

User Guide

TMP_DIS_1302_001 v0.5

EASY PLUG - VAULTIC ELIB FOR 1XX

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).



INSIDE REGISTERED CONFIDENTIAL PROPRIETARY - DO NOT COPY

Table of Contents

1	Glossary	7
2	Introduction	8
3	Reading this manual	9
4	VaultIC eLib Architecture	10
5	VaultIC eLib Communications Layer	11
6	Getting Started	12
6.1	Demonstration Code	12
6.2	Deployment Options	12
6.2.1	Library Linkage.....	12
6.2.2	Direct Source Code "Embedded" Linkage.....	12
7	VaultIC eLib Integration	13
7.1	PC Library Integration (All Platforms)	13
7.2	Windows Integration	13
7.3	Linux Integration	14
7.4	Mac OS Integration	14
7.5	Embedded Integration	14
8	Configuration & Customisation	15
8.1	Configurable Features	15
8.1.1	VLT_ENDIANNESSE	15
8.1.2	VLT_PLATFORM	15
8.2	Build Customisation	15
8.2.1	Fast CRC Calculations.....	15
8.2.2	Default ECDSA Curve Domain Parameters	16
8.3	Communication mode	16
8.4	Multi-slot support	16
9	VaultIC eLib Methods	17
9.1	Overview	17
9.1.1	Error & Status Code	17
9.2	Library Methods	17
9.2.1	VltInitLibrary	17
9.2.2	VltCloseLibrary.....	18
9.2.3	VltGetLibraryInfo	18
9.2.4	VltGetApi	19
9.2.5	VltGetStrongAuthentication	19
9.2.6	VltFindDevices	20
9.2.7	VltGetCrc16.....	21

9.3	Embedded Host Configuration Methods	22
9.3.1	VltApilnit	22
9.3.2	VltApiClose	23
9.3.3	VltCardEvent	23
9.3.4	VltSelectCard	24
9.4	Strong Authentication Service	24
9.4.1	Conditions of Use	25
9.4.2	VltStrongSetCryptoParams	25
9.4.3	VltStrongAuthenticate	26
9.4.4	VltStrongClose	26
9.4.5	VltStrongGetState	27
9.5	Base API	27
9.5.1	Identity Authentication	27
9.5.2	Cryptographic Services	32
9.5.3	VaultIC 100 Secure Counters	42
9.5.4	File System	45
9.5.5	Manufacturing Process	47
10	Client Stub Interfaces	55
10.1	vaultic_ecdsa_signer	55
10.2	vaultic_mem	55
10.2.1	host_memcpy	56
10.2.2	host_memset	56
10.2.3	host_memcmp	57
10.2.4	host_memxor	57
10.2.5	host_memcpyxor	58
10.2.6	host_lshift	59
10.3	vaultic_timer_delay	59
10.3.1	VltSleep	59
10.4	vaultic_iso7816_protocol	60
10.4.1	VltIso7816PtcInit	60
10.4.2	VltPtcClose	61
10.4.3	VltIso7816PtcSendReceiveData	61
10.5	vaultic_twi_peripheral	61
10.5.1	VltTwiPeripheralInit	62
10.5.2	VltTwiPeripheralClose	62
10.5.3	VltTwiPeripheralIoctl	62
10.5.4	VltTwiPeripheralSendData	63
10.5.5	VltTwiPeripheralReceiveData	64
11	VaultIC eLib Deployment	65
11.1	Binaries	65
11.2	Arch/embedded	66
11.3	Arch/pc	66
11.4	Build	66
11.5	Common	66
11.6	Device/vaultic_1XX_family	66

11.7	Docs	66
12	<i>Troubleshooting</i>	67
13	<i>Constraints & Limitations</i>	68
13.1	Multithreading/Multi-tasking	68
13.2	ECDSA Sign and Verify	68
13.3	One-Wire Interface	68
14	<i>Support & Contact Us</i>	69
	<i>Reference List</i>	71
	<i>Revision History</i>.....	73



INSIDE REGISTERED CONFIDENTIAL PROPRIETARY - DO NOT COPY

1. Glossary

Table 1-1.

Term	Detailed Description
API	Application Programming Interface
TWI	Two Wire Interface
DLL	Dynamic Link Library
SO	Shared Object library (typically seen in BSD Linux/Unix/MAC OS systems)
XOR	Bitwise Exclusive Or
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

