
Getting started with the X-CUBE-SAFE1 software package

Introduction

This user manual describes how to get started with the **X-CUBE-SAFE1** software package.

The **X-CUBE-SAFE1** software package is a software component that provides several demonstration codes, which use the **STSAFE-A110** device features from a host microcontroller.

These demonstration codes utilize the **STSAFE-A1xx** middleware built on the **STM32Cube** software technology to ease portability across different STM32 microcontrollers. In addition, it is MCU-agnostic for portability to other MCUs.

These demonstration codes illustrate the following features:

- Authentication
- Pairing
- Key establishment
- Local envelope wrapping
- Key pair generation



1 General information

The **X-CUBE-SAFE1** software package is a reference to integrate the **STSAFE-A110** secure element services into a host MCU's operating system (OS) and its application.

It contains the **STSAFE-A110** driver and demonstration codes to be executed on STM32 32-bit microcontrollers based on the Arm[®] Cortex[®]-M processor.

Note: *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

The **X-CUBE-SAFE1** software package is developed in ANSI C. Nevertheless, the platform-independent architecture allows easy portability to a variety of different platforms.

The table below presents the definition of acronyms that are relevant for a better understanding of this document.



2 STSAFE-A110 secure element

The [STSAFE-A110](#) is a highly secure solution that acts as a secure element providing authentication and data management services to a local or remote host. It consists of a full turnkey solution with a secure operating system running on the latest generation of secure microcontrollers.

The [STSAFE-A110](#) can be integrated in IoT (Internet of things) devices, smart-home, smart-city and industrial applications, consumer electronics devices, consumables and accessories. Its key features are:

- Authentication (of peripherals, IoT and USB Type-C® devices)
- Secure channel establishment with remote host including transport layer security (TLS) handshake
- Signature verification service (secure boot and firmware upgrade)
- Usage monitoring with secure counters
- Pairing and secure channel with host application processor
- Wrapping and unwrapping of local or remote host envelopes
- On-chip key pair generation

Refer to the [STSAFE-A110](#) datasheet available on the [STSAFE-A110](#) web page for additional information on the device.

3 STSAFE-A1xx middleware description

This section details the STSAFE-A1xx middleware software package content and the way to use it.

3.1 General description

The STSAFE-A1xx middleware is a set of software components designed to:

- interface the [STSAFE-A110](#) secure element device with an MCU
- implement the most generic [STSAFE-A110](#) use cases

The STSAFE-A1xx middleware is fully integrated within ST software packages as a middleware component to add secure element features (for example [X-CUBE-SBSFU](#) or [X-CUBE-SAFE1](#)).

It can be downloaded from the [STSAFE-A110](#) internet page through the **Tools & Software** tab.

The software is provided as source code under an ST software license agreement (SLA0088) (see License information for more details).

The following integrated development environments are supported:

- IAR Embedded Workbench® for Arm® (EWARM)
- Keil® Microcontroller Development Kit (MDK-ARM)
- STM32Cube IDE ([STM32CubeIDE](#))
- System Workbench for STM32 ([SW4STM32](#))

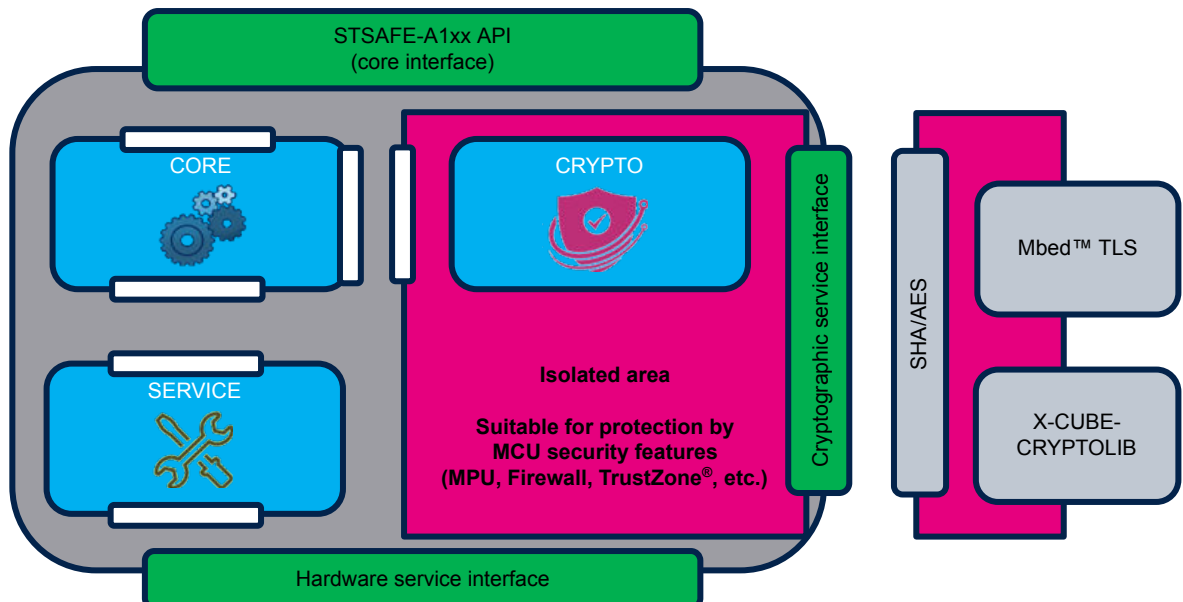
Refer to the release notes available in the package root folder for information about the supported IDE versions.

