NFCGate: Opening the Door for NFC Security Research with a Smartphone-Based Toolkit

Steffen Klee*
Secure Mobile Networking Lab
TU Darmstadt

Alexandros Roussos*
Secure Mobile Networking Lab
TU Darmstadt

Max Maass
Secure Mobile Networking Lab
TU Darmstadt

Matthias Hollick
Secure Mobile Networking Lab
TU Darmstadt

Abstract

Near Field Communication (NFC) is being used in a variety of security-critical applications, from access control to payment systems. However, NFC protocol analysis typically requires expensive or conspicuous dedicated hardware, or is severely limited on smartphones. In 2015, the NFCGate proof of concept aimed at solving this issue by providing capabilities for NFC analysis employing off-the-shelf Android smartphones.

In this paper, we present an extended and improved NFC toolkit based on the functionally limited original open-source codebase. With in-flight traffic analysis and modification, relay, and replay features this toolkit turns an off-the-shelf smartphone into a powerful NFC research tool. To support the development of countermeasures against relay attacks, we investigate the latency incurred by NFCGate in different configurations.

Our newly implemented features and improvements enable the case study of an award-winning, enterprise-level NFC lock from a well-known European lock vendor, which would otherwise require dedicated hardware. The analysis of the lock reveals several security issues, which were disclosed to the vendor.

1 Introduction

With the continuous advance of contactless applications, including payment and access control systems, the potential for abuse and security breaches is on the rise. This is exemplified by the recent increase of smartphone-based contactless payment transactions using Near Field Communication (NFC) [46, 58] and the expansion of NFC capabilities for apps on popular mobile platforms [4, 9]. Contactless payment systems have been shown to be susceptible to various attacks [10, 21, 22, 23, 48, 49, 55]. Research of potential attacks on NFC protocols commonly uses Android devices or dedicated hardware to capture NFC traffic. While dedicated hardware provides several advantages in versatility as well as advanced features, such as supported technologies, it is also expensive and can be difficult to use [37, 45]. In contrast, ordinary Android devices look less suspicious in public, are readily available, generally affordable, but usually limited in its set of features compared to dedicated hardware.

NFCGate, whose proof of concept (PoC) was originally presented as a demo by Maass et al. in 2015 [42], is a research toolkit for active NFC protocol analysis compatible with the Android platform, which can take advantage of the NFC stack and act as a programmable NFC device. It provides both the basic features for ordinary Android devices as well as advanced features similar to those found on dedicated hardware, while maintaining availability and affordability. Even though the original PoC was able to relay NFC traffic, clone card identifiers, and provided basic logging functionality, it had limited compatibility with NFC technologies and chipsets.

In this work, we describe the updated version of NFCGate [53], including its original and new functionality, with regards to networking and NFC communication. In particular, our contributions include:

- New modes: replay capabilities for captured NFC traffic, on-device mode for capturing traffic system-wide.
- Python-based plugin system for analyzing and modifying traffic on-the-fly.
- Improvements to the clone mode for duplicating static tag data of different NFC technologies.
- Extension of the relay mode for wormhole attacks.

Using NFCGate, we also conduct a security analysis of an NFC-based smart lock as a case study and evaluate the security of its protocol and implementation to demonstrate the toolkit. Since the lock is not compatible with Android's NFC stack by default, this case study would not be possible without NFCGate, which enables active attacks on the underlying protocol. Finally, we evaluate the additional latency generally introduced by the use of NFCGate in multiple real-world scenarios and discuss potential countermeasures.

*Both authors contributed equally to this research.

2 Background

In this section, we provide background information on the techniques and technologies used in NFCGate. We discuss NFC standards, the NFC software stack, and function hooking on Android.

2.1 NFC Standards

The Android NFC stack currently supports four basic types of technologies: NFC-A, NFC-B, NFC-F, and NFC-V [8].

Terminology While NFC endpoints come in various shapes and sizes, e.g. passive cards, active tags, or even card emulators, the communication in standards is usually defined between two endpoints: Proximity Integrated Circuit Card (PICC) and Proximity Coupling Device (PCD). A PICC is a device in the tag role and a PCD is an endpoint in the reader role. While establishing a connection between these endpoints, the PICC exchanges some *static tag data* with the PCD to initialize the communication. This data depends on the type of NFC technology and can be used to help the PCD decide between multiple available tags. To exchange data after initialization, messages are sent with the Application Protocol Data Unit (APDU) encapsulated in their payload.

NFC-A/B The ISO/IEC 14443 family of standards consists of four parts, describing the different layers in standardized NFC communication. While Part 1 describes the physical properties of NFC at the lowest level [29], Part 4 specifies the "half-duplex block transmission protocol" [30] used to transfer APDUs between NFC-enabled endpoints of type A and B. Several open and proprietary protocols are built on top of this protocol and allow for higher-level operations such as authentication, file access, and cryptographic computations to be performed on the tag. For example, one of these open protocols is defined in the ISO/IEC 7816-4 Interindustry Card Standard (ICS) [32, 33], while the widely used Mifare DESFire is a proprietary protocol [44].

