Application note

ST25DV-I2C crypto demonstration

Introduction

This application note describes how to develop an encrypted communication over NFC between a STM32 microntroller and a smartphone, thanks to the ST25DV-I2C series Dynamic NFC Tags.

The ST25DV-I2C is a dynamic NFC tag IC able to communicate with smartphone and NFC readers, and also with a microcontroller thanks to its I2C interface. Its fast transfer mode feature speeds up the communication between those two interfaces.

The following packages are available on www.st.com for this demonstration:

- STSW-ST25DV003 firmware
- STSW-ST25003 Android™ application

1 General information

This document is applied to STM32L476 Arm®-based devices.

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

2 Cryptographic keys overview

Several cryptographic keys are used in this demonstration.

Figure 1. "ST25DV-I2C Crypto Demo" keys overview

2.1 Keys used on ST25DV-I2C-DISCO side

Table 1. ST25DV-I2C-DISCO ECC key pair

Key name	ST25DV-I2C-DISCO ECC key pair
Type	Elliptic curve cryptography (ECC) key pair based on the curve 'prime256v1'.
Creation	Keys are part of the ST25DV-I2C-DISCO firmware. The public key is signed by the manufacturer to prove its authenticity.
Purpose	Asymmetric keys used when communicating with the Android™ phone. Both devices exchange their public key. Then they combine their own private key with the public key of the peer device to build a shared secret (ECDH). An AES session key is derived from this shared secret and a random number.

Table 2. ST25DV-I2C-DISCO AES session key

Key name	ST25DV-I2C-DISCO AES session key
Type	AES GCM key of 256 bits.
Creation	Session key created every time a ST25DV-I2C-DISCO board is tapped. A shared secret between the Android phone and the ST25DV-I2C-DISCO is established thanks to Diffie Hellman (ECDH). A session key is derived from the shared secret and a random number.
Purpose	Key used to encrypt all the exchanges between the Android phone and the ST25DV-I2C-DISCO. The same session key is created by the Android phone.

2.2 Keys used on Android side

Table 3. Android ECC key pair

Key name	Android ECC key pair
Type	ECC key pair based on the curve 'prime256v1'
Creation	Key pair created once, the first time that user ever launches this application. It is stored encrypted by the AES KeyStore key in the Android phone's shared preferences.
Purpose	Asymmetric keys used when communicating with the ST25DV-I2C-DISCO. Both devices exchange their public key. Then they combine their own private key with the public key of the peer device to build a shared secret (ECDH). An AES session key is derived from this shared secret and a random number.

Table 4. Manufacturer certificate

Key name	Manufacturer certificate
Type	Certificate containing ECC public key
Creation	Certificate stored in the assets of the Android application.
Purpose	This certificate is used by the Android application to check that the public key received from the ST25DV-I2C-DISCO is signed by the manufacturer.

Table 5. Android AES session key

Key Name	Android AES session key
Type	AES GCM key of 256 bits.
Creation	Session Key created every times user taps a ST25DV-I2C-DISCO board. A shared secret between the Android phone and the ST25DV-I2C-DISCO is established thanks to Diffie Hellman (ECDH). A session key is derived from the shared secret and a random number.
Purpose	Key used to encrypt all the exchanges between the Android phone and the ST25DV-I2C-DISCO. The same session key is created by the ST25DV-I2C-DISCO board.

Table 6. Android KeyStore key

Key name	Android KeyStore key
Type	AES CBC key of 256 bits.
Creation	Key created once, the first time that user ever launches this application.
Purpose	Key used to encrypt and decrypt the ECC key pair when it is saved in the Android phone shared preferences. This key is stored in Android KeyStore so nobody can retrieve it (only the "ST25DV-I2C Crypto Demo" application can use it). This is the master key of this application. The application cannot be used without it.

3 Keys management

Different methods can be implemented to provision the keys, depending on the use cases.

In the "ST25DV-I2C Crypto Demo", the first user of the ST25DV-I2C-DISCO board becomes the "Authorized User" of the product and nobody else is allowed to retrieve data or configure the product. The ECC public keys are used to identify the "Authorized User" and the ST25DV-I2C-DISCO. A login is also provided by the "Authorized User".