3.2 Architecture

This section describes the software components of the STSAFE-A1xx middleware software package.

The figure below presents a view of the STSAFE-A1xx middleware architecture and related interfaces.

Figure 1. STSAFE-A1xx middleware architecture



The middleware features three different interfaces:

- **STSAFE-A1xx API:** It is the main application programming interface (API), which provides full access to all the [STSAFE-A110](#) services exported to the upper layers (application, libraries and stacks). This interface is also referred to as the core interface because all the exported APIs are implemented in the CORE module. The upper layers that need to integrate the STSAFE-A1xx middleware must access the [STSAFE-A110](#) features through this interface.

- **Hardware service interface:** This interface is used by the STSAFE-A1xx middleware to reach the highest hardware platform independence. It includes a set of generic functions to connect the specific MCU, IO bus and timing functions. This structure improves the library code re-usability and guarantees easy portability to other devices.

Defined as weak functions, these generic functions must be implemented at application level following the example provided within the `stsafea_service_interface_template.c` template provided for easy integration and customization within the upper layers.

- **Cryptographic service interface:** This interface is used by the STSAFE-A1xx middleware to access platform or library cryptographic functions such as SHA (secure hash algorithm) and AES (advanced encryption standard) required by the middleware for some demonstrations.

Defined as weak functions, these cryptographic functions must be implemented at application level following the example provided with two different templates:

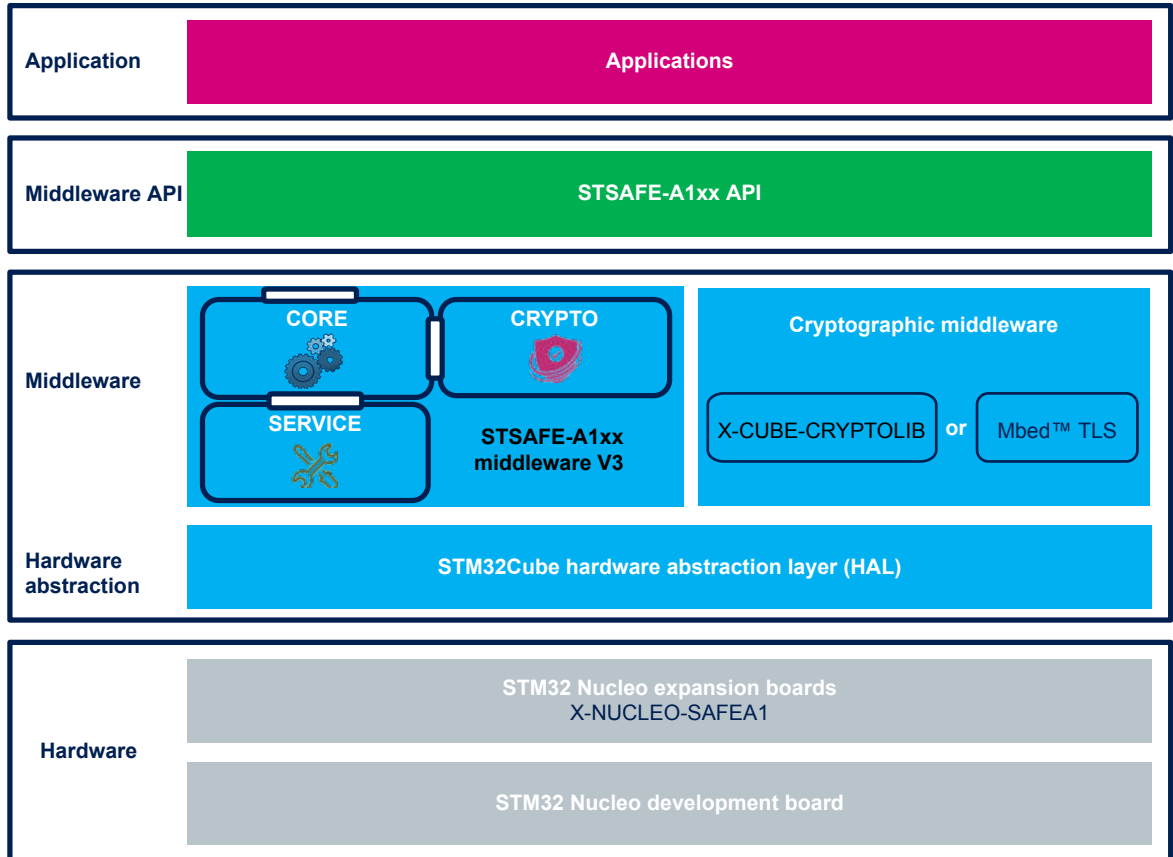
- `stsafea_crypto_mbedtls_interface_template.c` if the Arm® Mbed™ TLS cryptographic library is used;
- `stsafea_crypto_stlib_interface_template.c` if the ST cryptographic library is used;
- Alternative cryptographic libraries can be used by simply customizing the template source files. The template files are provided for easy integration and customization within the upper layers.

Note: *Arm and Mbed are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

Some examples of application integrating and using the STSAFE-A1xx middleware (such as [X-CUBE-SBSFU](#) or [X-CUBE-SAFE1](#)) are based on the STM32Cube hardware abstraction layer (HAL) for STM32 microcontrollers.

