Mikko Larke

Provisioning embedded systems with dual-interface NFC tags

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Bachelor’s Thesis

2 May 2022

Abstract

Author:	Mikko Larke
Title:	Provisioning embedded systems with dual-interface NFC tags
Number of Pages:	33 pages
Date:	2 May 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Smart Systems
Supervisors:	Keijo Länsikunnas, Principal Lecturer

Embedded systems are often designed to work infinitely without interference from the end-user. Therefore, it is not beneficial to include comprehensive user interfaces for these products. Still, while most system configurations can be programmed during product software development, some settings can usually be end-user dependent. Thus, a solution for providing this information is often needed.

Dual-interface NFC tags are small, cost-efficient, and end-user-friendly components that can be easily included in embedded system hardware. This type of NFC tag has both wired and wireless interfaces, enabling replacing embedded systems' internal user interface with a mobile device, such as a smartphone.

The first part of this thesis introduces the NFC technology. It declares the working principles of the technology and explains the difference between NFC readers and tags. NFC technologies security issues are discussed, and solutions for improving security over NFC data transfers are presented. Lastly, the advantages and disadvantages of dual-interface NFC tags as a platform for provisioning embedded systems are compared.

The second part of the thesis focuses on implementing an API library that can be included in existing software to add NFC-based provisioning. The API was designed to use Espressif ESP32 microcontroller and STMicroelectronics ST25DV based dual-interface NFC tag. First, the hardware components are introduced, and their key features are explained. Each of the functions included in the API library is introduced, and their usage is explained. The development process for including API in an existing program is presented, and a practical use case is shown.

Keywords: NFC, Embedded Systems, Provisioning

Tiivistelmä

Tekijä:	Mikko Larke
Otsikko:	Sulautettujen järjestelmien provisiointi kahden käyttöliittymän NFC-tunnisteilla
Sivumäärä:	33 sivua
Aika:	2.5.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Älykkäät järjestelmät
Ohjaajat:	Keijo Länsikunnas, ohjaava opettaja

Sulatetut järjestelmät on usein suunniteltu toimimaan jatkuvasti ilman käyttäjien syötteitä. Tämän takia kattavan käyttöliittymän lisääminen järjestelmään ei ole kannattavaa. Kuitenkin, vaikka suurin osa järjestelmän asetuksista voidaan asettaa jo laitteen ohjelmistoa kehitettäessä, osa laitteen asetuksista voi vaatia toimenpiteitä loppukäyttäjältä. Tämän takia usein on tarpeellista kehittää ratkaisu järjestelmän käyttöönottoa varten.

Kahden rajapinnan NFC-tunnisteet ovat pieniä, kustannustehokkaita ja käyttäjäystävällisiä komponentteja, jotka on helppo lisätä osaksi sulautettua järjestelmää. Tunnisteessa on sekä langallinen että langaton rajapinta, mikä mahdollistaa sulautetun järjestelmän sisäisen käyttöliittymän korvaamisen käyttäjän omalla mobiililaitteella, kuten älypuhelimella.

Insinöörityön ensimmäinen osa esittelee NFC-teknologian. Siinä esitetään teknologian yleiset toimintaperiaatteet sekä erot NFC-lukijoiden ja tunnisteiden välillä. NFC-teknologian tietoturvaan liittyvät uhat sekä yleisimmät ratkaisut niihin on esitetty. Lopuksi vertaillaan teknologian vahvuuksia ja heikkouksia sulautettujen järjestelmien provisioinnin näkökulmasta.

Insinöörityön jälkimmäisessä osassa keskitytään kehittämään API-kirjasto, joka voidaan sijoittaa osaksi olemassa olevaa ohjelmistoa NFC-pohjaisen provisionnin mahdollistamiseksi. API suunniteltiin toimimaan Espressif ESP32 -mikrokontrollerin ja STMicroelectronics ST25DV -pohjaisen NFC-tunnisteen kanssa. Aluksi kirjastoa tukeva laitteisto ja sen tärkeimmät ominaisuudet on esitelty. Kaikki kirjaston sisältämät funktiot sekä niiden käyttö on esitelty. Lopuksi API-kirjaston käyttöönottoprosessi sekä käytännöllinen esimerkki käyttöönotosta esitetään.

Avainsanat: NFC, Sulautetut järjestelmät, Provisiointi

Contents

List of Abbreviations

1 Introduction 1

2 Overview of NFC protocol 2

2.1 NFC readers 3
2.2 NFC tags 4
2.3 Security 6
2.4 Benefits and weaknesses on NFC based provisioning 7

3 Technology used in the thesis 8

3.1 Espressif ESP32 8
3.1.1 ESP-IDF 10
3.1.2 I2C 10
3.2 STMicroelectronics ST25DV 12
3.2.1 Wired access to EEPROM memory 13
3.2.2 Write and read operations 14
3.2.3 Memory management 16
3.2.4 GPO interrupt events 17
3.2.5 Security 18

4 Documentation of the provisioning API 22

4.1 Read functions 22
4.2 Write functions 23
4.3 Password functions 24
4.4 Configuration functions 25

5 Provisioning software in practice 26

5.1 Demonstrational software 27

6 Conclusion 30

References 32

List of Abbreviations

AES: Advanced encryption standard. Specification for data encryption.

API: Application programming interface. Software interface used to provide service for another piece of software.

ECC: Elliptic-curve cryptography. Public-key cryptography based on elliptic curves over finite fields.

ECDH: Elliptic-curve Diffie-Hellman. Key agreement protocol used for two devices to form shared public-private key pairs to create one shared secret.

EEPROM: Electronically erasable programmable read-only memory. Non-volatile memory used to store small amounts of data where individual bytes can be erased and reprogrammed.

ESP-IDF: Espressif IoT Development Framework. Complete development environment for ESP32 for development in C programming language.

GPIO: General-purpose input/output. Uncommitted digital signal pin that can be initialized either as input or output.

GPO: General purpose output. Uncommitted digital signal pin that can be initialized as output.

I2C: Inter-integrated circuit. Synchronous multi-target and multi-controller serial communication bus.

IoT: Internet of Things. System of computing devices that share information over the internet on their own.

NFC: Near-field communication. The wireless communication protocol is used for communication between electronic devices over a short distance.

RF: Radiofrequency. Information transferring method based on electromagnetic radiation spectrum between circuits.

SCL: Serial Clock. The line that carries the clock signal. Mandatory for I2C protocol data transfers.

SDA: Serial data line. The line that master and slave devices use to transfer data. Mandatory for I2C protocol data transfers.

SoC: System on a Chip. An integrated circuit that contains most components of some electronic system, such as a computer.

SPI: Serial peripheral interface. Synchronous serial communication interface for short-distance communication.

1 Introduction

Most embedded systems are designed to run on their own infinitely with minimal user influence. Therefore, adding comprehensive user interfaces for embedded systems isn't often beneficial. However, even if the firmware is designed to run without interference, the initial provisioning process of the device usually still requires some input from the end-user. This can become cumbersome or even practically impossible with a non-existing user interface or if multiple devices need to be configured in a bunch. Common features that might need to be provisioned by end-users include credentials for a wireless network, range of measurements, connection keys to third-party services, and so on.

One of the multiple ways to approach embedded systems provisioning is a dual-interface NFC tag, which can acquire and transfer information with wireless and wired interfaces. This commonly means the process of the end user transferring information with an NFC transfer capable device, such as a smartphone, to an NFC tag connected embedded system with a wired connection, like I2C or SPI protocol.

This thesis was created for Sulaon Oy to improve the user experience of their embedded devices. This was done both in research of dual-interfaced NFC technologies and by developing C-programming language-based API used for provisioning Espressif ESP32 based embedded systems with STMicroelectronics ST25DV dual-interface NFC tag. These hardware components were chosen together with the commissioning company considering components they already were using. The practical use of this provisioning system relays both on this API for reading information and the Android application for transferring information. However, development for the mobile application was left outside of the scope of this thesis since that was ordered from a third-party service with greater expertise in the development of mobile applications.

This thesis contains six chapters. The first chapter introduces the project, sets the scope, and declares the contents of the study. Chapter 2 describes the

basics of NFC protocol, the differences between static and dynamic tags, how to prevent security breaches, and states the benefits and disadvantages of using this method for provisioning embedded systems. In the third chapter, the hardware used in this project is introduced with more information on key features of selected components. Following that, in chapter 4, the provisioning API is introduced and documented, and in chapter 5, a demonstrative use case for API in practice is shown. Finally, chapter 6 summarises the study.

2 Overview of NFC protocol

NFC protocol is contactless technology used to communicate between two devices over radiofrequency. It is based on RFID protocol and uses a base frequency of 13.56 MHz. The NFC devices are designed to work over short distances of only a few centimeters, giving the NFC protocol a security advantage compared to other wireless protocols that can be accessed over greater distances. The key features of the NFC protocol are the ability to send information to another NFC capable device by modulating the signal of the RF field, retrieve information by sensing load modulation generated by other NFC devices, and transfer power from an active NFC device to a passive NFC device, as seen in figure 1. [1.]