In the "ST25DV-I2C Crypto Demo", the ECC key pairs are provisioned in the following way:

- For the ST25DV-I2C-DISCO, the ECC key pair is part of the firmware flashed into the program memory (including a manufacturer digital signature of the public key).
- For the smartphone, the ECC key pair is created when first launched, and kept forever.

This key provisioning method may apply for instance to sensors measuring personal data that are later synchronized with a smartphone. In that use case, the user pairs its smartphone with the device before usage, and its personal data is only accessible by this smartphone. Additionally the Android application can be configured to require a user authentication: fingerprint or pin code.

More complex use cases may be required, for instance it is possible to implement a multi-users system. The "Authorized User" can add some other users (identified by a login and an ECC public key), that require to use this product. The mechanism used for authentication remains the same. The only difference is that the microcontroller must save the login and public keys of several users (whereas it only saves the "Authorized User" credentials in this demonstration).

Another use case is, for a company, to sign the ECC public key of the Android phone. The device only accepts to communicate with a phone whose public key is signed by the manufacturer. For example, a gas counter containing a ST25DV-I2C series Dynamic NFC Tag and a STM32 microcontroller. A technician from the gas company may want to setup a secure connection with the microcontroller to configure the gas counter or retrieve data. The microcontroller accepts the connection only if the ECC public key contained in the technician Android phone is signed by the manufacturer of the gas counter.

4 Android™

4.1 Conscrypt crypto library

The "ST25DV-I2C Crypto Demo" application is using *Conscrypt* as the Java™ Security provider. Conscrypt is provided by Google®. It uses BoringSSL to provide cryptographic primitives for Java applications on Android.

Another widely used cryptographic library on Android is Bouncy Castle. However, the Bouncy Castle implementations of many algorithms are deprecated in Android 9.

4.2 Android KeyStore

The Android KeyStore is used to store sensitive information like cryptographic keys. The private key of Android ECC key pair must be stored in a secure place because anyone who get it, can use it to access the ST25DV-I2C-DISCO board.

This ECC key pair is usually created directly in the Android KeyStore. By this way, only the program who created it is able to use it for cryptographic operations and the key pair is non-exportable. This scheme is perfect when using the key to encrypt, decrypt or sign but it does not able to combine keys with algorithms like ECDH, so a workaround is needed for this purpose.

The Android ECC pair is stored encrypted in the "Android Shared Preferences". The encryption is done with a master key called AES KeyStore key created directly in the Android KeyStore. The "ST25DV-I2C Crypto Demo" application is the only one having access to this master key so it is the only one that is able to decrypt the Android ECC key pair available in the shared preferences.

Note: The AES session key is created every times the user tap the ST25DV-I2C-DISCO board so it is not stored in the non-volatile memory.

5 ST25DV-I2C-DISCO firmware

5.1 STM32 security features

The ST25DV-I2C-DISCO motherboard embeds a STM32L476 device which provides several protection mechanisms that are leveraged to protect sensible code such as the keys or the cryptographic processing.

Following protections are available:

- *Code write protection* is used to prevent any change in code ensuring the security.
- *Code readout protection* restricts access of the Flash memory and SRAM to user execution mode, when the MCU has booted from Flash memory (no access in debug mode or if the MCU boots from another memory).
- *Proprietary Code readout protection* prevents any data access to part of the code stored in Flash memory, this is used to prevent any access to the protected keys (only accessible by executing a dedicated getter function).
- *The firewall* is able to reset the MCU in case a protected memory (Flash memory or SRAM) is accessed while firewall is enabled. A specific call gate sequence must be executed to access the protected areas. The Firewall cannot be disabled once it has been enabled.
- *Memory protection unit*: the MPU is used to make an embedded system more robust by splitting the memory map for Flash memory and SRAMs into regions having their own access rights. In the "ST25DV-I2C Crypto Demo", MPU is configured in order to ensure that no other code is executed from any memories during code execution.
- *Tamper*: the anti-tamper protection is used to detect physical tampering actions on the device and to take related counter measures. In case of tampering detection, the application example forces a reboot.
- *Independent watchdog*: IWDG is a free-running down-counter. Once running, it cannot be stopped. It must periodically refresh before it causes a reset. This mechanism control execution duration.
- *Random number generator* is used to provide reliable random numbers for cryptographic processing.

These security features are available for other STM32 families, contact ST support to find the suitable STM32 MCU for user application.

