Journal of Surveillance, Security and Safety

Original Article

Open Access

Forge-resistant radio-frequency identification tags for secure internet of things applications

Luca Calderoni[1], **Dario Maio**[1], **Luciano Margara**[1], **Luca Spadazzi**[2]

[1]Department of Computer Science and Engineering, University of Bologna, Cesena 47522, Italy.
[2]Lab51 srl, Cesena 47522, Italy.

Correspondence to: Prof. Luca Calderoni, Department of Computer Science and Engineering, University of Bologna, via dell'Università, 50, Cesena 47522, Italy. E-mail: luca.calderoni@unibo.it

How to cite this article: Calderoni L, Maio D, Margara L, Spadazzi L. Forge-resistant radio-frequency identification tags for secure internet of things applications. *J Surveill Secur Saf* 2020;1:106-18. http://dx.doi.org/10.20517/jsss.2019.01

Received: 13 Dec 2019 **First Decision**: 1 Feb 2020 **Revised**: 10 Feb 2020 **Accepted**: 31 Mar 2020 **Available online**: 29 Oct 2020

Academic Editor: Michael G. Pecht **Copy Editor**: Jing-Wen Zhang **Production Editor**: Jing Yu

Abstract

Aim: Internet of Things (IoT) represents a key aspect within several application domains, and it enables growing opportunities for both organizations and end-users. Radio-frequency identification tags are probably the most relevant enabling solution for ubiquitous IoT systems and are often seen as a prerequisite for IoT itself. In this study, we analyzed one of the most promising radio-frequency identification tags to determine whether or not it represents a viable solution for secure IoT applications.

Methods: The study was conducted relying on an Android OS application developed within our laboratories, which helped us to inspect the chip and describe its logical data structure. We studied the capabilities of the tag in relation to the application protocol data unit it supports, and we described the cryptographic protocols with which it is equipped.

Results: This tag is resistant to forging activities, and it also preserves confidentiality and authenticity on exchanged data. We discussed several known privacy and security patterns that may be addressed relying on the tag we focused on and we underlined some deficiencies concerning chip cloning attack. Again, secure dynamic messaging and mirroring allow the surpassing of several privacy limitations.

Conclusion: In this paper we investigated the capabilities of the *NT4H2421Gx* tag. The deep Android inspection performed on the tag showed that it represents an option to rely on when we need to design secure IoT applications.

Keywords: Radio-frequency identification, NFC, internet of things, cryptographic protocols

1 INTRODUCTION

Internet of Things (IoT) has exploded in recent years, and the related security aspects are increasingly relevant[1,2]. Radio-frequency identification (RFID) represents the most adopted solution within the IoT domain[3,4]. The logistics industry is one of the earliest adopters of IoT and RFID solutions[5], while these technologies are now used in several application contexts, such as military and defense applications, supply chains, food industry and so forth. As an example, RFID tags may be applied to manage inventories, to reduce overstocks and to avoid understocks as well as to track the overall lifecycle of a product[6].

More recently, RFID tags have also been used for different applications, such as localization and personal identification. For example, electronic machine readable travel documents are equipped with RFID tags[7]. As this feature enables several cryptographic protocols to be applied during the communication between the tag and the reader, it also makes it possible to deliver automated border controls in crucial areas such as international airports. At the same time, localization and identification procedures based on RFID also imply privacy and traceability issues for the tag bearer[8-10].

Thus, the combination of RFID and cryptography is widely studied[11-14], and paving the way for a number of pervasive and secure applications. Among them, those aimed at preventing forgery and counterfeiting of trademark products represent a significant slice of the application sector. In recent years, the scientific community has therefore dedicated significant efforts to the design of techniques aimed to prevent malicious attacks against RFID technology[15-18]. Consequently, several efficient cryptographic protocols were proposed to deliver high-quality protection mechanisms for RFID-based applications.

The RFID industry tries to adapt its products so they can fit this rapid evolution and continues to produce new tags with smarter capabilities. Each RFID tag has different features, including the supported cryptographic protocols, the amount of data that it is able to store, the set of commands it can deal with and so forth. The design of a secure IoT application relying on RFID technology should be thus preceded by an in-depth study of tag capabilities. In this study, we focused on *NT4H2421Gx*[19], a recent RFID tag released by *NXP Semiconductors*, and we investigated its features extensively. The results showed that *NT4H2421Gx* represents a valid and promising solution for a wide number of secure IoT applications.

2 METHODS

In this section, we described the features of *NT4H2421Gx*. After a brief introduction to the general specifications of the tag, we investigated in depth its logical data structure, its application protocol data unit (APDU) and its core functionalities. Finally, we proposed a high-level comparison between this tag and other related ones.

The NXP's *NT4H2421Gx* tag is fully compliant with the *NFC Forum Type 4 IC* specification and relies on the *ISO/IEC 14443-4* contactless proximity protocol. The file system is compliant with *ISO/IEC 7816-4*[20]. The APDU is based on *ISO/IEC 7816-4* as well, while it preserves only three of the native commands. Each command included in the command set is tag specific.

2.1 Hardware layer

Contactless smart cards with microprocessors incorporate their own operating system, which is usually burned into the ROM module at the production stage. The tasks of the operating system are data transfer from and to the smart card, command sequence control, APDU interpretation, file management and