The figure below shows the STSAFE-A1xx middleware integrated in a standard STM32Cube application, running on an X-NUCLEO-SAFE1 expansion board mounted on an STM32 Nucleo board.

Figure 2. STSAFE-A1xx middleware in an STM32Cube application



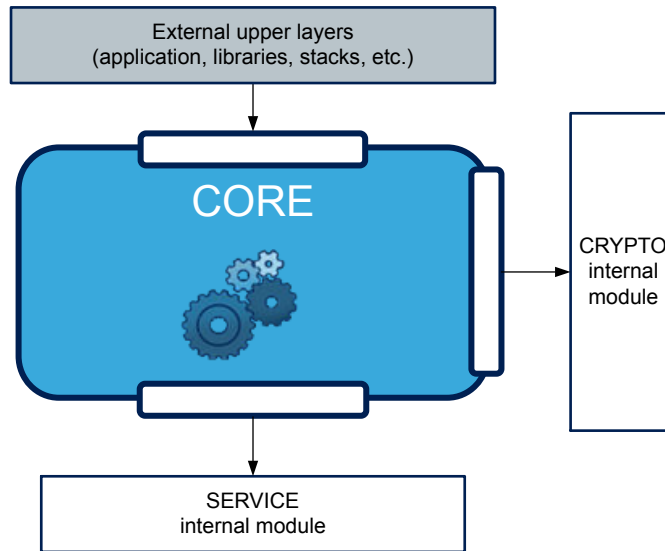
To provide the best hardware and platform independence, the STSAFE-A1xx middleware is not directly connected to the STM32Cube HAL, but through interface files implemented at application level (*stsafea_service_interface_template.c*, *stsafea_interface_conf.h*).

3.3 CORE module

The CORE module is the core of the middleware. It implements the commands called by the upper layers (application, libraries, stack and so on) in order to properly use the STSAFE-A1xx features.

The figure below presents a view of the CORE module architecture.

Figure 3. CORE module architecture



The CORE module is a multi-interface software component connected to:

- Upper layers: external connection through the exported APIs described in the two tables below;
- Cryptographic layer: internal connection to the CRYPTO module;
- Hardware service layer: internal connection to the SERVICE module;

The STSAFE-A1xx middleware software package provides a complete API documentation of the CORE module in the root folder (see *STSAFE-A1xx_Middleware.chm* file).

Refer to the [STSAFE-A110](#) datasheet for a brief explanation of the command set, to which the command APIs listed in the following table are related.

Table 1. CORE module exported API

API category	Function
Initialization configuration	StSafeA_Init <i>To create, initialize and assign the STSAFE-A1xx device handle.</i>
General-purpose commands	StSafeA_GetVersion <i>To return the STSAFE-A1xx middleware revision.</i>
	StSafeA_Echo <i>To receive the data passed in the command.</i>
	StSafeA_Reset <i>To reset the volatile attributes to their initial values.</i>
	StSafeA_GenerateRandom <i>To generates a number of random bytes.</i>
	StSafeA_Hibernate <i>To put the STSAFE-Axxx device in hibernation.</i>
Data partition commands	StSafeA_DataPartitionQuery

API category	Function
Data partition commands	<i>Query command to retrieve the data partition configuration.</i>
	StSafeA_Decrement <i>To decrement the one-way counter in a counter zone.</i>
	StSafeA_Read <i>To read data from a data partition zone.</i>
	StSafeA_Update <i>To update data through zone partition.</i>
Private and public key commands	StSafeA_GenerateSignature <i>To return the ECDSA signature over a message digest.</i>
	StSafeA_GenerateKeyPair <i>To generate a key-pair in a private key slot.</i>
	StSafeA_VerifyMessageSignature <i>To verify the message authentication.</i>
	StSafeA_EstablishKey <i>To establish a shared secret between two hosts by using asymmetric cryptography.</i>
Administrative commands	StSafeA_ProductDataQuery <i>Query command to retrieve the product data.</i>
	StSafeA_I2cParameterQuery <i>Query command to retrieve the I²C address and low-power mode configuration.</i>
	StSafeA_LifeCycleStateQuery <i>Query command to retrieve the lifecycle state (Born, Operational, Terminated, Born and Locked or Operational and Locked).</i>
	StSafeA_HostKeySlotQuery <i>Query command to retrieve the host key information (presence and host C-MAC counter).</i>
	StSafeA_PutAttribute <i>To put attributes in the STSAFE-Axxx device, such as keys, password, I²C parameters according to the attribute TAG.</i>
	StSafeA_DeletePassword <i>To delete the password from its slot.</i>
	StSafeA_VerifyPassword <i>To verify the password and remember the outcome of the verification for future command authorization.</i>
	StSafeA_RawCommand <i>To execute a raw command and receive the related response.</i>
Local envelope commands	StSafeA_LocalEnvelopeKeySlotQuery <i>Query command to retrieve local envelope key information (slot number, presence and key length) for the available key slots.</i>
	StSafeA_GenerateLocalEnvelopeKey <i>To generate a key in a local envelope key slot.</i>
	StSafeA_WrapLocalEnvelope <i>To wrap data (usually keys) that are entirely managed by the host, with a local envelope key and the [AES key wrap] algorithm.</i>
	StSafeA_UnwrapLocalEnvelope <i>To unwrap a local envelope with a local envelope key.</i>

