FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Application of NDEF messages in offline maintenance operations

Luís André Santos Correia Assunção

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. António Miguel Pimenta Monteiro

July 27, 2023

Application of NDEF messages in offline maintenance operations

Luís André Santos Correia Assunção

Mestrado em Engenharia Informática e Computação

July 27, 2023

Abstract

Infraspeak is a Portuguese company based in Porto that develops software to enhance maintenance operations. Their solution uses Near-Field Communication (NFC) to allow maintenance technicians to access information about the device that requires maintenance by reading an NFC tag attached to it. At the moment, such a tag contains only an identifier that allows information to be queried from a server, which is then displayed to the technician. However, this solution does not allow workers in locations with unreliable or inexistent internet connectivity to benefit from the solution.

This dissertation aims to improve Infraspeak's software, shifting the paradigm from a server-based solution to an offline-based solution, which still requires an eventual server connection. By storing the asset information directly on the NFC tag, it ways possible to create an offline information system. This modification hardens the system against power outages, network problems, server downtime, and many other possible problems, boosting its reliability and rendering it suitable for more widespread use cases. To store the information on the NFC tags, NFC Data Exchange Format (NDEF)[3] was used. In order to protect this information from unauthorised readings, cryptographical solutions were also investigated and implemented, considering the available size in the tag and desired secrecy level[16]. Related use cases to help the maintainers were also researched.

Creating this offline information system required an investigation of the functioning of NFC tags to select suitable tags to be used for the operations, as well as selecting critical information to be stored in the minimal space of NFC tags. At the same time, it requires implementing the solution in a novel Kotlin Multiplatform [34] environment for iOS and Android.

Acknowledgments

I want the thank the following people and institutions for the support provided during this project:

To Infraspeak for providing me with the opportunity to develop this project in their organization. I couldn't imagine a better internship environment.

To Artur Costa, my manager, for all the guidance, feedback and enthusiasm for the project.

To Professor António Miguel Pimenta Monteiro, for reviewing this document.

To all my friends for the good times spent during breaks and for sharing all highs and lows with me.

To my girlfriend, Carolina, for the amazing support in this journey. Your smile was always an inspiration and fuel to my motivation.

To my parents and my brother, for everything.

André Assunção

"Live as if you were to die tomorrow.
Learn as if you were to live forever."

Mahatma Gandhi

Contents

List of Figures

List of Tables

Listings

Abbreviations ans Symbols

AES	Advanced Encryption Standard
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CF	Chunk Flag
FEUP	Faculdade de Engenharia da Universidade do Porto (Faculty of Engineering of the University of Porto)
GNU	GNU's not Unix
HMAC	Hash-based Message Authentication Code
HTML	HyperText Markup Language
ID	IDentifier
IL	ID Length
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KMM	Kotlin Multiplatform Mobile
LZW	Lemper-Ziv-Welch
MAC	Message Authentication Code
MB	Message Begin
ME	Message End
MVP	Minimum Viable Product
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
OS	Operative System
PACK	Password ACKnowledgement
PoC	Proof of Concept
QR	Quick Response
RFID	Radio Frequency IDentification
SR	Short Record
TNF	Type Name Format
UI	User Interface
UID	Unique IDentifier
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
UUID	Universally Unique IDentidier
XOR	Exclusive OR

Chapter 1

Introduction

1.1 Background and Problem statement

Infraspeak is a company based in Porto, established in 2015. Since then, Infraspeak has been developing an intelligent maintenance operation software. An important piece of that software is Infraspeak Next, a mobile application that allows the user, typically a maintenance technician, to scan NFC tags attached to an asset, which can be either an equipment or a location, that requires maintenance and access all the information about that asset on their mobile device. Another use of the NFC tags is to allow the technicians to execute corrective and predictive maintenance procedures, access the information about the work orders to be executed and tasks related to that procedure and even prevent the user from completing the procedure before safety measures are implemented. All of this is incredibly efficient because, with a simple touch on the NFC Tag, all the information is promptly presented in a seamless and streamlined environment. To make this possible, each NFC tag stores a unique identifier that identifies the asset it is attached to. Then, the mobile device queries a central server to access all the information about that asset. Infraspeak also develops a web interface that allows control over field operations, which is primarily used by management teams.

The current architecture of Infraspeak Next has limitations on offline operations since that, in this scenario, the mobile device cannot query the server to get information on the asset identified by the NFC Tag.

1.2 Objectives

This dissertation aims to research ways of modifying Infraspeak Next and the corresponding NFC tags to make the system offline-based, allowing maintenance technicians to access information about the assets without an internet connection. Furthermore, it would also be beneficial to improve the NFC usage at the company by researching and implementing solutions to innovate the current usage of NFC Tags.

Another research topic this dissertation will focus on is the security of the proposed system since some assets might contain sensitive information. The system should also allow maintenance technicians to edit information on the assets securely.

This process includes designing the system, researching NFC tags, selecting a suitable NFC tag, and selecting what information the NFC tag should store.

Finally, a product should be developed. This product is an application that follows the design proposed in this dissertation and should be compatible with iOS and Android and developed in Kotlin Multiplatform Mobile.

The system should expect an eventual server connection to send all the data created during the maintenance session to the central server.

1.3 Benefits of solving the problem

An offline-based information system is beneficial in maintenance operations for various reasons. Firstly, it allows Infraspeak to expand its user base to maintenance technicians who work in remote locations without internet access, such as a mine.

Secondly, there are also situations when server connection might be impossible, for example, in the case of power outages, network problems, server downtime, attacks and many other scenarios. In this scenario, what would previously disable most of the maintenance operations or drastically increase their time, is to become no issue.

Finally, solving this problem boosts the reliability of Infraspeak Next, hardens the system against shortcomings and expands the user base of Infraspeak. It is, therefore, of great value to solve this problem.

Chapter 2

State of the Art

2.1 Near Field Communication

Near Field Communication is a short-range wireless technology that enables communication between devices over a distance of, at most, a few centimetres, but that usually leads to the devices touching rather than staying at a distance. It is based on RFID technology and allows for data exchange between devices such as smartphones, tablets, and other electronic devices.

One of the critical features of NFC is its ability to establish a connection between devices quickly and easily. Bringing two NFC-enabled devices close together enables them to establish a connection and exchange data without requiring manual pairing or configuration. This smooth user experience makes NFC an attractive option for many mobile payments, access control, and data exchange applications.

The development and specifications of NFC are driven by NFC Forum, which was created in 2004 in order to standardise NFC applications.

There are two communication modes for NFC:

- Active - The initiator device and the target device have their power source and communicate by alternating signal transmission;
- Passive - The target device does not have a power unit so the initiator device, typically a cellphone, generates radio signals and transmits power to the target device by induction, initiating the interaction.

There are six operating modes for NFC, specified by the NFC Forum:

- Card Emulation mode
- Reader / Writer mode
- Wireless Charging mode
- Host Card Emulation

- Peer-to-Peer mode
- Secure Element-based Card Emulation

This thesis will be focused on Reader / Writer mode since that is the operation mode that allows communication with NFC tags.

2.1.1 Tags

An NFC Tag is a passive NFC-compatible memory card containing an antenna and a small memory. Being a passive device, an NFC Tag cannot generate power.

NFC Tags can be read-only, rewritable or writable once. In rewritable tags, the user can set them to read-only, blocking further writing. Each NFC tag also has a read-only Unique Identifier that is attributed to each tag on manufacturing. The size of each UID depends on the tag type.

NFC Forum Tags host a formatted payload specified as an NDEF Record. These tags are available in a variety of sizes, shapes and capacities. However, they are divided based on differences in the communication protocol and data structure of each tag type, but their behaviour is similar. The NFC Forum has specified five types of tags:

- Type 1: The cheapest tag Type. It provides very little storage and does not provide data collision protection. In 2021 this tag type was removed from NFC Forum Technical Specification to simplify the implementation of future NFC-enabled devices. Its communication was based on NFC-A Technology;
- Type 2: It is similar to tag Type 1 but provides anticollision support;
- Type 3: The communication technology of this tag is based on NFC-F Technology, which is compatible with the Japanese Industrial Standard X 6319-4;
- Type 4: This is the biggest and fastest NFC tag type, which also translates to a higher cost. It can store up to 32KB and work at speeds up to 424 Kbps. This tag supports communication in either NFC-A or NFC-B Technology.
- Type 5: This is the most recent NFC tag defined by NFC Forum. It is based on NFC-V

The standard ISO14443 (for NFC tags) does not specify encryption or security for contactless communication. The developer must implement such a feature on the application level. Some proprietary tags provide numerous features such as hardware encryption, encrypted communication, and signatures. An example is NTAG® 424 DNA [37]

2.1.2 NFC Data Exchange Format

NDEF defines a format, maintained by the NFC Forum, and a set of rules to exchange information between NFC Forum devices or between an NFC Forum device and an NFC Forum tag. It provides

an abstraction of the actual storage technology, making it possible for every device to communicate without knowing the other device's characteristics.

The format consists of an NDEF message, the primary communication block. Each NDEF message can contain multiple NDEF records. Each NDEF record contains a header that contextualises the payload data and a payload that contains the data. Figure 2.1 shows the architecture of an NDEF message and Records.

Figure 2.2 shows the structure of an NDEF Record. It contains five flags [3]:

- Message Begin (MB): Indicates if the message begins;
- Message End (ME): Indicates if the message ends;
- Chunk Flag (CF): Indicates if the payload is distributed across multiple NDEF Records;
- Short Record (SR): Indicates if the records have 7 or 10 bytes;
- ID Length (IL): Indicates if the header contains ID length and ID.

An NDEF Record also contains a Type Name Format (TNF) field that specifies the type of information on the payload. There are eight options [3]:

- 0x0: Record is empty;
- 0x1: URI of NFC Forum well-known type;
- 0x2: Media-type;
- 0x3: Absolute URI;
- 0x4: NFC Forum external type;
- 0x5: Unknown;
- 0x6: Unchanged; Meaning it continues the payload of the last chunked record;
- 0x7: Reserved for future use.

Type Length, Payload Length and ID Length indicate the length of Type, Payload and ID fields, respectively.

The Type specifies the record type, expanding TNF.

The ID follows the structure indicated by TNF and indicates the type of the payload. The combination of TNF and Type allows the operative system to open an application when an NFC tag containing the type specified by the application is tapped.