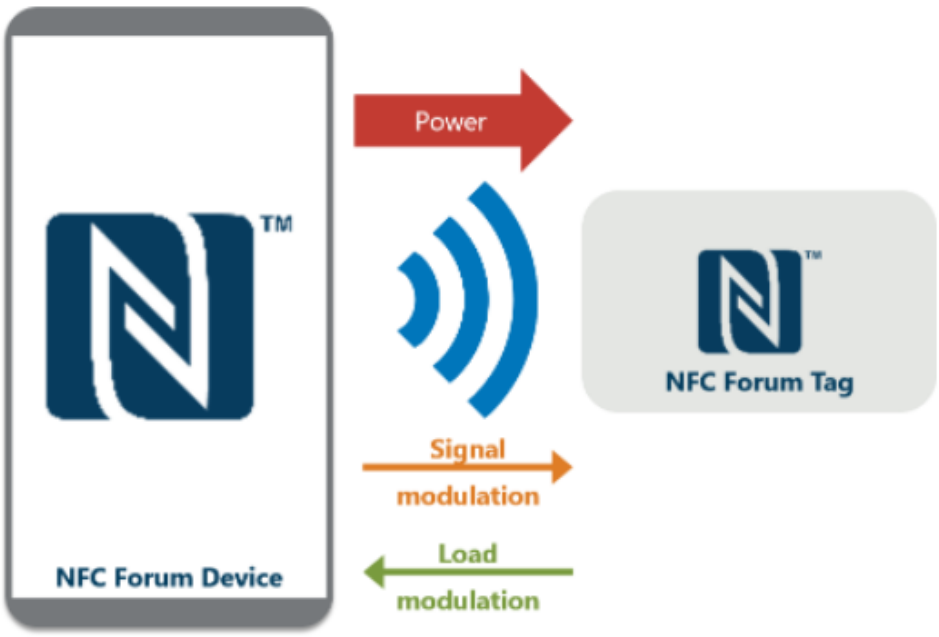

Figure 1. Process of exchanging information and power over NFC. [1.]

For the device to use the NFC protocol, it needs an NFC chip and EEPROM memory where information acquired over NFC is stored. NFC chips can broadly be divided into two main categories: NFC readers and NFC tags. On top of that, there are also NFC controllers, which are devices that can work as both NFC readers and NFC tags.

2.1 NFC readers

NFC readers are active devices that can act as transferring or receiving devices, and those are considered the main controllers over NFC communication between two devices. Their role in information transfer is to initiate communication, provide the power to the NFC device it connects to and send commands through a magnetic field. [2.] The general structure for an NFC reader can be seen in figure 2.

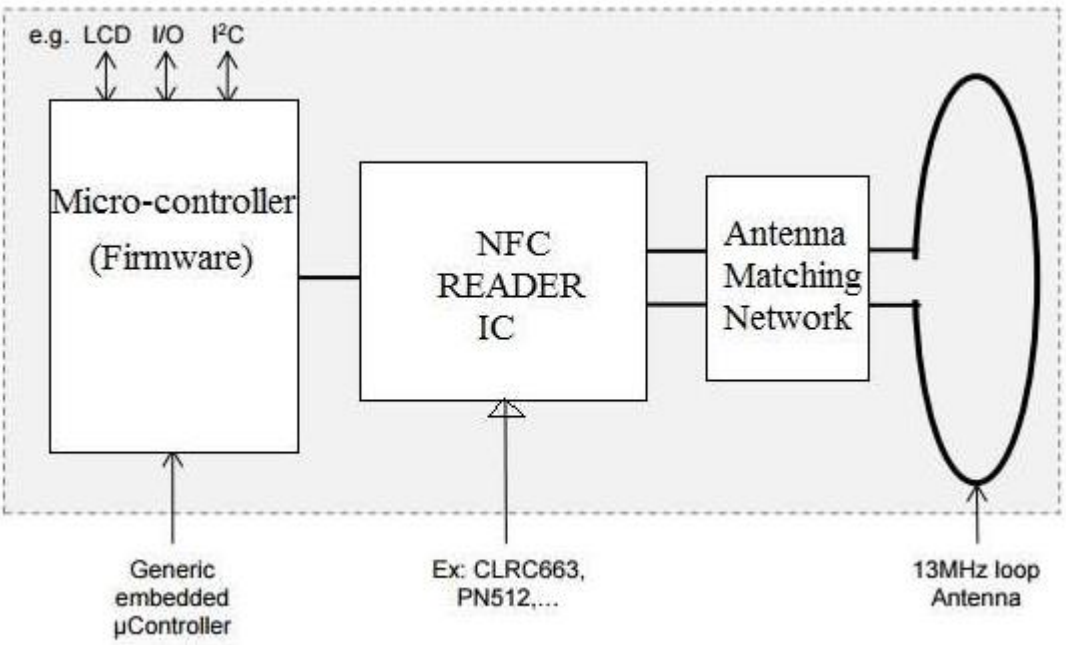

Figure 2. Block diagram for generic NFC reader. [3.]

NFC readers can be used in card emulation mode, peer-to-peer mode, and read/write mode. The card emulation causes the reader to act as a contactless smart card compliant with ISO/IEC 14443 standard. This enables the possibility to replace the most common smart cards with a single NFC reader. Peer-to-peer mode enables the communication between two active NFC devices as equals allowing both devices to serve as an endpoint. Reader/writer mode allows the active device to interact with passive NFC tags and transfer or acquire information. [4.]

2.2 NFC tags

NFC tags are passive devices that can be accessed with an NFC reader. NFC tags consist of the NFC chip, an antenna, and a substrate that ties other components together. NFC tags are powered through a magnetic field generated by the NFC reader, meaning that it doesn’t require its power source to be accessed. The general structure for an NFC tag can be seen in figure 3.

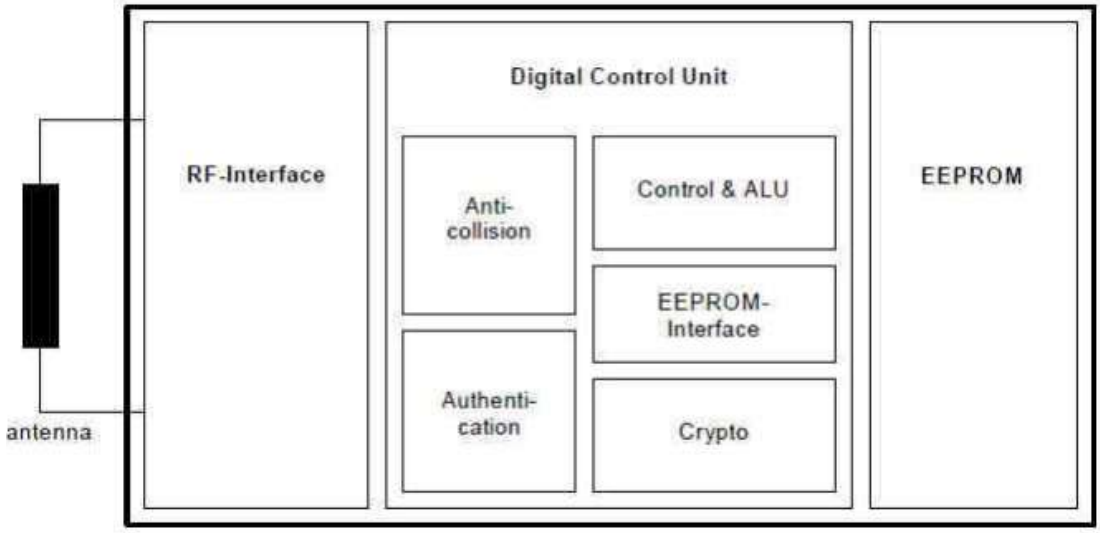

Figure 3. Block diagram for generic NFC tag. [3.]

During communication, the NFC tag responds to specific instructions sent by the NFC reader, which allows the tag, for example, to send information stored in the tag to read or write in new information provided by a reader. NFC tags can also have specific features programmed in, such as password protection, authentication information, or tamper detection. [4.]

NFC tags can be separated into single-interface tags and dual-interface tags. A more straightforward version from those is a single-interface tag, which can only communicate through NFC protocol. Tag activates when affected by the magnetic field generated by the NFC reader and acts depending on instructions provided by the reader. The dual interface tags are more complicated since they can be accessed by a wireless RF interface and a wired interface connecting the tag to a microcontroller. This allows the tag to work as a passthrough element for acquiring and delivering information between the microcontroller and the outside world. On top of that, the dual-interface NFC tags can use energy harvesting methods to pass power to the microcontroller from the same magnetic field that powers the NFC tag. [3.]

2.3 Security

Since NFC transfers information through radio waves, it leaves the transfer open for security breaches. Common threats are eavesdropping, where unauthorized personnel listens to classified information transferred, data manipulation, where information transferred is intentionally corrupted by a third party, or interception attack, where unauthorized personnel takes the role of middleman between NFC transfers to acquire authorized information and alter it. The short working range of NFC gives it reasonably good protection against breaches, but with RF receivers, it is still possible for attackers to access transfer up to ten meters away. [5.]

The most efficient way to protect NFC transfers is by establishing an encrypted channel between two devices before transferring information. To ensure that the encryption key used for the transfer can not be eavesdropped, it can not be sent directly between two NFC devices. A common way to prevent this is by creating a shared secret with the ECDH protocol. In this protocol, both devices have their own ECC key pair, consisting of private and public keys. As seen in figure 4, the devices then exchange their public keys with each other and combine their own private key with other devices' public key. This creates the same result on both devices even while their private keys differ from each other, creating a shared secret between devices. [6, p. 6.]

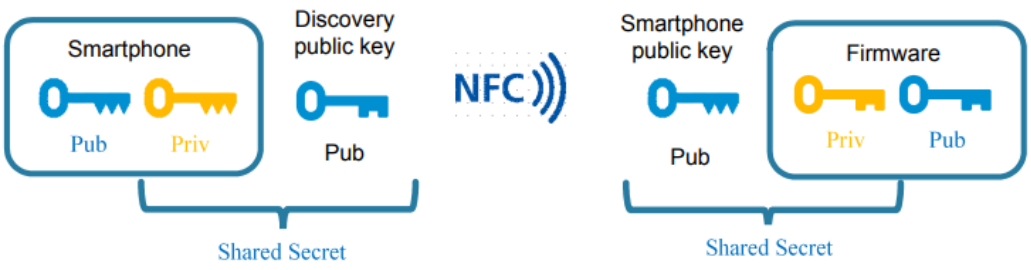

Figure 4. Creation of shared secret between smartphone and STM Discovery devkit. [6, p. 6.]