Table 2. Exported STSAFE-A110 CORE module APIs

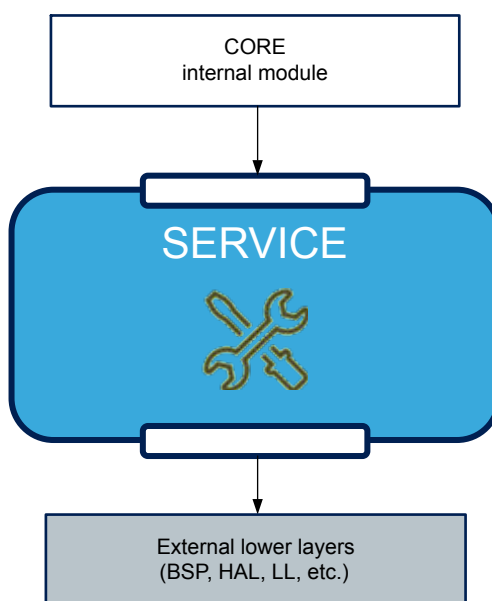
API category	Function
Command authorization configuration command	StSafeA_CommandAuthorizationConfigurationQuery <i>Query command to retrieve access conditions for commands with configurable access conditions.</i>

3.4 SERVICE module

The SERVICE module is the low layer of the middleware. It implements a full hardware abstraction in terms of MCU and hardware platform.

The figure below presents a view of the SERVICE module architecture.

Figure 4. SERVICE module architecture



The SERVICE module is a dual-interface software component connected to:

- External lower layers: such as BSP, HAL or LL. Weak functions must be implemented at external higher layers and are based on the *stsafea_service_interface_template.c* template file;
- Core layer: internal connection to the CORE module through the exported APIs described in the table below;

The STSAFE-A1xx middleware software package provides a complete API documentation of the SERVICE module in the root folder (see *STSAFE-A1xx_Middleware.chm* file).

Table 3. SERVICE module exported APIs

API category	Function
Initialization configuration	StSafeA_BSP_Init <i>To initialize the communication bus and the IO pins needed to operate the STSAFE-Axxx device.</i>
Low-level operation functions	StSafeA_Transmit <i>To prepare the command to be transmitted, and call the low-level bus API to be executed. Compute and concatenate a CRC, if supported.</i>
	StSafeA_Receive <i>To receive data from the STSAFE-Axxx by using the low-level bus functions to retrieve them. Check the CRC, if supported.</i>

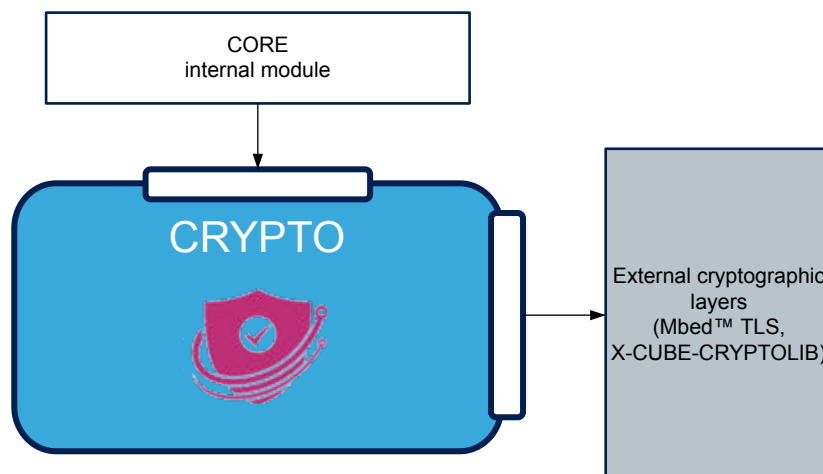
3.5 CRYPTO module

The CRYPTO module represents the cryptographic part of the middleware. It must rely on the platform's cryptographic resources.

The CRYPTO module is completely independent of the other middleware modules and, for this reason, can be easily encapsulated inside an isolated secure area suited to protection by MCU security features such as a memory protection unit (MPU), a firewall or a TrustZone®.

The figure below presents a view of the CRYPTO module architecture.

Figure 5. CRYPTO module architecture



The CRYPTO module is a dual-interface software component connected to:

- an external cryptography library: Mbed TLS and **X-CUBE-CRYPTOLIB** are currently supported. Weak functions must be implemented at external higher layers and are based on the:
 - `stsafea_crypto_mbedtls_interface_template.c` template file for the Mbed TLS cryptographic library;
 - `stsafea_crypto_stlib_interface_template.c` template file for the ST cryptographic library;
 Additional cryptographic libraries can be easily supported by adapting the cryptographic interface template file.
- the core layer: internal connection to the CORE module through the exported APIs described in the table below;

The STSAFE-A1xx middleware software package provides a complete API documentation of the CRYPTO module in the root folder (see `STSAFE-A1xx_Middleware.chm` file).

Table 4. CRYPTO module exported APIs

API category	Function
Cryptographic APIs	StSafeA_ComputeCMAC <i>To compute the CMAC value. Used on the prepared command.</i>
	StSafeA_ComputeRMAC <i>To compute the RMAC value. Used on the received response.</i>
	StSafeA_DataEncryption <i>To execute data encryption (AES CBC) on the STSAFE-Axxx data buffer.</i>
	StSafeA_DataDecryption <i>To execute data decryption (AES CBC) on the STSAFE-Axxx data buffer.</i>
	StSafeA_MAC_SHA_PrePostProcess <i>To pre- or post-process the MAC and/or SHA before transmission, or after reception of data from the STSAFE_Axxx device.</i>

3.6 Templates

This section gives a detail description of the templates available within the STSAFE-A1xx middleware software package.