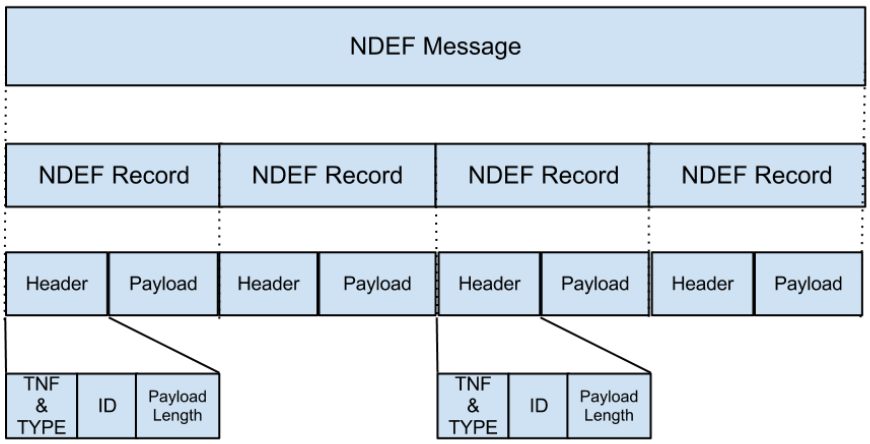


Figure 2.1: NDEF Message

2.1.3 Security

Security measures and attacks are always necessary to research when creating software. So, in this section, attacks on NFC devices are explored, more specifically NFC tags, and possible measures to prevent them. Depending on the context, some of these attacks do not pose a threat to the software being developed. For example, Eavesdropping is not a threat if the tag is supposed to be of open access to everyone.

2.1.3.1 Attacks

Eavesdropping: Being NFC an over-the-air technology, it is possible to eavesdrop on communications. Kirschenbaum [33] has proved that it is possible to build a low-cost antenna to eavesdrop on NFC communications up to 0.5m. Although impractical, this can be an issue in some environments. Eavesdropping is an attack on confidentiality.

Phishing: In this attack, attackers try to mislead a user into scanning and acting upon a malicious NFC tag. Let us imagine an NFC tag that connects to an Internet Access Point (this action requires user approval). If an attacker were to modify the tag into another that sends a message to activate a premium rate service (which also requires the user's approval), the user could be tricked to, out of habit, accept the prompt and subscribe to the service without realising.

Cloning: In this attack, attackers clone tag information, or in some cases, the entire tag, including Unique Identifier using programmable tags. This could disrupt services that rely on a tag to identify objects uniquely. Mulliner [5] identifies an attack on a group of vending machines that was enabled by the possibility of cloning the tag. In the described attack, every vending machine had a tag with its ID (ex: "VENDING1"). When a user scans the tag, they will be able to buy products on that machine. This way, if an attacker cloned the tag of Machine A and attached it to Machine

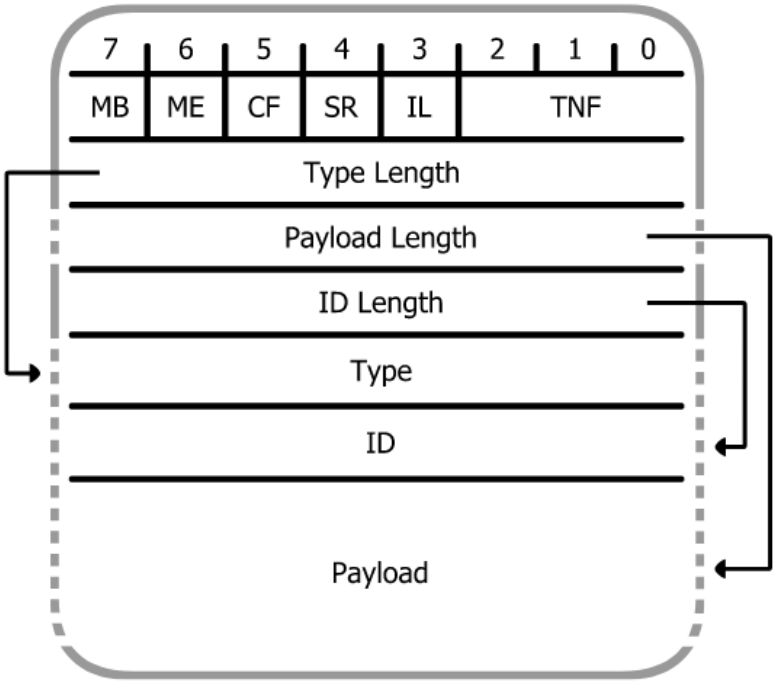


Figure 2.2: NDEF Record [14]

B, they could wait on Machine A until someone tries to buy something on Machine B but would instead scan Machine A's code, and the product would be dispensed on Machine A.

Physical modification: This is an attack on the physical infrastructure of an NFC tag. The vending machine attack described by Mulliner [5] could also be performed by simply removing the tag from vending machine A and attaching it to vending machine B without executing any spoofing or cloning attack. Destruction of the tag could also be considered a physical modification. It can be equivalent to a Denial of Service attack since it renders the tag unusable and its information lost.

2.2 Apple App Clips and Google Play Instant

Google Play Instant was launched in 2016 and allows users to interact with native applications without installing them. An Instant App has a limit of 15MB, meaning that, in most cases, it features only a small part of the parent application. Users can try Instant Apps without committing to the download of the parent app, which causes more interactions with the app and, hopefully, a bigger influx of users to the main app. Instant apps can be launched from the play store through a "Try now" button or through an NFC Tag, a QR Code or a link, by using Deep Links. [32]

During the presentation of this new feature in Google I/O 2016, one of the examples Google used was of a parking meter, which would have an NFC Tag that the user would scan in order to,

instantly, access the application which would allow the user to pay for parking without downloading it.

App Clips are the iOS version of Instant Apps, launched in 2021. Their objective is similar and it also works similarly, by allowing a user to have a native experience without downloading any application. [24]

2.3 Kotlin Multiplatform

Kotlin Multiplatform is a Kotlin-based framework maintained by JetBrains. It is still in the Beta phase of development. KMM allows for shared code in Android and iOS. However, contrary to most multiplatform programming frameworks, like Flutter or React Native, KMM requires platform-specific code. This is required to create native UIs or to use platform-specific APIs while sharing some common code, usually associated with business logic or any other code that can be shared between platforms. Kotlin Multiplatform Mobile is suitable for developing an application to read and write NFC tags since NFC APIs are platform specific, but the majority of the code will be common to both iOS and Android. Infraspeak Next is developed using KMM.

2.4 Related Work

Dünnebeil et al. in [9] researched a system to solve a similar problem to the one being researched in this dissertation. The project's goal was to design a system that would allow caregivers and emergency personnel to access patients' data on emergencies to prevent internet outages from putting people's lives at risk. The patients would wear the NFC tags, or the NFC tags would be present in very visible spots around their houses. The information on the NFC tags should be encrypted due to their sensitive nature. The research also proposes a key-sharing infrastructure to provide access to medical personnel from certain institutions.

Rios-Aguilar in [27] explains, as the title suggests, "Security Threats to Business Information Systems Using NFC Read/Write Mode". This article is of paramount importance to this dissertation since it is researching how to design a secure business information system using NFC Read-/Write mode. This article is also relatively recent, which provides good insight into state-of-the-art security threats.

Lesas in [21] provides a detailed guide to NFC, including how it works and how to program an Android application to read and write NFC tags. This is important because one of the goals of this dissertation is to create a mobile application to read and write NFC tags.

Chapter 3

Planning and Proposed Solution

3.1 Planning

The planning is divided into 6 tasks and can be consulted in 3.1. The subsections of this section explain each of the tasks.

3.1.1 Fieldwork

Working on top of an existing Infraspeak Next, it is paramount to deeply understand the product and its uses. The task of this research is to use the existing app, go on the field and see how maintenance operators use it and talk to the developers to understand how the app is developed.

3.1.2 Information Selection

After analysing the existing system, the first step is to select which data should be stored in the NFC tags; since the tags have rather small storage, this is important to guarantee the basic functioning of the system. This requires an understanding of the information stored in the already existing database. Information was planned to be characterised on three different levels:

- Necessary: Information needed for the system to perform tasks required by an MVP;
- Convenient: Information that allows the application to perform more additional tasks that might be desirable but are not crucial;
- Neutral: Information that does not serve a specific purpose when stored in NFC tags;

3.1.3 Proposed Solution Review

With the newly acquired information from the previous steps described in 3.1.1 and 3.1.2, the proposed solution in 3.2 was reviewed to analyse if it was still a suitable solution to implement a secure offline-based information system with NFC tags. A new solution was then designed, applying the necessary changes to the initially proposed solution.

After this review, the NFC tag to be used in the system was to be selected.

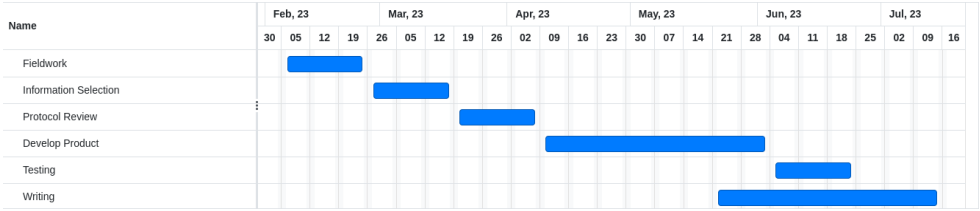


Figure 3.1: Gantt Diagram

3.1.4 Final product Development and Testing

The final goal of this dissertation is to develop a working product that solves the problem exposed during this document and integrates it into Infraspeak Next. Infraspeak Next is being developed in the novel Kotlin Multiplatform framework, and so is the product that originates from this dissertation. It is expected this product to be compatible with both iOS and Android.

It is paramount that the final product is thoroughly tested and verified to ensure it fulfils all of its use cases and is secure. The verification of the final product should include an informal validation with the final users of the product, in this case, the maintenance operators.

3.2 Proposed Solution

The proposed solution depends on various use cases that can only be entirely determined after the Fieldwork is done. However, in this section, the proposed solution will be based on the assumption that the maintenance operators can rewrite a tag and that the information must be kept confidential.

The first step towards a secure offline-based information system with NFC tags is an authentication method to guarantee that only authorised entities can write the NFC tags. To fulfil this requirement, the solution is expected to implement a challenge-response authentication, similar to what is proposed by Saeed and D.Walter in [16], but with the difference that it will be based on symmetric encryption and not public key encryption, saving memory space, processing power and time to implement the system. So, the NFC Tag will have a private key stored in read-protected memory, which will then use to create a cryptographic challenge. The reader should respond to the challenge. If the response is valid, writing permissions are granted. Otherwise, the communication is stopped.