The use of shared secret still has one flaw: The ECC key pairs of both devices are constant, which causes the shared secret to be always the same. This leaves the communication open for replay attacks, where the attacker records the entire NFC exchange and uses that to copy the shared secret. This can be prevented by adding a randomly generated number to the shared secret, which changes every time devices start to transfer information. This creates an AES Session key which is used to encrypt all the information transferred between two devices. For this to work, one of the connected devices needs to send its Login along with ECC public key when establishing the encrypted channel. Then, when information is to be transferred, this Login can be compared to its original private key and decrypt the message only if those match, as seen in figure 5. Therefore, even if the attacker accomplishes to eavesdrop the Login and the public key, which can not be encrypted, they still will not have access to the private key, preventing them from computing the shared secret or the AES session key. [6, p. 7.]

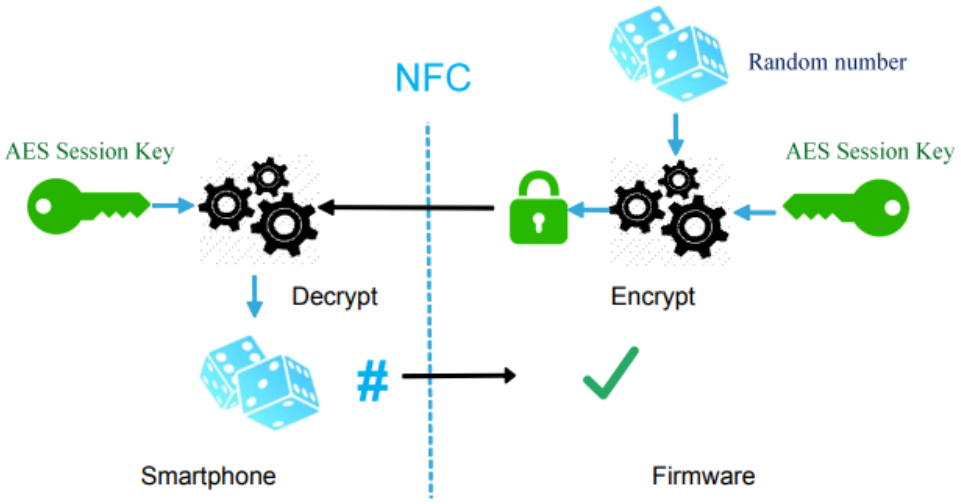

Figure 5. Authentication of devices within the encrypted channel. [6, p. 7.]

2.4 Benefits and weaknesses on NFC based provisioning

From all the variations provided by the NFC protocol, the best solution for provisioning embedded systems is by using read/write mode with a smartphone as an active reader and a dual-interface tag wired to a microcontroller as a

passive element. This has many advantages, such as security provided by limited working distance, easy use where a single tap transfers information, access to system diagnostics written to tag by an embedded system, and acquiring provisioning information from a smartphone. On the other hand, this way of provisioning still has some disadvantages, such as the need for a specifically tailored application on a smartphone to flexibly access tags and the lack of ability to send or acquire information remotely. For these reasons use of NFC as a provisioning method may not work in every situation, but in cases where physical access to devices to be provisioned is not an issue and remote access needs to be limited for better security, this is a viable option.

3 Technology used in the thesis

As part of this thesis, an API was designed to simplify accessing the NFC tag for the system to acquire provisioned information. Minimal hardware requirements for this provisioning process are a microcontroller, which acts as a container for source software, and an NFC tag, which acts as a transmitter platform of provisioned information. For the needs of the ordering company, the API was designed to work directly with Espressif ESP32 microcontroller programmed with C-programming language-based ESP-IDF framework and STMicroelectronics ST25DV NFC tag. However, even if the API is designed for these platforms, the base idea of accessing other tags with other microcontrollers is somewhat similar, so it can be utilized as starting point for other platforms. This chapter contains more specific information on both mentioned components and their features necessary for the provisioning process.

3.1 Espressif ESP32

ESP32 is an SoC-based general-purpose microcontroller provided with a comprehensive selection of peripherals, shown in full in the chips block diagram in figure 6.

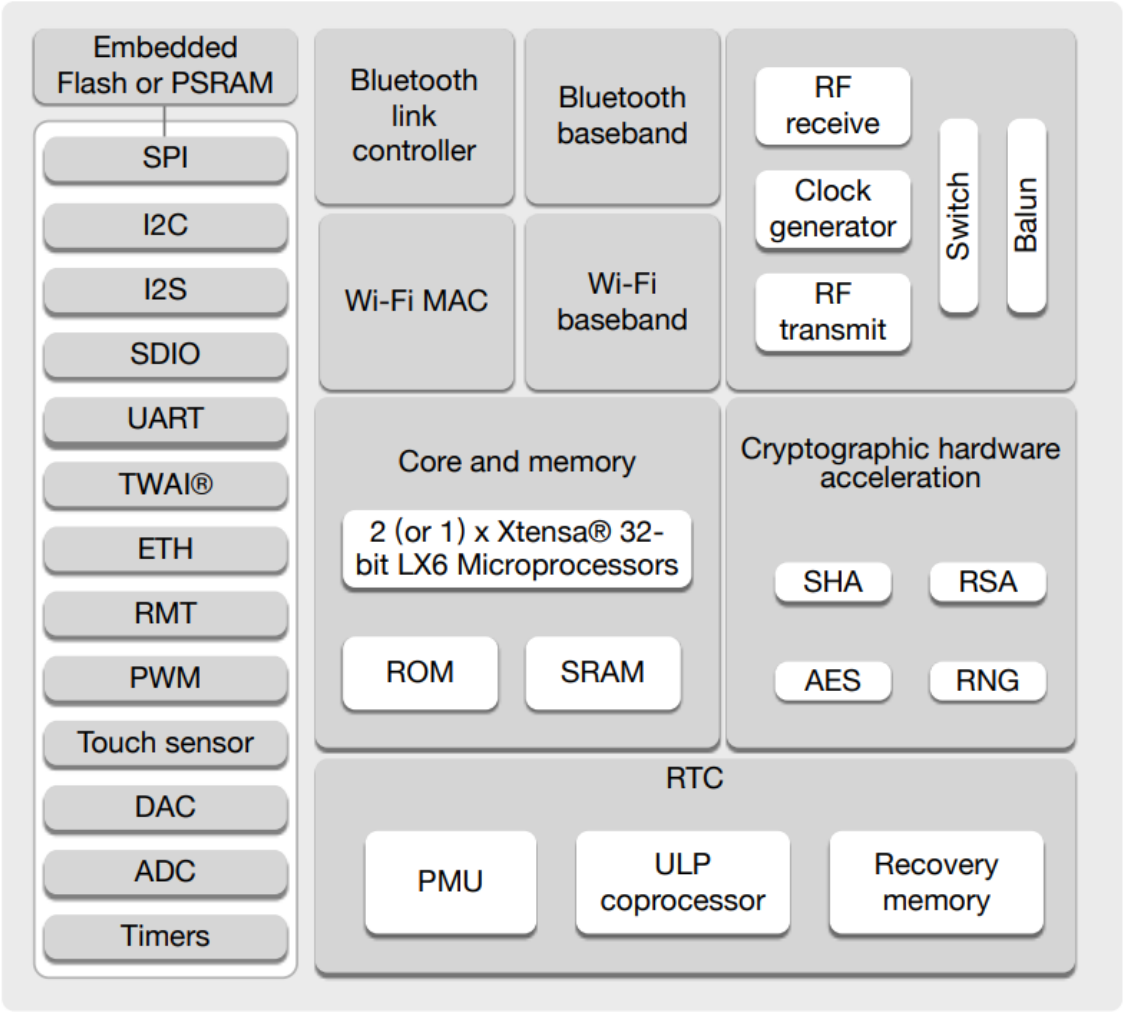

Figure 6. ESP32 functional block diagram. [7, p. 12.]

Chips' most notable advantages against other microcontrollers in a similar price range are its wireless capabilities, including WiFi and BLE protocols. Even though it is possible to design hardware just around the ESP32 SoC, a more common approach is to use one of the pre-made modules provided by Espressif. In addition to the actual SoC, these have external flash memory, crystal, and I-PEX antenna connector or pre-tuned PCB antenna. Furthermore, the module versions of the chip also contain pre-loaded low-level drivers, wireless protocol stacks, and FreeRTOS as the base operating system. [8.]

3.1.1 ESP-IDF

ESP-IDF is Espressif's official framework for ESP32 development. It was created to provide a comprehensive development environment for IoT applications. It includes everything from libraries for accessing controllers' peripherals to installation scripts and build tools. As part of IDF is also a comprehensive documentation library and extensive catalog of example software to guide software development. [9.]

3.1.2 I2C

I2C is a serial communication protocol used to transfer information between the master device, which in this case is the microcontroller, and one or more slave devices, which in this case is the NFC tag. The protocol uses only two lines, SDA and SCL, and the same lines can be used to connect multiple slaves to one master. Each slave device connected to the same I2C bus must have a unique address so the master device can send commands to a specific slave. This slave address is commonly 7-bit long, with 128 unique addresses available. However, if more addresses are needed, the I2C protocol also supports 10-bit slave addresses providing 1024 unique addresses.

I2C handles data in a specific message format, which can be broken into different data frames, as seen in figure 7. Each frame has its purpose. The start condition indicates the beginning of data transfer by switching SDA from high to low before SCL switches from high to low. This is followed by the address frame of the slave device that the master needs to connect. The last bit of the slave address is always used to inform the slave if data is to be retrieved or written to the slave. After that, the message contains one or more 8-bit data frames with the information to be transferred or received. Lastly, the message ends with a stop condition that indicates the end of data transfer and frees the bus for other transfers by switching SDA from low to high after SCL switches from low to high. [10.]