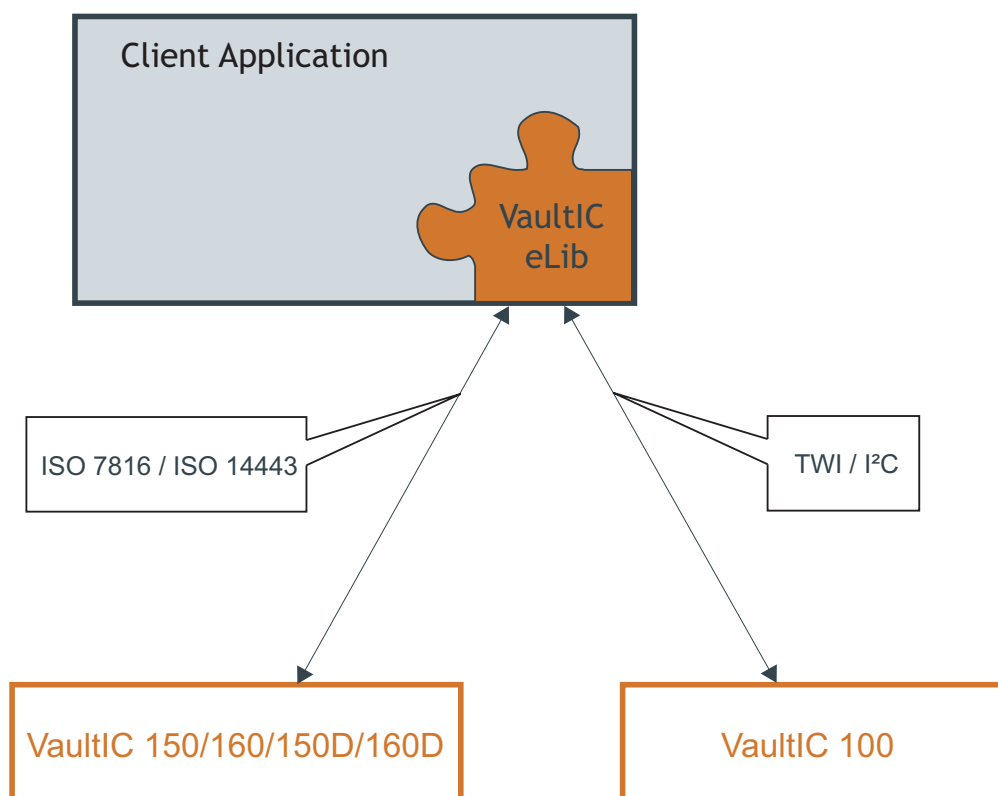
2. Introduction

The **VaultlC 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 **VaultlC eLib** is to provide a programmatic interface to the command set supported by the **VaultlC Security Module** product family. As the **VaultlC Security Module** supports a wide range of communication interfaces, the primary function of the **VaultlC 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 **VaultlC eLib** provides a unique and easy to use interface that allows the customer to readily integrate **VaultlC Security Module** functionality in their own applications. This interface attempts to shield the customer application from the complexity of the underlying **VaultlC 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 VaultIC1XX 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.

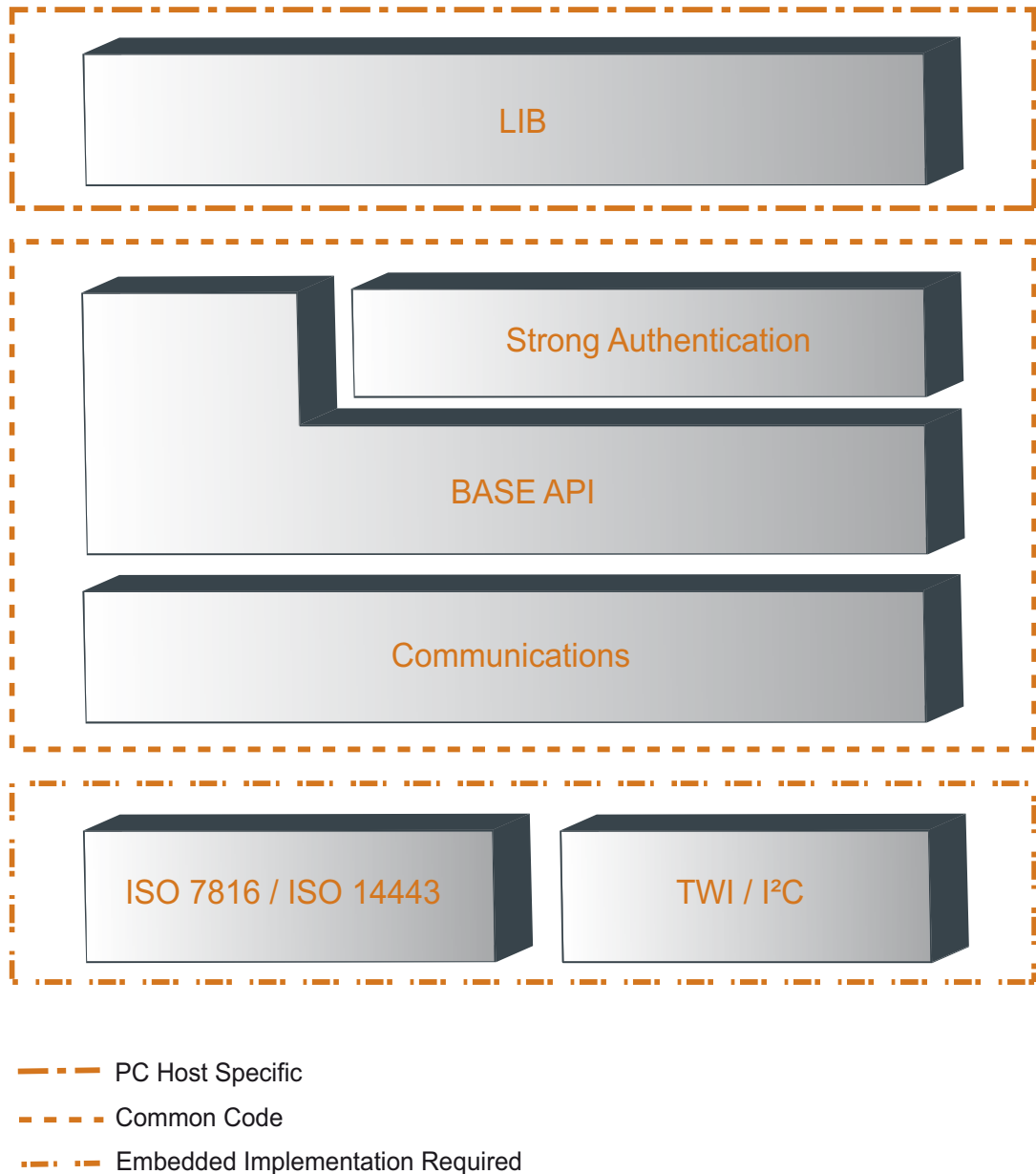
Support & Contact Us

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.