The communication between active and passive devices will be based on NDEF messages. The first NDEF Record's payload will consist of encrypted element data. It is crucial to encrypt the data to protect the confidentiality of the information. The encryption algorithm will be a symmetric encryption algorithm with a shared secret key. The encryption will only encrypt the payload of the NDEF record, which still allows the Operative system to read the information stored in the header of the NDEF Records and act upon it with, for example, opening Infraspeak Next on touch with the NFC tag. The OS could not act upon an encrypted header because it would not be capable of accessing the encrypted information. This encryption step does not prevent eavesdropping, but such an attacker cannot decrypt the data, rendering it unusable for a third party since no clear text

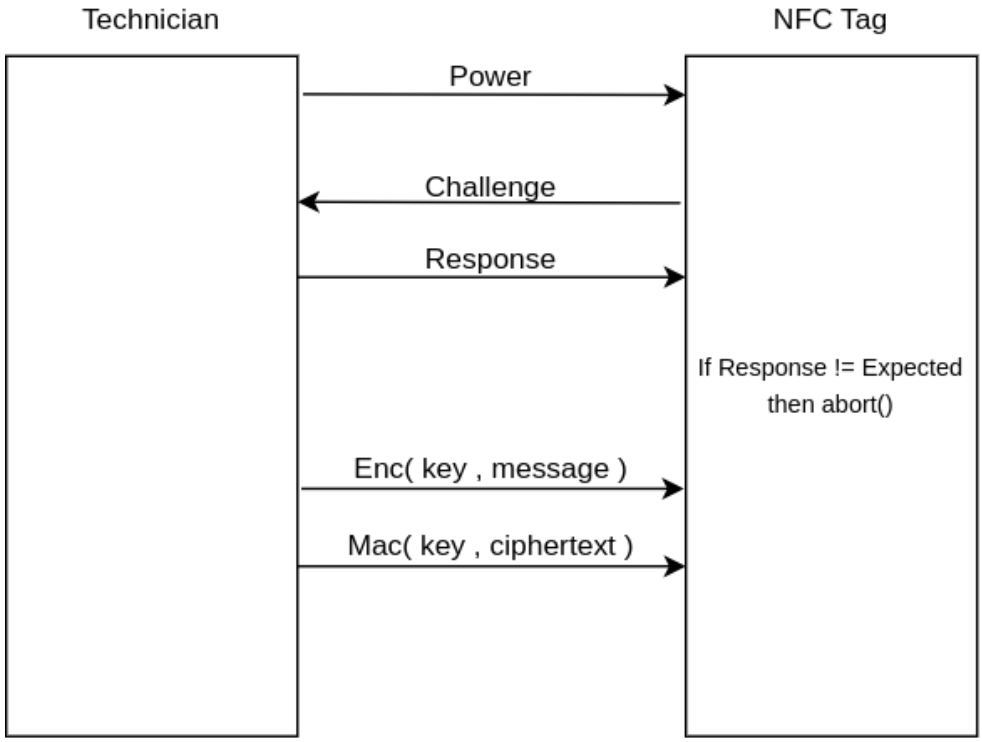


Figure 3.2: Proposed solution diagram (Writing)

data is exchanged in the communication. Further description about key sharing can be found in subsection 3.2.2

A second NDEF record will contain a MAC of the payload of the first NDEF record using the same key. An eventual reader will validate the MAC and only proceed to decrypt the first payload, following an Encrypt-then-MAC approach. The MAC provides integrity since any message that passes the integrity check is guaranteed to have been produced by someone with access to the secret key, which in this case, means a member of the organisation.

The specific characterization of the cryptographic protocols should only take place when the size of the NFC tag and the information to be stored is known since these constraints affect the usage of different algorithms.

A final step is to store all the necessary information regarding the interaction in a local database to send to the central server when an internet connection is available.

A diagram of a writing operation following the proposed solution is presented in figure 3.2

3.2.1 Comparing with other solutions

Another possible solution to build a secure offline-based information system based on NFC tags would be to locally store all the information about the assets in every device that will be used to scan NFC tags. The tags would still only store their unique identifier. This approach has the advantage that NFC tags are cheaper because they do not require significant memory or process cryptographic algorithms. However, it makes it more difficult to synchronise all devices when a

change is made in any of the assets, making it slightly more dependent on an internet connection in order to send to the other devices information that could have been stored in the tag. Another issue with this solution is the scalability problem associated with storing information about all assets.

3.2.2 Key Sharing infrastructure

It is still challenging to design a key-sharing infrastructure since it requires a deep understanding of the use cases, which will be acquired during the fieldwork phase.

However, two possible solutions are proposed in the subsection.

3.2.2.1 Role-based key sharing

Let us assume that certain assets should only be accessed by a specific group of employees based on their permissions. One key-sharing infrastructure could be to have a secret key shared amongst all the entities with the same permission. The keys would be stored on the device that belonged to that entity, and assets that require permission A would have their NFC tags with the secret key corresponding to permission A, for example. Each employee could have multiple keys stored in their device on a protected keystore.

This solution allows for a genuinely offline experience.

3.2.2.2 Unique element key

Another key-sharing infrastructure could be to have a unique secret key for each element. All the secret keys would be stored in a central server, and when an employee was scheduled to fix a particular element, the secret key corresponding to that element would be earlier transmitted to their device to be later used.

This solution allows for much more fine-grained access control but requires an internet connection before accessing any element, which might not be suitable for some use cases.

3.2.2.3 Conclusion

Knowing the benefits of each solution, in this context, it is more beneficial to use role-based key sharing, because there is a great focus from the company on allowing the user to operate in an offline environment as much as possible.

Chapter 4

Fieldwork

During the Fieldwork phase, I worked on getting to know more about Infraspeak and Infraspeak Next. During this time, I was introduced to an onboarding process where I got to use Infraspeak Next like a real maintenance technician in a very hands-on approach. Later I got to speak with different colleagues from different departments (Product, Engineering, Customer Success, Finance) to deeply understand the problem being tackled in this dissertation. I also developed some features for Infraspeak Next that were later developed for production. The goal of developing real features for the application was to get to know more about the application and to get used to the work methods and engineering practises that I will have to later implement when developing the proof of concept for the final solution.

4.1 Infraspeak history

The idea behind Infraspeak was born in a FEUP dissertation [10]. The author, Luís Carlos Martins, would later create the company based on the project developed in his master's thesis. Luís was a Civil Engineer and had very little formal education in programming but still single-handedly created the first versions of Infraspeak's technician app, only available for Android. Only after the company started to grow, some software engineers were hired.

4.2 Infraspeak Next

Infraspeak Next is a recent (less than two years) engineering project being developed in Infraspeak to merge all the existing apps in a single, well-structured mobile application, available both in iOS and Android, re-writing all the pre-existing applications. This aims to create an easily maintainable application that can grow with the company without posing a significant bottleneck to the development. The vast majority of tasks assigned to the Mobile team are focused on developing Infraspeak Next while the technician app is only maintained and no longer has added features.

There is a great effort inside the team to build an app that is scalable and that follows the best software engineering practices.

One of the more noticeable practises of Infraspeak Next development is that of constant delivery. The Mobile team tries hard to create a new release for the application every week. This means that there is an urge to deliver quick value to the client, even if only small changes. This is also very evident in the company's motto "Move fast, break things".

4.2.1 Offline Support

At Infraspeak, there is already a big focus on providing offline support. However, it focuses mainly on storing information on the device storage, which was explored in 3.2.1, where the conclusion was that it is not a perfect solution due to scalability issues.

One solution that is used at Infraspeak is to store in the mobile device only part of the data. For example, a work order can contain images, documents, chat messages and many other informations. There are millions of work orders in Infraspeak's database. To prevent the application from becoming huge in size and consuming an unbearable amount of network and device resources, Infraspeak Next only parses the 20 last work orders to store in the mobile device and then stores any other work order that the user interacts with. By default, the images, documents and chat messages are not stored in the device, unless the user opens them while they have access to the network. The work orders are never deleted from the device unless there is an application deletion. This method rapidly scales up not only the allocated size of the application but also the network resources consumed to synchronize all of these work orders.

Infraspeak Next allows a user to create or update work orders in an offline environment. This is done using an algorithm that stores the requests needed to execute the needed actions and that is later sent to the server when the network is available again.

4.3 NFC in Infraspeak

Currently, NFC tags have different purposes at Infraspeak. First of all, they identify assets. This is very useful since a user can consult the information about an element or create a Work Order with a single tap without having to search from a list with thousands of different assets. This functionality is also one of the biggest selling points of Infraspeak since it simplifies so much the process. NFC Tags can also be used to log in, instead of using a username and a password.

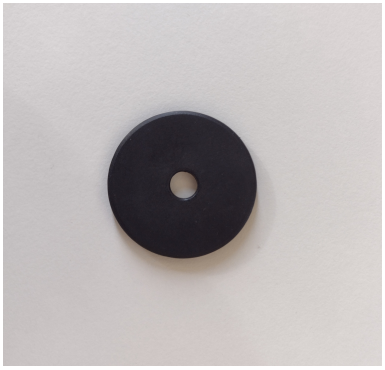
As it is possible to see in figure 4.1, Infraspeak sells six different NFC Tags to their clients. Tag 1 (4.1a) was the first ever used tag and is now not very used. It was meant to be versatile but had technical problems when placed in aluminium objects. Tag 2 (4.1b) is now the most used NFC Tag. It is more robust than Tag 1 and also works on all surfaces. Tag 3 (4.1c) is only used for login operations and is usually held in keychains. Tags 4 to 6 (4.1d, 4.1e, 4.1f) are meant to be robust, waterproof and shock-proof while also protecting the NFC components against chemical damage. These tags are a niche and also more expensive than the previous ones and less discreet and, therefore, not so used.

Tag 2 is the model NTAG213 by NXP Semiconductors, which means it supports password write or read protection using a 32-bit password [20].

(a) Infraspeak Tag 1

(b) Infraspeak Tag 2

(c) Infraspeak Tag 3



(d) Infraspeak Tag 4

(e) Infraspeak Tag 5

(f) Infraspeak Tag 6

Figure 4.1: NFC tags used by Infraspeak

Table 4.1: Information on NFC tags used by Infraspeak - Part 1

Image	NFC Forum Tag Type	Technology	Proprietary Version
4.1a	Type 2	NFC-A	NTAG213
4.1b	Type 2	NFC-A	NTAG213
4.1c	Type 2	NFC-A	NTAG213
4.1d	Not a NFC Forum Tag Type	NFC-V	ICODE SLIX2
4.1e	Type 2	NFC-A	NTAG216
4.1f	Type 2	NFC-A	MF0ICU1

Table 4.2: Information on NFC tags used by Infraspeak - Part 2

Image	Available Memory	Usage	Password Protection
4.1a	137 B	0.4%	Yes
4.1b	137 B	98%	Yes
4.1c	137 B	0.3%	Yes
4.1d	321 B	0.4%	Yes
4.1e	868 B	0.6%	Yes
4.1f	46 B	0.3%	No

Table 4.2 shows various information about the tags described. The available memory values refer to available user memory, this means that it excludes the reserved space for the tag UUID, which is considered system memory. It is also important to notice that NDEF format requires 8 bytes per NDEF message to be reserved for the NDEF message header.

4.4 Challenges to the project

Considering the previous sections, some challenges arise to the previously designed project. First of all, all the NFC tags are used both by the technician app and by Infraspeak Next. Meaning that any solution developed using the NFC Tags and Infraspeak Next should require no to little intervention on the technician app, which poses a great challenge to backward compatibility.

A second challenge comes from the memory size of the currently used NFC Tags. Since Infraspeak already sold more than 650 000 NFC Tags to their clients, it is infeasible to just switch them all to bigger memory tags. In light of constant delivery and the need that Infraspeak finds in constantly upgrading their product to their existing clients, a significant part of the project should focus on delivering value to Infraspeak or its clients, making use of the small memory available on the tags.

