Application Note

TPR0691A

VAULTIC4XX ELIB PROGRAMMER'S GUIDE

SEAL SQ
semiconductors + quantum

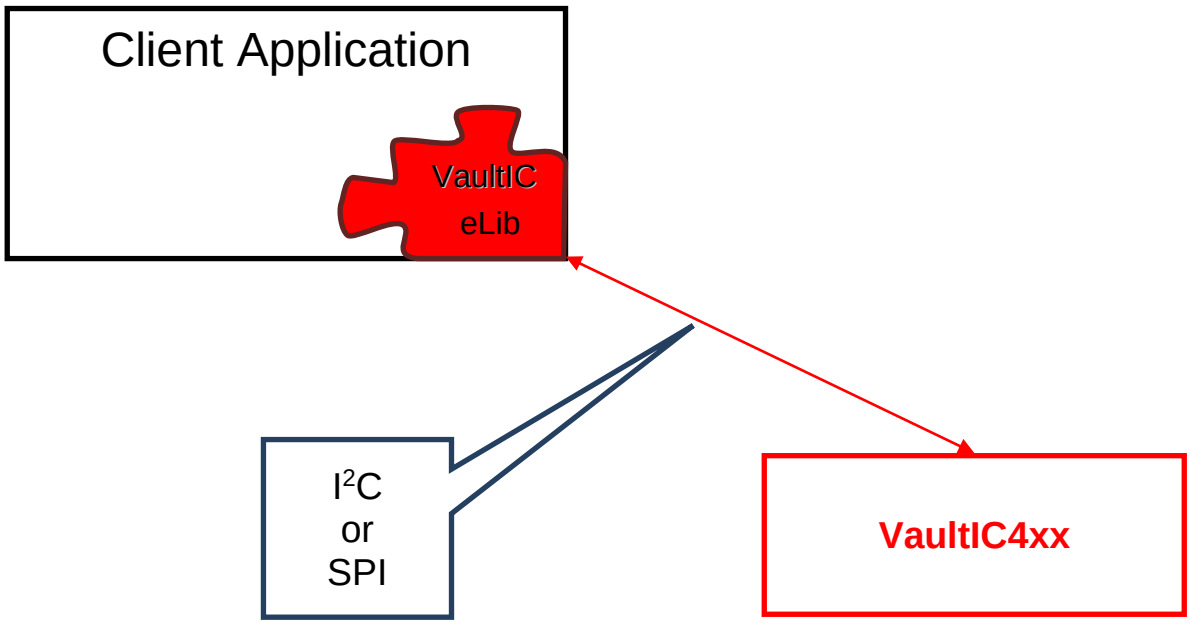
Table of Contents

1 Introduction

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

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

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



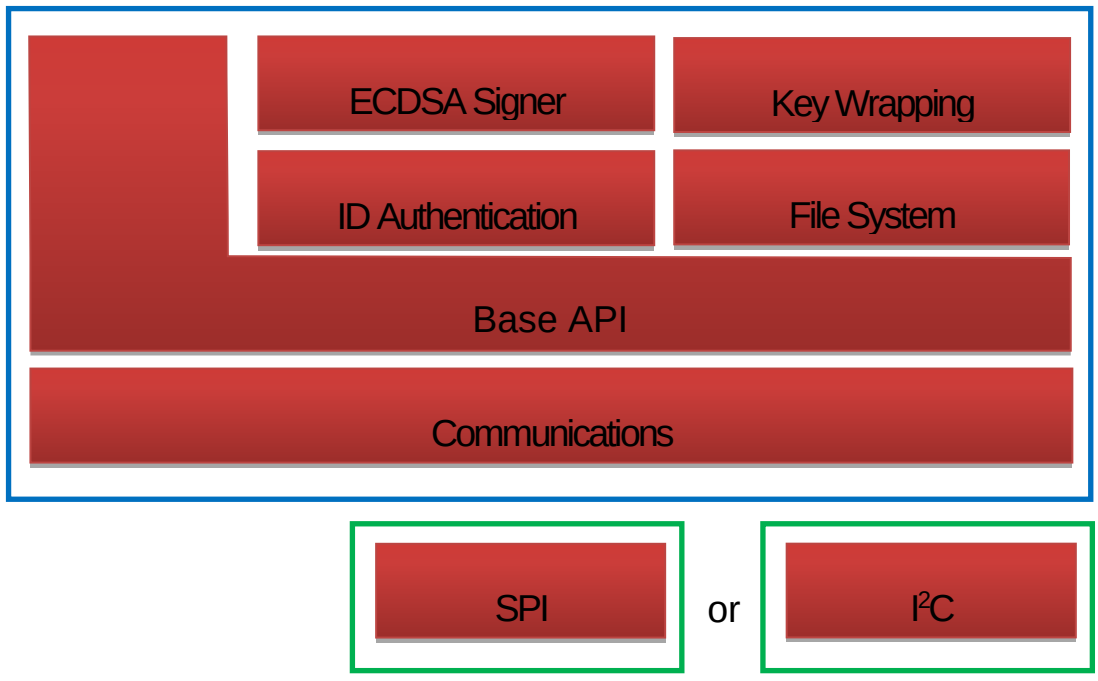
The VaultIC eLib is delivered as ANSI C source code for Windows based systems and for embedded targets.