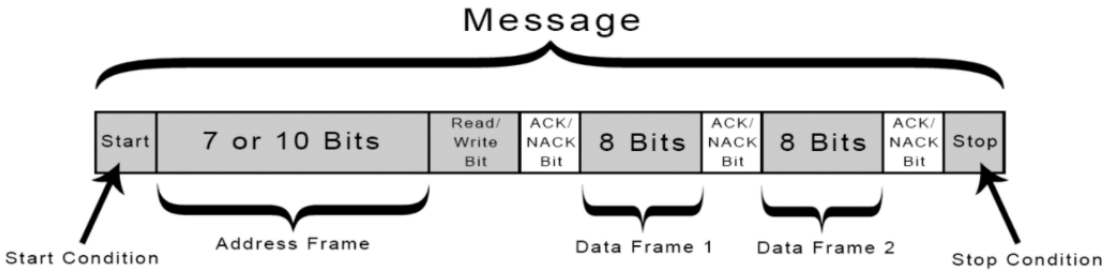

Figure 7. Breakdown of I2C message. [10.]

ESP32 contains two I2C ports that can be used either in master or slave mode, but only the master mode is relevant to the provisioning API. Using these ports is done with a dedicated I2C driver included in ESP-IDF. Before using the driver, it needs to be configured and installed. Driver configuration is done by selecting one of the two available I2C ports and setting up parameters of the i2c_config_t structure. After that, the configuration can be initialized by passing the previously initialized port and structure to the i2c_param_config() function. After the driver is configured, it is ready for installation by calling the i2c_driver_install() function with parameters for the selected port and mode. Detailed configuration structure and installation function are shown in listing 1.

```
i2c_config_t conf = {
    .mode = I2C_MODE_MASTER,
    .sda_io_num = I2C_MASTER_SDA_IO,
    .sda_pullup_en = GPIO_PULLUP_ENABLE,
    .scl_io_num = I2C_MASTER_SCL_IO,
    .scl_pullup_en = GPIO_PULLUP_ENABLE,
    .master.clk_speed = I2C_MASTER_FREQ_HZ,
};
i2c_param_config(I2C_NUM_0, &conf);

i2c_driver_install(I2C_NUM_0, I2C_MODE_MASTER,0,0,0);
```

Listing 1. Configuration and installation example for I2C driver with ESP-IDF

After that, the system is ready for I2C communication. To create I2C messages, ESP-IDF provides a command link, which is a container where each frame necessary for a proper I2C message can be stored before sending it to the slave. [7.] The command link can be used by the I2C master to send data to the slave, as seen in figure 8, or to read data from the slave, as seen in figure 9.

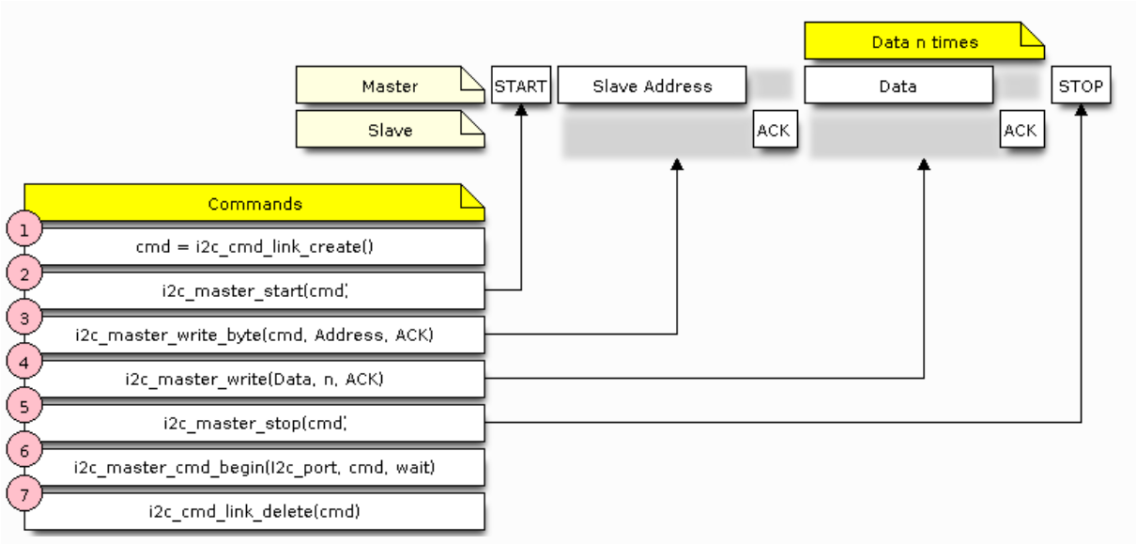

Figure 8. Master Write with command link. [11.]

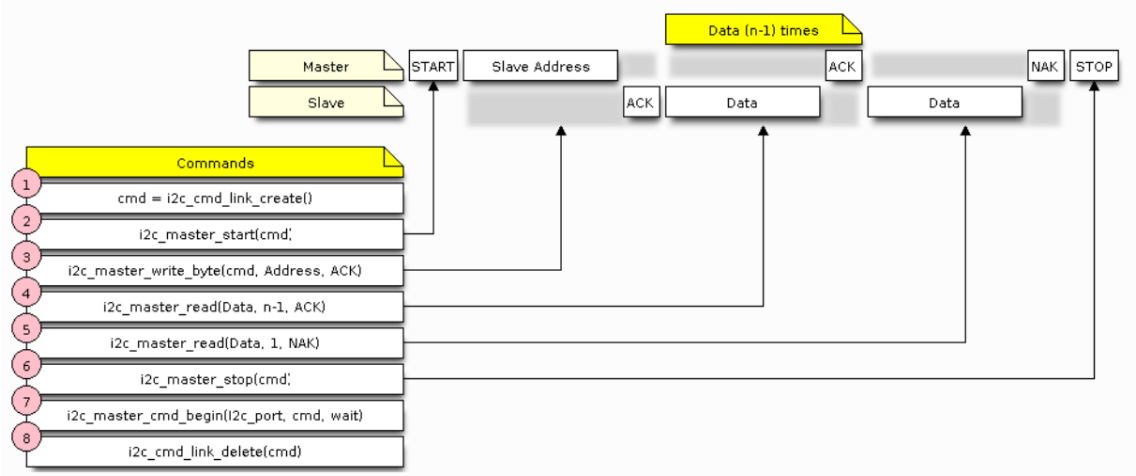

Figure 9. Master Read with command link. [11.]

3.2 STMicroelectronics ST25DV

ST25DV is a dual-interface NFC tag with contactless RF and wired I2C interfaces, 64-bit password protections for both I2C and RF interfaces, EEPROM memory with separated user memory and system registers, and integrated energy harvesting and interrupt capabilities. The tag supports both NFC Type 5 and ISO25693 RFID specifications, and it comes in three different variants by the size of the user memory allocated to EEPROM, which are 4 Kbit, 16 Kbit, and 64 Kbit. The tag also provides a GPO pin for implementing interrupt

pulses and energy harvesting features. Previous specifications are shown in full in the chips block diagram in figure 10.

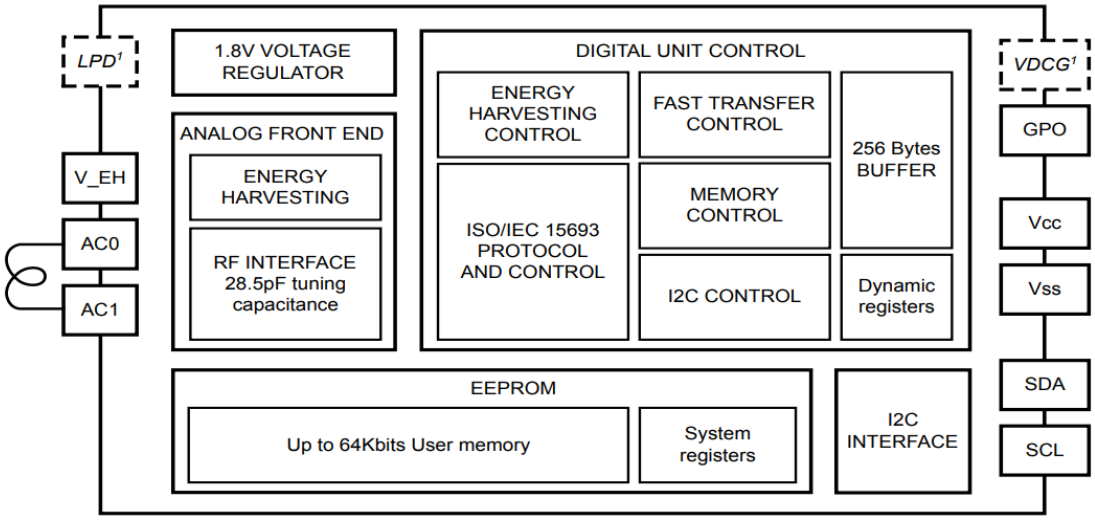

Figure 10. ST25DV block diagram. [12, p. 3.]

This part of the thesis concentrates on aspects of ST25DV that are necessary for using the provisioning API, which is wired access to tags EEPROM memory, writing and reading from the tag, modifying the management of chips memory, and setting up security features for both I2C and RF interfaces. Each aspect is studied from the perspective of a wired interface since that is viable for the provisioning API.