Another challenge is that, currently, different tags have different sizes, meaning that any solution should have this in consideration. This can be done by, at least, not trying to write more content to the tag than it can contain but also, in a better scenario, making use of the additional memory of the bigger tags to provide additional value. In other words, in the perfect scenario, the protocol should adapt itself to the tag it is reading from or writing to in order to maximize performance. Either way, the focus should be on developing a solution to be used with the tag 4.1b since it fills 98% of the tags' usage.

Chapter 5

Information selection

In this chapter, an analysis of the information that could potentially be stored in the NFC tags will be made. Firstly, in section 5.1 all information will be presented and then it will be researched which information is useful and provides value to Infraspeak. Then, in section 5.2 how such information can be efficiently stored in the NFC Tags. Finally, in section 5.3 taking into account, the available memory space, a final decision regarding the use of the memory will be made.

5.1 Available information

In the context of the company, an element can be either an equipment or a location and both are similar to one another, meaning that both can be associated with an NFC tag and that both have similar attributes. On the JSON listing 5.1 it is showed the format of the data structure. The type is always "element" for the assets. The id is the same as the element_id, but the element_id is stored as an integer. The type can only be, in this context, "EQUIPMENT" or "LOCATION". The observation is a string that can be quite big. Of all of these fields, the only one that can potentially store sensitive information is the observation field, which can contain technical details about the element.

Each element can also have relationships with other tables of the database. Each relationship stores the type of the other end of the relationship and the respective id. Relationships can include auditStats, categories, category, economics, client, element, elementCharacteristics, entity, failures, files, local, location, maintenancePolicy, meteringRegistry, openFailures, otherCosts, profilePicture, registry, stocks, tasks, housekeeperStatus, housekeeperAssetStatus.

Other tables, other than the element itself can also contain useful information to be displayed in an offline environment. An example would be the reason behind the last work order related to that element or the date or information of the next scheduled work order. However, in this project, there is a very restrictive space constraint so it is impossible to store all information in the NFC Tag. Having this in mind, and after consulting the engineering and product team of Infraspeak, we concluded that the only table, other than the element, that can be valuable for storing in the NFC Tag is the Category table.

```
{
    "data": {
      "type": "element",
      "id": "9384",
      "attributes": {
        "element_id": 9384,
        "type": "EQUIPMENT",
        "code": "44921AB",
        "entity_id": 12,
        "observation": null,
        "has_nfc": true,
        "has_bar_code": false,
        "date_deleted": "2022-09-03 12:25:37",
        "failures_count": 0,
        "barcode": null,
        "last_modified": "2021-11-03 20:44:05",
        "local_id": 14548,
        "latitude": 0,
        "longitude": 0
      }
    }
}
```

Listing 5.1: JSON of element (example)

```
{
  "data": [
    {
      "type": "category",
      "id": "90540",
      "attributes": {
        "category_id": 90540,
        "type": "TYPE_OF_EQUIPMENT",
        "has_children": false,
        "name": "Centrifugal Pumps",
        "code": "03",
        "full_code": "SFG20.45.03",
        "parent_id": 90537,
        "entity_id": 12,
        "is_real": true,
        "all_technical_skills": true
      }
    }
  ]
}
```

Listing 5.2: JSON of category (example)

The JSON listing 5.2 shows an example data of a category. The category_id is an integer, the type is a string that can only have two values "SOFT_MAINTENANCE" and "TYPE_OF_EQUIPMENT". The name is a string that can have any size. The code is a string that can have any size but is usually small. The full code is the concatenation of the parents of the category and the category with dots in between. The fields parent_id and entity_id are both integers.

5.2 Storing Information

Since the storage space of the NFC tags is so small, the way the information is stored can cause a huge impact on the quantity of information that can be stored.

First of all, there is no need to store the identifiers of the variables. The application should read a sequence of bytes and link its contents to the respective identifier based on their order.

A boolean can easily be stored in a single bit and the "type" of the element, given that it can only contain two different values, can also be stored in a single bit.

The "has_nfc", "has_bar_code" fields are useless since this project will be implemented only in assets associated with NFC Tags, and the barcode is used as a substitute in cases where the clients do not want to use NFC tags, which is very rare. Following the same line, the "barcode" field is also useless.

The "latitude" and "longitude" are also useless since that when a user is scanning the NFC tag they are already present in the location of that equipment and do not need to know how to get there or where it is.

Generally, 32-bit unsigned integers are more than enough for the integers represented in both the Asset and the Category. However, there is one exception that can benefit from storing a 64-bit unsigned integer, which is the element_id of the Asset. At this moment, Infraspeak stores more than 650 000 assets in its database. Since Infraspeak is still a startup, it would be possible, considering an exponential growth of the company, that the number of assets reaches 4 294 967 295, which is the number in which 32-bit integers overflow. It is unlikely, but a possibility, considering the number of assets that exist everywhere.

Another field that could be worth storing would be an integer representing the version of the message stored in the tag. In this scenario, information can be stored in the tag without being the most updated version of such information. In order to keep track of which information is being retrieved when scanning the tag, a version number can be useful.

Furthermore, there needs to be a field to identify which role the user should have in order to modify the tag, following the key-sharing protocol discussed in 3.2.2. The application would then, based on that value select the key necessary to write to the NFC Tag or display an error message transmitting the lack of permissions. An integer is sufficient for this value.

5.2.1 String storage

Storing strings in such a small space is an incredible challenge. Some algorithms can significantly reduce the size of strings. Some classical examples are the Huffman Encoding [1] and the Lempel-Ziv-Welch (LZW) algorithm [2] and, more modernly, the gzip algorithm [31]. However, all of these three suffer from the same problem, they are not efficient for small-size strings and sometimes even increase the total size of the string. Even if these algorithms can reduce 50% of the size of a 400-byte string, which is roughly one-fifth of the size of this subsection, it would fill the NFC tag memory almost entirely with little space for strings that do not compress so well, making this approach inviable. Some modern approaches to the problem of compressing small strings like Unishox [26] and Smaz [28] also pose a problem to this project. These algorithms require a dictionary to be stored in the device in order to work. These dictionaries are language specific, which means that Infraspeak, being a company with a lot of international clients and looking to expand to even more countries, would need to store a dictionary for each language they support, which would represent a significant impact on application size.

However, there are some exceptions when looking into storing strings. Taking a look at the JSON structure 5.1. The date_deleted and the last_modified fields are strings but have a fixed size of 10 bytes since they follow a rigid format. A better compression could be applied to that string, storing it as a Unix timestamp, which represents the number of seconds since 1970-01-01 00:00:00 UTC. For example, "2022-09-03 12:25:37" could be translated to 1662204337. Historically, Unix timestamp is stored in a signed 32-bit integer, however, that poses a problem since the integer will overflow in the year 2038. To prevent that from happening, it is possible to use a 32-bit

unsigned integer, which will only overflow in 2106 or a 64-bit integer, which would, practically, not overflow. Using an unsigned 32-bit integer, this optimization reduces the size of the string by 60%.

Finally, it is impossible to completely store the description field, regardless of its importance, since there is no limit to its size and there are no guarantees that, even compressed, it would fit in the available size. At most, it would be possible to truncate it and only store part of the information.

5.2.2 Infraspeak link

It is possible to launch both App Clips and Instant Apps from NFC Tags. It is also possible to launch both of them using the same NFC Tag, containing the same link, by associating the link domain to the Associated Domains in the iOS application and creating an intent filter in the Android application. Considering that Infraspeak's website is "https://infraspeak.com", such a link would need 14 bytes to be stored in an NDEF Message. This comes from the fact that the NDEF Message header already contains the URI type, "https://" in this case and the payload would only contain "infraspeak.com".

5.3 Conclusion

Considering the usage of App Clips and Instant app, the final selected information contained in the NFC Tag would require 1 NDEF record, for the Infraspeak website's link. Regarding the asset the NFC Tag is associated with, it is stored directly in memory without the need for an NDEF record. This means that 22 bytes are already reserved, 8 for the NDEF Record header and 14 for the link itself, leaving 115 bytes left that can store information.

With these 115 free bytes or 920 bits, the proof of concept application could store the element_id (64 bits), the type (1 bit), the entity_id (32 bits), the date_deleted (32 bits), the failures_count (32 bits), the last_modified (32 bits) and the local_id (32 bits). This sums up a total of 225 bits. From the category table, it could store the category_id (32 bits), the type (1 bit), the has_children field (1 bit), the parent_id (32 bits), the entity_id (32 bits), the is_real (1 bit) and the all_technical_fields (1 bit). Additionally, the version number (32 bits) will also be stored. In total, using 325 bits from the category and the element tables plus the version number and the role number, leaving 579 bits free. Since this project will be implemented as a proof of concept, it is not necessary to completely fill the available memory. The stored values from the element and category tables are sufficient to prove the concept and be useful in the context of Infraspeak's needs.

Chapter 6

Features and improvements

In this chapter, the features and improvements that will be added to Infraspeak Next, following the research done in this dissertation, will be presented.

6.1 Offline Support

In terms of offline support, the main focus of this project is to allow technicians to consult information about assets without an internet connection. The old Technician app allowed technicians to scan an NFC Tag and consult all kinds of information about the element in front of them, requiring, however, an internet connection to do so. While Infraspeak Next does not yet support this feature, it will in the near future. Each tag will store relevant information regarding the element and its category so that a technician can have a general overview of the element in front of them and be presented with useful information to complete their function. This feature is of major importance in the product of Infraspeak, so providing offline support to it creates palpable value for the company and its clients.

6.2 Password Protection

Knowing that the vast majority of Infraspeak's NFC tags support password protection, it is possible to allow a user to securely re-write the information in the NFC tag. This is useful since there are some information fields that do not contain static values, such as the "last_modified" field or that can be later modified for one reason or another. Password protection is still of major importance even if nothing was written on the NFC Tags. At this point, Infraspeak's tags are writable and do not store anything, but any person can write to the tag's memory. This allows an attacker to store links, plain text messages, instant app links, etc. in the tag, which can potentially damage Infraspeak's reputation. In an even worse scenario, the tags' memory can enable phishing attempts, since a user that scans the tag would easily be misled into thinking the information contained in the tag is official information from Infraspeak since the tags contain the logo of Infraspeak.

This kind of protection does, however, present some problems.

Firstly, knowing that the password is only 32-bit long, it is deemed insecure by modern standards. Nonetheless, the fact that an attacker would have to physically interact with the NFC Tag in order to crack the password somewhat mitigates partially this issue. Furthermore, it is possible to define a maximum failed authentications limit to prevent brute force.

The second issue comes from eavesdropping. In this context, the password is transmitted over clear text to the NFC Tag. An eavesdropper could passively get access to the password if they stand close enough to the NFC Tag. This scenario is highly unlikely since it would require the attacker to be so close to the NFC Tag and user that it would easily be noticeable. This issue only happens in tags of type NTAG21x. While the tags of type ICODE SLIX2 use a challenge-response authentication protocol that does not disclaim the password publicly.

