life.augmented

# AN5582

Application note

# TruST25 embedded library

## Introduction

TruST25 is a set of industrialization processes, software tools, and specific products deployed by STMicroelectronics to guarantee and verify the identity of an ST25 NFC/RFID tag. The TruST25 digital signature is based on a digital signature embedded in the tag that can be verified by any user. This signature is generated and verified using cryptographic tools.

The TruST25 embedded library is a middleware that can be used from an ST25R reader application to verify the digital signature of the ST25 tags.

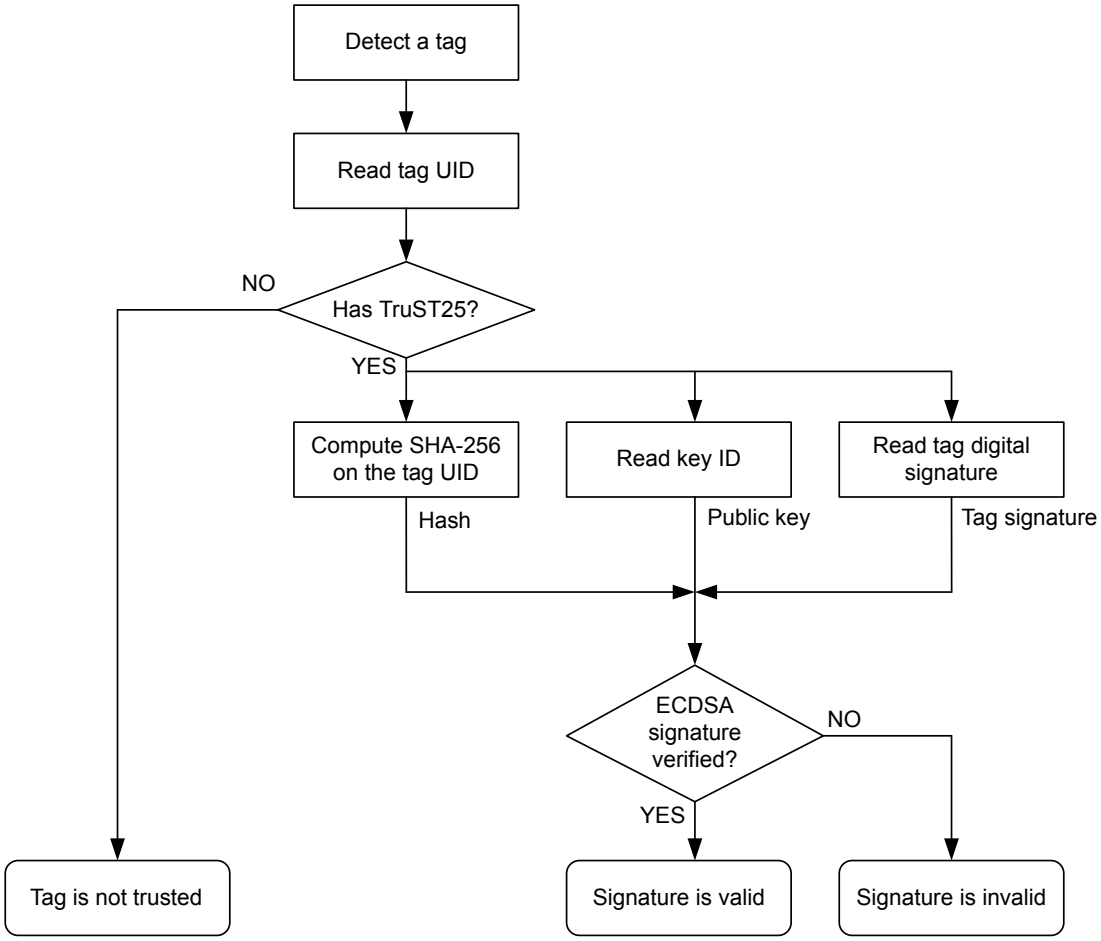
This document describes how to benefit from the TruST25 embedded library in an application.

The TruST25 embedded library is part of the ST25 NFC embedded library package (STSW-ST25R-LIB). The STSW-ST25R-LIB embedded software provides middlewares and their associated examples that can be reused when developing an application with ST25R products.

# 1 TruST25 digital signature verification process

The following flowchart explains the TruST25 digital signature verification process.