All the templates listed in the table below are provided inside the `Interface` folder available at the root level of the middleware software package.

Template files are provided as examples to be copied and customized into the upper layers, in order to easily integrate and configure the STSAFE-A1xx middleware:

- Interface template files provide example implementations of the `__weak` functions, offered as empty or partially empty functions inside the middleware. They must be properly implemented in the user space or in the upper layers according to the cryptographic library and to the user's hardware choices.
- Configuration template files provide an easy way to configure the STSAFE-A1xx middleware and features that can be used in the user application, such as optimizations or specific hardware.

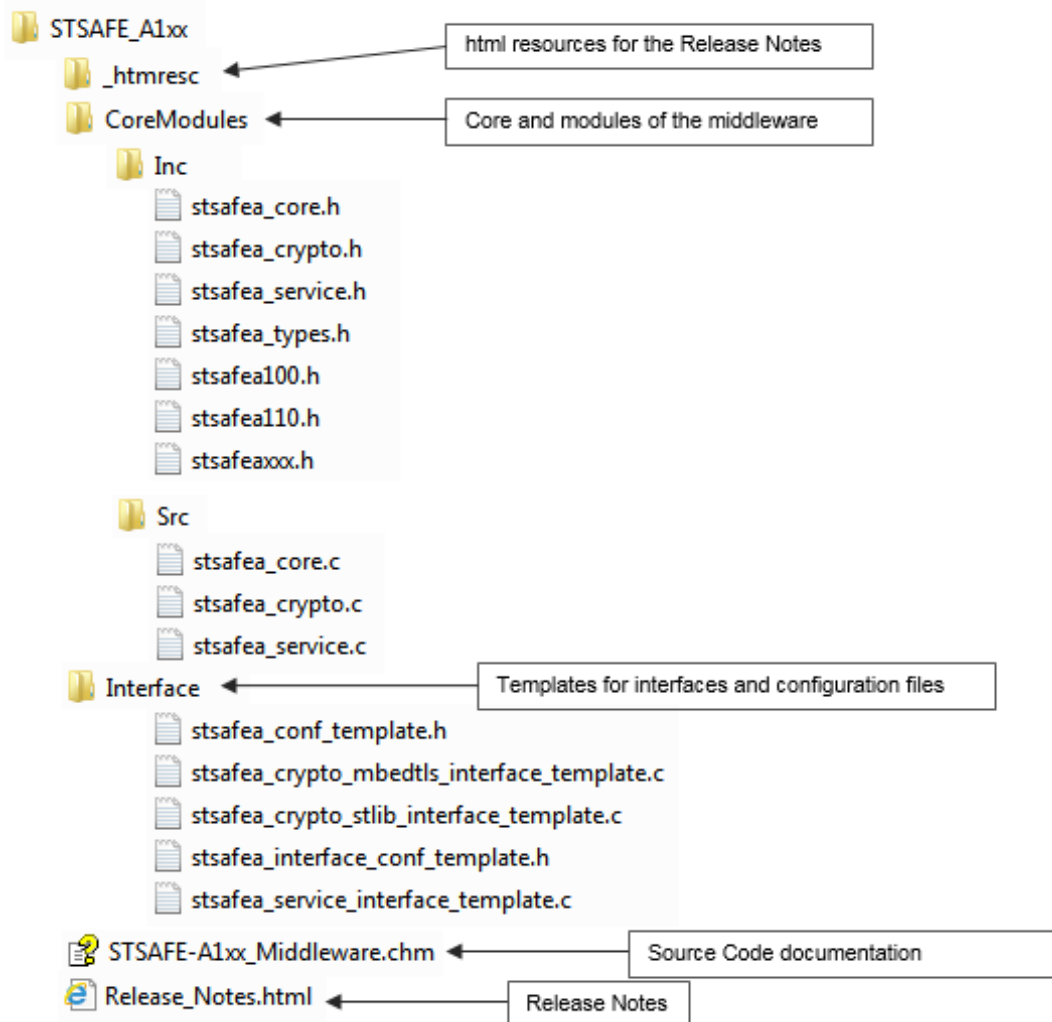
Table 5. Templates

Template category	Template file
Interface templates	stsafea_service_interface_template.c <i>Example template to show how to support the hardware services required by the STSAFE-A middleware and offered by the specific hardware, low-level library or BSP selected in the user space.</i>
	stsafea_crypto_mbedtls_interface_template.c <i>Example template to show how to support the cryptographic services required by the STSAFE-A middleware and offered by the Mbed TLS cryptographic library (key management, SHA, AES, etc.).</i>
	stsafea_crypto_stlib_interface_template.c <i>Example template to show how to support the cryptographic services required by the STSAFE-A middleware and offered by the STM32 cryptographic library software expansion for STM32Cube (X-CUBE-CRYPTOLIB) (key management, SHA, AES, etc.).</i>
Configuration templates	stsafea_conf_template.h <i>Example template to show how to configure the STSAFE-A middleware (in particular for optimizations purposes).</i>
	stsafea_interface_conf_template.h <i>Example template to show how to configure and customize the interface files listed above.</i>

3.7 Folder structure

The figure below presents the folder structure of the STSAFE-A1xx middleware software package.