2 Architecture

The VaultIC eLib has been designed to be easily deployed on a variety of platforms.

Most 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 below and more information is available in the API documentation on the USB stick included in the development kit.



3 Package contents

The VaultIC eLib source code is provided on the USB flash drive included in the VaultIC4xx development kit (also including demonstration codes). In the figure below, the directories related to the VaultIC eLib API are in bold characters.

```
vaultic_elib_4xx
├─ docs
├─ src
│   ├─ arch
│   │   ├─ embedded
│   │   │   ├─ common
│   │   │   ├─ stm32(*)
│   │   │   └─ raspberry(*)                          * depending on kit
│   │   │
│   │   └─ pc(*)
│   │
│   ├─ common
│   │   └─ comms
│   │
│   ├─ crypto
│   └─ device
│       └─ vaultic_4XX_family
└─ CHANGES.TXT
```

- **changes.txt**: list all changes between the VaultIC eLib releases.
- **docs**: contains the documentation related to the VaultIC eLib package i.e. this document and the API functions description.
- **src**: contains all source code files related to the VaultIC eLib API.
  - **src/arch/embedded**
    The embedded subfolder of the arch folder contains source files for an embedded host microcontroller, and a main implementation. These files require a target specific implementation to obtain a functional ***VaultIC eLib***.
  - **src/arch/pc**
    The PC subfolder of the arch folder contains source files specific to Windows platform.
  - **src/common**
    The common folder contains all of the platform independent source code files. These files should not require any modification to compile on the selected host target.
  - **src/crypto**
    contains source code related to the host cryptographic operations. These files should not require any modification to compile on the selected host target, but they can be optimized e.g. if the host platform has specific cryptographic resources available.
  - **src/device/vaultic_4XX_family**
    The *vaultic_4XX_family* subfolder of the device folder contains family specific files that are different as a result of interface and behavioral differences between the VaultIC families.

4 Building the VaultIC eLib API

The VaultIC eLib is designed to support several SEALSQ devices and several host platforms.

Depending on the development kit, the VaultIC eLib delivered is already configured to be functional with a VaultIC4xx on different types of platforms:

- Windows OS
- embedded targets based on ARM Cortex M3
- raspberry Pi OS

An easy way to compile and try these default configurations is explained in the Getting Started document present on the USB flash drive of the development kit.

Additionally, you can customize the VaultIC eLib API compilation by setting variables and you can migrate and compile the API on a new embedded platform by adapting some functions.

These options and the functions to modify are described in the following sections.

4.1 Configuration & Customization

The VaultIC eLib can be configured thanks to several variables defined in the C code (#define). Depending on the development kit, these variables can be found in:

- file *'vaultic_elib_4xx\src\device\vaultic_4XX_family\vaultic_4xx\VaultIC_config.h'*
- the IDE **preprocessor** field (depending on your IDE) or the **makefile**
  If it is more convenient for your project, some variables defined in the **preprocessor** field can be fixed in the previous header file (edit it and uncomment needed variables)

Some of these variables are mandatory and others are optional, these variables and the way to use them are described below.

4.1.1 The target device

The target device is defined by the variable TARGETCHIP_VAULTIC_4xx (where 4xx is dependent on the VaultIC device). This variable must not be changed. If you want to use another VaultIC than the ones provided in the development kit, please ask for the corresponding development kit in which the eLib will be already setup for the device present in the kit.