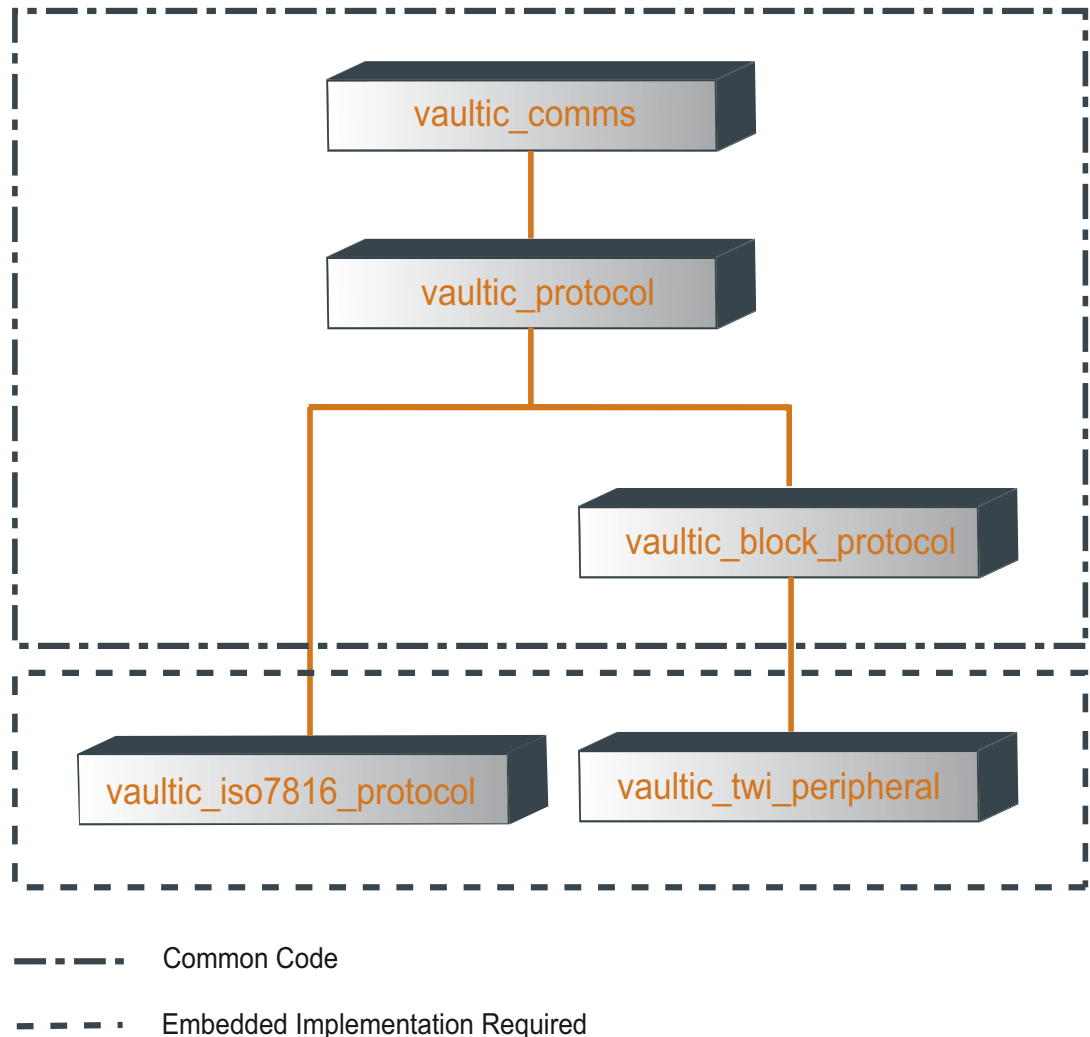


[Figure 4-1](#) has links to the relevant sections of the document to better describe the specific areas of the **VaultIC eLib**. The “Common Code” can be configured to only include functionality that is required by the client application. Instructions on how to achieve this can be found in the [Configuration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



The module within the “Embedded Implementation Required” i.e. `vaultic_iso7816_protocol`, `vaultic_twi_peripheral` map to the client stub interfaces detailed in the [Client Stub Interfaces](#) section.

The modules within the “Common Code” 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_PersonaliseSampleCode	Demonstrates how to personalize the VaultIC device to use a user-defined curve in non-FIPS mode
Xxx_FipsPersonaliseSampleCode	Demonstrates how to personalize the VaultIC device to use a built-in curve in FIPS mode
Xxx_AuthenticateSampleCode	Demonstrates how to perform strong mutual authentication using user-defined curve domain parameters in non-FIPS mode
Xxx_FipsAuthenticateSampleCode	Demonstrates how to perform strong mutual authentication using built-in domain parameters in FIPS mode.
Xxx_SampleCode	Gives examples of calling all the base API functions and services
Xxx_ActivateSampleCode	Demonstrates how to activate a deactivated VaultIC device.
Xxx_DeactivateSampleCode	Demonstrates how to deactivate 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_i2csapi/

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

The PCSC lite library for Linux can be found in <http://pcsc-lite.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)

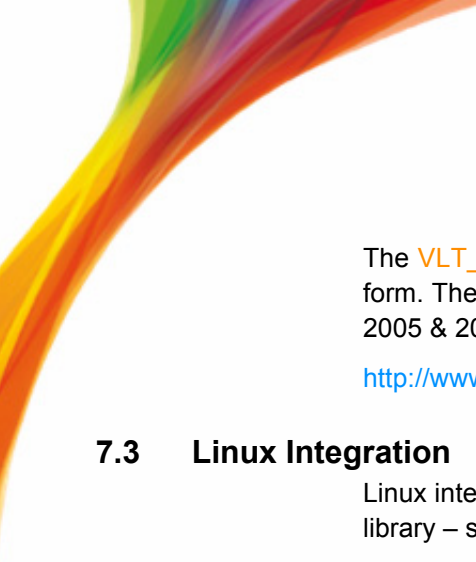
The libraries on Windows, Linux and MacOS export the following functions as the library entry points:

- **VltInitLibrary**
- **VltCloseLibrary**
- **VltGetLibraryInfo**
- **VltGetApi**
- **VltGetStrongAuthentication**
- **VltFindDevices**
- **VltGetCrc16**

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 configure 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 **VaultIC eLib** is configured by calling the following method:

- **VltApilnit**

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_ecdsa_signer

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 endianness 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.

8.2 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 described in the following sections.