Figure 6. Project file structure



3.8 How to: integration and configuration

This section describes how to integrate and configure the STSAFE-A1xx middleware in the user application.

3.8.1 Integration steps

Follow these steps to integrate the STSAFE-A1xx middleware in the desired application:

- Step 1: Copy (and optionally rename) the *stsafea_service_interface_template.c* file and either of *stsafea_crypto_mbedtls_interface_template.c* or *stsafea_crypto_stlib_interface_template.c* to the user space according to the cryptographic library that has been added to the application (whatever the cryptographic library selected/used by users, they can even create/implement their own cryptographic interface file from scratch by adapting the suitable template).
- Step 2: Copy (and optionally rename) the *stsafea_conf_template.h* and *stsafea_interface_conf_template.h* files to the user space.
- Step 3: Make sure to add the right includes in your main or any other user space source file that needs to interface the STSAFE-A1xx middleware:

```
#include "stsafea_core.h" #include "stsafea_interface_conf.h"
```
- Step 4: Customize the files used in the three steps above according to user preferences.

3.8.2 Configuration steps

In order to properly configure the STSAFE-A1xx middleware in the user application, ST provide two different configuration template files to be copied and customized in the user space according to the user's choices:

- *stsafea_interface_conf_template.h*: This example template is used to and shows how to configure the cryptographic and service middleware interfaces in the user space through the following `#define` statements:
 - `USE_PRE_LOADED_HOST_KEYS`
 - `MCU_PLATFORM_INCLUDE`
 - `MCU_PLATFORM_BUS_INCLUDE`
 - `MCU_PLATFORM_CRC_INCLUDE`
- *stsafea_conf_template.h*: This example template is used to and shows how to configure the STSAFE-A middleware through the following `#define` statements:
 - `STSAFEA_USE_OPTIMIZATION_SHARED_RAM`
 - `STSAFEA_USE_OPTIMIZATION_NO_HOST_MAC_ENCRYPT`
 - `STSAFEA_USE_FULL_ASSERT`
 - `USE_SIGNATURE_SESSION` (for the STSAFE-A100 only)

Follow these steps in order to integrate the STSAFE-A1xx middleware in the desired application:

- Step 1: Copy (and optionally rename) the *stsafea_interface_conf_template.h* and *stsafea_conf_template.h* files to the user space.
- Step 2: Confirm or modify the `#define` statement of the two above-mentioned header files according to the user platform and cryptographic choices.

4 Demonstration software

This section illustrates demonstration software based on the STSAFE-A1xx middleware.

4.1 Authentication

This demonstration illustrates the command flow where the STSAFE-A110 is mounted on a device that authenticates to a remote host (IoT device case), the local host being used as a pass-through to the remote server.

The scenario where the STSAFE-A110 is mounted on a peripheral that authenticates to a local host, for example for games, mobile accessories or consumables, is exactly the same.

Command flow

Note: For demonstration purposes, the local and remote hosts are the same device here.

1. Extract, parse and verify the STSAFE-A110's public certificate stored in the data partition zone 0 of the device in order to get the public key:
 - Read the certificate using the STSAFE-A1xx middleware through the STSAFE-A110's zone 0.
 - Parse the certificate using the cryptographic library's parser.
 - Read the CA certificate (available through the code).
 - Parse the CA certificate using the cryptographic library's parser.
 - Verify the certificate validity using the CA certificate through the cryptographic library.
 - Get the public key from the STSAFE-A110 X.509 certificate.
2. Generate and verify the signature over a challenge number:
 - Generate a challenge number (random number).
 - Hash the challenge.
 - Fetch a signature over the hashed challenge using the STSAFE-A110's private key slot 0 through the STSAFE-A1xx middleware.
 - Parse the generated signature using the cryptographic library.
 - Verify the generated signature using the STSAFE-A110's public key through the cryptographic library.
 - When this is valid, the host knows that the peripheral or IoT is authentic.

4.2 Pairing

This code example establishes a pairing between an STSAFE-A110 device and the MCU it is connected to. The pairing allows the exchanges between the device and the MCU to be authenticated (that is, signed and verified). The STSAFE-A110 device becomes usable only in combination with the MCU it is paired with.

The pairing consists of the host MCU sending a host MAC key and a host cipher key to the STSAFE-A110. Both keys are stored to the protected NVM of the STSAFE-A110 and should be stored to the flash memory of the STM32 device.

By default, in this example, the host MCU sends well-known keys to the STSAFE-A110 (see command flow below) that are highly recommended to use for demonstration purposes. The code also allows the generation of random keys.

Moreover, the code example generates a local envelope key when the corresponding slot is not already populated in the STSAFE-A110. When the local envelope slot is populated, the STSAFE-A110 device allows the host MCU to wrap/unwrap a local envelope to securely store a key on the host MCU's side.

Note: The pairing code example must be executed successfully prior to executing all the following code examples.

Command flow

1. Generate the local envelope key in the STSAFE-A110 using the STSAFE-A1xx middleware.
By default, this command is activated. Be aware that uncommenting the following define statements in the `pairing.c` file deactivates the local envelope key generation:

```
/* #define _FORCE_DEFAULT_FLASH_ */
```

This operation occurs only if the STSAFE-A110's local envelope key slot is not already populated.