The third problem is that one of the types of NFC Tags that Infraspeak sells does not provide password protection mechanisms or any other writing-protection mechanism as can be seen in table 4.2. Furthermore, this specific tag also has the lowest amount of memory (46 bytes), which means that it is also impossible to implement cryptographic operations to ensure that the information is authentic. Therefore, either the information is written to tag with the accepted risk that it can be re-written by a third party or it is not written at all, losing offline support capabilities. In this project, it will be assumed that this specific tag will not be written.

6.3 Automatic Application Launch

After analysing the current state of Infraspeak's product and talking to the Product team, I realized that having an Automatic Application Launch using the NFC Tags would be greatly appreciated by the users. By declaring an intent filter, the Android operating system automatically opens the Infraspeak Next app when it detects an NFC Tag containing an NDEF Message with Infraspeak website URL. This is called deep linking. To further improve this behaviour, Infrapeak should upload an assetlinks.json to the server serving the link, which will allow the Android operative system to know that Infraspeak's app is the true owner of that link. These links are then called Android App Links. By the Android wiki [29], "Android App Links are simply HTTP deep links that your website is verified to own so that the user doesn't need to choose which app to open". In iOS, using custom URLs will provide the same experience.

6.4 Instant Experiences

Instant experiences like iOS App Clips and Android Instant apps also bring a lot of value to Infraspeak and its clients. After having meetings with the Product team at Infraspeak, we reached the conclusion that a valuable instant experience to be added to Infraspeak Next would be the ability for an unauthenticated user to report problems in equipments.

First of all, it is great for marketing purposes. Any person that finds an Infraspeak tag in a building will be able to interact with it without knowing what Infraspeak is or does and without installing any application.

Furthermore, it also brings value to clients since it allows for more automatic and agile operations. Let's consider an Infraspeak client, a hotel, for example, which is a fairly common type of client. Currently, if a guest of the hotel found a problem with an equipment in their room they would have to go to the reception and report the problem there. The hotel employee would then write the report in Infraspeak Next. By using instant experiences, the hotel guest can make the report autonomously, providing an overall seamless experience to the guest, faster fix times and an increase in satisfaction by all parties involved.

Both Android and iOS provide documentation that allows a developer to create an instant experience link. In Android, the process of creating a link for the Instant App is very similar to that of creating an Android App Link. The only difference is that the intent filter of the Instant App should enable both HTTP and HTTPS protocols. Usually, the intent filter is only useful once an app is installed, but for instant Apps, Google Play stores the intent filters in order to detect which app should be launched when a certain link is scanned, that is why intent filters are so important in Android Instant Apps.

In iOS, Infraspeak should upload the apple-app-associaton-file, which points the operative system towards a full App or an App Clip, depending if the main app is installed or not.

Chapter 7

Proof of concept

7.1 First Version

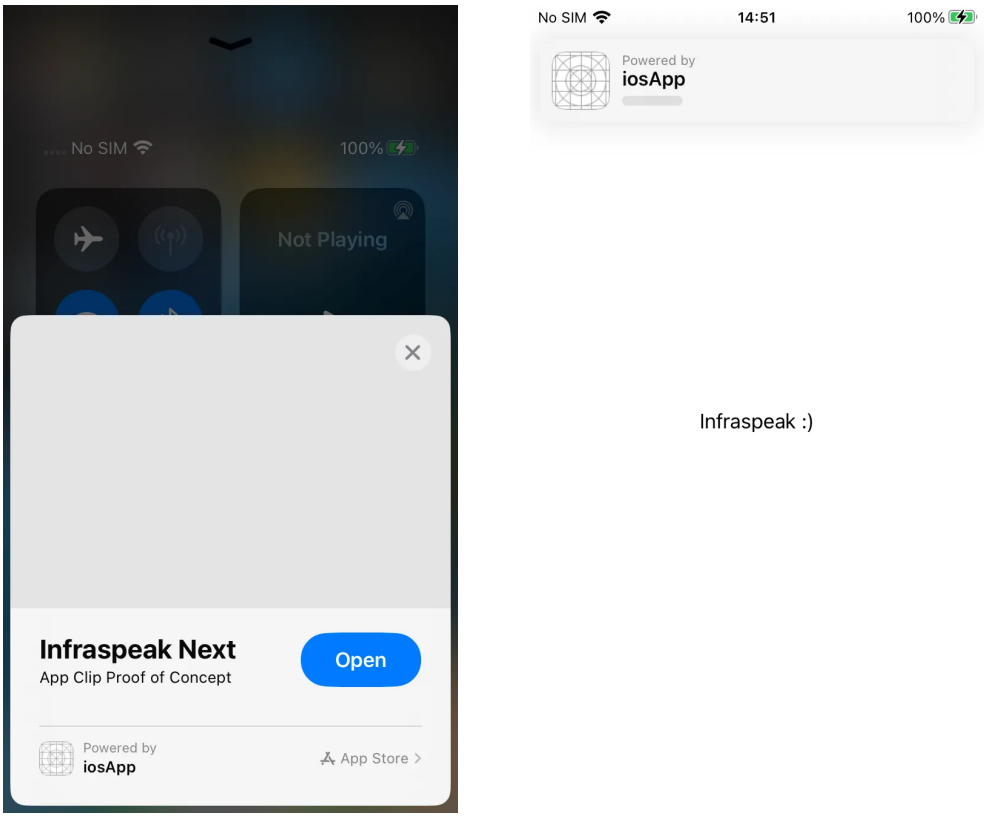
The focus of the first version of the Proof of Concept is to enable basic NFC features, while not yet reading and writing the element/category data.

7.1.1 Write the link

The first step towards enabling instant experiences and, generally, linking the NFC Tags to Infraspeak's website is to write the link in all NFC Tags. Since most tags are already deployed, the system should identify if a given tag stores an NDEF record containing the URL of Infraspeak's website. If it doesn't, the application will write it. This process allows Infraspeak to update the information stored in all tags, given that the users use an updated version of the app. Since previous versions of the app do not use information stored in the tag and only use the Unique Identifier of the tag, this process will not have an impact on any operation (past, current or future).

From a programming point of view, the Android application should detect the NFC intent by overriding the `onNewIntent()` function. At this point, there are two possible branches. Either the intent is `ACTION_NDEF_DISCOVERED` or it isn't. In the case that it isn't, it means that there are no NDEF Records stored in the tag. Therefore the program will create an NDEF Record with the URI "https://infraspeak.com". Then it will create an NDEF Message with that NDEF Record and finally write to the tag the NDEF Message. The second option happens when the NFC tag already has stored an NDEF Message. In that case, the application will parse the messages and store them as an array. If the first message is not the link, then it will create an NDEF Record with the link and append the rest of the previously read NDEF Records to it, forming an NDEF Message that is then written to the tag.

Following the KMM structure, both platforms (iOS and Android) independently implement the native NFC libraries, but all the data processing is done by Core, the shared module between platforms. This way, it allows for a native approach to OS actions, like NFC reading and writing, making development easier and more seamless, while at the same time avoiding code repetition in shared data processing functions that do not require OS-specific behaviour.



(a) App Clip Card (b) App Clip Interface

Figure 7.1: App Clip interfaces from the first iteration of the PoC

7.1.2 Instant Experiences

During this first iteration of the proof of concept, some explorational applications were developed. The first was an example of an Apple App Clip. The goal of the first experience with app clips was to understand how they can be used and their limitations. It is safe to say that it was a success since I was able to develop an app clip associated with Infraspeak Next that launches when an NFC tag with the respective URL is scanned. To do this, it is necessary to add a new target (App Clip) to the current XCode project and develop the app clip inside of the newly created module. For now, a very simple application was created, displaying only static text, as shown in figure 7.1b. The next step is to configure a local experience [39]. A local experience in iOS allows a developer to test the launch of an App Clip without launching the application into App Store Testing. It is possible to configure the link that will launch the instant experience, the title, the subtitle, a photo and the text in the button prompting action. There is a known issue that prevents some iPhones from correctly displaying the photo selected in the local experience form. It happens that the iPhone being used to develop this project is affected by that problem and because of that the image displayed in the App Clip card is always a white background. After configuring the local experience, the developer can scan an NFC tag that contains an NDEF message of the URL selected in the local experience. That action will display an App Clip card. The App Clip card generated from this first iteration of

the Proof of Concept can be seen in 7.1a. The name of the "main" app appears as iOSApp because that's the name of the module in XCode, but, in production, it would appear as "Infraspeak Next" and clicking the banner would redirect the user to Infraspeak Next in the App Store. Clicking "Open" will then open the App Clip without any installation process.

There is however a drawback that will prevent these instant experiences from being deployed to production for now. The problem is that the instant experience would require the backend API to accept unauthenticated or guest account reports, which is currently not possible. These decisions and execution do not belong to the scope of this dissertation so it won't be possible, for now, to deploy the project to production. For this reason, the Instant Experiences developed in this dissertation will be built as a proof of concept to show how an Instant Experience can seamlessly be launched from an NFC Tag and how it would potentially look when deployed. Because of that, it is not deemed a priority to develop an Instant experience for both iOS and Android, since the concept behind it is similar. Furthermore, creating the Instant Experience for Android is more demanding in the sense that, to test it, it is needed to deploy the application to a testing environment in the Play Store, while the process of testing the iOS App Clip is much easier, as described above. So, in this project, the Instant App in Android will not be developed as it adds little to no value to the final proof of concept.

7.2 Second Version

The second step of the Proof of concept will go more in-depth in each of the developed areas, refining the concepts and developing User Interfaces.

7.2.1 Password Protection

As mentioned in 4.3, the great majority of NFC Tags used in Infraspeak belong to the model NTAG213 by NXP Semiconductors. With this information, it is possible to enable password protection in the tags. Reading the datasheet [20], we can get useful information about low-level commands sent to the tag. Each OS provides different methods to interact with the NFC tags by sending low-level commands. Therefore, the implementation of password protection is OS-specific.

7.2.1.1 NFC Tag Low-Level concepts, commands and interactions

Table 7.1, from the NTAG21X datasheet [20], showcases all the available low-level commands used to interact with the tags used in the project that follows the ISO 14443 standard. In this project, commands FAST_READ(0x3A), WRITE(0xA2) and
PWD_AUTH(0x1B) will be used.

The authentication process of the NFC tag of the model NTAG21X is described in the datasheet[20]. Firstly, the users send a command PWD_AUTH, represented by the hexadecimal number 0x1B, followed by 4 bytes containing the password, in hexadecimal. If the password is correct,

Command	ISO/IEC 14443	NFC FORUM	Command code (hexadecimal
Request	REQA	SENS_REQ	26h (7 bit)
Wake-up	WUPA	ALL_REQ	52h (7 bit)
Anticollision CL1	Anticollision CL1	SDD_REQ CL1	93h 20h
Select CL1	Select CL1	SEL_REQ CL1	93h 70h
Anticollision CL2	Anticollision CL2	SDD_REQ CL2	95h 20h
Select CL2	Select CL2	Select CL2	95h 70h
Halt	HLTA	SLP_REQ	50h 00h
GET_VERSION	-	-	60h
READ	-	READ	30h
FAST_READ	-	-	3Ah
WRITE	-	WRITE	A2h
COMP_WRITE	-	-	A0h
READ_CNT	-	-	39h
PWD_AUTH	-	-	1Bh
READ_SIG	-	-	3Ch

Table 7.1: NTAG21x command overview [20]

the tag will send a 2 bytes password acknowledgement (PACK) and the tag will pass from an ACTIVE state to an AUTHENTICATED state. This PACK is a way for the user to authenticate the tag because the PACK is set by the user upon defining the password. If the PACK is the same as the PACK initially defined by the user, both tag and user are authenticated. As said previously, this method is rather insecure due to the low size of the password and PACK. To mitigate this issue, NTAG213 allows a user to define an AUTH_LIM, which is a maximum wrong password attempt limit. It can be set between 0 and 7. 0 represents unlimited tries. When the limit is reached the tag locks itself to a state similar to a read-only, which cannot be reverted. The tag allows for both write or read/write protection. This behaviour can be defined by setting the PROT bit (page 40 (0x28), bit 0) to either 0, write protection, or 1, read and write protection.