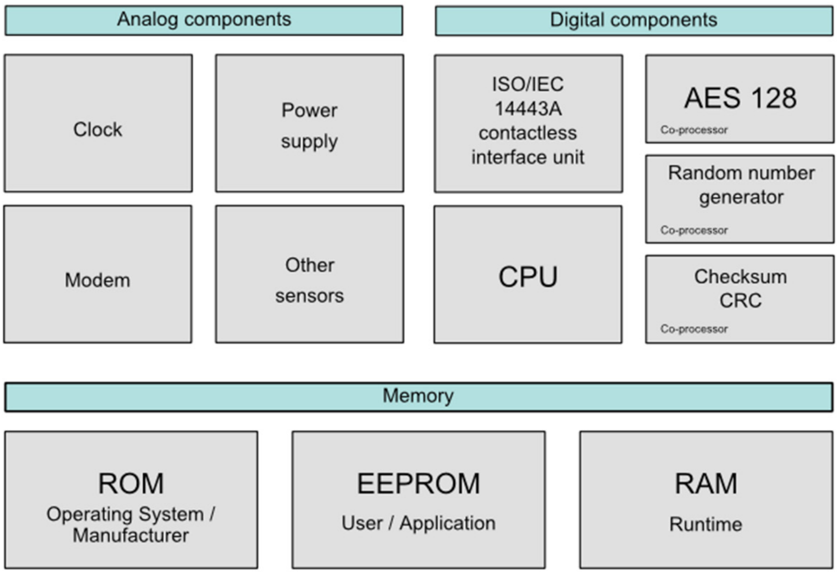


Figure 1. A high-level hardware block diagram of the *NT4H2421Gx* tag

cryptographic algorithm execution (e.g., encryption, authentication)[21]. Concerning *NT4H2421Gx*, a high-level block diagram depicting its hardware components is provided in Figure 1.

Usually, a command processing sequence within a smart card operating system undergoes the following flow. At the *physical layer*, commands sent from the reader to the tag are received through the radio frequency interface, according to *ISO/IEC 14443-2A*. The packets are processed at the *transport layer* according to *ISO/IEC 14443-3A*: error detection and correction are performed by the I/O manager, which relies on the CRC co-processor. If the packet is deemed correct, its payload is extracted and processed at the *application layer*, relying on *ISO/IEC 7816-4* or proprietary APDU commands. When secure messaging applies, the payload is decrypted or checked for integrity. These procedures are enhanced by the AES and RNG co-processors. When the APDU manager is not able to recognize the command, the return code manager generates the appropriate return code and sends it back to the reader. Conversely, if a valid command is received, the system executes the instructions which correspond to the command code, according to the APDU. When the command implies some access to the EEPROM, this is performed exclusively by the file management system and the memory manager, which convert all symbolic addresses into the corresponding physical addresses of the memory area. The file manager is also responsible for verification of access conditions, depending on the addressed data.

2.2 Logical data structure

Concerning the file system, *NT4H2421Gx* complies with *ISO/IEC 7816-4*. Specifically, it is equipped with a master file (MF), a dedicated file (DF) and three elementary files (EF). The logical data structure mounted on the tag we focused on is depicted in Figure 2.

The first file is also known as the *capability container* (CC) file and it is formatted in accordance with *NFC Forum* specifications[22]. This file specifies the mapping version and the maximum size of command APDU and response APDU data size. Moreover, this file contains some metadata concerning the other two files included in the user memory. For each of them, this file specifies the name of the file, the overall byte size and the access conditions which need to be met to access the file. The "Results" section provides a deep look at the CC file.

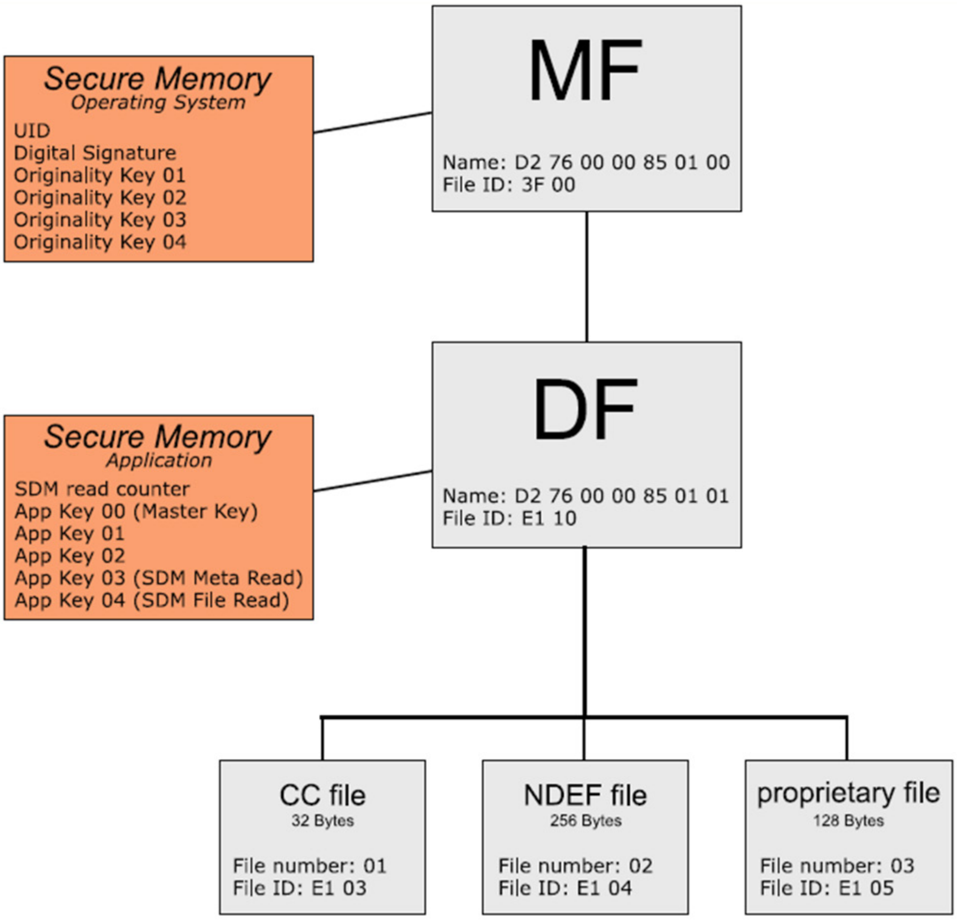


Figure 2. The file system mounted in the user memory. The three *elementary files* listed under the *dedicated file* are *standard data files*, according to *ISO/IEC 7816-4*. MF: master file; DF: dedicated file; CC: capability container; SDM: secure dynamic messaging; UID: unique tag identifier; NDEF: NFC data exchange format; NFC: near field communication

The second file is also known as the *NDEF* file and contains an NDEF-formatted message[23]. NDEF is a lightweight, binary message format that can be used to encapsulate one or more application-defined payloads of arbitrary type and size into a single message construct. Each payload is composed of a type, length, and optional *id*. Just as an example, identifiers may be represented by URIs, MIME media types, or other NFC-specific types. This file is also designed to support *secure dynamic messaging* (SDM) and *data mirroring*. These options extend the security and privacy features offered by this tag and will be discussed in the next sections.