3.2.1 Wired access to EEPROM memory

The wired interface of the ST25DV tag works over the I2C protocol as a slave device, and it contains two physical memory banks with their own slave addresses, which are referenced as device select codes by the manufacturer of the tag. As shown in figure 11, both tag's device select codes start with a 4-bit long static device type identifier which in binary is 1010. This follows with a 3-bit long device select address, which in binary is 'E2'11, where E2 is 1 when accessing tags system memory, or 0 when accessing user memory, dynamic

registers, or fast transfer mailbox. The last bit of device select code is the read/write bit, set to 1 for a read operation and 0 for a write operation.

	Device type identifier (1)				Chip Enable address			RW̄
	b7	b6	b5	b4	b3	b2	b1	b0
Device select code	1	0	1	0	E2(2)	1	1	RW̄

1. *The most significant bit, b7, is sent first.*
2. *E2 is not connected to any external pin. It is however used to address the ST25DVxxx as described in Section 4 Memory management.*
   - *E2 = 0, access to user memory, Dynamic registers or Mailbox.*
   - *E2 =1, access to system area.*

Figure 11. The formula for ST25DV device select code. [12, p. 66.]

For more practical use of device select codes, those can be converted to static hexadecimal values. Since the master sets the last bit, it can be left out of the conversion. Therefore, the device select code for accessing system memory is in binary 1010111, which converts to 0x57 in hexadecimal, and the device select code for accessing other parts of memory is in binary 1010011, which converts to 0x53 in hexadecimal.

3.2.2 Write and read operations

After specifying the accessed memory bank with device select code, the operation is selected by setting up the least significant bit of device select code to 0 for writing and 1 for reading. Following that for both operations is the selection of memory addresses to access. Each accessible data byte in memory has a 16-bit address, and for accessing that, the address needs to be sent to the tag in two 8-bit long frames starting from the most significant byte.

In the case of writing data to the tag's memory, the master device can start sending bytes to be written right after the address is selected. The tag can receive up to 256 sequential bytes in one command, and after transferring each byte, the tag's internal address counter increments automatically. The tag detects the end of the write operation by receiving the stop condition from the

master device. A detailed representation of the write operation is shown in figure 12.

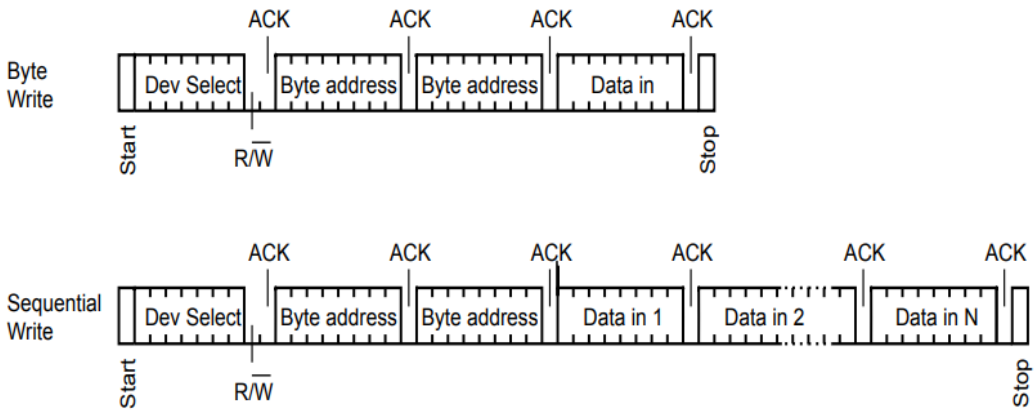

Figure 12. Writing sequence for ST25DV. [12, p. 68.]

There are two different methods when reading from the tag: current address read and random address read. The difference between these is that the current address read accesses the data where the tags internal address counter was previously left to, while random address read starts with a dummy write to a specific address to set the address counter to a specific memory register before the read operation. To execute the current address read, the master needs to send one or more data out commands right after the device selects code following the stop condition without acknowledging the byte. However, executing the random access read is more complicated since the internal address counter needs to be loaded to the requested address before being read. This is implemented by sending the address to be read right after the device select code. After that master needs to send another start condition without stop condition followed by the device select code. Then the data out commands can be implemented similarly with the current address read. A detailed representation of reading sequences can be seen in figure 13.

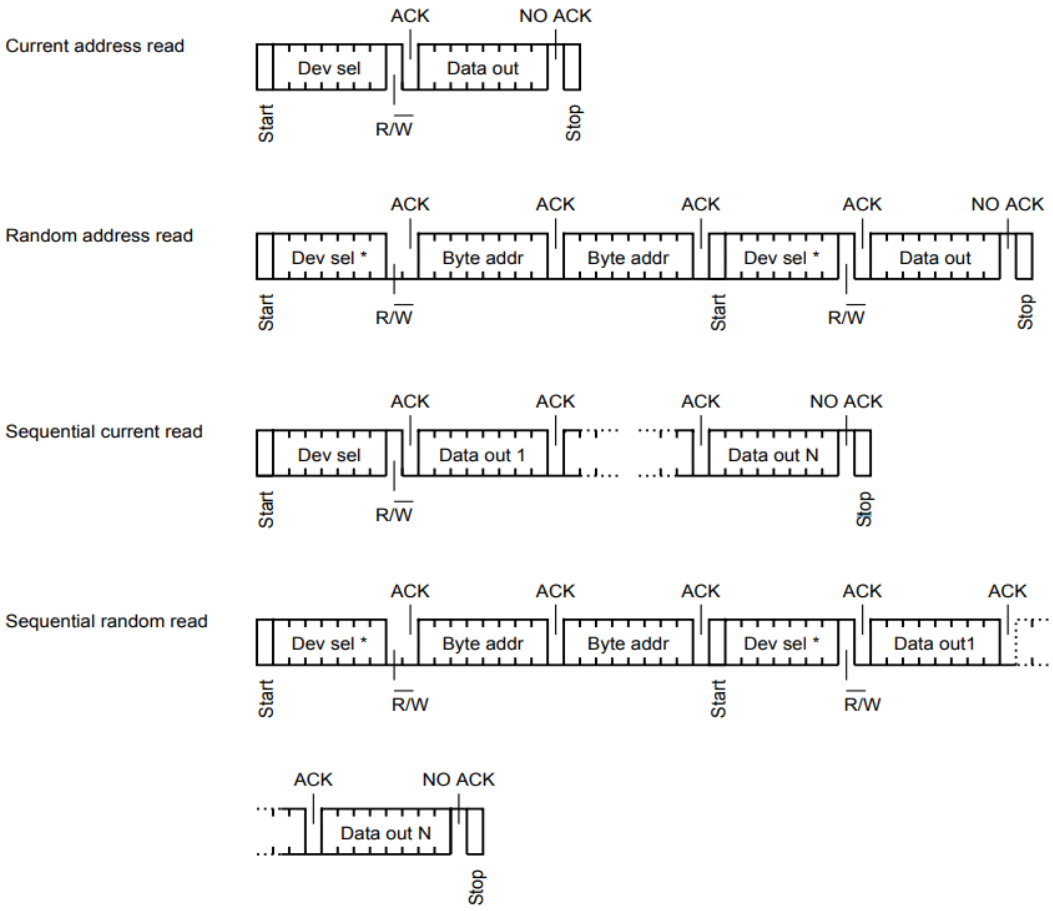

Figure 13. Reading sequences for ST25DV. [12, p. 71.]

3.2.3 Memory management

The tag's user memory can be separated up to four areas. Each area can have different access privileges, and sequential read or write operations are automatically blocked if the end of area is reached. Each custom-sized user area is defined by the last address accessible from a given area, which can be configured by changing ENDA1, ENDA2, and ENDA3 registers located in tags system memory. Each of these register values represents an address of user memory that separates two areas from each other. These register values must be used in the given order, and the value can't exceed the maximum size of tags user memory, as shown in figure 14. Reducing the number of memory areas is implemented by setting values of one or more area end addresses to the last available address of user memory. However, since each area is contiguous to each other, the process of decreasing memory areas must start

from the last register. By default, each end address is set to the last accessible byte, which results in the existence of only one area that includes the entire user memory.

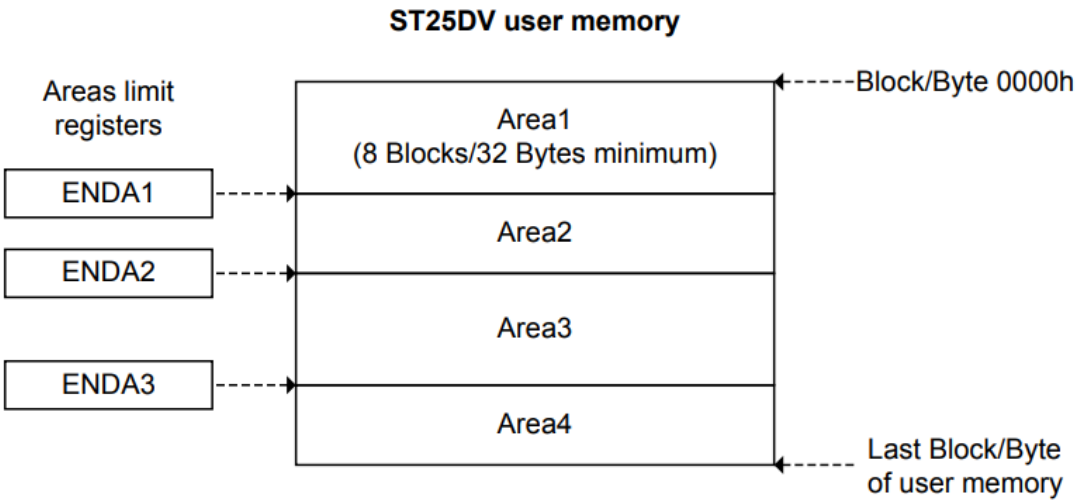

Figure 14. Representation of tags user memory management. [12, p. 13.]