In the delivered eLib demonstration projects it is specified in the **preprocessor** variables or in the **makefile** and this variable must not be removed or modified.

4.1.2 Enable a communication protocol

The VaultIC eLib can manage several communication modes. On the VaultIC4xx devices the two available communication modes are I2C and SPI.

- The I2C communication interface is enabled by the variable USE_TWI.
- The SPI communication interface is enabled by the variable USE_SPI.

In the delivered eLib demonstration projects, these variables can be specified in the **preprocessor** variables or can be input parameters of the **makefile**.

In some demonstration kits (depending on the host), the choice of the interface is done at runtime from the user app, by enabling the variable SPI_SEL_CTRL_BY_GPIO (to control with a GPIO the interface used when starting the VaultIC chip), and by enabling both USE_TWI and USE_SPI.

Except if required, it is recommended to **NOT USE** this mechanism in your final application, i.e. SPI_SEL_CTRL_BY_GPIO **SHALL NOT** be enabled, and you **SHALL** use either USE_TWI or USE_SPI **(and not both)**.

The variable USE_TWI or USE_SPI can also be fixed in the file *'VaultIC_config.h'* by uncommenting the right line. You must define at least one of these two variables.

In case of platform migration, once this variable is set, it is necessary to adapt the drivers as explained in section Functions requiring a migration.

The VaultIC eLib is available for other SEALSQ devices with other communication protocols, this document is related to the VaultIC4xx device, so only I2C or SPI communications are possible

4.1.3 Enable VCC control of the VaultIC4xx

The power supply of the VaultIC4xx can be controlled by a GPIO of the host. In the delivered eLib this function is enabled thanks to the variable VCC_CTRL_BY_GPIO located in the **preprocessor** field or in the **makefile**.

This variable can also be set in the file *'VaultIC_config.h'* by uncommenting the right line.

In case of platform migration, once this variable is set, it is necessary to adapt the drivers as explained in section Functions requiring a migration.

This variable **should** be removed from the **preprocessor** field or makefile if the host platform doesn't control the VCC of the VaultIC4xx.

4.1.4 Enable RST control of the VaultIC4xx

On some VaultIC4xx a reset pin is available. This reset can be controlled by a GPIO of the host. In the delivered eLib this function is enabled thanks to the variable RST_CTRL_BY_GPIO located in the **preprocessor** field or in the **makefile**.

This variable can also be set in the file *'VaultIC_config.h'* by uncommenting the right line.

In case of platform migration, once this variable is set, it is necessary to adapt the drivers as explained in section Functions requiring a migration.

When supported by the chip it is recommended to use the RST management to ensure a good control of the device.

This variable **should** be removed from the **preprocessor** field or **makefile** if the host platform doesn't control the RST of the VaultIC4xx.

4.1.5 Enable or disable services

To customize your VaultIC eLib API you need to go to the *'src/device/vaultic_4XX_family/vaultic_4xx'* folder to modify the *'VaultIC_config.h'* file.

In this configuration file there is a list of services that can be enabled or disabled to optimize the size of the eLib API.

To enable a service just set the variable to

- VLT_ENABLE

To disable a service just set the variable to

- VLT_DISABLE

For the VaultIC4xx the service list is described in the following sections.

4.1.5.1 *Default ECDSA Curve Domain Parameters*

Some code and data space can be saved by disabling inclusion of the curves that are not needed. For example, if you use the VaultIC4xx with only a 283-bit Koblitz curve, you can disable the inclusion of the other curve types by making the following declarations in *'VaultIC_config.h'*:

```
#define VLT_ENABLE_ECDSA_B163       VLT_DISABLE
#define VLT_ENABLE_ECDSA_B233       VLT_DISABLE
#define VLT_ENABLE_ECDSA_B283       VLT_DISABLE
#define VLT_ENABLE_ECDSA_B409       VLT_DISABLE
#define VLT_ENABLE_ECDSA_B571       VLT_DISABLE
#define VLT_ENABLE_ECDSA_K163       VLT_DISABLE
#define VLT_ENABLE_ECDSA_K233       VLT_DISABLE
#define VLT_ENABLE_ECDSA_K283       VLT_ENABLE
#define VLT_ENABLE_ECDSA_K409       VLT_DISABLE
#define VLT_ENABLE_ECDSA_K571       VLT_DISABLE
#define VLT_ENABLE_ECDSA_P192       VLT_DISABLE
#define VLT_ENABLE_ECDSA_P224       VLT_DISABLE
#define VLT_ENABLE_ECDSA_P256       VLT_DISABLE
#define VLT_ENABLE_ECDSA_P384       VLT_DISABLE
#define VLT_ENABLE_ECDSA_P521       VLT_DISABLE
```