The third file is a proprietary NXP file which is read- and write-protected and contains raw data. At the production stage, access to this file is restricted using two different application keys, one for reading operations and one for writing operations. This condition is better exemplified in the "Results" section.

The RFID device also includes nine cryptographic keys, designed to be used as advanced encryption standard (AES) keys[24]. Four keys are provided at the tag level (MF). They are also referred to as *originality keys*. The other five keys are instead included at the application level (DF) and are referred to as *application keys*. Originality keys are stored in ROM and may never be removed or updated after chip production. Conversely, application keys are part of the user memory (EEPROM) and may be updated to customize the tag for application-specific scenarios. Each of these nine keys may be used to perform an authentication procedure between the tag and a reader. Moreover, to update any of the app keys, a successful

Table 1. AES keys installed on NT4H2421Gx tag

Key	Length	Location	Key n	Update	Authentication	Notes
Originality key 1	128 bits	ROM	0x01	×	√	
Originality key 2	128 bits	ROM	0x02	×	√	
Originality key 3	128 bits	ROM	0x03	×	√	
Originality key 4	128 bits	ROM	0x04	×	√	
Application key 1	128 bits	EEPROM	0x00	√	√	App master key
Application key 2	128 bits	EEPROM	0x01	√	√	
Application key 3	128 bits	EEPROM	0x02	√	√	
Application key 4	128 bits	EEPROM	0x03	√	√	SDM meta read
Application key 5	128 bits	EEPROM	0x04	√	√	SDM file read

While App Master Key is always identified by code 0x00 at the dedicated file level, SDM-related keys may be identified by each of the application keys (i.e., it is not mandatory to use key 0x03 and 0x04 as reported in this table). SDM: secure dynamic messaging

authentication through the first application key is required. This key is also referred to as *App Master Key*. A complete list of the aforementioned keys is provided in Table 1.

Finally, it is important to point out that the tag ROM also contains the *unique tag identifier* (UID), composed of 7 bytes, and a 56-byte *digital signature*, which was computed by NXP at the production stage and burned in the memory. This digital signature lays at the basis of the strong anti-forging functionalities provided by the *NT4H2421Gx* tag and will be discussed in the next section.

2.3 Application protocol data unit

An APDU consists of the instruction set used by the reader and the tag during communication. Each procedure that is performed during communication relies on a combination of APDU commands. APDU instructions are divided into *command APDUs* and *response APDUs*. The former ones are sent by the reader to the tag while the latter are sent back by the tag to the reader.

NT4H2421Gx APDU is based on the *ISO/IEC 7816-4* standard. However, the majority of available commands are proprietary and are programmed through original *ISO/IEC 7816-4* command wrapping. Specifically, only three of the native commands are preserved.

The complete *NT4H2421Gx* command set is provided in Table 2. Please note that some of the listed commands are composed of more than one part. For instance, the *GetVersion* command is divided into *GetVersion part1*, *GetVersion part2* and *GetVersion part3*. These details do not add much to the discussion on the subject and are therefore omitted for brevity.

2.4 Comparison

NT4H2421Gx is a robust and versatile tag and provides a wide range of desirable features within the IoT domain. As summed up in Table 3, this tag was introduced by NXP to surpass several limitations that afflicted tags belonging to older generations. NTAG is the market-leading portfolio of NFC tag solutions for the consumer and industrial segments of IoT. These tags offer different levels of security and different functionalities as well, to address a wide range of applications.

NT4H2421Gx supports NDEF-formatted messages to be stored in the user memory. NDEF records may be combined with UID mirroring, UID randomization and SDM to cover a broad range of user requirements, including privacy preservation. Thanks to several co-processors, this tag also provides authentication functionalities and secure messaging. Both of them rely on AES-128 cryptography. Memory access is subject to a mixture of user-driven and manufacturer-driven permissions and relies on AES-128 authentication as well. Forging attempts are averted by the manufacturer's digital signature (56 bytes), which is computed against the UID at the production stage and is embedded into the tag.

Table 2. NT4H2421Gx command set

Category	Command	Class	Description
Basic r/w functionalities	ISOSelectFile	ISO/IEC 7816-4	Select MF, DF or EF
	ISOReadBinary	ISO/IEC 7816-4	Read data from a data file (EF)
	ISOReadBinary	ISO/IEC 7816-4	Write data to a data file (EF)
	ReadData	Proprietary	Read data from a data file (EF)
	WriteData	Proprietary	Write data to a data file (EF)
Authentication	AuthenticateEV2First	Proprietary	Perform AES three-pass authentication
	AuthenticateEV2NonFirst	Proprietary	Perform AES three-pass authentication
	AuthenticateLRPFirst	Proprietary Proprietary	Perform LRP three-pass authentication
	AuthenticateLRPNonFirst		Perform LRP three-pass authentication
Key management	GetKeyVersion	Proprietary	Get version of the specified key
	ChangeKey	Proprietary	Update key, version and reset counters
Digital signature	Read_Sig	Proprietary	Get the tag digital signature
Metadata management	GetVersion	Proprietary	Get tag metadata (UID, producer)
	GetCardUID	Proprietary	Get the unique 7-byte tag UID
	GetFileCounters	Proprietary	Get the SDM read counter
	GetFileSettings	Proprietary	Get file metadata (access rights, SDM)
	ChangeFileSettings	Proprietary	Set file metadata (access rights, SDM)
	SetConfiguration	Proprietary	Set tag mode (LRP, random ID)