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 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 a slightly slower, but smaller code implementation can be used by setting this definition as

```
#define VLT_ENABLE_FAST_CRC16CCIT VLT_DISABLE
```

8.2.2 Default ECDSA Curve Domain Parameters

The VaultIC1XX device family supports selection of one of set of default FIPS compliant ECDSA curve domain parameters. The API also defines these domain parameters for use in the sign/verify phases of strong authentication. Some code and data space can be saved by disabling inclusion of the curves that are not needed. For example, if we know that all devices will be pre-personalised to use the 283-bit Koblitz curve, we can disable the inclusion of the other curve types by making the following declarations in vaultic_config.h:

```
/* only need 283-bit Koblitz curve support */
#define VLT_ENABLE_ECDSA_K233 VLT_DISABLE
#define VLT_ENABLE_ECDSA_K283 VLT_ENABLE
#define VLT_ENABLE_ECDSA_B233 VLT_DISABLE
#define VLT_ENABLE_ECDSA_B283 VLT_DISABLE
```

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. VaultlC eLib Methods

This section provides information on the methods supported by the **VaultlC 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.

Strong Authentication Service

These services simplify use of the authentication protocols supported by the **VaultlC Security Module**. In any anti-counterfeiting type application these may be the only methods that the application might call. These services are implemented using the appropriate Base API methods.

Base API

This section describes the Base API methods which supply a one to one mapping with the commands exposed by the **VaultlC Security Module**. As an example the **VltSubmitPassword** method provides the interface to the Submit Password command documented in the **VaultlC 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 **VaultlC eLib** library while status values smaller than **VLT_OK** are the APDU status words that are returned by the **VaultlC 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 **VaultlC 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.



The *VltInitLibrary* method is used to initialize the host PC library, by establishing communications with the **VaultlC Security Module**. See the sample code for an example of using this function.

Requirements

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

9.2.2 VltCloseLibrary

The *VltCloseLibrary* method is responsible for closing the **VaultlC 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.



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* 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.



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.



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 VltGetStrongAuthentication

The *VltGetStrongAuthentication* 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_STRONG_AUTHENTICATION structure.

Syntax

```
VAULTIC_STRONG_AUTHENTICATION* VltGetStrongAuthentication( void );
```

Parameters

None

Return Value

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



The *VltGetStrongAuthentication* 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 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 *VltApilnit* 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.



The *VltFindDevices* method is a helper function that can be called before calling *VltApilnit*, 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 VltInit method.



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>  
        <peripheral idx="01">2237367164</peripheral>  
    </interface>  
</devices>
```

Requirements

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

9.2.7 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.



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.



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 TWI_PARAMS(x) \
x.Params.VltBlockProtocolParams.VltPeripheralParams.PeriphParams.VltTwiParams
#define BLOCK_PARAMS (x) x.Params.VltBlockProtocolParams
VLT_INIT_COMMS_PARAMS commsParams;
VLT_STS status = VLT_FAIL;
// Setup the comms init structure for TWI.
commsParams.u8CommsProtocol = VLT_TWI_COMMS;
BLOCK_PARAMS(commsParams).u16BitRate = 400;
BLOCK_PARAMS(commsParams).u8ChecksumMode = BLK_PTCL_CHECKSUM_SUM8;
TWI_PARAMS(commsParams).u16BusTimeout = 450;
TWI_PARAMS(commsParams).u8Address = 0x5E;
TWI_PARAMS(commsParams).u32msTimeout = 4000;

```

```
// Setup the max wait time for fips mode for the VaultIC 1XX to become
// operational Appendix H in the VaultIC datasheet lists the required wait
// times for other device modes.
BLOCK_PARAMS(commsParams).u16msSelfTestDelay = 800;
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.

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(
    VLT_PU8 pu8ReaderName,
    DWORD dwTimeout,
    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.



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(  
    SCARDHANDLE hScard,  
    SCARDCONTEXT hCtx,  
    DWORD dwProtocol  
);
```

Parameters

`hScard` [in]
Smart card handle value

`hCtx` [in]
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.

Requirements

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

9.4 Strong Authentication Service

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

- Unilateral Authentication
- Mutual Authentication in FIPS mode
- Mutual Authentication in non-FIPS mode

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.

9.4.1 Conditions of Use

If this service is used, the following rules **must** be adhered to:

If the **VltStrongAuthenticate** method is successfully called, the **VltStrongClose** 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**
- **VltExternalAuthenticate**

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

9.4.2 VltStrongSetCryptoParams

The *VltStrongSetCryptoParams* method is used to initialise the cryptographic parameters structure with one of the builtin set supported by the device

Syntax

```
VLT_STS VltStrongSetCryptoParams(
    _in VLT_U8 paramId,
    _in VLT_SA_CRYPTOPARAMS* pCryptoParams
);
```

Parameters

paramId [in]

Identifier indication which parameter set is required. For ECDSA based authentication on the VaultIC1XX device the following identifiers are available:

```
VLT_ECDSA_CURVE_B233 - for 233-bit random binary curve
VLT_ECDSA_CURVE_K233 - for 233-bit Koblitz curve
VLT_ECDSA_CURVE_B283 - for 283-bit random binary curve
VLT_ECDSA_CURVE_K283 - for 283-bit Koblitz curve
pCryptoParams [in/out]
```

A pointer to a user allocated VLT_SA_CRYPTOPARAMS structure which will receive the parameter data

Return Value

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



Caution

This method should only be used when the VaultIC device has been pre-initialised to use one of the built-in default cryptographic parameter sets. Refer to the device technical datasheet for more details.

Requirements

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

See Also

[VltStrongClose](#)

9.4.3 VltStrongAuthenticate

The *VltStrongAuthenticate* method is used to invoke the requested authentication protocol

Syntax

```
VLt_STS VltStrongAuthenticate(  
    _in VLt_SA_CRYPT_PARAMS* pCryptoParams,  
    _in VLt_SA_PROTOCOL_PARAMS* pProtocolParams  
);
```

Parameters

pCryptoParams [in]

Cryptographic parameters

pProtocolParams [in]

Authentication protocol parameters

Return Value

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



This service is used to perform either unilateral or mutual authentication. Both FIPS and non-FIPS modes are supported.

The [VltStrongSetCryptoParams](#) method can be used to pre-initialise the VLt_SA_CRYPT_PARAMS structure when using one of the default parameter sets built into the VaultIC Security Module.