The "ST25DV-I2C Crypto Demo" only uses native STM32 protections against outer and inner attacks to prevent compromising sensible data. For a higher level of security, an addition of an STSAFE secure element provides secure functions such as:

- Authentication
- TLS secure channel key establishment
- Data and certificate storage
- Signature verification

More information is available on www.st.com.

5.2 Security implementation in firmware

5.2.1 Firmware overview

The "ST25DV-I2C Crypto Demo" firmware reuses part of the X-Cube SBSFU software package (available on www.st.com) to enable the STM32 features (such as Firewall, Readout protection).

The "ST25DV-I2C Crypto Demo" firmware is divided in two different projects/binaries:

- The "Secure Engine"
- The "ST25DV-I2C Crypto Demo", above the "Secure Engine" binary

5.2.2 "Secure Engine" firmware: SECoreBin

The SECoreBin library is built around the *se_crypto_services.c* file which handles all the required cryptographic operations, described hereafter.

Other files in the library are reused from the SBSFU example, and they structure the firmware so it can be protected by STM32 firewall IP and readout protections, mainly:

- A Call gate entry mechanism is provided to only allow legitimate cryptographic instructions from the application to pass the firewall. Interruptions are disabled during secure processing.

- The ECC keys and public key signature are embedded in the executed code, and can thus be protected by the proprietary code readout protection (no data access to this section is allowed).
- The computed "Shared Secret" and the "Session key" are isolated in a dedicated SRAM area which is also protected by the firewall (same for the "Secure Engine" execution stack).

The "Secure Engine" module uses the STM32 cryptographic library (see Section 5.2.4 STM32 cryptographic library) to perform the low level cryptographic processing.

All accesses to the "Secure Engine" are performed by calling the *SE_CallGate* function, providing a service ID and the associated parameters. The function returns 0 (SE_ERROR) in case of failure, or a non-zero value for a success (SE_SUCCESS)

Following tables describe the services provided by the "Secure Engine" module.

Table 7. SE_INIT_ID

ID	SE_INIT_ID
Parameters	uint32_t SystemCoreClock: the system core clock frequency
Description	Initialize the required crypto peripherals on the STM32L476: the CRC and the random number generator (RNG). SystemCoreClock is required to setup a local Tick function.

Table 8. SE_CRYPTO_GENERATE_AES_KEY_ID

ID	SE_CRYPTO_GENERATE_AES_KEY_ID
Parameters	uint8_t* publicKey: A pointer to a 65 bytes buffer, storing the peer device public key
Description	Combine the peer device public key and the firmware private key (running the elliptic curve Diffie Hellman method) to get a 256 bit key, used as a "Shared Secret" in the demonstration. It relies on the STM32 Crypto library function ECCgetPointCoordinate.

Table 9. SE_CRYPTO_DERIVE_KEY_ID

ID	SE_CRYPTO_DERIVE_KEY_ID
Parameters	uint8_t* salt: pointer to the buffer used to return the 32 bytes of random data that has been used for the derivation. salt size must be at least 32 bytes. uint32_t* length: pointer to return the number of bytes of the salt (must be 32).
Description	This method defines a 32 bytes random number (salt) and uses it to derive, from the Shared Secret, the Session Key. The random 32 bytes salt, has to be communicated to the peer device in order for it to do the same derivation and get the same Session Key.

Table 10. SE_CRYPTO_KEY_AVAILABLE_ID

ID	SE_CRYPTO_KEY_AVAILABLE_ID
Parameters	Crypto_Key_Status_t* keyStatus: The current AES key status CRYPTO_KEY_STATUS_UNDEF = 0, crypto service has not been initialized yet CRYPTO_NO_KEY = 1, no Shared Secret computed yet CRYPTO_SHARED_SECRET_DEFINED = 2, a Shared Secret has been computed CRYPTO_SESSION_KEY_DEFINED = 3, a session key has been derived from the Shared Secret
Description	Getter for the AES Key status. Used in the "ST25DV-I2C Crypto Demo" to know if a "Shared Secret" has already been defined.

Table 11. SE_CRYPTO_RESET_KEY_ID

ID	SE_CRYPTO_RESET_KEY_ID
Parameters	Crypto_Key_Status_t status: the expected status to reset CRYPTO_KEY_STATUS_UNDEF = 0, crypto service has not been initialized yet CRYPTO_NO_KEY = 1, no Shared Secret computed yet CRYPTO_SHARED_SECRET_DEFINED = 2, Shared Secret computed CRYPTO_SESSION_KEY_DEFINED = 3, a session key has been derived from Shared Secret
Description	This method allows the application to reset the Shared Secret and session key. If status is greater than current status, nothing is done. Shared Secret and session keys are erased from memory if new status requires it.