In this context, the write protection is the most appropriate measure since a write/read protection would block the OS ability to read NDEF messages and open run Instant Experiences or automatically open the app.

To write contents to the tag's memory, a user can only write 4 bytes at a time, sending a WRITE command (0xA2), the page where the contents will be written to and 4 bytes representing the content to be written. The READ (0x30) command works similarly but the data is not transmitted by the user but rather by the tag and it reads 16 bytes at a time. However, there is another command, FAST_READ, that allows the user to read all pages between a defined first page and a defined last page, making the reading process more convenient.

To set the password for an NFC tag, the user needs to write the 2 bytes PACK to page 44 (0x2C), the 4 bytes password to page 43 (0x2B), set the PROT bit to 0 and define AUTH_LIM, both on the first byte of page 40 (0x28). For now, the AUTH_LIM will be 0 because it can have irreversible consequences and, as a proof of concept, it is not desirable to lock tags permanently. This process considers that either the user is already authenticated or the tag is not yet password

```
1    tag = intent?.getParcelableExtra(NfcAdapter.EXTRA_TAG) as? Tag)
        ?: return
2
3    val mifare = MifareUltralight.get(tag)
4    mifare.connect()
```

Listing 7.1: Kotlin code to connect to an NFC Tag

protected, otherwise, it will fail.

The NFC Tag of type ICODE SLIX2, following the ISO 15693 standard, has different low-level commands and different password-protection mechanisms. The reading and writing commands function in a similar manner, and their hexadecimal values can be consulted in the official datasheet [25]. Given that the great majority of the tags are not of this type, as it can be seen in table 4.2, the explanations regarding its functioning will not be as in-depth. There is an important point regarding password authentication in this type of NFC Tags. It implements a simple challenge-response authentication. The first step in this process is for the application to send a GET_RANDOM_NUMBER (0xB2) command to the tag and it will respond with a 16-bit-long random number. After that, an XOR operation is performed with the number, repeated once and the 32-bit-long password. The result of the operation is transmitted to the tag, finishing the password authentication process. This challenge-response authentication is a rather insecure model of performing the authentication since both the random number and the result of the operation are transmitted in clear text. This means that a passive attacker could eavesdrop on the communications and get both of these values. With this information, it is possible to get the original password and later perform that challenge-response authentication with 100% success rate. This comes because of the mathematical properties of the XOR operation, which dictates that A XOR B = C, C XOR A = B, being A, B and C binary numbers with the same length. This is, in fact, the method used to encrypt and decrypt OTP (One-Time-Password) cyphers. Those are deemed perfectly secure because any party involved, with the exception of the sender and receiver do not have access to the key, only the ciphertext. Any person with at least two of any between ciphertext, clear text and the key is able to get the other. In the context of the tag, the ciphertext is the result of the XOR operation and the key is the random number, being then possible to get the clear text password.

7.2.1.2 Android Low-Level Interaction

When sending low-level commands to NFC Tags, the developer is no longer working with the NDEF abstraction and needs to interact directly with the underlying technology.

```
1    val content = byteArrayOf(0x31.toByte(), 0x32.toByte(), 0x33.
        toByte(), 0x34.toByte())
2    val response = mifare.transceive(byteArrayOf( 0xA2.toByte(), 40.
        toByte(), content[0], content[1], content[2], content[3])
```

Listing 7.2: Kotlin code to write to an NFC Tag - NFCA Library

The tags used, NTAG213, use MifareUltralight technology, which uses NFC-A technology. For this reason, when developing in Android the developer can opt to use more generic libraries to interact with NFC-A technology or use more specific libraries to interact with MifareUltralight proprietary technology. Both are very similar with only minor differences.

The first step to interact with the tag is to connect to it. Listing 7.1 shows the code to connect to a MifareUltralight tag. From here, to authenticate using a password, the developer should use the transceive function [36], which allows to send commands to the tag and get a response. In this case, the command to be used is PWD_AUTH (0x1B) and the password is "pass" or 0x70617373 in hexadecimal. Therefore the code is `val response = mifare.transceive(byteArrayOf( 0x1b.toByte(), 0x70.toByte(), 0x61.toByte(), 0x73.toByte(), 0x73.toByte()))`. This is only a minimal reproducible example since the actual code would not use "Magic Numbers" as commands or data, since that makes the code difficult to read and later debug. The developer should then check if the returned value, the PACK, is equal to the PACK initially defined, meaning that the tag is authentic. After that, the program should read from or write to the tag. At this point, there are two alternatives, to use general NFCA libraries or use the specific MifareUltralight libraries. Both approaches do the same thing, but MifareUltralight libraries provide higher-level functions to interact with the tag. Listing 7.2 shows the more low-level way to write the string "1234" to page 40 of the NFC tag, while 7.3 shows how to write the same string to the same page using a Mifare Ultralight specific library. The second option is easier to read, write and debug so that is the one to be used in this project

7.2.1.3 iOS Low-Level Interaction

The process of using the tag and executing low-level commands in iOS is the same, but the code and the OS interaction vary.

```
1    val content = byteArrayOf(0x31.toByte(), 0x32.toByte(), 0x33.
        toByte(), 0x34.toByte())
2    val response = mifare.writePage(40, content)
```

Listing 7.3: Kotlin code to write to an NFC Tag - Mifare Ultralight Library

```
1
2     func startSession(){
3         session = NFCTagReaderSession(pollingOption: .iso1443,
              delegate: self)
4         session?.begin()
5     }
```

Listing 7.4: Swift code to start an NFC reader session

```
1
2     func tagReaderSession(_ session: NFCTagReaderSession, didDetect
          tags: [NFCTag]){
3         let tag = tags.first!
4         session.connect(to:tag) { error in
5             if error != nil {
6                 self.stopScan()
7                 return
8             }
9             //at this point, the tag is already connected
10
11            if case let .mifare(mifareTag) = tag {
12                let hexcommand: [UInt8] = [0x30, 0x28]
13                mifareTag.sendMifareCommand(commandPacket: Data(
                      hexCommand)) {(responseData, error) in
14                    if let responseString = String(data:responseData
                          , encoding: .ascii)
15                    {
16                        print("Response Data: \(responseString)")
17                    }
18                }
19            }
20
21        }
22    }
```

Listing 7.5: Swift code to connect to and read from tag

While in Android, the system is always listening to incoming NFCs and communicates them to the application via intent, in iOS, it is needed to explicitly call the NFC reader in order to read a tag. Listing 7.4 shows how to start an NFC reader session. It uses polling and it filters the incoming tags to only those following ISO 14443 and ISO 15693 formats. The developer can select more formats, these are the ones used by Infraspeak's NFC Tags.

Listing 7.5 shows the code to connect to an NFC Tag and then read an ASCII string on page 40 (0x28). To send any other command, the same process is used, using `sendMifareCommand`[38] function. This function allows the developer to send, for example, the PWD_AUTH command as well.

7.2.2 Writing to and reading from the tag

In order to write assets' information to the NFC Tag, a data class for the asset was created. The class could be initialized with the primitives corresponding to each field of the class or it could be initialized from binary data in hexadecimal format in order to read it directly from the NFC Tag. Then, there is a private method of the class that transforms the class into binary data in order to write it to the NFC Tag.

Figure 7.2 shows the flowchart of a write operation for the tags following the ISO 14443 standard that Infraspeak sells (NTAGx and MF0ICU1). The first step is, again, to connect to the tag, but then the program should detect which tag is being read. Since all the tags, at this point of the flow, follow the ISO 14443 standard and belong to the Mifare family, one way to distinguish them is by using their memory sizes. To get the memory size of an NFC Tag, a possibility is to send an NDEF command querying the tag for the size of the NDEF Message, which takes over the entire space of the tag, as long as it is NDEF formattable. With this storage information, if the detected size is the same as the size of the MF0ICU1 Tag(4.1f), it aborts the connection, as mentioned in section 6.2. From here, all the tags are of the type NTAGx and it is possible to use either the size of the tags or the GET_VERSION (0x60) command to identify its specific type. At this point, some tags will be password-protected and some will not. Sending a PWD_AUTH command to a tag that is not password protected will result in a fatal I/O error that will finish the NFC connection. Therefore, one way to identify if the tag is password-protected or not is to read the AUTH0 memory block. AUTH0 defines from which memory page the tag is password protected. If AUTH0 is bigger than the user memory size, parts of the tag are password-protected. If it is not password-protected, the flow is to then write the PWD, PACK, the AUTH_LIM and the AUTH0 memory spaces in order to protect the tag with a password and then write the Infraspeak link, which does not need to ever be updated. In it is password protected, the flow is to then authenticate to the tag using the AUTH command and verify the returned PACK. At this point, the tag is already in an authenticated state and the program can freely write to the memory. Then, it is necessary to convert the Asset object into its binary form and write it directly to the tag's memory. Since the WRITE command only writes 4 bytes at a time, it is necessary to iterate over the binary data to write it all to the NFC Tag using the necessary amount of WRITE commands, while checking if it is not trying to write to undesirable memory positions like the LOCK bits

or the PASSWORD bits or any other configuration bits that can affect the tag's behaviour. It is possible to perform those checks by using the size of the NDEF Message, which describes the user memory size.

Finally, the tag disconnects and the writing process is concluded.

Regarding the NFC Tag of type ICODE SLIX2, the difference to the process described above lies in the fact that ICODE SLIX2 does not have an AUTH0 flag. One way to get the security information about the memory blocks is to use the Get Multiple Block Security Status command (0x2C), which allows the developer to know which blocks are password-protected and locked.

Writing the password is also different, as mentioned in section 6.2. There is no need to set PACK, AUTH0 and PROT bits, because this tag does not support those.

These processes were tested multiple times with password-protected tags, tags without password protection, empty tags and tags with old values stored and they worked flawlessly every time.

Reading from the tag is a much more straightforward process and is also the same for all types of tags, changing only the hexadecimal commands to be sent. In this project, password protection only implements write protection so reading can be freely done without password authentication. The reading begins at the defined block after the Infraspeak link is written (0x0C) and gets the necessary data using the FAST_READ command. The data, in hexadecimal format, is then used to initialize the Asset class. The algorithm also does not attempt to read from MF0ICU1 tags.