On successful completion of a mutual authentication, the user will be authenticated (logged into) to *VaultIC Security Module*, providing access to additional methods. Please see the technical datasheet for more details of permissions required to execute certain commands.

Requirements

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

See Also

[VltStrongClose](#)

9.4.4 VltStrongClose

The *VltStrongClose* method closes the strong authentication service and cancels any existing authentication with the *VaultIC Security Module*.

Syntax

```
VLt_STS VltStrongClose ( void );
```

Return Value

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



Logs the current user out of the VaultIC Security Module and frees up any resources associated with the authentication.

Requirements

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

See Also

[VltStrongAuthenticate](#)

9.4.5 VltStrongGetState

The *VltStrongGetState* method returns the state of the strong authentication service.

Syntax

```
VLT_STS VltStrongGetState (
    _out VLT_PU8 pu8State
);
```

Parameters

pu8State [out]

A pointer to a user allocated variable which will receive the enumerated value representing the current state:

```
#define VLT_USER_NOT_AUTHENTICATED (VLT_U8)0x00
#define VLT_USER_AUTHENTICATED (VLT_U8)0x01
```

Return Value

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



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

Requirements

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

9.5 Base API

9.5.1 Identity Authentication

9.5.1.1 VltSubmitPassword

This method authenticates a user using a password.

Syntax

```
VLT_STS VltSubmitPassword(
    _in VLT_U8 u8PasswordLength,
```

```
_in const VLT_U8 *pu8Password
);
```

Parameters

u8PasswordLength [in]
Password Length. (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.

Requirements

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

See Also

[VltCancelAuthentication](#)

9.5.1.2 VltManageAuthenticationData

This 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.



Refer to the *VaultIC Secure 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

9.5.1.3 VltGetAuthenticationInfo

This 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_CREATOR
- VLT_USER

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.

Requirements

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

9.5.1.4 VltCancelAuthentication

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

Syntax

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

Return Value

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



Logs out the currently authenticated user.

Requirements

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

9.5.1.5 VltInternalAuthenticate

This method can be used alone to perform a generic Device Unilateral Authentication of the *VaultIC Security Module*, or may be followed by a *VltExternalAuthenticate* method call in order to perform a Mutual authentication with the host.

Syntax

```
VLT_STS VltInternalAuthenticate(
    _in VLT_U8 u8UserID,
    _in VLT_U8 u8RoleID,
    _in const VLT_U8 *pu8HostChallenge,
    _out VLT_PU8 pu8DeviceChallenge,
    _inout VLT_PU16 pu16SignatureLength,
    _out VLT_PU8 pu8Signature
);
```

Parameters

u8UserID [in]

Operator ID. Possible values are:

- 0x00
- VLT_USER

u8RoleID [in]

Role ID. Possible values are

- 0x00
- VLT_APPROVED_USER

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.

Requirements

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

See Also

[VltExternalAuthenticate](#)

9.5.1.6 VltExternalAuthenticate

This method is used after [VltInternalAuthenticate](#) to complete a generic Mutual Authentication.

Syntax

```
VLT_STS VltExternalAuthenticate(  
    _in VLT_U8 u8UserID,  
    _in VLT_U8 u8RoleID,  
    _in VLT_U8 u8HostChallengeLength,  
    _in const VLT_U8 *pu8HostChallenge,  
    _in VLT_U16 u16HostSignatureLength,  
    _in const VLT_U8 *pu8HostSignature  
);
```

Parameters

u8UserID [in]

Operator ID. Possible values are:

- VLT_USER

u8RoleID [in]

Role ID. Possible values are

– VLT_APPROVED_USER

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.



This command is used after a [VltInternalAuthenticate](#) to complete a Mutual Authentication protocol. This command gets a signature from a host and a host challenge Ch, and attempts verification. It returns the result of this verification.

Requirements

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

See Also

[VltInternalAuthenticate](#)

9.5.1.7 VltInitializeAlgorithm

This 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_DIGEST_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.



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.



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*



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

VltGenerateSignature

VltVerifySignature

VltComputeMessageDigest

VltInternalAuthenticate

9.5.2 Cryptographic Services

9.5.2.1 VltPutKey

This 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.



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



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

Requirements

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

See Also

[VltReadKey](#)

[VltInitializeAlgorithm](#)

9.5.2.2 VltReadKey

This 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.



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

Requirements

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

See Also

[Cryptographic Services](#)

[VltInitializeAlgorithm](#)

9.5.2.3 VltGenerateAssurance

This method generates an assurance message for private key possession assurance.

Syntax

```
VLT_STS VltGenerateAssurance(  
    _inout VLT_PU8 pu8SignerIdLength,  
    _inout VLT_PU8 pu8SignerID,  
    _out VLT_ASSURANCE_MESSAGE* pAssuranceMsg,  
);
```

Parameters

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.

Requirements

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

9.5.2.4 VltVerifyAssurance

This method verifies an assurance message for private key possession assurance.

Syntax

```
VLT_STS VltVerifyAssurance(  

```

```

_in VLT_U8 u8SignedAssuranceMsgLength,
_in VLT_PU8 pu8SignedAssuranceMsg,
);

```

Parameters

u8SignedAssuranceMsgLength [in]
The length of the signed assured message

pu8SignedAssuranceMsg [in]
The signed assured.

Return Value

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



This command verifies an assurance message for private key possession assurance.



This command can only be made by a user who's role is approved.

9.5.2.5**VltGenerateSignature**

This method generates a signature from a message.

Syntax

```

VLT_STS VltGenerateSignature(
_in VLT_U32 u32MessageLength,
_in const VLT_U8 *pu8Message,
_inout VLT_PU16 pul6SignatureLength,
_in VLT_PU8 pu8Signature
);

```

Parameters

u32MessageLength [in]
Message length.

pu8Message [in]
Message buffer.

pul6SignatureLength [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.



Note

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



Caution

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.

Requirements

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

See Also

VltInitializeAlgorithm

VltVerifySignature

9.5.2.6

VltUpdateSignature

This 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.



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](#)

9.5.2.7 *VltComputeSignatureFinal*

This method finishes a multiple-part signature operation, returning the signature.