- Define two 128-bit numbers to use as the host MAC key and the host cipher key.

By default, golden known keys are used. They have the following values:

```
0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF
/* Host MAC key */
0x11, 0x11, 0x22, 0x22, 0x33, 0x33, 0x44, 0x44, 0x55, 0x55, 0x66, 0x66, 0x77, 0x77, 0x88, 0x88
/* Host cipher key */
```

To activate random key generation, add the following define statement to the `pairing.c` file:

```
#define USE_HOST_KEYS_SET_BY_PAIRING_APP 1
```

- Store the host MAC key and the host cipher key to their respective slot in the [STSAFE-A110](#).
- Store the host MAC key and the host cipher key to the STM32's flash memory.

4.3 Key establishment (establish secret)

This demonstration illustrates the case where the [STSAFE-A110](#) device is mounted on a device (such as an IoT device), which communicates with a remote server, and needs to establish a secure channel to exchange data with it.

In this example, the STM32 device plays the role of both the remote server (remote host) and the local host that is connected to the [STSAFE-A110](#) device.

The goal of this use case is to show how to establish a shared secret between the local host and the remote server using the elliptic curve Diffie-Hellman scheme with a static (ECDH) or ephemeral (ECDHE) key in the [STSAFE-A110](#).

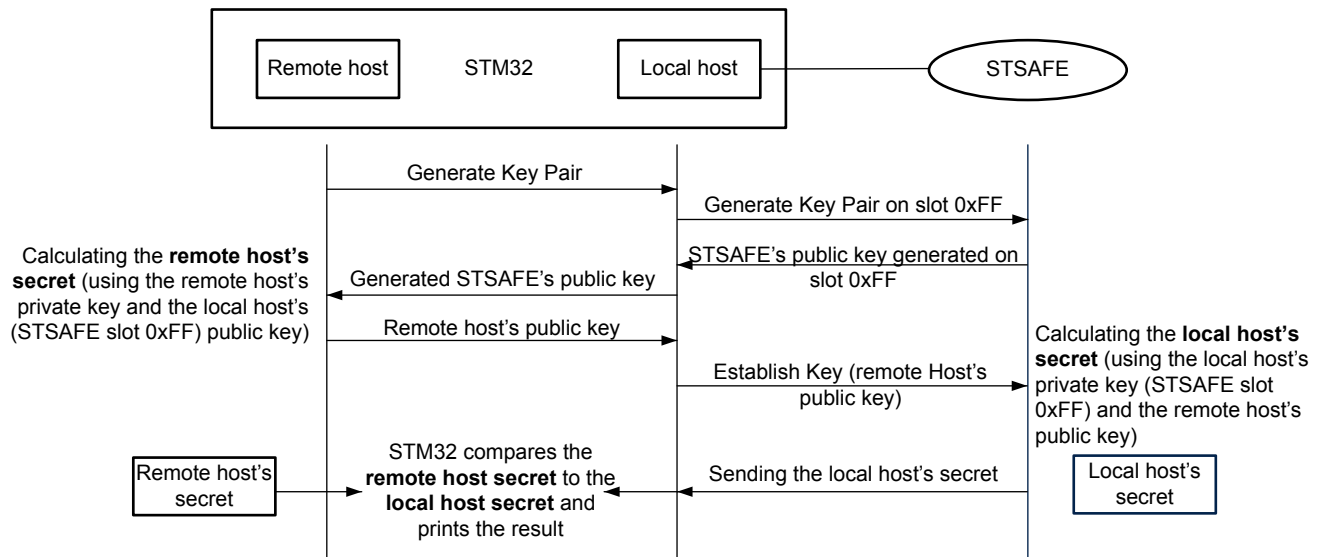
The shared secret should be further derived to one or more working keys (not illustrated here). These working keys can then be used in communication protocols such as TLS, for example for protecting the confidentiality, integrity and authenticity of the data that are exchanged between the local host and the remote server.

Command flow

Figure 7. Key establishment command flow illustrates the command flow.

- The remote host's private and public keys are hard-coded in the code example.
- The local host sends the `StSafeA_GenerateKeyPair` command to the [STSAFE-A110](#) to generate the key pair on its ephemeral slot (slot 0xFF).
- The [STSAFE-A110](#) sends back the public key (which corresponds to slot 0xFF) to the STM32 (representing the remote host).
- The STM32 computes the remote host's secret (using the STSAFE device's public key and the remote host's private key).
- The STM32 sends the remote host's public key to the [STSAFE-A110](#) and asks the [STSAFE-A110](#) to compute the local host's secret using the `StSafeA_EstablishKey` API.
- The [STSAFE-A110](#) sends back the local host's secret to the STM32.
- The STM32 compares the two secrets, and prints the result. If the secrets are the same, the secret establishment is successful.

Figure 7. Key establishment command flow



4.4 Wrap/unwrap local envelopes

This demonstration illustrates the case where the [STSAFE-A110](#) wraps/unwraps the local envelope in order to securely store a secret to any non-volatile memory (NVM).

Encryption/decryption keys can be securely stored in that manner to additional memory or within the [STSAFE-A110](#)'s user data memory.

The wrapping mechanism is used to protect a secret or plain text. The output of wrapping is an envelope encrypted with an AES key wrap algorithm, and that contains the key or plain text to be protected.

Command flow

Note: The local and remote hosts are the same device here.