Table 12. SE_CRYPTO_ENCRYPT_ID

ID	SE_CRYPTO_ENCRYPT_ID
Parameters	uint8_t* data: pointer on the data to encrypt uint32_t length: number of bytes to encrypt uint8_t* enc_data: pointer to the buffer to be used for encrypted data. enc_data size must be at least length + 28 bytes. int32_t* enc_length: pointer to return the number of bytes of the encrypted data (includes 12 bytes of IV and 16 bytes of GMAC)
Description	Defines the initialization vector nonce, encrypts the provided data and computes the GMAC, using the STM32 Crypto library AES_GCM_Encrypt_Init, AES_GCM_Encrypt_Append, AES_GCM_Encrypt_Finish functions.

Table 13. SE_CRYPTO_DECRYPT_ID

ID	SE_CRYPTO_DECRYPT_ID
Parameters	uint8_t* data: pointer on the data to decrypt uint32_t length: number of bytes to decrypt (starting by 12 bytes of the IV and ending with the 16 bytes of GMAC) uint8_t* dec_data: pointer to the buffer to be used for decrypted data. dec_data size must be at least length - 28 bytes. int32_t* dec_length: pointer to return the number of bytes of the decrypted data
Description	Extracts the initialization vector, decrypts the provided data and checks the GMAC validity, using the STM32 Crypto library AES_GCM_Decrypt_Init, AES_GCM_Decrypt_Append, AES_GCM_Decrypt_Finish functions.

Table 14. SE_CRYPTO_HASH_ID

ID	SE_CRYPTO_HASH_ID
Parameters	uint8_t* data: pointer on the data to be used as an input for the Hash uint32_t length: number of bytes to be considered in data buffer uint8_t* hash_data: pointer to the buffer to be used for the hash value. hash_data size must be at least 32 bytes. int32_t* hash_length: pointer to return the number of bytes of the hash.
Description	Generates a SHA256 Hash value from the given data, using the STM32 Crypto library SHA256_Init, SHA256_Append, SHA256_Finish functions.

Table 15. SE_CRYPTO_GET_RANDOM_NUMBER_ID

ID	SE_CRYPTO_GET_RANDOM_NUMBER_ID
Parameters	Crypto_Random_Number_t pointer (16 bytes buffer)
Description	Uses the random number generator (RNG) peripheral of the STM32L476 to set 16 bytes with random values.

Table 16. SE_CRYPTO_GET_PUBLIC_KEY_ID

ID	SE_CRYPTO_GET_PUBLIC_KEY_ID
Parameters	uint8_t * publicKey: 65 bytes buffer used to store the Public Key
Description	Executes a function to retrieve the ECC public key

Table 17. SE_CRYPTO_GET_PUBLIC_KEY_SIGNATURE_ID

ID	SE_CRYPTO_GET_PUBLIC_KEY_SIGNATURE_ID
Parameters	uint8_t * publicKeySig: 71 bytes buffer used to store the Public Key Signature
Description	Executes a function to retrieve the ECC public key signature

Table 18. SE_CRYPTO_GET_LOGIN_ID

ID	SE_CRYPTO_GET_LOGIN_ID
Parameters	Crypto_AuthenticationEnv_t * auth: a pointer to an authentication structure
Description	Reads the authentication data stored in the Flash memory and returns it in "auth". The authentication structure embeds the peer device login and its associated PublicKey.

Table 19. SE_CRYPTO_SET_LOGIN_ID

ID	SE_CRYPTO_SET_LOGIN_ID
Parameters	Crypto_AuthenticationEnv_t * auth: a pointer to an authentication structure
Description	Stores the provided authentication data in the Flash memory. The authentication structure embeds the peer device login and its associated PublicKey.

Table 20. SE_LOCK_RESTRICT_SERVICES

ID	SE_LOCK_RESTRICT_SERVICES
Parameters	None
Description	Restrict the "Secure Engine" services.