Syntax

```
VLT_STS VltComputeSignatureFinal(
    _inout VLT_PU16 pul6SignatureLength,
    _out VLT_PU8 pu8Signature
);
```

Parameters

`pul6SignatureLength [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.

Requirements

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

See Also

[VltUpdateSignature](#)

9.5.2.8 *VltVerifySignature*

This 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.



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



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.

Requirements

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

See Also

VltInitializeAlgorithm

VltGenerateSignature

9.5.2.9 VltUpdateVerify

This 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.



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.

Requirements

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

See Also

[VltComputeVerifyFinal](#)

[VltUpdateSignature](#)

9.5.2.10 *VltComputeVerifyFinal*

This 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 length.

pu8Signature [in]

Signature buffer

Return Value

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

Requirements

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

See Also

[VltUpdateVerify](#)

9.5.2.11 *VltComputeMessageDigest*

This 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.



The **VltInitializeAlgorithm** command 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

9.5.2.12 VltUpdateMessageDigest

This 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.



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.

Requirements

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

See Also

VltComputeMessageDigestFinal

9.5.2.13 *VltComputeMessageDigestFinal*

This 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.

Requirements

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

See Also

[VltUpdateMessageDigest](#)

9.5.2.14 *VltGenerateRandom*

This 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.



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

Requirements

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

9.5.2.15 VltGenerateKeyPair

This 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 VLT_U8 u8PrivateKeyGroup,  
    _in VLT_U8 u8PrivateKeyIndex,  
    _in const VLT_KEY_GEN_DATA *pKeyGenData  
);
```

Parameters

u8PublicKeyGroup [in]

Public key group index

u8PublicKeyIndex [in]

Public key index

u8PrivateKeyGroup [in]

Private key group index

u8PrivateKeyIndex [in]

Private key index

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.



This command can be used only on CREATION mode

Requirements

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

9.5.3 VaultIC 100 Secure Counters

9.5.3.1 VltSetSecureCounters

This command initialize the content of the counters.

Syntax

```
VLT_STS VltSetSecureCounters(  
    VLT_U8 u8CounterNumber,  
    VLT_U8 u8CounterGroup,
```

```
VLT_U8 u8CounterValueLen,
VLT_PU8 pu8CounterValue);
```

Parameters

u8CounterNumber [in]

Counter number.

u8CounterGroup [in]

Counter Group.

u8CounterValueLen [in]

Counter value length.

pu8CounterValue [in]

Counter value.

Return Value

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



This command is accessible only if the VaultIC module is in CREATION state and the Counter Mode choice is set to DIRECT mode.

9.5.3.2 VltDecrementCounter

Depending of Counter Mode choice, this method:

- Decrements one counter with the provided amount in COUNTER mode.
- Reads the content of the counter in DIRECT mode.

Syntax

```
VltDecrementCounter(
    VLT_U8 u8CounterNumber,
    VLT_U8 u8CounterGroup,
    VLT_COUNTER_DATA* pCounterData,
    VLT_COUNTER_RESPONSE* pCounterResponse );
```

Parameters

u8CounterNumber [in]

Counter number.

u8CounterGroup [in]

Counter Group.

pCounterData [in]

Nounce and amount values.

pCounterResponse [out]

Returned counter value, RND value and signature.

Return Value

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



In DIRECT mode, The amount must be set to zero, other values are forbidden.

9.5.3.3 VltIncrementCounter

Depending of Counter Mode choice, this method:

- Increments one counter with the provided amount in COUNTER mode.
- Writes directly in the counter the provided amount in DIRECT mode.

Syntax

```
VltIncrementCounter(  
    VLT_U8 u8CounterNumber,  
    VLT_U8 u8CounterGroup,  
    VLT_COUNTER_DATA* pCounterData,  
    VLT_U16 u16Signaturelen,  
    VLT_PU8 pu8Signature,  
    VLT_COUNTER_RESPONSE* pCounterResponse );
```

Parameters

u8CounterNumber [in]

Counter number.

u8CounterGroup [in]

Counter Group.

pCounterData [in]

Nounce and amount values.

u16Signaturelen [in]

Signature length.

pu8Signature [in]

Signature buffer.

pCounterResponse [out]

Returned counter value, RND value and signature.

Return Value

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



In COUNTER mode, The command is performed only if no overflow is detected.

9.5.4 File System

9.5.4.1 *VltSelectFile*

This method selects a file.

Syntax

```
VLT_STS VltSelectFile (
    _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.



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.

Requirements

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

9.5.4.2 *VltWriteFile*

This method writes data to the currently selected file.

Syntax

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

Parameters

pu8Data [in]

Data to write

u8DataLength [in]

Length of the data (1..255)

Return Value

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



Updates part of the contents of the current file, from the current file position.

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](#)

9.5.4.3 VltReadFile

This 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.



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.



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](#)

9.5.4.4 VltSeekFile

This 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.



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

[VltWriteFile](#)

[VltReadFile](#)

9.5.5 Manufacturing Process

9.5.5.1 VltGetInfo

This 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.



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`

9.5.5.2 *VltSelfTest*

This 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.



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.



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



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`

9.5.5.3 *VltSetStatus*

This 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
- VLT_TERMINATED

Return Value

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

Requirements

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

9.5.5.4 *VltSetConfig*

This method sets VaultIC configuration parameters.

Syntax

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

Parameters

u8ConfigItem [in]

Value of the parameter to set, possible value are:

- VLT_FIPS_MODE
- VLT_ECDSA_DOMAIN_PARAM_CHOICE
- VLT_LOCK_SUBMIT_PASSWORD
- VLT_USER_FILE_ACCESS
- VLT_COMMS_CHANNEL_ACCESS
- VLT_RF_BAUD_RATE
- VLT_RF_TIMEOUTS

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.



Note

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



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

Requirements

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

9.5.5.5 *VltDeactivate*

This method changes the chip life cycle state to DEACTIVATED.