MF: master file; DF: dedicated file; EF: elementary file; LRP: leakage-resilient primitive; SDM: secure dynamic messaging; UID: unique tag identifier; AES: advanced encryption standard

Table 3. Comparison of three NXP tags designed for the IoT domain

Tag type	NDEF	Secure messaging	SDM	Random ID	Digital Sig.	Authentication	Memory access protection
NT4H2421Gx	√	√	√	√	√	√	√
NTAG21x	√	×	×	×	√	×	√
NTAG210×	×	×	×	×	√	×	×

IoT: internet of things; SDM: secure dynamic messaging

NTAG21x is protected by the same digital signature principle, while it relies on a different, weaker elliptic curve, which produces a 32-byte signature. NDEF and memory access protection are provided as well, while, for the latter, access is granted on a 32-bit password basis instead of the more reliable AES-128 authentication. The other features are not provided by this tag.

Concerning the last type, *NTAG210μ* does not provide any of the listed features, apart from the 32-byte digital signature.

Finally, none of the tags provides strong protection against chip cloning attacks. Concerning *NT4H2421Gx*, while a cloning attempt is not straightforward, since it implies that the malicious party needs to learn the AES originality keys, it is not impossible. Further considerations on the subject are provided in the "Discussion" section.

3 RESULTS

To effectively check the tag properties and some of its core functionalities, we designed a mobile application on the basis of Android OS, which uses the NFC sensor of the smartphone as a tag reader. The customized *NT4H2421Gx* tag was provided by *lab51 srl*.

In this section, we exemplified some of the APDU commands executed by the mobile application, and we stressed the digital signature verification process, as it represents the more reliable feature in relation to anti-forging. In the following, the content of each command and each response is proposed in hexadecimal format.

First of all, DF was selected through the standard *ISOSelectFile* command (see Table 2 for reference). Subsequently, the *GetVersion* command was addressed to acquire some basic information on the tag

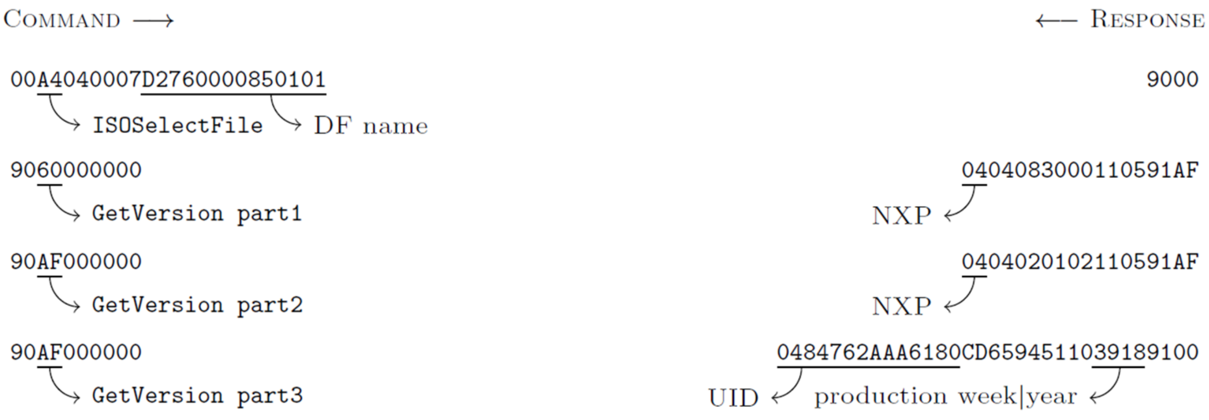


Figure 3. The complete communication trace concerning the *GetVerison* command. UID: unique tag identifier; DF: dedicated file

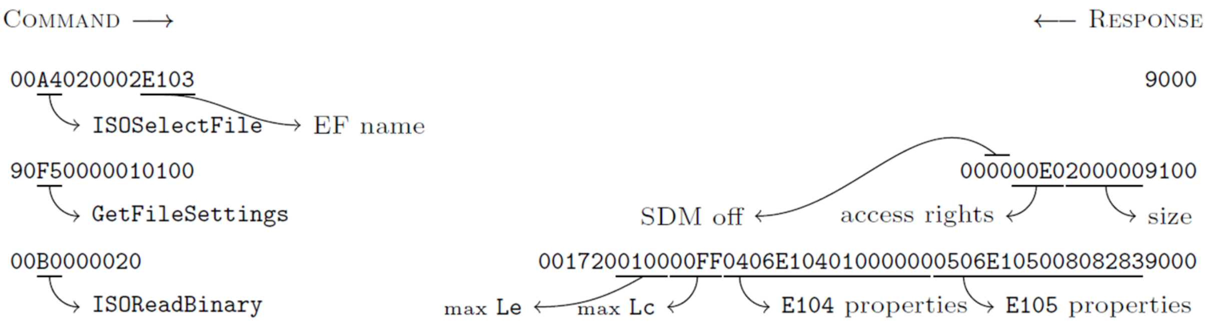


Figure 4. The complete communication trace concerning the commands performed against the *capability container* file. EF: elementary file; SDM: secure dynamic messaging

studied. The complete communication trace is provided in Figure 3. According to the returned data, the tag was produced during the 39th week of 2018 by NXP. The most important information included in the answer is the tag ID: as the tag studied is not configured with the *random ID* setting, the third response includes the real 7-byte UID. This condition may lead to a privacy breach and will be further discussed in the "Discussion" section.