4.1.5.2 *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. Consequently, this requires data space. If this is an issue then slightly slower, but smaller code implementation can be used by setting this definition as:

```
#define VLT_ENABLE_FAST_CRC16CCIT   VLT_DISABLE
```

4.1.5.3 *SHA services*

The VaultIC4xx eLib API can manage several hash algorithms which are all enabled by default:

```
#define VLT_ENABLE_SHA224           VLT_ENABLE
#define VLT_ENABLE_SHA256           VLT_ENABLE
#define VLT_ENABLE_SHA384           VLT_ENABLE
#define VLT_ENABLE_SHA512           VLT_ENABLE
```

But you can disable the algorithms you don't use to optimize code size by setting the corresponding variables to `VLT_DISABLE`.

4.1.6 Debug and demonstration features

Some variables defined in the **preprocessor**/**makefile** only have an interest for the debug of the demonstration kit. These variables **must** be removed in a final application.

4.1.6.1 Enable communication protocol debug information

The VaultIC eLib sends and receives messages to/from the VaultIC4xx thanks to a block protocol communication based on APDU commands.

It is possible to print the APDU messages sent and received in a debug terminal thanks to the variable TRACE_APDU.

Additional communication details can be obtained by enabling the variable TRACE_BLOCK_PTCL in order to display the content of all blocks exchanged with the VaultIC4xx and/or TRACE_BLOCK_PTCL_ERRORS in order to display more information in case of errors.

You can use these variables during the debug process (add it in the **preprocessor** field or **makefile**), but it **must** be removed for the embedded production firmware.

The use of these variables requires an implementation of **printf** method.

4.1.6.2 Fast host cryptographic calculation

The host platform may have to do cryptographic calculations to exchange information with the VaultIC4xx (for example during ECDSA operation or mutual authentication). These calculations can be time consuming, so the development kit uses OpenSSL library or a cryptographic library optimized for ARM (in which only B163 curves are supported) depending on the targeted platform, respectively Windows or Embedded.

- The OpenSSL host crypto is enabled by the variable USE_OPENSSL_CRYPTO.
- The ARM crypto library is enabled by the variable USE_FAST_ARM_CRYPTO.

In the demonstration code, one of these two variables is specified in the **preprocessor** field or **makefile** depending on the platform.

Unless you use one of these libraries, you **must** remove these variables from the **preprocessor** field or **makefile**.

4.2 Platform migration

4.2.1 Supported platforms

The VaultIC eLib source code is written in ANSI C. It is possible to integrate the VaultIC eLib for execution on multiple platforms. Depending on the development kit the platforms directly supported are:

- Windows OS (32 and 64 bits)
- embedded targets based on ARM Cortex
- Raspberry Pi OS

The definition of the platform is done thanks to the variable VLT_PLATFORM.

With the VaultIC4xx device two choices are possible:

- VLT_WINDOWS
- VLT_EMBEDDED

But you don't have to set this variable as it will be automatically set depending on the IDE you are using.

Indeed, if you are using a windows IDE like **Visual Studio** the variable _WIN32 is already set (whatever the compilation is 32 or 64 bits) and the following variable will be configured:

```
#define  VLT_PLATFORM VLT_WINDOWS
```

If the IDE you use is not configured as a windows OS IDE, then the embedded choice will be done for the platform:

```
#define  VLT_PLATFORM VLT_EMBEDDED
```

4.2.2 Functions requiring a migration

To be functional with a specific embedded target, the VaultIC eLib requires the implementation of specific functions related to timing and communication interface.

These functions are defined in the following C files:

- *'src/arch/embedded/**<host_name>**/vaultic_timer_delay.c'*
- *'src/arch/embedded/**<host_name>**/vaultic_control_driver.c'*
- *'src/arch/embedded/**<host_name>**/vaultic_twi_driver.c'*
- *'src/arch/embedded/**<host_name>**/vaultic_spi_driver.c'*

Depending on the delivered eLib, <**host_name**> is "*stm32/stm32l1*" or "raspberry/*pi3*". In case of migration, you can copy these files in your own new directory and use them as skeleton.