Since the range of 16-bit user memory exceeds the 8-bit size registers of system memory, some conversion is needed. The resolution between user memory and system registers is 32 bytes, which means that when the value of system registers increases by one digit, the value of user memory increases by 32. Therefore, the ENDA1, ENDA2, and ENDA3 register values can be calculated by dividing the selected user memory address by 32.

3.2.4 GPO interrupt events

When accessing information stored in the tag's memory by the microcontroller, it needs to be notified when new information is passed to the tag. The tag has a dedicated GPO to send interrupt signals to the microcontroller. The tag can be configured to send interrupt in multiple situations, such as from dedicated command from the RF interface, when an RF field is detected to appear or disappear, and automatically after the end of the write operation from the RF interface.

Selecting GPO mode can be done by accessing tags system memory on address 0x0000. This register takes in an 8-bit value, where the least significant bit enables or disables the use of the GPO pin, and the rest are used to select which mode is to be used, as seen in figure 15.

Bit	Name	Function	Factory Value
b0	RF_USER_EN	0: disabled 1: GPO output level is controlled by Manage GPO Command (set/reset)	0b
b1	RF_ACTIVITY_EN	0: disabled 1: GPO output level changes from RF command EOF to response EOF.	0b
b2	RF_INTERRUPT_EN	0: disabled 1: GPO output level is controlled by Manage GPO Command (pulse).	0b
b3	FIELD_CHANGE_EN	0: disabled 1: A pulse is emitted on GPO, when RF field appears or disappears.	1b
b4	RF_PUT_MSG_EN	0: disabled 1: A pulse is emitted on GPO at completion of valid RF Write Message command.	0b
b5	RF_GET_MSG_EN	0: disabled 1: A pulse is emitted on GPO at completion of valid RF Read Message command if end of message has been reached.	0b
b6	RF_WRITE_EN	0: disabled 1: A pulse is emitted on GPO at completion of valid RF write operation in EEPROM.	0b
b7	GPO_EN	0: GPO output is disabled. GPO is High-Z (open drain) or 0 (CMOS) 1: GPO output is enabled. GPO outputs enabled interrupts.	1b

Figure 15. Protocol for formatting 8-bit value for GPO mode. [12, p. 36.]

3.2.5 Security

ST25DV provides extensive capabilities for data protection built upon one 64-bit password for wired I2C connection and up to 3 64-bit passwords for wireless RF connection. Without presenting the password, the static system registers can't be accessed. Furthermore, each area of user memory can be protected with separate rules for read and write privileges by the I2C password or any of the three RF passwords. The only exception is the first area of user memory, which must always be readable by both I2C and RF connections.

The protection level of I2C is controlled over one 8-bit register located in the tags system memory section in address 0x000B, where every two bits contain a separate partition of user memory in the manner shown in figure 16.

Bit	Name	Function	Factory Value
b1-b0	RW_PROTECTION_A1	00: Area 1 I^2C access: Read always allowed, Write always allowed 01: Area 1 I^2C access: Read always allowed, Write allowed if I^2C user security session is open 10: Area 1 I^2C access: Read always allowed, Write always allowed 11: Area 1 I^2C access: Read always allowed, Write allowed if I^2C user security session is open	00b
b3-b2	RW_PROTECTION_A2	00: Area 2 I^2C access: Read always allowed, Write always allowed 01: Area 2 I^2C access: Read always allowed, Write allowed if I^2C user security session is open 10: Area 2 I^2C access: Read allowed if I^2C user security session is open, Write always allowed 11: Area 2 I^2C access: Read allowed if I^2C security session is open, Write allowed if I^2C security session is open	00b
b5-b4	RW_PROTECTION_A3	00: Area 3 I^2C access: Read always allowed, Write always allowed 01: Area 3 I^2C access: Read always allowed, Write allowed if I^2C user security session is open 10: Area 3 I^2C access: Read allowed if I^2C user security session is open, Write always allowed 11: Area 3 I^2C access: Read allowed if I^2C security session is open, Write allowed if I^2C security session is open	00b
b7-b6	RW_PROTECTION_A4	00: Area 4 I^2C access: Read always allowed, Write always allowed 01: Area 4 I^2C access: Read always allowed, Write allowed if I^2C user security session is open 10: Area 4 I^2C access: Read allowed if I^2C user security session is open, Write always allowed 11: Area 4 I^2C access: Read allowed if I^2C security session is open, Write allowed if I^2C security session is open	00b

Figure 16. Protocol for formatting 8-bit value for I2C security. [12, p. 51.]

On the other hand, since the RF side can be protected with three separate passwords, its security is split into 4-bit registers for each of four user memory areas. These registers are in tags system memory and are located in address 0x0004 for the first user memory area, 0x0006 for the second area, 0x0008 for the third area, and 0x0010 for the fourth and last area. Contents for each register are similar to each other and follow the way shown in figure 17.

Bit	Name	Function	Factory Value
b1-b0	PWD_CTRL_A1	00: Area 1 RF user security session can't be open by password 01: Area 1 RF user security session is open by RF_PWD_1 10: Area 1 RF user security session is open by RF_PWD_2 11: Area 1 RF user security session is open by RF_PWD_3	00b
b3-b2	RW_PROTECTION_A1	00: Area 1 RF access: Read always allowed / Write always allowed 01: Area 1 RF access: Read always allowed, Write allowed if RF user security session is open 10: Area 1 RF access: Read always allowed, Write allowed if RF user security session is open 11: Area 1 RF access: Read always allowed, Write always forbidden	00b
b7-b4	RFU	-	0000b

Figure 17. Protocol for formatting 4-bit value for RF security. [12, p. 48.]

Both I2C and RF can update the protection conditions for both connection interfaces, but the passwords for each interface can only be changed with the given interface. Therefor this section will only concentrate on presenting and changing the password of the I2C interface. Gaining access to protected information by the I2C interface is allowed by sending specific present password command from the master device. After the start condition, the master sends the device select code 0x57 dedicated for tags system memory with the least significant bit set as write bit. This follows by tags I2C password address bytes 0x09 and 0x00. After this, the 64-bit password, separated into eight 8-bit chunks, is sent twice to the tag separated with validation code 0x09. The reason for sending the password two times is to prevent data corruption during transfer. Finally, after acquiring the stop condition, the tag compares the presented password to a password saved in system memory and grants access if they match. After presenting the password, all tag registers are available until the session is closed by presenting an incorrect password by the same present password command. The whole sequence for presenting passwords can be seen in figure 18.

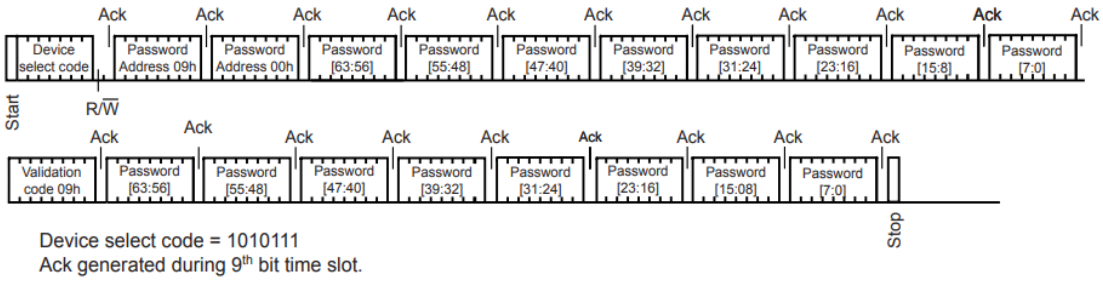

Figure 18. Sequence for presenting password over I2C. [12, p. 72.]

Writing a new password for the tag requires the security session to be already opened by the present password command. Once the security session is open, the password can be changed by almost identical commands to the present password command, the only difference being that the validation code is changed to be 0x07. The whole sequence for writing passwords can be seen in figure 19.

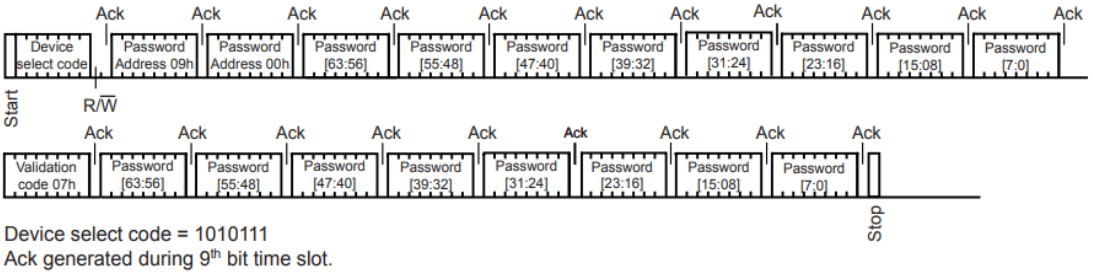

Figure 19. Sequence for writing password over I2C. [12, p. 73.]

Even with the multiple passwords used to access secure information, the system still has some flaws, including the lack of data encryption while presenting passwords, which may be acquired through eavesdropping attacks. Since this tag does not have an internal crypto chip and if the short working range of NFC does not construct a secure enough environment, the methods declared in chapter 2.3 can be implemented to encrypt the exchange of information.

4 Documentation of the provisioning API

For seamless use of the ST25DV tag as a provisioning interface for the ESP32 microcontroller, an API was created. The API contains helpful enumerations and definitions for most common registers and values, all necessary functions for writing and reading tags over I2C, presenting and writing passwords, configuring tags system registers, and parsing incoming and outcoming data needed for the provisioning process.