**Figure 1. TruST25 digital signature verification process**




The TruST25 digital signature verification process is managed through the following steps:

1. Wait for a tag being detected.
2. Read the tag's UID. The UID is usually known from the previous tag detection step.
   – From this UID, verify that the tag is an ST25 tag supporting the digital signature.
   – If the tag does not have TruST25 support, the process ends with the status "Tag not trusted".
3. If the ST25 tag supports the digital signature, compute the SHA-256 of its UID.
4. Read the Key ID, defining which ECC key is used to generate the digital signature.
5. Compute the signature from the hash and the ECC public key defined by the key ID, using the ECDSA on the secp128r1 elliptic curve.
6. Read the digital signature from the tag.
7. If the computed and read signatures are the same, the process ends with the "valid signature" status and the tag can be trusted by the application. If the signatures are different, the process ends with the "invalid signature" status, and the tag must be considered as nongenuine by the application.

# 2 TruST25 embedded library

The TruST25 embedded library provides an implementation of the process described in Section 1 that can be run on an MCU. The TruST25 embedded library API can be called by an embedded application to verify the digital signature of an ST25 tag. The TruST25 process is run in a blocking way, returning when the digital signature verification process is over.

## 2.1 TruST25 APIs

The TruST25 embedded library offers a single function to verify any ST25 tag supporting the digital signature. This function is implemented in the `trust25_signature.c` file.

**Table 1. `trust25_signature.c` file**

| Function | `trust25_signature_status_t`<br>`TruST25_SignatureVerification(trust25_tag_type_t tagType,`<br>`uint8_t* devUid,`<br>`uint32_t nbUidDigit);` |
|---|---|
| Parameters | `tagType`: input the type of the tag to be verified<br>• `TRUST25_TAG_NFCA_T2T`: for NFC Forum Type 2 tag<br>• `TRUST25_TAG_NFCA_T4T`: for NFC Forum Type 4 tag<br>• `TRUST25_TAG_NFCV`: for NFC Forum Type 5 tag<br>`devUid`: input the tag's UID<br>`nbUidDigit`: input the number of bytes of the tag's UID |
| Returns | `TRUST25_SIGNATURE_VALID`: when the verification process is successful<br>`TRUST25_SIGNATURE_INVALID`: when the tag and the computed signature are different<br>`TRUST25_NOT_TRUSTED`: when the tag does not support the digital signature<br>`TRUST25_SIGNATURE_UNDEFINED`: when an error prevented the verification |
| Description | This function runs the TruST25 digital signature verification process. |

## 2.2 TruST25 middleware interface file

The TruST25 embedded library is delivered as an STM32Cube middleware. This means that the library is made independent of the hardware: MCU and NFC reader. An interface header file defines the lowest layer API for the middleware, to be implemented for the targeted hardware.

A template for this Interface header file is available here:

`Middlewares/ST/TruST25/trust25_interface_template.h`.

The Interface files for the following ST25R readers are available in the TruST25 embedded library.

`Projects/STM32L476RG-Nucleo/Applications/X-NUCLEO-NFC0xA1/trust25_signature/Src/trust25_interface.c` implementation is the same for all ST25R readers:

- X-NUCLEO-NFC06A1 featuring the ST25R3916 NFC reader
- X-NUCLEO-NFC05A1 featuring the ST25R3911 NFC reader
- X-NUCLEO-NFC03A1 featuring the ST25R95 NFC reader

The interface for the TruST25 middleware is described in the following Table 2 to Table 8.

**Table 2. TruST25_SHA256_HASH_DigestCompute function**

| | |
|---|---|
| **Function** | `trust25_status_t`<br>`TruST25_SHA256_HASH_DigestCompute(uint8_t* InputMessage,`<br>`uint32_t InputMessageLength,`<br>`uint8_t *MessageDigest,`<br>`int32_t* MessageDigestLength);` |
| Parameters | `InputMessage`: The buffer containing the data to work on.<br>`InputMessageLength`: The number of bytes in the buffer containing the data to work on.<br>`MessageDigest`: The buffer used to return the SHA-256.<br>`MessageDigestLength`: The number of bytes of the SHA-256. |
| Returns | `TRUST25_SUCCESS`: SHA-256 is successfully computed.<br>`TRUST25_ERROR`: SHA-256 computation fails. |
| Description | This function computes the SHA-256 of the provided InputMessage. |

