_MEM_BLOB *pInData
);
```

Parameters

`pInData [out]`

The data buffer to place the received data from the VaultIC Security Module.

Return Value

Upon successful completion a `VLT_OK` status will be returned otherwise the appropriate error code will be returned.

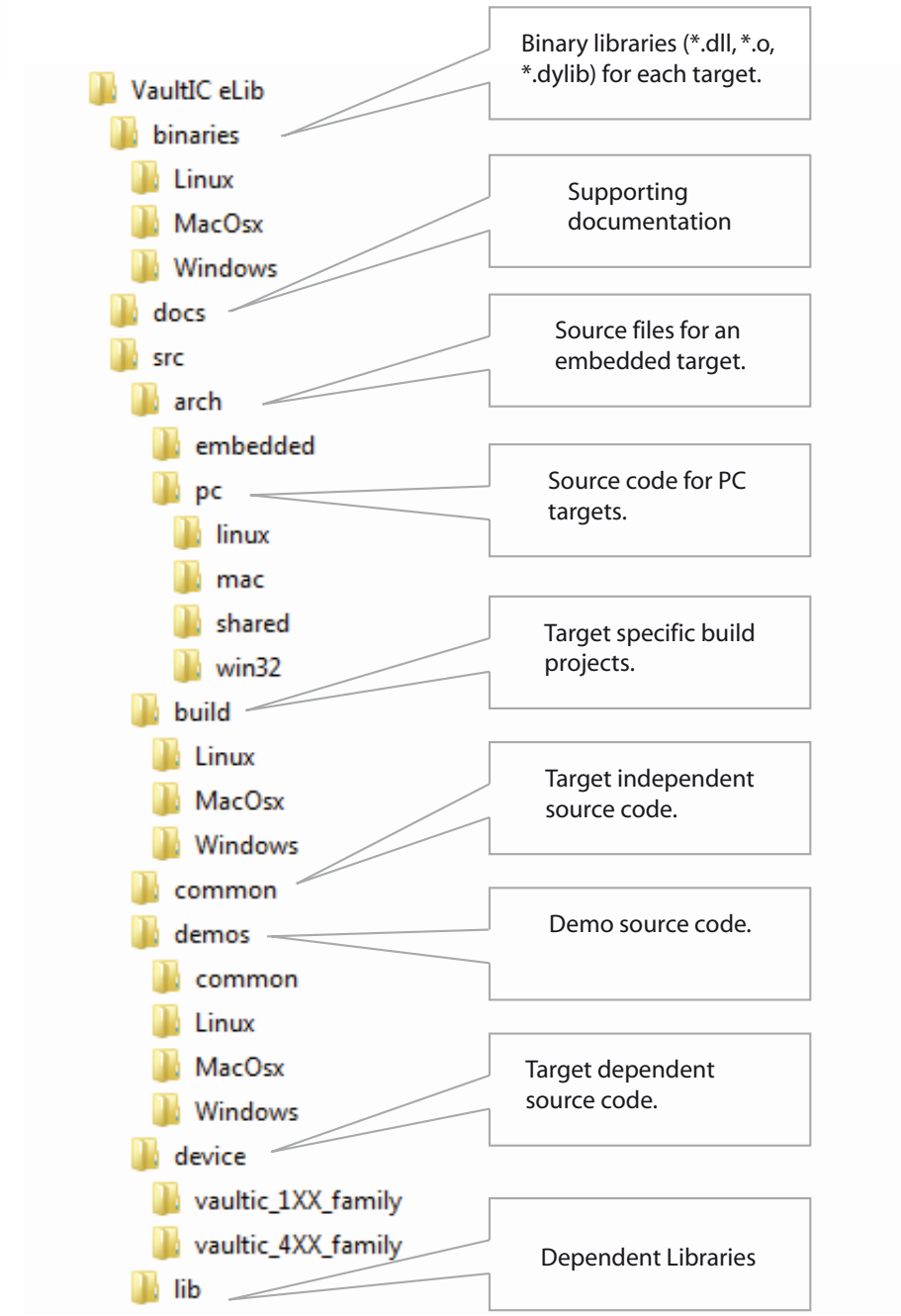
Note

This method should receive the data from the VaultIC Security Module and populate the p*InData* argument. The value of the `u32msTimeout` value passed in the `VLT_INIT_COMMS_PARAMS` structure to the VltTwiPeripheralInit method should be used to determine how long the method will continue to wait for a response from the VaultIC. This value should be large enough to wait for the S-TimeRequest S-Block.

11. VaultIC eLib Deployment

The API is deployed using an installer package. One the installer has been run the folder structure below is created in the user-defined installation folder.