Syntax

```
VLT_STS VltDeactivate (  
    _in VLT_PU8 pu8KeyId,  
    _in VLT_U8 u8KeyIdLength,  
    _in VLT_PU8 pu8KeyData,  
    _in VLT_U8 u8KeyDataLength,  
);
```

Parameters

pu8KeyId [in]

Data buffer containing Key Id value

u8KeyIdLength [in]

Length of key Id buffer

pu8KeyData [in]

Data buffer containing deactivation key value

u8KeyDataLength [in]

Length of key data buffer

Return Value

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

Requirements

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

See Also

[VltActivate](#)

9.5.5.6 *VltActivate*

This method changes the chip life cycle state to ACTIVATED.

Syntax

```
VLT_STS VltActivate (  
    _in VLT_PU8 pu8Challenge,  
    _in VLT_U8 u8ChallengeLength,  
    _in VLT_PU8 pu8KeyData,
```

```

    _in VLT_U8 u8KeyDataLength,
);

```

Parameters

`pu8Challenge [in]`
 Data buffer containing challenge value
`u8ChallengeLength [in]`
 Length of challenge buffer
`pu8KeyData [in]`
 Data buffer containing activation key value
`u8KeyDataLength [in]`
 Length of key data buffer

Return Value

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



When this command succeed, chip life cycle is set to ACTIVATED.



A `VltGetChallenge` command must be sent before this command.

Requirements

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

See Also

[VltGetChallenge](#)

[VltDeactivate](#)

9.5.5.7 VltGetChallenge

This method return a 16-bytes challenge used for re-activation process.

Syntax

```

VLT_STS VltGetChallenge (
    _out VLT_PU8 pu8Challenge,
    _out VLT_U8 u8ChallengeLength
);

```

Parameters

`pu8Challenge [out]`
 Data buffer containing challenge value
`pu8ChallengeLength [out]`

Length of challenge buffer

Return Value

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

Requirements

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

See Also

[VltActivate](#)

9.5.5.8 VltTestCase1

This 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.



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](#)

9.5.5.9 VltTestCase2

This 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.



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](#)

9.5.5.10 *VltTestCase3*

This 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.



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](#)

9.5.5.11 VltTestCase4

This 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.



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](#)

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_ecdsa_signer

The API provides an example implementation of ECDSA signing and verifying code. This is a fully ANSI 'C' implementation derived from freely available source code. As such, it is not warranted or guaranteed to be either secure or efficient in terms of speed of execution or code size. It is likely that a host environment may have access to a more secure or efficient cryptographic library. In which case, the provided ECDSA can be replaced.

The ECDSA sign/verify interface is declared in **vaultic_ecdsa_signer.h** and the default implementation is provided in the **vaultic_ecdsa_signer.c**.

The implementation uses the following files:

Table 10-1.

vaultic_bigdigits.c & .h	A big integer arithmetic implementation
vaultic_ff2n.c & .h	Finite-field arithmetic functions in polynomial basis
vaultic_elliptic-ff2n.c & .h	Elliptic curve arithmetic functions
vaultic_sha256.c & .h	A software implementation of SHA-256 message digest

These maybe removed from the API build if an alternative cryptographic implementation is used.

Alternative implementations of the following interface functions will be required:

```
VLT_STS EcdsaSignerInit(
    const VLT_ECDSA_DOMAIN_PARAMS* pDomainParams,
    const VLT_ECDSA_PRIVATE_KEY* pPrivateKey,
    const VLT_ECDSA_PUBLIC_KEY* pPublicKey,
    VLT_U8 u8OpMode);
VLT_STS EcdsaSignerClose( void );
VLT_STS EcdsaSignerDoFinal(
    VLT_PU8 pu8Message,
    VLT_U32 u32messageLen,
    VLT_U32 u32messageCapacity,
    VLT_PU8 pu8Signature,
    VLT_PU32 pu32SignatureLen,
    VLT_U32 u32SignatureCapacity );
```

Please refer to the implementation in **vaultic_ecdsa_signer.c** for more details. These functions are used exclusively in the strong authentication service defined in **vaultic_ecdsa_strong_authentication.c**.

10.2 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 API code.

10.2.1 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.

Remarks

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`.

10.2.2 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.

Remarks

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`.

10.2.3 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.

Remarks

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`.

10.2.4 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.

Remarks

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`.

10.2.5 `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.

Remarks

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`.

10.2.6 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.

Remarks

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.

10.3 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.

10.3.1 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.

Remarks

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.