Depending on the interface communication you choose, you only need to migrate the I^2C(TWI) or the SPI file.

4.2.2.1 Timing management

The file *'vaultic_timer_delay.c'* contains functions for managing basic timer operations.

```
void VltSleep (VLT_U32 uSecDelay);
```

This method must be adapted to the targeted platform and must wait for *"uSecDelay"* microseconds before returning.

```
void VltTimerStart(VLT_U32 msDelay);
```

This method must be adapted to start a timer configured to count until msDelay milliseconds.

```
void VltTimerStop(void);
```

This method must be adapted to stop the timer started by **VltTimerStart()** function.

```
VTL_BOOL VltTimerIsExpired(void);
```

This method must be adapted to return TRUE if the timer reaches the delay defined in **VltTimerStart()** function.

4.2.2.2 *Control management*

Power control

If the VaultIC power supply is controlled by the host through a GPIO (so the variable VCC_CTRL_BY_GPIO is set) then the following methods present in the file *'vaultic_control_driver.c'* must be adapted:

```
void        VltControlPowerOn(void);

void        VltControlPowerOff(void);
```

In these methods, define the actions needed by the host to switch on or off the VaultIC power supply (it can usually be a set to 1 or 0 of a GPIO).

Reset control

If the VaultIC reset pin is controlled by the host through a GPIO (so the variable RST_CTRL_BY_GPIO is set) then the following methods present in the file *'vaultic_control_driver.c'* must be adapted:

```
void        VltControlResetLow(void);

void        VltControlResetHigh(void);

void        VltControlUninit(void);
```

In these methods, define the actions needed by the host to configure the GPIO used to control the VaultIC reset pin.

4.2.2.3 *I²C management*

You don't need to read this section if you choose the SPI as communication interface.

The effort needed to migrate the I²C driver is certainly the most time-consuming. If you are not familiar with I²C, it is highly recommended to read the I²C section of the VaultIC datasheet and to focus attention on the physical layer part before starting the migration of the I²C driver. You will also need to pay attention to the I²C part in the datasheet of your targeted CPU/system.

The following file contains the methods and the definitions needed to manage the I²C communication interface:

'vaultic_twi_driver.c'

The following functions must be implemented:

```
uint16_t    VltTwiDriverInit(uint8_t u8I2cAddress, uint16_t u16BitRate);

void        VltTwiDriverDeInit(void);

uint16_t    VltTwiDriverSendBytes(uint8_t   u8I2cAddress,
                                  uint8_t * pu8BytesToSend,
                                  uint16_t  u16NumBytesToSend,
                                  uint32_t  u32BusTimeOut);

uint16_t    VltTwiDriverReceiveBytes(uint8_t   u8I2cAddress,
                                     uint8_t * pu8Buffer,
                                     uint16_t  u16NumBytesToReceive,
                                     uint32_t  u32BusTimeOut,
                                     uint32_t  u32ResponseTimeOut);

uint16_t    VltTwiDriverWakeUpVaultIc (uint16_t   u16BusTimeout);
```

In the embedded implementation you can see a TWI_TRACE_ERRORS variable that can be used during debug step of your I²C driver migration. This define is not mandatory and must be removed in the production version, but you can follow the function skeleton and comments to implement it in case of need.

The define TWI_TRACE_ERRORS is specific to the Raspberry Pi implementation and is dedicated to printing error messages.

4.2.2.4 *SPI management*

You don't need to read this section if you choose the I²C as communication interface.

The effort needed to migrate the SPI driver is certainly the most time-consuming. If you are not familiar with SPI, it is highly recommended to read the SPI section of the VaultIC datasheet and to focus attention on the physical layer part before starting the migration of the SPI driver. You will also need to pay attention to the SPI part in the datasheet of your targeted CPU/system.

The following file contains the methods and the definitions needed to manage the SPI communication interface:

- *'vaultic_spi_driver.c'*

The following functions must be implemented:

```
uint16_t    VltSpiDriverInit(uint16_t u16BitRate)

void        VltSpiDriverDeInit(void)

uint16_t    VltSpiDriverSendBytes(uint8_t * pu8BytesToSend,
                                  uint16_t  u16NumBytesToSend,
                                  uint32_t  u32BusTimeOut)

uint16_t    VltSpiDriverReceiveBytes(uint8_t * pu8Buffer,
                                     uint16_t  u16NumBytesToReceive,
                                     uint32_t  u32BusTimeOut)

void        VltSpiSlaveSelectLow(void);

void        VltSpiSlaveSelectHigh(void);
```

In the embedded implementation you can see a SPI_TRACE_ERRORS variable that can be used during debug step of your SPI driver migration. This define is not mandatory and has to be removed in the production version, but you can follow the function skeleton and comments to implement it in case of need.