1. Generate random data assimilated to a local envelope.
2. Wrap the local envelope using the [STSAFE-A110](#)'s middleware.
3. Store the wrapped envelope.
4. Unwrap the wrapped envelope using the [STSAFE-A110](#)'s middleware.
5. Compare the unwrapped envelope to the initial local envelope. They should be equal.

4.5 Key pair generation

This demonstration illustrates the command flow where the [STSAFE-A110](#) device is mounted on a local host. A remote host asks this local host to generate a key pair (a private key and a public key) on slot 1 and then to sign a challenge (random number) with the generated private key.

The remote host is then able to verify the signature with the generated public key.

This demonstration is similar to the [Authentication](#) demonstration with two differences:

- The key pair in the [Authentication](#) demonstration is already generated (on slot 0), whereas, in this example, we generate the key pair on slot 1. The [STSAFE-A110](#) device can also generate the key pair on slot 0xFF, but only for key establishment purposes.
- The public key in the [Authentication](#) demonstration is extracted from the certificate in zone 0. In this example, the public key is sent back with the [STSAFE-A110](#)'s response to the `StSafeA_GenerateKeyPair` command.

Command flow

Note: For demonstration purposes, the local and remote hosts are the same device here.

1. The host sends the `StSafeA_GenerateKeyPair` command to the **STSAFE-A110**, which sends back the public key to the host MCU.
2. The host generates a challenge (48-byte random number) using the `StSafeA_GenerateRandom` API. The **STSAFE-A110** sends back the generated random number.
3. The host computes the hash of the generated number using the cryptographic library.
4. The host asks the **STSAFE-A110** to generate a signature of the computed hash using the `StSafeA_GenerateSignature` API. The **STSAFE-A110** sends back the generated signature.
5. The host verifies the generated signature with the public key sent by the **STSAFE-A110** in step 1.
6. The signature verification result is printed.

Revision history

Table 6. Document revision history

Date	Revision	Changes
09-Dec-2019	1	Initial release.
13-Jan-2020	2	Removed <i>Licence information</i> section.
07-Feb-2022	3	<p>Updated list of features illustrated by demonstration codes in Introduction.</p> <p>Removed <i>List of acronyms</i> table and inserted <i>glossary</i> at the end.</p> <p>Small text change and updated colors in Figure 1. STSAFE-A1xx middleware architecture.</p> <p>Updated Figure 2. STSAFE-A1xx middleware in an STM32Cube application.</p> <p>Updated Table 1. CORE module exported API.</p> <p>Removed <code>StSafeA_InitHASH</code> and <code>StSafeA_ComputeHASH</code> from Table 4. CRYPTO module exported APIs.</p> <p>Updated Section 3.8.2 Configuration steps.</p> <p>Updated Section 4.2 Pairing.</p> <p>Updated Section 4.3 Key establishment (establish secret).</p> <p>Added Section 4.5 Key pair generation.</p> <p>Small text changes.</p>

Glossary

AES Advanced encryption standard

ANSI American National Standards Institute

API Application programming interface

BSP Board support package

CA Certification Authority

CC Common Criteria

C-MAC Command MAC

ECC Elliptic curve cryptography

ECDH Elliptic curve Diffie–Hellman

ECDHE Elliptic curve Diffie–Hellman - ephemeral

EWARM IAR Embedded Workbench® for Arm®

HAL Hardware abstraction layer

IAR Systems® World leader in software tools and services for embedded systems development.

IDE Integrated development environment. A software application that provides comprehensive facilities to computer programmers for software development.

IO Input/output

IoT Internet of things

I²C Inter-integrated circuit

LL Low-level drivers

MAC Message authentication code

MCU Microcontroller unit

MDK-ARM Keil® microcontroller development kit for Arm®

MPU Memory protection unit

NVM Nonvolatile memory

OS Operating system

SE Secure element

SHA Secure Hash algorithm

SLA Software license agreement

ST STMicroelectronics

TLS Transport layer security

USB Universal serial bus

Contents

1	General information	2
2	STSAFE-A110 secure element	3
3	STSAFE-A1xx middleware description	4
3.1	General description	4
3.2	Architecture	4
3.3	CORE module	7
3.4	SERVICE module	9
3.5	CRYPTO module	10
3.6	Templates	11
3.7	Folder structure	12
3.8	How to: integration and configuration	13
3.8.1	Integration steps	13
3.8.2	Configuration steps	13
4	Demonstration software	14
4.1	Authentication	14
4.2	Pairing	14
4.3	Key establishment (establish secret)	15
4.4	Wrap/unwrap local envelopes	16
4.5	Key pair generation	16
	Revision history	18
	List of tables	21
	List of figures	22

List of tables

Table 1.	CORE module exported API	7
Table 2.	Exported STSAFE-A110 CORE module APIs	9
Table 3.	SERVICE module exported APIs	9
Table 4.	CRYPTO module exported APIs	10
Table 5.	Templates	11
Table 6.	Document revision history	18

List of figures

Figure 1.	STSAFE-A1xx middleware architecture	4
Figure 2.	STSAFE-A1xx middleware in an STM32Cube application	6
Figure 3.	CORE module architecture.	7
Figure 4.	SERVICE module architecture	9
Figure 5.	CRYPTO module architecture.	10
Figure 6.	Project file structure	12
Figure 7.	Key establishment command flow	16

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2022 STMicroelectronics – All rights reserved