The following step consists in the selection of the CC file. The application checks the file settings through the *GetFileSettings* command and subsequently reads the full file content using the standard *ISOReadBinary* command. The communication trace involved is provided in Figure 4. The information returned by the *GetFileSettings* command shows that the SDM is not enabled for this file. Again, the CC file has a size of *20:00:00*, which means it is composed of 32 bytes, as it should be interpreted with least significant byte encoding. Concerning the access rights to the file, the response shows that the *E103* file is subject to the *00:E0* access policy. According to NXP specifications, it means that this file is free to read (*E*), while other operations (write and change file permissions) need to be preceded by authentication through the key number *0x00* (the App Master Key). *ISOReadBinary* asks the tag for 32 bytes from the aforementioned file. The answer states that the CC effectively occupies 23 bytes only (*00:17*). Here, we may see that the file system comprises two more files, named *E104* and *E105*. The first one occupies 256 bytes and may be read and written without any authentication (*00:00*). Note that this access notation differs from the one returned by the *GetFileSettings* command as it is intended to be in accordance with the NFC Forum specifications. The latter file occupies 128 bytes. The access conditions for this file are set to *82:83*. These numbers fall in the proprietary range, concerning NFC Forum access policies. Specifically, it means that read operations need to be preceded by authentication with the application key number *0x02*. The same applies to write operations, with key number *0x03*.

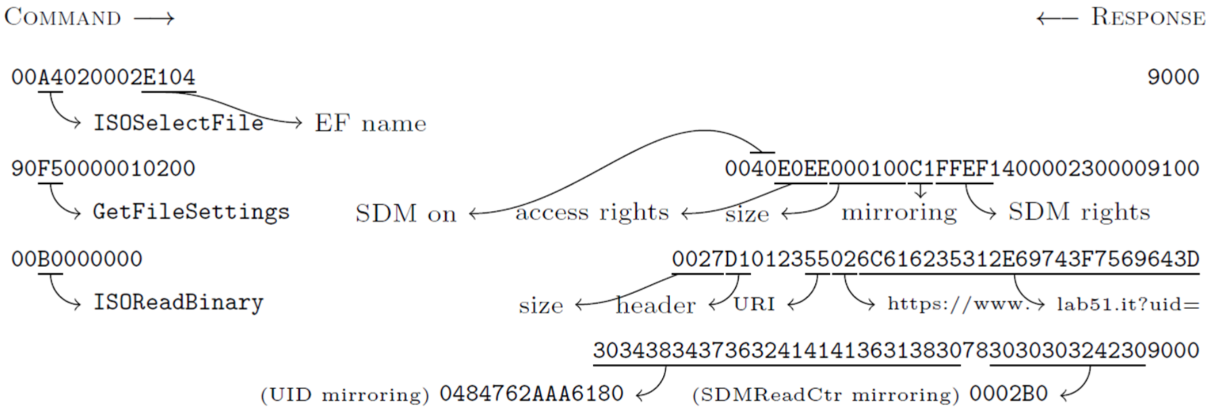


Figure 5. The complete communication trace concerning the commands performed against the NDEF file. EF: elementary file; SDM: secure dynamic messaging; UID: unique tag identifier

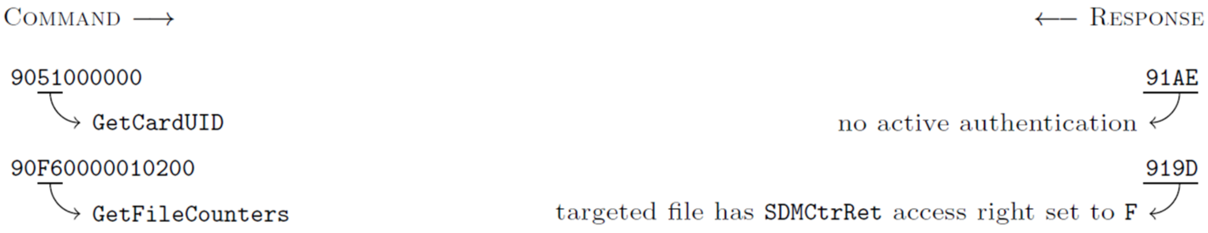
The next step consists of the inspection of the NDEF file. After the file selection, the application checks the file settings through the *GetFileSettings* command and, subsequently, reads the full file content using the standard *ISOReadBinary* command. The complete communication trace is provided in Figure 5. The information returned by the *GetFileSettings* command shows that, differently from the CC file, SDM is enabled for this file. Specifically, the file metadata shows that two attributes are supposed to be mirrored inside the NDEF file: the tag UID, stored at offset *14:00:00* (i.e., 20 following the decimal notation) and the SDM read counter, stored at offset *23:00:00* (i.e., 35 following the decimal notation). Both of them are stored in ASCII encoding. SDM access rights are set to *FF:EF*; this means that the UID and the *SDMReadCtr* are stored as plaintext within the NDEF file. Moreover, no run time encryption is applied to these data when the NDEF file is read through the *ISOReadBinary* or *ReadData* commands. Again, the *GetFileCounters* command is disabled. Moreover, metadata indicate that the overall dimension of the NDEF file is 256 bytes (*00:01:00*), and the access conditions are set to *E0:EE*. This access policy reflects the one included in the CC file for the NDEF file, as it states that the file may be updated and read with no restrictions (*E*). This setting suggests that the default file access rights were not changed after chip personalization.

Concerning the file content, the file effectively occupies 39 bytes (*00:27*). The file stores a single NDEF record having header *D1*. Hence, this record is a *short record* of a *well-known type*. The specific type is a URI (*55*) and the payload length is 35 (*23*). The first byte of the URI is *02*, which is an abbreviation for "*https://www.*". The remaining bytes contain the rest of the URI and the mirrored UID and *SDMReadCtr*, stored in ASCII encoding, as depicted in Figure 5.

To check the correctness of the APDU implementation in relation to the tag access logic, we also tested two more commands: *GetCardUID* and *GetFileCounters*. Both commands correctly return an error code. In the first case, this is due to the fact that the command was executed when the tag and the reader were not under authenticated mode. The error returned by the latter is instead related to the *SDM access rights* reported in Figure 5: as the *SDMCtrRet* is set to *F*, the *GetFileCounters* command is disabled. The error codes are provided in Figure 6.