4.2.2.5 *Host cryptographic calculations*

If your host application needs to perform cryptographic calculations, you will need to develop your own cryptographic functions or to integrate a cryptographic library.

You can contact your local SEALSQ FAE if you need more information or help on this topic.

4.3 Guideline to migrate eLib on another CPU/platform/IDE

This section describes the steps needed to migrate the eLib on another platform (than the one included in the development kit).

Due to the number of existing platforms this guideline cannot be exhaustive and should be used as a to-do list to quickly start the migration. Depending on the targeted host (number of timers, I^2C capabilities …) and IDE, advanced development and debug can be needed.

4.3.1 Files and directories copy

Copy the API in your source code location. The VaultIC eLib API is inside the directory *'src'*.

Duplicate the directory *'src/arch/embedded/<host_name>'* in *'src/arch/embedded/<new_host_name>'*.

The files in this directory will be used as skeletons for the new host.

You should also have a look at the file *'main.c'* and one of the demonstration C files to see how the VaultIC eLib API is initialized and used.

4.3.2 IDE configuration

All the C files located in the *'src'* directory need to be compiled. Depending on your IDE you will need to import them or to specify you want to compile them.

The following directories contain the needed .h header files and depending on your IDE it can be necessary to add these paths in the dedicated environment variable:

- *'src/arch/embedded/common'*
- *'src/common'*
- *'src/crypto'*
- *'src/device/vaultic_4XX_family'*
- *'src/device/vaultic_4XX_family/crypto'*
- *'src/device/vaultic_4XX_family/auth'*

The following variable must be added to the **preprocessor** compilation options:

- TARGETCHIP_VAULTIC_4xx (where 4xx is the VaultIC device name)

In the preprocessor compilation options or in the *'VaultIC_config.h'* file define one of these two variables:

- USE_TWI (for the I^2C) or USE_SPI

You could also need the following variables to be defined:

- VCC_CTRL_BY_GPIO
  Set this variable if the host needs to control the VaultIC power supply.
- RST_CTRL_BY_GPIO
  Set this variable if the host needs to control the VaultIC reset pin (not available on all devices).
- TRACE_APDU / TRACE_BLOCK_PTCL / TRACE_BLOCK_PTCL_ERRORS
  Debug features based on **printf** and **scanf** implementation, if your new platform doesn't manage such functions, you should remove these variables in a first time.

4.3.3 Files modification

As explained in paragraph Functions requiring a migration the following files need to be adapted to the new platform. You can use the provided versions located in the development kit as a skeleton:

- *'vaultic_mem.c'* if the new platform is not ANSI C compatible
- *'vaultic_timer_delay.c'*
- *'vaultic_twi_driver.c'* **or** *'vaultic_spi_driver.c'*
- *'vaultic_control_driver.c'*

Glossary

TERM	Detailed Description
APDU	Application Protocol Data Unit
API	Application Programming Interface
CRC CCITT	Cyclic Redundancy Check – Consultative Committee for International Telegraphy and Telephony
ECDSA	Elliptic Curve Digital Signature Algorithm
GPIO	Global Purpose Input Output
IDE	Integrated Development Environment
TWI	Two Wire Interface – another name for the I^2C interface
USB	Universal Serial Bus
VaultIC eLib API	The programmatic interface to allow communication with the VaultIC Device. Can be short named eLib or API in the document
VaultIC Security Module	The VaultIC device is a hardware cryptographic module.

History

Version	Date	Comments
A	Aug 10, 2023	Creation

Headquarters

SEALSQ
Arteparc de Bachasson - Bat A
Rue de la Carrière de Bachasson
CS 70025
13590 Meyreuil - France
Tel: +33 (0)4-42-370-370
Fax: +33 (0)4-42-370-024

Product Contact

Web Site
www.SEALSQ.com

Technical Support
dl_e-security@wisekey.com

Sales Contact
sales@wisekey.com