4.1 Read functions

The first selection of functions is read functions, which are used to acquire data from the tag. The first of these functions is the st25dv_readByte function, which can read a single character from the tag's EEPROM memory and store it in a character variable. This function takes in the device select code, and register address to determine where data to be read is located within the tag's memory and a pointer to the variable where acquired information is to be stored.

Along with a function for reading a single byte from the tag, the API also contains two different ways to read multiple characters from a tag with a single function. The first of these is st25dv_readSequence, which can be used to read a selected number of characters from the tag. This is useful when only part of the data stored in the selected user area needs to be accessed and the length of data to be retrieved is known. This function takes in four parameters: device select code, register address, a pointer to character array where data is to be stored, and the number of bytes to read. Since the tag's internal address counter didn't work reliably during sequential read, the sequential read was implemented by manually storing data retrieved with st25dv_readByte function to buffer array with manually incremented register address, which was run in loop for the length of characters to be read.

The other way to read multiple characters from the tag is with the st25dv_readUntilNull function, which can read data from the tag until NULL character 0x00 is reached. This is useful when the length of data to be read is

unknown, and the data written in the selected user memory area can be read in its entirety. This is the preferred method for reading the tag's information since, in most use cases, the length of data can vary, and data written in the tag can be separated into four user areas. Using this function is otherwise similar to the st25dv_readSequence function, except the number of characters to read doesn't need to be passed to the function since the looping mechanism looks for NULL character 0x00 instead of cycles determined by data length.

4.2 Write functions

The second selection of functions implemented within API is used to write data to tag from the microcontroller. The simplest one is the st25dv_writeByte function, which writes a single character to tag EEPROM memory. The function takes in three values as parameters: the device select code, register address to be written, and the character to be written.

Another write function included in API is st25dv_writeSequence, which can be used to write a sequence of characters to tag. Like the st25dv_writeByte function, this function takes in the device select code and register address. However, instead of a single character, it takes in a character array containing text to be written in the tag along with the number of characters to be written.

The third function under the category of write functions is st25dv_eraseRange, which is used clear a given area from tags user memory by writing NULL character 0x00 to selected register addresses. Since this function targets tags user memory, it doesn't need to take in device select code as a parameter. However, the function still takes in the register address for the first byte to be erased and the number of bytes to be erased.

To properly update content on the tag, the API also contains the function st25dv_setContent, which can easily write content to the tag's user memory while ensuring that any previous content on the given memory partition is erased. This function takes in two parameters: the memory address where writing is to be started and the pointer to the content to be written on the tag.

The function begins to read the current content starting from the given address and ending either in NULL character 0x00 or the end of the partition. Then it compares the current content with the new content, and if they are the same, the operation is immediately terminated. Otherwise, the function compares current and new content lengths and erases the excessive characters if the current content is longer than the new content. After this, the function writes the new content to the tag.

4.3 Password functions

The third selection of functions contains tools for presenting passwords to open or close I2C security sessions and for setting a new password. Presenting the password is done with the st25dv_presentPassword function, which takes in eight 8-bit values stored in an array, forming one 64-bit password. Other parameters are not required since the device select code and the register address are always fixed values. This function has two purposes: opening a security session by presenting the same combination of bytes as one stored in the tag's system memory or closing the security session by presenting any other combination of bytes compared to bytes stored in the tag. To indicate which operation was implemented, the function prints to ESP32 information log message, which tells if the security session is open or closed right after the password was presented.

Another function in the password category is st25dv_setPassword, which is used to change an existing password, and it takes in the new password in a similar format as the function for presenting a password. Before this function can be used, the tag's security session needs to be opened by the present password function. After the security session is open, the function reads the current password from the tag and compares that to the password to be set. If passwords are similar, the operation is terminated. Otherwise, the new password will be written to the tag.

4.4 Configuration functions

The final selection of functions is used to configure the tag's initial settings. The first of the initial configuration functions are st25dv_setPartitions, which is used to set the end addresses for each user memory area. The function takes in three parameters, which are the system register values for each division point between partitions. Suppose any of the parameters are set to the maximum acceptable value, which is the system register value for the last address on user memory. The division point is placed at the end of the memory area, thus decreasing the number of partitions. The function checks if given parameters exceed the last address on user memory and adjusts each parameter to the last acceptable value. Then, to prevent data corruption, the function sets end address 2 and end address 3 to the last acceptable address of user memory if they already aren't set on that. After that, each of the new end addresses are stored in the tags system memory. Since end areas must always be presented in a fixed order, the function writes the second and the third end addresses only if their registers are set to be greater than the previous end address.

The second function for tag initials configuration is st25dv_setAreaProtection, which sets the level of protection to the given partition of user memory. This function takes in two parameters: an index of the user memory area to change and code containing the number of the password to be used, and code for a specific combination of read and write permissions shown in figure 12. To use this function, the security session needs to be opened by the present password function. The function first checks if the given index is viable and then reads the current code for security level. After that, the current security code is compared to the new security code, and if they are different, the new value is updated to the tag.

The last configuration function is st25dv_setGpoMode, which is used to disable or enable interrupt outputs on the selected mode. This function takes in only one parameter, which is an enumerated value for setting GPO mode. The function first reads the current GPO mode from the tag and then compares that to the passed value and updates the new value if they are different.

5 Provisioning software in practice

A few steps need to be taken to use the given API as part of more complex software. Firstly, the I2C bus must be configured with the GPIO pins connected to tags SDA and SCL lines. When the bus is configured, the tag can be initialized by dividing user memory into a proper number of partitions, giving each partition appropriate permissions for read and write operations from the RF interface, and finally writing initial content for each partition. After this, the provisioning software is ready to be used. A flowchart for simple example software where timers are used to periodically check one partition of tag for updates over RF and write system diagnostics to another partition in figure 20.

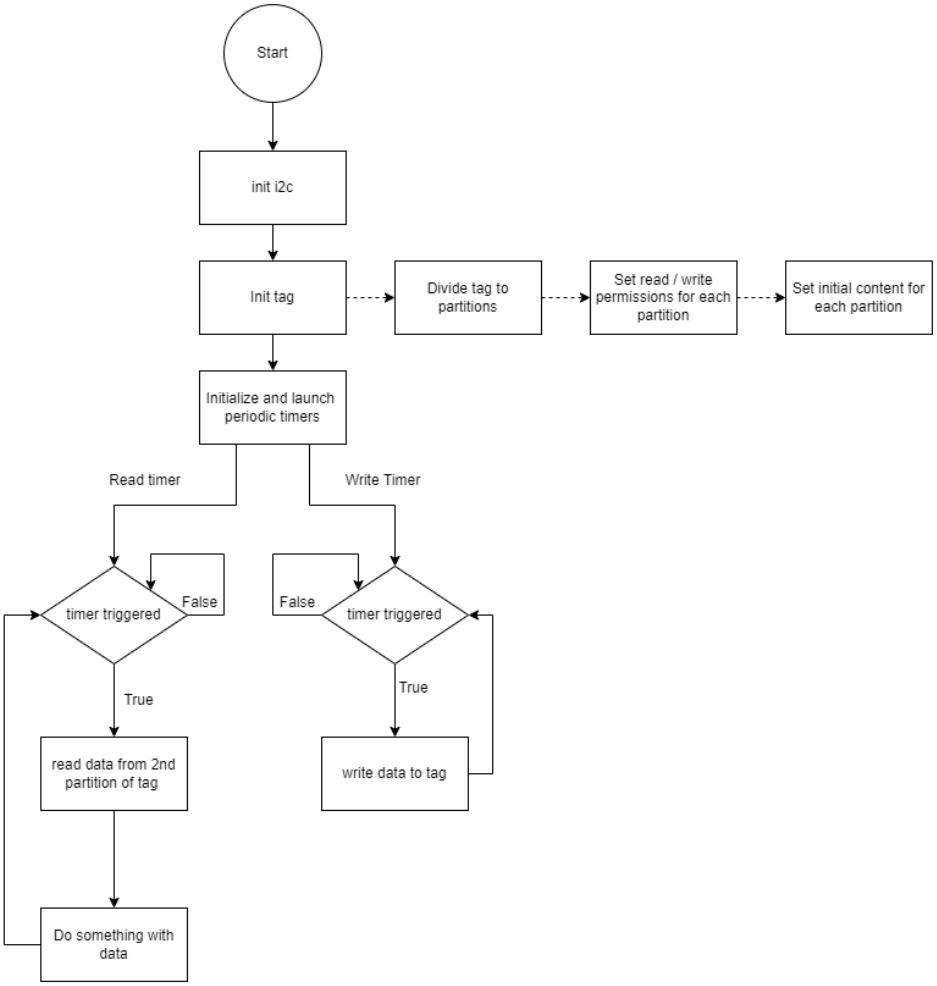

Figure 20. Process for provisioning.

5.1 Demonstrational software

A simple software was implemented to demonstrate the practical use of the provisioning API. The software will use a 4Kbit variation of the ST25DV tag to transfer information between the ESP32 microcontroller and an Android smartphone. The ST25 NFC Tap application provided an RF connection to the tag in this demonstration. In this demonstration, the tag is divided into four partitions. Each of these partitions is set to an equal size, which gives 128 bytes for each partition. This can be implemented by setting partition end addresses to 0x0080, 0x100 and 0x180, which converted to register values in system memory are 0x03, 0x07 and 0x0B. After the tag is divided into proper partitions, each partition can be set for initial content and proper read and write privileges. To do this, the I2C security session needs to be opened by calling the st25dv_presentPassword function.