Finally, we run the *Read_Sig* command to verify the digital signature and to prove the compliance of this tag with respect to chip forging. The related communication trace is listed in Figure 7.

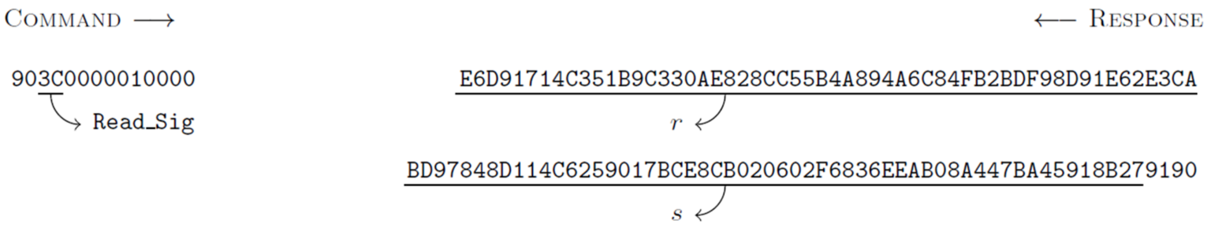
According to NXP, the digital signature relies on elliptic curve cryptography[25] and was produced for the tag UID using an *ECDSA* algorithm with the elliptic curve *secp224r1*. As the name suggests, this curve implies keys of 224 bits (i.e., 28 bytes). Thus, the digital signature is composed of two parts: the first part is


COMMAND ⟶ ⟵ RESPONSE

9051000000 → GetCardUID | 91AE → no active authentication

90F60000010200 → GetFileCounters | 919D → targeted file has SDMCtrRet access right set to F

Figure 6. The complete communication trace concerning the *GetCardUID* and *GetFileCounters* commands

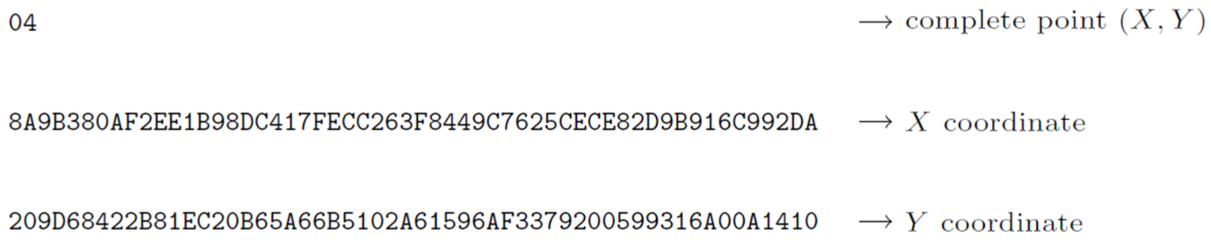

COMMAND ⟶ ⟵ RESPONSE

903C0000010000 → Read_Sig

E6D91714C351B9C330AE828CC55B4A894A6C84FB2BDF98D91E62E3CA → r

BD97848D114C6259017BCE8CB020602F6836EEAB08A447BA45918B279190 → s

Figure 7. The complete communication trace concerning the *Read_Sig* command


04 ⟶ complete point (X, Y)

8A9B380AF2EE1B98DC417FECC263F8449C7625CECE82D9B916C992DA ⟶ X coordinate

209D68422B81EC20B65A66B5102A61596AF3379200599316A00A1410 ⟶ Y coordinate

Figure 8. NXP public key for the elliptic curve *secp224r1*

28 bytes long and refers to the *r* parameter, and the second part is 28 bytes long as well and refers to the *s* parameter. The corresponding *public key* which should be used to verify the digital signature is provided by NXP and includes the *X* and *Y* coordinates of a point on the curve, plus an additional control byte[26]. The public key is provided in Figure 8.

The verification procedure was written within the Android application relying on the *Bouncy Castle Cryptographic Library* (https://www.bouncycastle.org).

To correctly test the digital signature, the raw bytes returned by the *Read_Sig* command need to be encoded in DER; otherwise, they cannot be handled by the java library used to operate the verification.

The verification procedure may be summed up as follows:
1. add the *Bouncy Castle* security provider;
2. create an empty data structure based on the *secp224r1* curve;
3. load the elliptic curve point from the raw bytes containing the NXP public key;
4. generate the elliptic curve public key accordingly;
5. prepare a *Signature* object with the aforementioned public key;
6. set the message to be verified as the tag UID;
7. encode the tag digital signature with DER encoding;
8. perform the digital signature verification on the DER-encoded signature.

The Android algorithm correctly verifies the digital signature. The originality check based on strong asymmetric cryptography is thus passed.

4 DISCUSSION

In this section, we discuss some notable security and privacy patterns that may be addressed using the *NT4H2421Gx* tag.

4.1 Communication channel security

The most commonly known security functionalities are based on three-pass mutual authentication and rely on AES symmetric cryptography. The authentication procedure is initiated by the *AuthenticateEV2First* or *AuthenticateLRPFirst* command.

When the reader and the tag are in the authenticated state, they are able to communicate using each command included in the command set. Performing a successful authentication proves that the reader possesses one of the cryptographic keys listed in Table 1. In authenticated mode, each APDU command is protected by *secure messaging*. Thus, message payloads are encrypted using the adopted AES key, and a *message authentication code* is attached as well. It follows that the communication channel is secured with respect to sniffing/eavesdropping attacks. Three-pass mutual authentication and secure messaging ensure confidentiality, integrity and trust. Of course, as they rely on symmetric AES cryptography, they suffer the key distribution problem, which is notably relevant within this field[27]. Some strategies should be adopted to provide the readers with one or more AES keys.

Finally, the *SetConfiguration* command may be used to enable the leakage-resilient primitive (LRP) mode (note that it is not possible to revert the tag to simple AES mode). Under LRP mode, three-pass authentication is started by the *AuthenticateLRPFirst* command and may rely on *originality keys* as well. LRP mode relies on a slightly different AES algorithm which is designed to resist side-channel attacks. An in-depth discussion on this subject falls out of the scope of this work.