Figure 11-1.



11.1 Binaries

A collection of binaries for each of the PC platforms supported, Windows, Linux, MAC. The format of those binaries are, DLL, so, dylib.

11.2 Arch/embedded

The embedded subfolder of the arch folder contains source files specifically for the IAR development environment. The files contain the target specific stubbed interfaces, for the embedded host microcontroller, and a main implementation. These files require a target specific implementation to obtain a functional VaultIC eLib.

11.3 Arch/pc

The PC subfolder of the arch folder contains source files specifically for the PC development environment. These files are further broken down into Linux, Mac, shared, and win32.

These files provide the implementation for communications and target specific library code for the VaultIC eLib.

11.4 Build

The build folder contains development environment projects for each of the supported platforms:

- Embedded – IAR embedded project, target independent.
- Linux – Netbeans .so library development project.
- Mac – Netbeans .dylib development project.
- Win32 – Visual studio 2005,2012, DLL library development project.

11.5 Common

The common folder contains all of the platform independent source code files. These files should require no modification to compile on the selected host target.

11.6 Device/vaultic_4XX_family

The *vaultic_4XX_family* subfolder of the device folder contains family specific files that are different as a result of interface and behavioural differences between the VaultIC families.

11.7 Docs

The Docs folder contains this document, and other documents that help the host developer implement their intended application.

12. Troubleshooting

The VltLibraryInit method is returning a failure code.

1. If you are using the Total Phase Aardvark or Cheetah, make sure the dll or so, is in the executable folder, or on the system search path.
2. If you are using a PCSC supported reader, ensure the Microsoft PCSC and appropriate reader drivers are installed and working.
3. Check the delay for the self tests is sufficient. If you are unsure how long the delay should be, please contact Inside Secure VaultIC support representative.
4. Check the block protocol parameters and TWI parameters are setup correctly. If you are unsure what values to use, please contact Inside Secure VaultIC support representative.

Communications using the ISO7816 T1 protocol fail

1. If you are using the Linux shared object or Mac OS dynamic library, the ISO7816 T1 protocol does not work. This issue is due to the pcsc-lite library.
2. If you are using USB communications, the VaultIC firmware does not support the ISO7816 T1 protocol.

Communications randomly fail

1. Check the block protocol parameters and SPI or TWI parameters are setup correctly. If you are unsure what values to use, please contact Inside Secure VaultIC support representative.

Methods are returning Errors when the correct arguments have been supplied

1. If the arguments passed to VaultIC Raw API methods are correct, and an error is returned. Check the `VAULT_IC_VERSION` defined in *vaultic_config.h* matches the firmware version of the VaultIC Security Module. If you are unsure of the firmware version, please contact Inside Secure VaultIC support representative.
2. If the arguments passed to VaultIC Raw API methods are correct, and an error is returned. Wipe the file system and users from the device, and personalise the VaultIC Security Module. If the issue persists or you are unsure how to wipe the file system, please contact Inside Secure VaultIC support representative.

13. Operational Constraints

13.1 Multithreading/Multi-tasking

The implementation of the VaultIC eLib does not support multithread or multitasking access.

The VaultIC eLib supports a variety of platforms; their underlying resources are vastly different in implementation and behaviour. Any attempt to unify these differences will be resource costly without any real benefit as their efficiency is likely to suffer.

13.2 Ciphers

The VaultIC eLib provides a number of basic ciphers such as AES, DES, and TDES, every effort has been made to ensure these implementations are fit for purpose. However, the security of these implementations and key management considerations depend on the specific target environment. These implementations are provided as they are, however it solely the responsibility of the end user to ensure they are integrated in a secure environment and are used in a secure manner.

14. Support & Contact Us

You can contact us by sending an email to e-security@insidefr.com

Reference List

[R1]	VaultIC Technical Datasheet	TPR0544

Revision History

Document Details

Title: Easy Plug - VaultIC eLib for 4XX

Literature Number: TMP_DIS_1302_001 v0.5

Date: Nov 19, 2013

- **Revision 1.0 :**
  - First release

Headquarters

Inside Secure
Arteparc de Bachasson - Bat A
Rue de la Carrière de Bachasson
CS 70025
13590 Meyreuil - France
Tel: +33 (0)4-42-905-905
Fax: +33 (0)4-42-370-198

Product Contact

Web Site
www.insidesecure.com

Technical Support
e-security@insidefr.com

Sales Contact
sales_web@insidefr.com