5.2.3 "ST25DV-I2C Crypto Demo" application layer

This part of the firmware implements the "ST25DV-I2C Crypto Demo" itself, managing the protocol and transfer over NFC, and relying on the "Secure Engine" for cryptographic processing.

It reuses some parts of the X-Cube SBSFU firmware:

- The boot procedure, including security checks
- The "Secure Engine" call gate API, to interact with the "Secure Engine" while the firewall is enabled
- The Target protection module, which enables the different protections available on the STM32.

By default, only the firewall and the MPU protections are enabled (SFU_FWALL_PROTECT_ENABLE and SFU_MPU_PROTECT_ENABLE macros defined in main.h). It may be useful to unset the firewall protection for debugging (as the firewall resets the STM32 each time the debugger tries to access protected code or data).

Other STM32 protections can be enabled by defining the corresponding macro:

- SFU_WRP_PROTECT_ENABLE to apply write protections
- SFU_RDP_PROTECT_ENABLE to apply readout protections
- SFU_PCROP_PROTECT_ENABLE to apply proprietary readout protections
- SFU_MPU_PROTECT_ENABLE to enable the memory protection unit
- SFU_FWALL_PROTECT_ENABLE to enable the firewall (for full protection WRP, RDP – level2 - & PCROP must be set)
- SFU_TAMPER_PROTECT_ENABLE to use tamper protection
- SFU_DAP_PROTECT_ENABLE to disable debug pins
- SFU_DMA_PROTECT_ENABLE to disable DMA
- SFU_IWDG_PROTECT_ENABLE to enable independent watchdog

For production, please follow the instructions:

1. Activate all required security protections: SFU_xxx_PROTECT_ENABLE
2. Deactivate verbose mode: SFU_VERBOSE_DEBUG_MODE
3. Deactivate SFU_FWIMG_BLOCK_ON_ABNORMAL_ERRORS_MODE
4. Deactivate SECBOOT_OB_DEV_MODE
5. Activate SFU_FINAL_SECURE_LOCK_ENABLE to configure RDP level 2

Read Protection Level 2 is mandatory to achieve the highest level of protection and to implement a "Root of Trust".

It is user's responsibility to activate it in the final software to be programmed during the product manufacturing stage.

In production mode, the boot checks the option byte values (RDP, WRP and PCROP) and blocks execution in case a wrong configuration is detected.

5.2.4 STM32 cryptographic library

The "ST25DV-I2C Crypto Demo" "Secure Engine" relies on the STM32 cryptographic library in X-CUBE-CRYPTOLIB package (available on www.st.com).

This library runs on all STM32 series with the firmware implementation and for dedicated devices some algorithms are supported with hardware acceleration to optimize the performance and the footprint.

Note: This library provides a random number generation API that can be used on STM32 not having hardware implementation of RNG.

The supported Crypto algorithms are (only the algorithms in **bold** font are used in the "ST25DV-I2C Crypto Demo"):

- AES-128, AES-192, **AES-256** bits:
  - ECB (Electronic Codebook Mode)
  - CBC (cipher-block chaining) with support for ciphertext stealing
  - CTR (counter mode)
  - CFB (cipher feedback)
  - OFB (output feedback)
  - CCM (counter with CBC-MAC)
  - **GCM (Galois counter mode)**

  - CMAC
  - KEY WRAP
  - XTS (XEX-based tweaked-code book mode with cipher text stealing)
- ARC4
- DES, TripleDES:
  - ECB (electronic code book mode)
  - CBC (cpher-block chaining)
- HASH functions with HMAC support:
  - MD5
  - SHA-1
  - SHA-224
  - **SHA-256**
  - SHA-384
  - SHA-512
- ChaCha20
- Poly1305
- CHACHA20-POLY1305
- Random engine based on DRBG-AES-128
- RSA signature functions with PKCS#1v1.5
- RSA encryption/decryption functions with PKCS#1v1.5
- ECC (elliptic curve cryptography):
  - Key generation
  - **Scalar multiplication (the base for ECDH)**
  - ECDSA
- ED25519
- Curve25519

5.2.5 "Secure Engine" and demonstration footprint

The following table shows the "ST25DV-I2C Crypto Demo" firmware code footprint.

The "Secure Engine" module numbers only consider the "Secure Engine" part of the firmware, and the main contributor to the data footprint is the isolated and protected stack of 2 KBytes.