4.2 Privacy implications

The GDPR specifically includes the term *online identifiers* within the definition of what constitutes personal data. These objects may include information relating to the device that an individual is using, such as applications, tools or protocols. To this end, the GDPR *Recital 30* shows a shortlist as an example and explicitly includes *RFID tags*. To comply with the latest privacy requirements, a good tag should thus be allowed to hide its UID under specific circumstances, since this UID may be sniffed out by unauthorized readers, threatening the user's privacy.

The *random ID* feature provided by *NT4H2421Gx* implements this requirement. This setting may be triggered through the *SetConfiguration* command, and prevent the UID to be unveiled through the *GetVersion* command. Specifically, when the tag is in *random ID* mode, a 4-byte random ID substitutes the 7-byte UID within the *GetVersion* response. There are two more options to learn the tag UID: using the *GetCardUID* command or reading it out from the NDEF file when UID mirroring is active. The first option does not represent a privacy breach, as the *GetCardUID* command is not allowed when the reader is not authenticated (see Figure 6 for reference). Concerning the latter option, it should be pointed out that UID mirroring is not mandatory, and moreover, the mirrored UID may be stored as ciphertext within the NDEF file. While the examined tag mirrors the UID as plaintext (see Figure 5 for reference), a proper change in the NDEF file settings (through the *ChangeFileSettings*) would encrypt the UID.

4.3 Chip cloning

The ability of a malicious stakeholder to clone the tag is probably the most dangerous event concerning *NT4H2421Gx*. The only countermeasure provided by the tag is represented by the inability of the attacker to copy the four *originality keys* which are stored in the ROM. These keys may be used accessing the MF level to perform a successful authentication, proving the tag originality. Unfortunately, it is evident from

NXP documentation that these symmetric AES keys are sometimes shared with NXP's licensees to check if the tags are genuine[26]. This information could be maliciously used to produce a complete clone of a genuine NXP tag.

To this end, please note that the tag UID and the corresponding NXP digital signature may be acquired through a legitimate tag inspection (as described in the "Results" section) and copied to the cloned tag as well.

To overcome this issue, further security protocols should be adopted. A significant example is represented by electronic passports[15]. The guidelines for e-Passport issuance and management are provided by the International Civil Aviation Organization (ICAO), and include a detailed description of the security protocols and the logical data structure used to store and arrange data into the RFID chip. To prevent chip cloning attacks, ICAO designed the *Active Authentication* security protocol. This protocol relies on asymmetric cryptography and requires a dedicated key pair. Briefly, during the chip's customization phase, the secret key is stored in the chip's secure memory, while the public key is stored in one of the chip's elementary files. When the reader needs to check whether or not the chip is genuine, it sends a random nonce to the chip, which signs it using the private key as signing key, according to the adopted cypher. The reader then reads the chip's public key from the corresponding EF and decrypts the string. On a positive match, the protocol succeeds. As the private key is stored in the chip's secure memory, it is very hard to read for an attacker. Moreover, as the protocol relies on asymmetric cryptography, there is no need for the licensees to handle the private key. This missing piece (the private key) and the introduced protocol represent a strong defense against chip cloning attacks. A similar solution could be adopted to strengthen the security features of *NT4H2421Gx*.

4.4 Tag forging

When we talk about tag forging, we refer to the ability of an attacker to produce a new tag from scratch claiming that it is genuine and that it is produced by some trusted organization (such as NXP). This procedure differs from the cloning one, as in this case the attacker does not copy the same tag UID in the forged chip, where the aim is to couple the tag with a new different UID.

The deep inspection performed on the *NT4H2421Gx* tag proved that this technology is strongly resistant with respect to forging activities. In fact, the *Read_Sig* command provides the reader with a digital signature which was computed signing the tag UID with an NXP elliptic curve *private key* (see Figure 7 for reference). Hence, to forge the tag, the attacker should sign the new UID with the same private key and should store the resulting signature in the tag ROM. Differently from symmetric AES keys, this private key never leaves the NXP hardware security module. As such, to forge a genuine NXP chip, the attacker must be able to break strong asymmetric encryption (which is usually deemed impossible under reasonable settings).

4.5 Soft security settings

To facilitate user experience and tag interoperability, this tag also supports a soft security setting named SDM. This feature may be set up for a single file (namely the NDEF one) through the *ChangeFileSettings* command. Besides, as depicted in Figure 5, SDM is enabled in the tag studied. SDM allows for confidential and integrity-protected data exchange, without requiring a preceding authentication. The NDEF file content may be accessed without any authentication. Encrypting part of the file content (together with tag UID or *SDMReadCtr*) is a valid option to reach the maximum interoperability with any RFID/NFC reader, while preserving some form of security. As predictable, when the involved application context requires strong security settings, SDM should not be considered a valid option.

This work could be extended according to several directions. From a theoretical point of view, a formal validation of the experimental results presented in this article would be an interesting open issue. Furthermore, a future research direction could involve further investigation of which countermeasures may be set up in this chip to handle chip cloning attacks better. Following the ICAO principles designed for electronic machine readable travel documents, a viable solution could consist of a novel protocol relying on asymmetric cryptography. Furthermore, this tag supports notable features that enhance privacy and also implement soft security settings, which increase tag interoperability. From a practical and application point of view, a good option could be to design and implement stateless systems (from the user's perspective) able to preserve some form of security and confidentiality while enabling tag inspection. Such a system could rely on smartphones NFC sensors and should be independent of a dedicated end-user application on the smartphone itself. This setting should exploit the SDM feature provided by the tag.