**Table 3. TruST25_ECDSA_VerifySignature function**

| | |
|---|---|
| **Function** | `trust25_signature_status_t TruST25_ECDSA_VerifySignature(`<br>`const uint8_t *InputMessage,`<br>`uint32_t InputMessageLength,`<br>`const uint8_t * sign_r,`<br>`int32_t sign_r_size,`<br>`const uint8_t * sign_s,`<br>`int32_t sign_s_size,`<br>`const uint8_t * pub_x,`<br>`int32_t pub_x_size,`<br>`const uint8_t * pub_y,`<br>`int32_t pub_y_size);` |
| Parameters | `InputMessage`: The buffer containing the signed data.<br>`InputMessageLength`: The number of bytes in the signed data.<br>`sign_r`: The R part of the signature.<br>`sign_r_size`: The number of bytes in the R part of the signature.<br>`sign_s`: The S part of the signature.<br>`sign_s_size`: The number of bytes in the S part of the signature.<br>`pub_x`: X coordinate of the public key to be used for the verification.<br>`pub_x_size`: The number of bytes of the X coordinate of the public key.<br>`pub_y`: Y coordinate of the public key to be used for the verification.<br>`pub_y_size`: The number of bytes of the Y coordinate of the public key. |
| Returns | `TRUST25_SIGNATURE_VALID`: The signature is successfully verified.<br>`TRUST25_SIGNATURE_INVALID`: The signature is not valid.<br>`TRUST25_SIGNATURE_UNDEFINED`: An error occurred during the signature verification. |
| Description | This function verifies the provided ECDSA signature (R and S parts) against the provided InputMessage and public key. |

**Table 4. TruST25_T4A_SelectFile function**

| Function | `trust25_status_t TruST25_T4A_SelectFile(uint16_t fileId);` |
|---|---|
| Parameters | `fileId`: The ID of the file to be selected. |
| Returns | `TRUST25_SUCCESS`: The specified file is successfully selected.<br>`TRUST25_ERROR`: An error occurred when selecting the specified file. |
| Description | This function selects a file on the currently selected T4 tag. |

**Table 5. TruST25_T4A_ReadData function**

| Function | `trust25_status_t TruST25_T4A_ReadData(uint8_t* data,`<br>`uint16_t offset,`<br>`uint16_t length);` |
|---|---|
| Parameters | `data`: The buffer to return the read data.<br>`offset`: The offset where to start reading the data.<br>`length`: The number of bytes to be read. |
| Returns | `TRUST25_SUCCESS`: The data is successfully read.<br>`TRUST25_ERROR`: An error occurred when reading the data |
| Description | This function reads the data from the selected file on the currently selected T4 tag. |

**Table 6. TruST25_T5_SendReceiveCmd function**

| Function | `trust25_status_t TruST25_T5_SendReceiveCmd(uint8_t * tx,`<br>`uint32_t tx_length,`<br>`uint8_t *rx,`<br>`uint16_t rx_length);` |
|---|---|
| Parameters | `tx`: The buffer containing the command to be sent.<br>`tx_length`: The number of bytes in the command to be sent.<br>`rx`: The buffer to return the response.<br>`rx_length`: The number of expected bytes in the response (fewer bytes trigger an error, extra bytes are discarded). |
| Returns | `TRUST25_SUCCESS`: The command is successful and the expected number of bytes is responded.<br>`TRUST25_ERROR`: The command returns an error, or the response length is too short. |
| Description | This function sends a command to a T5 tag and returns the response. |

**Table 7. TruST25_T2_ReadBlock function**

| Function | `trust25_status_t TruST25_T2_ReadBlock(uint16_t block_address,`<br>`uint8_t *rx);` |
|---|---|
| Parameters | `block_address`: The address of the block to read.<br>`rx`: The buffer to return the response. |
| Returns | `TRUST25_SUCCESS`: The command is successful and the expected number of bytes is responded.<br>`TRUST25_ERROR`: The command returns an error, or the response length is too short. |
| Description | This function reads a block from a T2 tag. |

Table 8. **TruST25_T2_SendReceiveCmd function**

| Function | `trust25_status_t TruST25_T2_SendReceiveCmd(uint8_t * tx, uint32_t tx_length, uint8_t *rx, uint16_t rx_length);` |
|---|---|
| Parameters | `tx`: The buffer containing the command to be sent.<br>`tx_length`: The number of bytes in the command to be sent.<br>`rx`: The buffer to return the response.<br>`rx_length`: The number of expected bytes in the response (fewer bytes trigger an error, extra bytes are discarded). |
| Returns | `TRUST25_SUCCESS`: The command is successful and the expected number of bytes is responded.<br>`TRUST25_ERROR`: The command returns an error, or the response length is too short. |
| Description | This function sends a command to a T2 tag and returns the response. |

## Revision history

Table 9. **Revision history**

| Date | Revision | Changes |
|---|---|---|
| 17-Dec-2020 | 1 | Initial release. |

# Contents

# List of tables

# List of figures