The "ST25DV-I2C Crypto Demo" minimum set of feature disables the display and the picture transfer, which requires a large amount of code and data, while being useless on a real application. It can be compiled by having the CRYPTO_DEMO_MINIMUM_FEATURE macro defined.

The "ST25DV-I2C Crypto Demo" full features provides the number for the global demonstration.

The total lines provide numbers for "ST25DV-I2C Crypto Demo" including the "Secure Engine".

Table 21. "ST25DV-I2C Crypto Demo" firmware memory usage

Module	Code and read-Only	Data
Secure Engine	28.6 KBytes	2.3 KBytes
"ST25DV-I2C Crypto Demo" firmware with minimum set of features (without picture/ display support)	61.5 KBytes	5.7 KBytes
Total (minimum features)	90 KBytes	8 KBytes
"ST25DV-I2C Crypto Demo" firmware with full features (with pictures and display)	149 KBytes	81.5 KBytes
Total (full feature)	177.6 KBytes	83.8 KBytes

5.3 Configuring the keys

5.3.1 Generating keys and signature

Several security elements must be defined in the fimware to run the "ST25DV-I2C Crypto Demo" security scheme:

- An ECC key pair (in the "ST25DV-I2C Crypto Demo", the prime256v1 elliptic curve is used).
- A digital signature of the public key, signed using the manufacturer key (a manufacturer key has been specifically created for the demo).

Openssl or a similar tool can be used to generate each of these elements.

Generate the key pair:

```
openssl ecparam -name prime256v1 -genkey -noout -out ecc_keys.pem
```

See the generated keys:

```
openssl ec -in ecc_keys.pem -text
```

The public key, displayed by the previous command, needs then to be formatted as a binary file and this binary file must be signed with the manufacturer key:

```
openssl dgst -sha256 -sign <Manufacturer secret key as a PEM file> -hex -out publicKeySignatu
re.txt <Public key in binary format>
```

The *publicKeySignature.txt* file contains a valid digital signature for the ECC public key in a DER format (hex / encapsulated format).

5.3.2 Implementing keys in firmware

The private key, the public key and its digital signature must be implemented as executable code such as in the *se_key.s* file (in the SECoreBin project).

For instance:

```
EXPORT SE_ReadKey
SE_ReadKey
PUSH {R4-R11}
MOVW R4, #0x1ec5
MOVT R4, #0x2dae
MOVW R5, #0x7d68
MOVT R5, #0x41ce
MOVW R6, #0x5955
MOVT R6, #0xfa87
MOVW R7, #0x42ff
MOVT R7, #0x93bc
MOVW R8, #0xd035
MOVT R8, #0xcd9f
MOVW R9, #0x3ec5
MOVT R9, #0x6b20
MOVW R10, #0xb9c2
MOVT R10, #0x2e38
MOVW R11, #0x4033
MOVT R11, #0xc4f2
STM R0, {R4-R11}
POP {R4-R11}
BX LR
```

Note: The MOVT/MOVW instructions are little endian on STM32 (byte order must be reversed).

With this implementation the keys can be protected by the "Proprietary Code" readout protection of the STM32 (PCROP is enabled for the corresponding code section).

6 Frequently asked questions

- *Can someone use the login and the public key sent by the Android phone to get access to the ST25DV-I2C-DISCO data?*

  No.

  The main point is to protect access to the private keys used by the Android phone and the ST25DV-I2C-DISCO board. Someone spying the NFC connection can see the login used and the public keys exchanged but cannot get the corresponding private keys. Without the private keys, the hacker cannot find the "Shared Secret", and neither can find the AES session key derived from the "Shared Secret"; so its authentication fails.

- *What is the Login used for?*

  Basically, the exchange of the public key is enough. This public key is in binary format so adding a "Login" makes it more explicit when the access is granted or refused to a phone. The phone model is used as the "Login" name.

- *When doing the key exchange, is there a risk of “Man in the Middle” attack?*

  No.

  The comunication is protected against “Man in the middle” attack because the Android phone checks that the STM32 public key is signed by the manufacturer of the device. If there was a malicious person between the Android phone and the STM32 (which is very unlikely when using an NFC connection), the hacker can replace the STM32 public key by its own public key but this key is not signed by the manufacturer so it is not accepted by the Android phone.

Revision history

Table 22. **Document revision history**

Date	Version	Changes
27-May-2019	1	Initial release.

Contents

List of tables

List of figures