In a conclusion, in this paper we investigated the capabilities of the *NT4H2421Gx* tag. To effectively check the tag properties and some of its core functionalities, we designed a mobile application based on *Android OS* which uses the NFC sensor of the smartphone as a tag reader. This application allowed us to read the memory of the aforementioned chip at the bit level, and to discuss its core functionalities and settings in relation to the most common security and privacy patterns. In the final part of the paper we considered each of these aspects separately to stimulate the research community regarding these topics. Concluding, the deep Android inspection performed on the *NT4H2421Gx* tag showed that it represents an option to rely on when we need to design secure IoT applications. This tag is resistant to forging activities, and it also preserves confidentiality and authenticity on exchanged data. Again, SDM and mirroring enable stateless applications (from the user's perspective) to be delivered and also allow the surpassing of several privacy limitations.

DECLARATIONS

Authors' contributions

Made substantial contributions to conception and design of the study and performed data analysis and interpretation: Calderoni L
Provided technical and material support: Spadazzi L
Supervised the work: Maio D, Margara L

Availability of data and materials

Not applicable.

Financial support and sponsorship

None.

Conflicts of interest

All authors declared that there are no conflicts of interest.

Ethical approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Copyright

REFERENCES

1. Conti M, Dehghantanha A, Franke K, Watson S. Internet of things security and forensics: challenges and opportunities. Future Generation Comp Syst 2018;78:544-6.
2. Palmieri P, Calderoni L, Maio D. Private inter-network routing for wireless sensor networks and the internet of things. Proceedings of the Computing Frontiers Conference (CF'17). ACM, New York, USA; 2017;396-401.
3. Mehta R, Sahni J, Khanna K. Internet of things: vision, applications and challenges. Procedia Computer Sci 2018;132:1263-9.
4. Jia X, Feng Q, Fan T, Lei Q. RFID technology and its applications in Internet of Things (IoT). Proceedings of the 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), Yichang; 2012:1282-5.
5. Lee YM, Cheng F. Exploring the impact of RFID on supply chain dynamics. Proceedings of the 36th conference on Winter simulation; 2004 Dec 5-8; Washington, DC, USA; 2004. pp. 1145-52.
6. Wu L, Liu S, Zhao B, Wu W, Zhu B. The research of the application of the binary search algorithm of RFID system in the supermarket shopping information identification. J Wireless Com Network 2019;27:1-10.
7. International Civil Aviation Organization. Doc 9303 - Machine readable travel documents. 7th ed. ICAO; 2015.
8. Avoine G, Calderoni L, Delvaux J, Maio D, Palmieri P. Passengers information in public transport and privacy: can anonymous tickets prevent tracking? Int J Information Management 2014;34:682-8.
9. Chothia T, Smirnov V. A traceability attack against e-passports. In: Sion R, editor. Financial cryptography. Lecture notes in computer science. Springer; 2010. pp. 20-34.
10. Ma D, Saxena N, Xiang T, Zhu Y. Location-aware and safer cards: enhancing RFID security and privacy via location sensing. IEEE Trans Distributed Secure Computing 2013;10:57-69.
11. Yang SJ, Huang X. Certain types of M-fuzzifying matroids: a fundamental look at the security protocols in RFID and IoT. Future Generation Computer Systems 2018;86:582-90.
12. Halevi T, Li H, Ma D, Saxena N, Voris J, et al. Context-aware defenses to RFID unauthorized reading and relay attacks. IEEE Transactions Emerging Topics Computing 2013;1:307-18.
13. He D, Zeadally S. An analysis of RFID authentication schemes for internet of things in healthcare environment using elliptic curve cryptography. IEEE Int Things J 2015;2:72-83.
14. Li N, Mu Y, Susilo W, Guo F, Varadharajan V. Vulnerabilities of an ECC-based RFID authentication scheme. Security Comm Networks 2015;8:3262-70.
15. Calderoni L, Maio D. Cloning and tampering threats in e-passports. Expert Syst Appl 2014;41:5066-70.
16. Gandino F, Montrucchio B, Rebaudengo M. Tampering in RFID: a survey on risks and defenses. Mobile Netw Appl 2010;15:502-16.
17. Gao L, Zhang L, Lin F, Ma M. Secure RFID authentication schemes based on security analysis and improvements of the USI protocol. IEEE Access 2019;17:1360-6.
18. Aghili SF, Mala H, Kaliyar P, Conti M. SecLAP: secure and lightweight RFID authentication protocol for Medical IoT. Future Gener Comput Syst 2019;101:621-34.
19. NXP Semiconductors: NT4H2421Gx - NTAG 424 DNA. Tech. rep., NXP (January 2019), rev. 3.0. Available from https://www.nxp.com/docs/en/data-sheet/NT4H2421Gx.pdf [Last accessed on 3 Jun 2020]
20. International Organization for Standardization Electrotechnical Commission. Organization, Security and Commands for Interchange. 3th ed. 2013.
21. Finkenzeller K, Muller D. RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication. 3th ed. Wiley; 2010.
22. NFC Forum: Type 4 Tag Operation Specification. 2011. Available from https://nfc-forum.org/product/nfc-forum-type-4-tag-specification-version-1-1/ [Last accessed on 3 Jun 2020]
23. NFC Forum: NFC Data Exchange Format (NDEF). 2006. Available from https://nfc-forum.org/product/nfc-data-exchange-format-ndef-technical-specification/ [Last accessed on 3 Jun 2020]
24. Daemen J, Rijmen V. The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer; 2002.
25. Miller VS. Use of elliptic curves in cryptography. In: Williams HC, editor. Advances in Cryptology - CRYPTO'85, Proceedings. CRYPTO 1985. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg; 1985;218:417-26.
26. NXP Semiconductors: AN12196 - NTAG 424 DNA and NTAG 424 DNA TagTamper features and hints. Tech. rep., NXP (July 2019), rev. 1.5. Available from https://www.nxp.com/docs/en/application-note/AN12196.pdf [Last accessed on 3 Jun 2020]
27. Ng CY, Susilo W, Mu Y, Safavi-Naini R. Practical RFID ownership transfer scheme. J Computer Security 2011;19:319-41.