Since the read privileges of the first memory partition can't be restricted, it is best suited for non-discrete information such as device model name. This information should not be changed after the initial configuration, so write privileges from RF are closed. Sequence for setting initial content and these privileges for the first partition are shown in listing 2.

```
esp_err_t ret;
char content[40] = "Demo model name";

//Set 1st partition RF write always forbidden
ret = st25dv_setAreaProtection(1,0x03);
if(ret!=ESP_OK) ESP_LOGE(TAG, "Failed to set protection to 1st area");

//Set 1st partition content to devices generic model name
ret = st25dv_setContent(0x0000, content);
if(ret!=ESP_OK) ESP_LOGE(TAG, "Failed to set content of 1st area");
```

Listing 2. source for the first partition of user memory.

The second partition of the tag contains the information that needs to be provisioned to the device. Since the proper information is provided later by an RF interface, the initial content can be set to some placeholder value. The partition password is configured for both read and write privileges to ensure that only authorized devices can change or see the provisioned information. To prevent continuous polling for this information, tags GPO interrupts are enabled

in writing mode. The sequence for initial content and security privileges for this partition is shown in listing 3.

```
esp_err_t ret;
char content[40] = "ssid=WifiName;pwd=WifiPassword";

//Set 2nd partition RF write and read protected with password
ret = st25dv_setAreaProtection(2,0x06);
if(ret!=ESP_OK) ESP_LOGE(TAG, "Failed to set protection to 2nd area");

//Set 2nd partition content to placeholder data for information to be
provisioned
ret = st25dv_setContent(0x0080, content);
if(ret!=ESP_OK) ESP_LOGE(TAG, "Failed to set content of 2nd area");

//Enable GPO in writing mode
st25dv_setGpoMode(RF_WRITE_ENABLE) //enumeration for 0xC0
```

Listing 3. Source code for the second partition of user memory.

The third partition of the tag contains system diagnostics written from the device for easy access over the RF interface. As initial content, the starting values of selected diagnostics are written and updated later. Since these values are only to be updated through a wired connection, RF write can be disabled, and if the diagnostics are not considered public, information can be read and secured with a password. The sequence for initial content and security privileges for this partition is shown in listing 4.

```
esp_err_t ret;
int temp = getTemp(); //Mockup for retrieving temperature
char uptime[10] = parseUptime(); //Mockup for parsing system uptime
char content[40] = ""; //Placeholder for string containing system diagnostics
sprintf(content, “uptime=%s;temp=%d”, uptime, temp);

//Set 3rd partition RF write forbidden and read protected with password
ret = st25dv_setAreaProtection(3,0x07);
if(ret!=ESP_OK) ESP_LOGE(TAG, "Failed to set protection to 3rd area");

//Set 3rd partition content to placeholder data for system diagnostics
ret = st25dv_setContent(0x0100, content);
if(ret!=ESP_OK) ESP_LOGE(TAG, "Failed to set content of 3rd area");
```

Listing 4. Source code for the third partition of user memory.

The fourth and last partition in this demonstration is left empty for a possibility of the further need for storage during further development of the device. Even if the partition is currently not used, it should be password protected on write privileges to prevent unauthorized devices from taking advantage of the empty

partition. The sequence for initial content and security privileges for this partition is shown in listing 5.

```
esp_err_t ret;

//Set 4th partition RF write and read protected with password
ret = st25dv_setAreaProtection(4,0x06);
if(ret!=ESP_OK) ESP_LOGE(TAG, "Failed to set protection to 4th area");
```

Listing 5. Source code for the fourth partition of user memory

After initial configurations, the security session can be closed, and the tag can be left to work in the background. The first of two background operations is the tag updating system diagnostics periodically. A periodic timer integrated into ESP-IDF is launched to trigger a callback on reasonable periods. During callback, the content on the third partition of the tag is replaced with a string containing the updated diagnostics. Timer initialization and callback are shown in listing 6.

```
const esp_timer_create_args_t periodic_timer_args =
{
.callback = &periodic_timer_callback,
.name = "periodic"
};

esp_timer_handle_t periodic_timer;
ESP_ERROR_CHECK(esp_timer_create(&periodic_timer_args, &periodic_timer));
ESP_ERROR_CHECK(esp_timer_start_periodic(periodic_timer, 30 * 1000000));

static void periodic_timer_callback(void* arg)
{
    char content[40];
    // Retrieve diagnostics and store them
    sprinf(content, “uptime=%s;temp=%d”, parseUptime(), getTemp());
    esp_err_t ret = st25dv_setContent(0x0101, content);
     if(ret!=ESP_OK) ESP_LOGE(TAG, "Failed to set content of 3rd area");
}
```

Listing 6. Source code for periodic timer for storing diagnostics to tag

The second of the operations running in the background is for updating the provisional information. Since the only information written to tag over RF is the provisional information, this can be implemented with an interrupt triggered by a successful RF write operation. When an interrupt is triggered, it informs the provisioning task by setting up a binary semaphore. When the task detects the changed state of the semaphore, it reads the tags updated information and

uses it to change system parameters. Source code for configuring the interrupt and triggering it in a task can be seen in listing 7.

```
static void isr_handler(void* arg)
{
    xSemaphoreGive(binary_sem);

    if( xHigherPriorityTaskWoken )
     {
          /* Actual macro used here is port specific. */
          portYIELD_FROM_ISR();
     }
}


void provisioningTask(void *arg)
{
    SemaphoreHandle_t binary_sem;
    char buf[254] = "";

    init_intr(); //mockup for setting up interrupt service

    while (1){

     if(xSemaphoreTake(binary_sem, portMAX_DELAY) == pdTRUE)
        {
            //Read information from the second partition of user memory
            st25dv_readUntillNull(0x53,0x0080,&buf);

            //Do something with the retrieved data
          }
    }
}
```

Listing 7. Source code for configuring interrupt to inform task when the tag is updated.

6 Conclusion

In this thesis, NFC technology was introduced and presented as a provisioning mechanism for embedded systems. Design and working principle for both NFC tags and readers was shown, and common ways to enhance the security of NFC information transfer were presented. Also, the advantages and disadvantages of NFC as a provisioning method were discussed. The theory was taken to practice with the creation and documentation of the API library, which is used to provision Espressif ESP32 microcontroller with STMicroelectronics ST25DV dual-interface NFC tag. This was underlaid with a detailed description of those components and their key features. The proof-of-concept for provisioning embedded systems with dual-interface NFC tag was

conducted for Sulaon Oy as part of their software development for embedded systems.

Provisioning embedded systems with NFC tags has many benefits, and it works splendidly in situations where the device to be provisioned is easily accessed. The configurations can be implemented with a smartphone by simply moving it near the device that needs to be configured. Securing the RF transfers with passwords and storing confidential information on specific register addresses on the tag combined with a short working distance of NFC gives the device good protection.

The API was created for Sulaon Oy to be implemented as part of their embedded system development. The company has reviewed the work, and it has been approved for use, and the API has already been implemented as part of the products the company is creating. In the further development of the project, porting the API for other microcontrollers and programming languages is being considered to extend the use of NFC provisioning on more products designed by the company.

Overall, the results of this thesis were successful. The dual-interface NFC tags were proven to be beneficial for embedded systems provisioning. Even with the few disadvantages, it can be recommended to be used in situations where end-users need to quickly set up device configurations or monitor system diagnostics. The tag can be accessed by simply placing the smartphone with a dedicated provisioning application near the embedded device, ensuring maximum ease of use even for end users with minimal technical knowledge.

References

1 About the technology. 2022. Online Material. NFC Forum. <https://nfc-forum.org/what-is-nfc/about-the-technology>. Accessed: 8.4.2022.

2 What is an NFC chip? 2022. Online Material. STMicroelectronics. <https://www.st.com/content/st_com/en/support/learning/essentials-and-insights/connectivity/nfc/nfc-chips.html>. Accessed: 8.4.2022.

3 NFC Tag vs NFC reader-Difference between NFC tag and NFC reader. 2012. Online Material. RF Wireless World. < https://www.rfwireless-world.com/Terminology/NFC-tag-vs-NFC-reader.html>. Accessed: 16.4.2022.

4 Dardé, Laurent. 2014. Start a Conversation with NFC: Three Operation Modes. Online material. <https://www.nxp.com/company/blog/start-a-conversation-with-nfc-three-operating-modes:BL-START-CONVERSATION-WITH-NFC>. 24.11.2014. Accessed: 8.4.2022.

5 NFC Security. Online Material. Electronics Notes. <https://www.electronics-notes.com/articles/connectivity/nfc-near-field-communication/security.php> Accessed 18.4.2022.

6 ST25DV-I2C cryptographic demonstration. 2020. Online Material. STMicroelectronics. <https://www.st.com/resource/en/user_manual/um2684-st25dvi2c-crypto-demonstration-stmicroelectronics.pdf> Accessed 18.4.2022.

7 ESP32 Series Datasheet. 2022. Online Material. Espressif Systems. <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Accessed: 19.3.2022

8 Litingtun, Shawn. 2021. Introduction to the ESP32 WiFi/Bluetooth Wireless Microcontroller. Online material. <https://predictabledesigns.com/introduction-to-the-esp32-wifi-bluetooth-wireless-microcontroller>. 23.6.2021. Accessed: 19.3.2022.

9 Minatel, Pedro. 2021. ESP-IDF Development Tools Guide — Part I. Blog post. <https://blog.espressif.com/esp-idf-development-tools-guide-part-i-89af441585b>. 9.6.2021. Accessed: 20.3.2022.

10 Campbell, Scott. 2016. Basics of the I2C communication protocol. Online material. < https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol>. Accessed: 20.3.2022.

11 Inter-Integrated Circuit (I2C). ESP-IDF Programming Guide. Espressif Systems. <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2c.html>. Accessed 20.3.2022.

12 ST25DVxxx datasheet. 2021. Datasheet. STMicroelectronics. <https://www.st.com/resource/en/datasheet/st25dv04k.pdf>. 9.2.2021. Accessed: 20.3.2022.