7.2.3 Instant Experiences Improvements

7.2.3.1 Opening the experience based on tag UID

One significant enhancement introduced in instant experiences is the capability to retrieve the Unique Identifier (UID) of an NFC Tag during the launch of the experience. This functionality empowers the application to tailor the presented information based on the scanned NFC Tag, thereby enabling a personalized user experience. For instance, if a specific NFC Tag possesses the UID "12345" and corresponds to an air conditioner belonging to company ABC, when the user scans the tag and initiates the instant experience, the interface can dynamically display a message such as "Report a problem with the air conditioner to Company ABC." Subsequently, the user can provide detailed observations and submit the report by utilizing a designated button. Leveraging the NFC Tag UID, the application can accurately associate the report request with the specific air conditioner and transmit it to the server. The availability of the NFC Tag UID is crucial in ensuring the accurate identification and processing of user requests within the context of the instant experience.

Both Android and iOS allow this behaviour by sending information to the instant app using URL parameters. So, the NFC Tag should, instead of storing the Infraspeak link, store the link to Infraspeak's website with the NFC Tag UID as a URL parameter, which occupies between 14 and 16 bytes, depending on the tag. Since there is enough space in all tags to accommodate this

```
1   @main
2   struct iosAppClipApp: App {
3       var body: some Scene {
4           WindowGroup {
5               ContentView()
6                   .onContinueUserActivity(
                        NSUserActivityTypeBrowsingWeb) { userActivity
                         in
7                       guard let incomingURL = userActivity.
                            webpageURL,
8                             let component = NSURLComponents(url:
                                  incomingURL,
                              resolvingAgainstBaseURL: true)
9                       else { return }
10                  }
11          }
12      }
13  }
```

Listing 7.6: Swift code to receive the link that launches the app clip

new requirement (at least 499 bits, as mentioned in 5.3), it poses no problem to the selection of information made preciously in 5.

Listing 7.6 shows the Swift code needed to access the URL that invoked the app clip. In this example, the URL is stored in the incoming URL and the query containing the NFC Tag UID can be accessed using the code expression `component.queryItems?.first?.value`.

7.2.3.2 Interface

To make the App Clip Proof of Concept as close to reality as possible, excluding the already mentioned impossibility to send unauthenticated requests to the API, an interface was developed for the App Clip. The goal of this interface is to be as simple as possible and as intuitive as possible in order because the end users of it are people that never had any previous interaction with Infraspeak and only want to report a problem, seamlessly and efficiently.

The functionality was primarily based on the already existing Work Order creation screen, available in Infraspeak Next, where a user can submit Work Orders, containing the type of the problem, which can be selected from a dropdown, a description field, and a section to submit documents and photos to better describe the problem.

The interface also contains very basic information about the asset being scanned in order to prevent mistakes.

The interface was based on already existing designs at Infraspeak and used modular components of the Design System of the company.

Screenshots of the interface can be seen in figure 7.3.

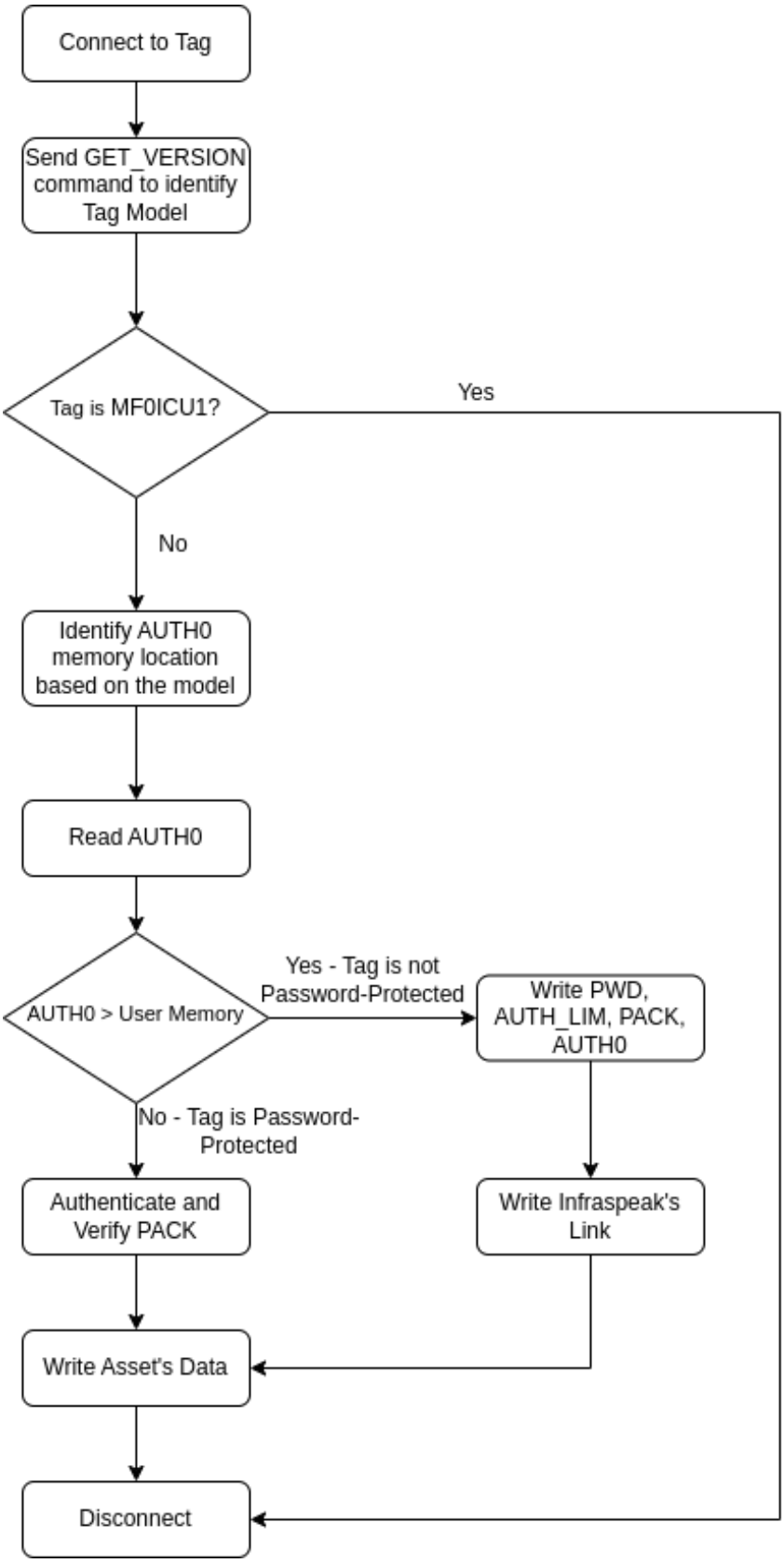


Figure 7.2: NFC Tag write operation flowchart for NTAG21x

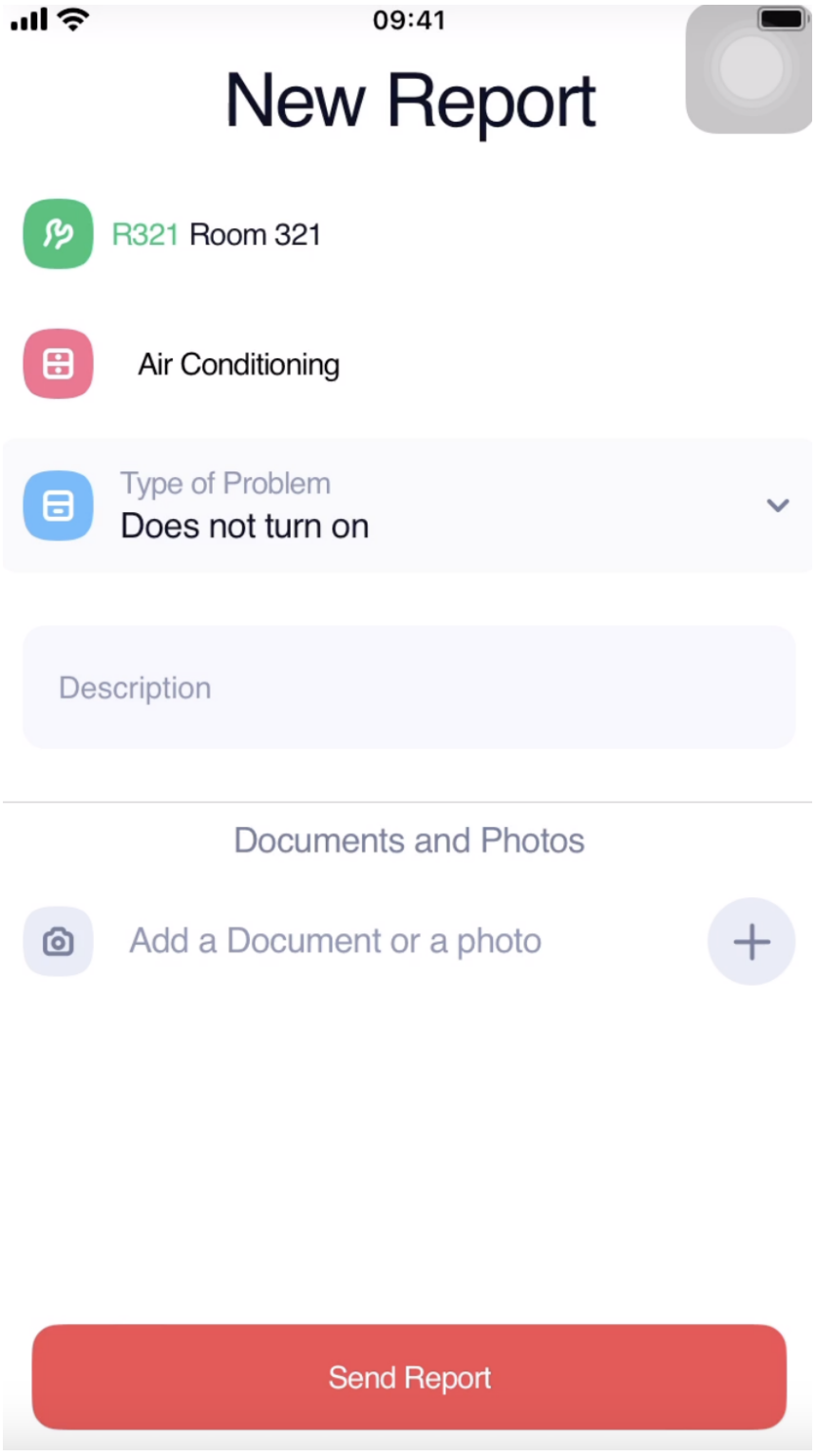


Figure 7.3: App Clip Interface - Final Version

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The work developed in this project did reach its main goal of researching and implementing a modification to the NFC infrastructure at Infraspeak. The costs of launching it into production are minimal because all the tags are already deployed and the PoC is functional.

The Instant experiences, developed in the form of an App Clip, open novel ways of interacting with customers and even with non-customers, boosting the impact Infraspeak has on businesses.