The transport layer also includes the negotiation of a Frame Waiting Time (FWT), which is specified by the PICC as i, and allows the PCD to retransmit a message if no response was received within the interval [30, Section 7.2]. It is defined as follows:

$$FWT_i = \left(\frac{256 \cdot 16}{13.56\,\text{MHz}}\right) \cdot 2^i \qquad 0 \leq i \leq 14$$

NFC-F Similarly, the JIS X 6319-4 standard specifies, among other things, the physical properties and the transmission protocol for "high-speed proximity cards" [34], which Sony FeliCa (NFC-F) complies with. Despite its similarities to ISO/IEC 14443, this standard is incompatible and requires both PICC and PCD to support the tag technology.

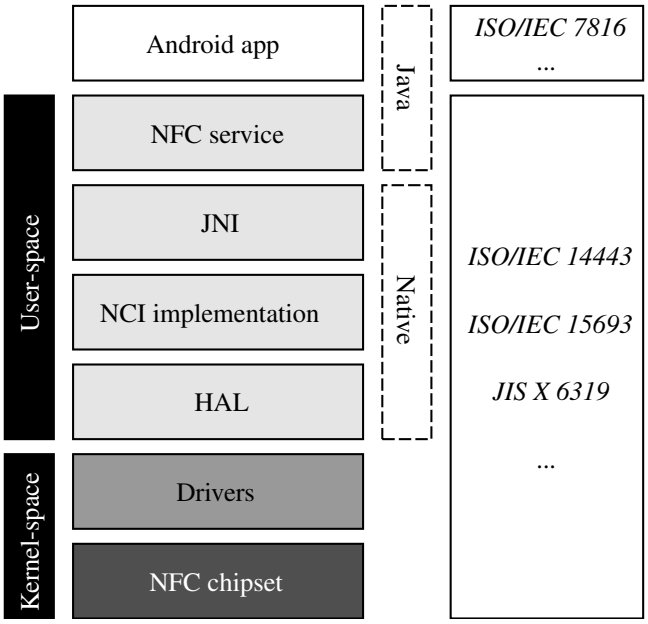

Figure 1: Architecture of the Android NFC stack.

NFC-V In contrast to the previous technologies, the vicinity standard ISO/IEC 15693 (NFC-V) [31] achieves significantly more distance than ISO/IEC 14443, but trades some throughput for it. Due to its different physical properties resulting from the operation at a greater distance, this standard is incompatible to ISO/IEC 14443 and does not mandate the implementation of any transport protocol beyond exchanging the static tag data.

NCI The NFC Controller Interface (NCI) standard [43] specifies the high-level communication protocol between an NCI-conforming NFC controller and the application processor using various underlying transport mechanisms, e.g., I^2C. Most importantly, NCI defines sending configuration streams to the NFC Controller (NFCC), which modify a standardized setting with each option in the stream. Setting these options can change the way the NFCC presents itself to other readers during card emulation, including emulating static tag data. While the sending of such streams is not supported as an API by Android, it exists internally and could be accessed through binary instrumentation.

2.2 Android NFC Stack

The Android NFC stack in general consists of multiple layers ranging from the underlying NFC chipset to the high-level programming interfaces available through the Android SDK.

Figure 1 outlines the architecture of the Android NFC stack. The lowest level consists of the NFC chipset and the NFC controller, which interacts with the controller-specific kernel driver, usually via a transport bus like I^2C or UART. The

communication then leaves the kernel-space and continues in the user-space with the hardware abstraction layer (HAL) that chipset manufacturers are required to implement. This eliminates the need for chipset-specific adjustments of the user-space NFC stack. As of now, the Android Open Source Project ships with Broadcom and NXP HAL implementations [1]. It consists of native libraries and Java framework classes. Of those native libraries, the NCI library communicates with the HAL and delivers results to the NFC service using the Java Native Interface (JNI). When using the NFC API in the Android SDK, every app communicates with the privileged NFC service.

2.3 Android HCE

Host Card Emulation (HCE) allows Android-based smartphones with NFC to emulate tags for all use-cases where the phone takes the role of the tag, e.g. Google Pay [2]. In order to support multiple HCE applications for different purposes on a single smartphone, Android employs a complex Application Identifier (AID) routing mechanism. This requires Android to terminate the ISO/IEC 7816-4 ICS connection and intercept the initial `SELECT` message containing the AID, which is then used to route the requests to the correct application registered for the AID. This artificial limitation prevents Android from accepting any requests not conforming to ICS or not starting with the expected `SELECT` message.

2.4 Android Function Hooking

Hooking is used to intercept regular function calls within foreign applications to modify or extend existing functionality. By injecting instructions into the target function, hooking allows the modification of its control-flow, parameters, return value, and behavior [41]. Because Android applications consist of Java code with optional native code connected via the JNI, two types of function hooking exist.