10.4 vaultic_iso7816_protocol

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

10.4.1 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 pul6MaxSendSize,  
    _out VLT_PU16 pul6MaxReceiveSize
```

Parameters

`pInitCommsParams` [in]

The communications initialisation parameters.

`pOutData` [in]

This parameter is not used.

`pInData` [in]

This parameter is not used.

`pul6MaxSendSize` [out]

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

`pul6MaxReceiveSize` [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.



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 `pul6MaxSendSize` argument should be populated with the maximum number of APDU Data In bytes that the host can send. The `pul6MaxReceiveSize` argument should be populated with the maximum number of APDU Data Out bytes that the host can receive.



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

10.4.2 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.



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

10.4.3 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.



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.

10.5 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 initialising 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.

Remarks

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

10.5.3 VltTwiPeripheralIoctl

The `VltTwiPeripheralIoctl` method is responsible 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.

Remarks

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.

Remarks

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.

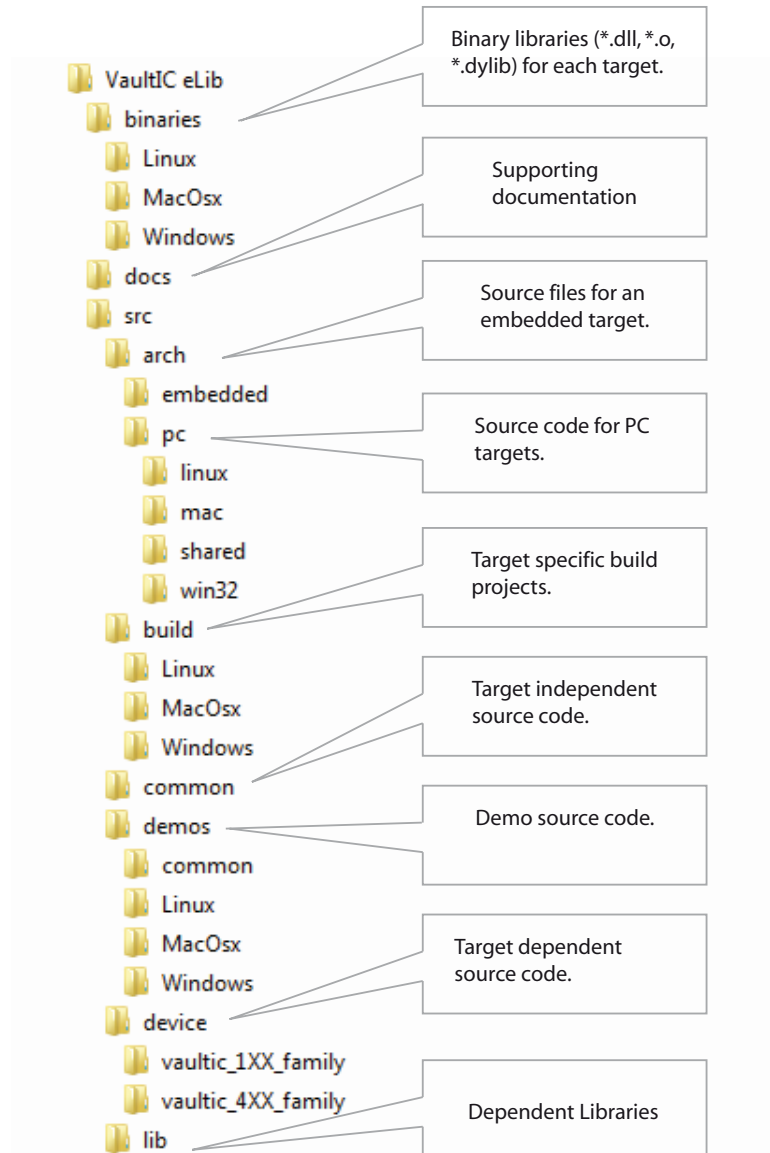
Remarks

This method should receive the data from the **VaultIC Security Module** and populate the *pInData* 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. VaultlC eLib Deployment

The API is deployed using an installer package. Once 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_1XX_family

The *vaultic_1XX_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, 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 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. Wipe the file system and users from the device, and personalise the VaultIC Security Module.
2. If the issue persists or you are unsure how to wipe the file system, please contact Inside Secure VaultIC support representative.



13. Constraints & Limitations

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 ECDSA Sign and Verify

The **VaultIC eLib** provides an example implementation of ECDSA sign and verify functions. Every effort has been made to ensure that this implementation is fit for purpose. However, the security of this implementation and key management considerations depend on the specific target environment. These implementations are provided “as is” and it is solely the responsibility of the end user to ensure they are integrated in a secure environment and are used in a secure manner.

13.3 One-Wire Interface

The API does not support communications using the One-Wire Interface since any implementation would be host and interface hardware specific.

14. Support & Contact Us

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

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

Disclaimer: All products are sold subject to Inside Secure Terms & Conditions of Sale and the provisions of any agreements made between Inside Secure and the Customer. In ordering a product covered by this document the Customer agrees to be bound by those Terms & Conditions and agreements and nothing contained in this document constitutes or forms part of a contract (with the exception of the contents of this Notice). A copy of Inside Secure's Terms & Conditions of Sale is available on request. Export of any Inside Secure product outside of the EU may require an export Licence.

The information in this document is provided in connection with Inside Secure products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Inside Secure products. EXCEPT AS SET FORTH IN INSIDE SECURE'S TERMS AND CONDITIONS OF SALE, INSIDE SECURE OR ITS SUPPLIERS OR LICENSORS ASSUME NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL INSIDE SECURE BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, LOSS OF REVENUE, BUSINESS INTERRUPTION, LOSS OF GOODWILL, OR LOSS OF INFORMATION OR DATA) NOTWITHSTANDING THE THEORY OF LIABILITY UNDER WHICH SAID DAMAGES ARE SOUGHT, INCLUDING BUT NOT LIMITED TO CONTRACT, TORT (INCLUDING NEGLIGENCE), PRODUCTS LIABILITY, STRICT LIABILITY, STATUTORY LIABILITY OR OTHERWISE, EVEN IF INSIDE SECURE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Inside Secure makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Inside Secure does not make any commitment to update the information contained herein. Inside Secure advises its customers to obtain the latest version of device data sheets to verify, before placing orders, that the information being relied upon by the customer is current. Inside Secure products are not intended, authorized, or warranted for use as critical components in life support devices, systems or applications, unless a specific written agreement pertaining to such intended use is executed between the manufacturer and Inside Secure. Life support devices, systems or applications are devices, systems or applications that (a) are intended for surgical implant to the body or (b) support or sustain life, and which defect or failure to perform can be reasonably expected to result in an injury to the user. A critical component is any component of a life support device, system or application which failure to perform can be reasonably expected to cause the failure of the life support device, system or application, or to affect its safety or effectiveness.

The security of any system in which the product is used will depend on the system's security as a whole. Where security or cryptography features are mentioned in this document this refers to features which are intended to increase the security of the product under normal use and in normal circumstances.

© Inside Secure 2013. All Rights Reserved. Inside Secure®, Inside Secure logo and combinations thereof, and others are registered trademarks or tradenames of Inside Secure or its subsidiaries. Other terms and product names may be trademarks of others.

The products identified and/or described herein may be protected by one or more of the patents and/or patent applications listed in related datasheets, such document being available on request under specific conditions. Additional patents or patent applications may also apply depending on geographic regions.



Reference List

[R1]	VaultIC 441 Technical Datasheet	TPR0551
------	---------------------------------	---------



INSIDE REGISTERED CONFIDENTIAL PROPRIETARY - DO NOT COPY



Revision History

Document Details

Title: Easy Plug - VaultIC ELib for 1XX

Literature Number: TMP_DIS_1302_001 v0.5

Date: Nov 19, 2013

- **Revision 1.0 :**
 - First release