The new offline-support capabilities will come in handy to a variety of clients worldwide and might open new business opportunities for Infraspeak in the offline-first environment.

The password-protection feature, although standard, definitely improves the security of the NFC tags and their contents and prevents future problems related to the misuse of the NFC Tags by malicious actors.

The research developed here also improves the knowledge about NFC at Infraspeak, since all the used NFC technologies and thorough explanations of their functioning are present in this document. Furthermore, the extensive documentation created will also help further develop the PoC with minimal overhead.

Researching and creating the PoC in the context of an internship at Infraspeak was incredibly beneficial for the work, for the author and for the company, showcasing how industry and academia can come together to solve real-world problems and create innovative solutions, together.

All in all the work here developed was a success and positively impacted the NFC usage at Infraspeak.

8.2 Future Work

Despite the great success of the research, it is important to notice that there are improvements that can be made to it and development that can still take place in the code.

First of all, it would be important, in the future, to use more powerful and secure NFC Tags. Right now, there still is information that should be stored in the NFC Tags but does not have space

to be stored. This can be solved by simply choosing an NFC Tag with more internal memory. Furthermore, it was shown, in this research, that all tags used at Infraspeak are insecure and can easily have their passwords eavesdropped. This is a security issue that can be solved by using tags that support cryptographic operations and that would allow a secure challenge-response authentication to take place. A more in-depth description of this process can be found in 3.2. Using NFC Tags with bigger memory sizes also presents new challenges. Some steps that need to be taken after upgrading the hardware include paying attention to the confidentiality of the information stored in the tags because newer information can contain sensitive fields. Furthermore, as the tags grow in memory size, so does the time needed to transmit and write the information. Transmitting big amounts of information through NFC can be a slow process. At the moment this is not an issue since that, at most, only less than 150 bytes of information are transmitted at a time, which happens incredibly fast.

Secondly, in order for the Instant Experiences to be useful, there needs to be an endpoint that allows the requests to be accepted by the server. The specific way of doing that is yet to be researched and debated.

Thirdly, technology is an ever-evolving area. There are always new updates, and new possibilities, especially in recently adopted technologies such as NFC. Surely, new ways to improve Infraspeak's operation using NFC features will arise and then new research will need to be conducted to adapt the system.

Bibliography

[1] David A. Huffman. “A Method for the Construction of Minimum-Redundancy Codes”. In: *Proceedings of the IRE* 40.9 (Sept. 1952), pp. 1098–1101. ISSN: 2162-6634. DOI: 10.1109/JRPROC.1952.273898.

[2] Welch. “A Technique for High-Performance Data Compression”. In: *Computer* 17.6 (June 1984), pp. 8–19. ISSN: 1558-0814. DOI: 10.1109/MC.1984.1659158.

[3] NFC Forum. *NFC Data Exchange Format (NDEF) - Technical Specification*. 2007.

[4] Gerald Madlmayr et al. “NFC Devices: Security and Privacy”. In: *2008 Third International Conference on Availability, Reliability and Security*. 2008 Third International Conference on Availability, Reliability and Security. Mar. 2008, pp. 642–647. DOI: 10.1109/ARES.2008.105.

[5] Collin Mulliner. “Vulnerability Analysis and Attacks on NFC-Enabled Mobile Phones”. In: *2009 International Conference on Availability, Reliability and Security*. 2009 International Conference on Availability, Reliability and Security. Mar. 2009, pp. 695–700. DOI: 10.1109/ARES.2009.46.

[6] NXP Semiconductors. *NFC Forum Type Tags*. 2009.

[7] Erkki Siira, Tuomo Tuikka, and Vili Tormanen. “Location-Based Mobile Wiki Using NFC Tag Infrastructure”. In: *2009 First International Workshop on Near Field Communication*. 2009 First International Workshop on Near Field Communication. Feb. 2009, pp. 56–60. DOI: 10.1109/NFC.2009.8.

[8] Michael Roland and Josef Langer. “Digital Signature Records for the NFC Data Exchange Format”. In: *2010 Second International Workshop on Near Field Communication*. 2010 Second International Workshop on Near Field Communication. Apr. 2010, pp. 71–76. DOI: 10.1109/NFC.2010.10.

[9] Sebastian Dünnebeil et al. “Encrypted NFC Emergency Tags Based on the German Telematics Infrastructure”. In: *2011 Third International Workshop on Near Field Communication*. 2011 Third International Workshop on Near Field Communication. Feb. 2011, pp. 50–55. DOI: 10.1109/NFC.2011.18.

[10] Luís Carlos Silva Veiga Martins. "Informática na manutenção de edifícios : utilização de sistemas de identificação por RFID". In: (2011). URL: https://repositorio-aberto.up.pt/handle/10216/61849 (visited on 03/22/2023).

[11] NFC Forum. *Type 2 Tag Operation*. 2011.

[12] NFC Forum. *Type 3 Tag Operation*. 2011.

[13] NFC Forum. *Type 4 Tag Operation*. 2011.

[14] Michael Roland, Josef Langer, and Josef Scharinger. "Security Vulnerabilities of the NDEF Signature Record Type". In: *2011 Third International Workshop on Near Field Communication*. 2011 Third International Workshop on Near Field Communication. Feb. 2011, pp. 65–70. DOI: 10.1109/NFC.2011.9.

[15] Naser Hossein Motlagh. "Near Field Communication (NFC) - A Technical Overview". May 28, 2012. DOI: 10.13140/RG.2.1.1232.0720.

[16] Muhammad Qasim Saeed and Colin D. Walter. "Off-Line NFC Tag Authentication". In: *2012 International Conference for Internet Technology and Secured Transactions*. 2012 International Conference for Internet Technology and Secured Transactions. Dec. 2012, pp. 730–735.

[17] Carlos Tiago Rocha Babo. "Generic and Parameterizable Service for Remote Configuration of Mobile Phones Using Near Field Communication". 2013.

[18] Tom Igoe, Don Coleman, and Brian Jepson. *Beginning NFC: Near Field Communication with Arduino, Android, and Phonegap*. First edition. Beijing: O'Reilly, 2014. 233 pp. ISBN: 978-1-4493-7206-4.

[19] Sufian Hameed, Usman Murad Jamali, and Adnan Samad. "Integrity Protection of NDEF Message with Flexible and Enhanced NFC Signature Records". In: *2015 IEEE Trustcom/BigDataSE/ISPA*. 2015 IEEE Trustcom/BigDataSE/ISPA. Vol. 1. Aug. 2015, pp. 368–375. DOI: 10.1109/Trustcom.2015.396.

[20] "NTAG213/215/216 NFC Forum Type 2 Tag Compliant IC with 144/504/888 Bytes User Memory". In: 2015 (2015).

[21] Anne-Marie Lesas. *The Art and Science of NFC Programming*. Hoboken, NJ: ISTE Ltd/John Wiley and Sons Inc, 2016. ISBN: 978-1-78630-057-7.

[22] Carlos Bermejo and Pan Hui. *Steal Your Life Using 5 Cents: Hacking Android Smartphones with NFC Tags*. Comment: 7 pages, 5 figures. May 5, 2017. arXiv: 1705.02081 [cs]. URL: http://arxiv.org/abs/1705.02081 (visited on 01/21/2023). preprint.

[23] Manmeet (Mandy) Mahinderjit Singh, Ku Adzman, and Rohail Hassan. "Near Field Communication (NFC) Technology Security Vulnerabilities and Countermeasures". In: *International Journal of Engineering and Technology* 7 (Dec. 9, 2018), pp. 298–305. DOI: 10.14419/ijet.v7i4.31.23384.

[24] *Learn More About App Clips*. Apple Support. Mar. 24, 2021. URL: https://support.apple.com/en-us/HT212238 (visited on 04/10/2023).

[25] NXP Semiconductors. *ICODE SLIX2 - Datasheet*. Dec. 1, 2021.

[26] Arundale Ramanathan. *Unishox: A Hybrid Encoder for Compressing Short Unicode Strings*. Version 1.0.2. Nov. 2021. DOI: 10.5281/zenodo.5573864. URL: https://github.com/siara-cc/Unishox2 (visited on 03/31/2023).

[27] Sergio Rios-Aguilar, Marta Beltrán, and González-Crespo Rubén. "Security Threats to Business Information Systems Using NFC Read/Write Mode". In: *Computers, Materials & Continua* 67.3 (2021), pp. 2955–2969. ISSN: 1546-2226. DOI: 10.32604/cmc.2021.014969. URL: https://www.techscience.com/cmc/v67n3/41595 (visited on 01/24/2023).

[28] Salvatore Sanfilippo. *Antirez/Smaz*. Mar. 24, 2023. URL: https://github.com/antirez/smaz (visited on 03/31/2023).

[29] *Create App Links for Instant Apps*. Android Developers. URL: https://developer.android.com/training/app-links/instant-app-links (visited on 06/14/2023).

[30] *Direct Links on NFC Tags Does Not Work to Open Instant App. Using App Store Link Is Not Sufficient. [113304120] - Visible to Public - Issue Tracker*. URL: https://issuetracker.google.com/u/1/issues/113304120 (visited on 04/13/2023).

[31] GNU Project. *Gzip - GNU Project - Free Software Foundation*. URL: https://www.gnu.org/software/gzip/ (visited on 03/31/2023).

[32] *Google Play Instant*. Android Developers. URL: https://developer.android.com/topic/google-play-instant (visited on 04/10/2023).

[33] Ilan Kirschenbaum and Avishai Wool. *How to Build a Low-Cost, Extended-Range RFID Skimmer*. URL: https://eprint.iacr.org/2006/054.pdf (visited on 01/21/2023). preprint.

[34] Kotlin. *Kotlin Multiplatform | Kotlin*. URL: https://kotlinlang.org/docsmultiplatform.html (visited on 10/20/2022).

[35] *Linking to Google Play*. Android Developers. URL: https://developer.android.com/distribute/marketing-tools/linking-to-google-play (visited on 04/13/2023).

[36] *MifareUltralight*. Android Developers. URL: https://developer.android.com/reference/android/nfc/tech/MifareUltralight (visited on 05/19/2023).

[37] NXP Semiconductors. *NTAG 424 DNA | 424 DNA TagTamper – Advanced Security and Privacy for Trusted IoT Applications | NXP Semiconductors*. URL: https://www.nxp.com/products/rfid-nfc/nfc-hf/ntag-for-tags-labels/ntag-424-dna-424-dna-tagtamper-advanced-security-and-privacy-for-trusted-iot-applications:NTAG424DNA (visited on 01/23/2023).

[38] *sendMiFareISO7816Command(_:completionHandler:)* Apple Developer Documentation. URL: https://developer.apple.com/documentation/corenfc/nfcmifaretag/3153114-sendmifareiso7816command (visited on 05/25/2023).

[39] *Testing the Launch Experience of Your App Clip*. Apple Developer Documentation. URL: https://developer.apple.com/documentation/app_clips/testing_the_launch_experience_of_your_app_clip (visited on 05/05/2023).