Java Function Hooking. Since the Java byte code is executed in a Java virtual machine, it is independent of the underlying platform architecture. As a consequence, function hooking in Java is also platform-independent. While the complex process architecture on Android requires significant engineering effort to allow modifications to its virtual machine, the well-known hooking frameworks Xposed [50] and EdXposed [51] accomplish general Java function hooking by replacing the virtual machine entirely. These frameworks enable application extensions to alter the behavior of any other application on the system.

Native Function Hooking. In contrast to Java code, native instructions are tied to the processor architecture. Android applications using native code have to provide binaries for all targeted architectures. Consequently, hooking native code requires platform-specific techniques as well. In order to hook native functions, a hooking application substitutes instructions in the target process with the desired behavior. Of the several options that exist for function hooking techniques [38, 41, 56], the method we use is *procedure linkage table* hooking provided by the xHook library [28].

3 Related Work

While many software-based tools for the analysis of NFC-based protocols have been developed, these tools are mostly tied to their specific use-case. In addition, they operate on high levels in the NFC software stack, so that their control of the underlying chipset is limited to that of the available operating system APIs. One of the first software-based NFC tools for relaying APDUs over the network dates back to 2011 [21] and uses a Nokia 6131 as the reader and a BlackBerry 9900 as the card emulator. It shows the applicability of relay attacks on applications using ISO/IEC 14443 APDUs and compares timings of smart cards and the relay.

Another software-based tool [55] uses HCE on the widely available Android operating system to passively (unmodified) relay APDUs over the network. With this approach, the tool is able to successfully relay a MasterCard contactless transaction from New York (USA) to Madrid (Spain). Due to limitations on the Android platform, the tool requires the first APDU to be a ICS `SELECT` command with a previously registered AID. While a separate Xposed module can be used to circumvent the AID restriction, the tool cannot bypass the ICS requirement. Additionally, the tool cannot emulate static tag data due to a lack of platform support.

In a similar way, the discontinued NFCProxy tool [40] uses basic Android HCE to relay NFC traffic. Due to its short development time, NFCProxy lacks advanced features and bypasses for Android platform restrictions. Despite its shortcomings, NFCProxy is used in a PoC attack on the payment systems Visa payWave (qVSDC) and EMV contactless [10]. Table 1 gives a quick comparison of the aforementioned tools with respect to their features and properties.

Since the introduction of Google Wallet, NFC-based mobile payment systems have become a point of interest for security researchers. One particular analysis of Google Wallet [49] uses a combination of software- and hardware-based tools to relay APDUs from the phone's secure element to a dedicated hardware-based card emulator. This approach requires non-trivial modifications to the operating system in order to access the secure element. Since its publication, Google has fixed this particular attack vector and introduced HCE, eliminating the need for using the secure element. Google Wallet (now Google Pay) has since moved to HCE [24], which has seen widespread usage [58].

With the rise of electronic vehicles in Germany, a surge of integrated and cheap charging stations from different providers attracts the interest of security researchers. Due

Table 1: Feature comparison of NFCGate to other NFC tools.

Tool	Protocols	Availability	Usability and Handling	Price
NFCProxy [40], [21, 55]	Only ISO/IEC 7816 APDUs	Android	Inconspicuous, no additional hardware	$
Proxmark3 [45]	Any on ISO/IEC 14443	Dedicated hardware	Suspicious, requires USB host	$$$
ChameleonMini [37]	Any on ISO/IEC 14443	Dedicated hardware	Suspicious, requires USB host	$$
NFCGate	**Any on ISO/IEC 14443**	**Android (rooted)**	**Inconspicuous, no additional hardware**	**$**

to the simplicity of the charging stations, they do not feature regular payment processing. Instead, many providers rely on lightweight customized billing systems with a supplied token for user authentication. For example, some providers use NFC-based smartcards as hardware tokens as analyzed by Dalheimer [18]. He shows that despite the usage of Mifare Classic tags supporting cryptographic operations, some systems only use static tag data as authentication for billing. This allows cloning the tag using a hardware-based NFC tool [37, 45], which emulates the static tag data.

The security of contactless payment systems generally relies on the assumption of the limited range of NFC. However, the ReCoil attack [52] shows that this premise does not hold. Using a passive (unpowered) relay consisting of antenna coils and a waist band, the NFC communication range can be extended up to 49.6 cm. This allows an attacker to relay the NFC traffic over a distance, thus bypassing physical security measures.

4 Implementation

While the Android operating system acts as an integrated NFC reader enabling apps to communicate with tags freely, tag emulation is much more limited. The HCE functionality of Android allows the emulation of applications based on ISO/IEC 7816 ICS, but has no option to mimic static tag data, which restricts its functionality for NFC security analysis. NFCGate is an Android application that circumvents these restrictions by using symbol hooking in managed (Java) and native (C/C++) code to gain low-level access to the Android NFC stack. It is compatible with devices supporting the Xposed [50] or EdXposed [51] hooking framework.

As the code of NFCGate has undergone many changes, we focus on the technical implementation of the following features of NFCGate, where Table 2 distinguishes the original PoC from the current version of NFCGate:

- *Standardized logging format.* Logging NFC traffic in the standardized packet capture format *pcapng* [54] enables the use of packet analyzers with advanced dissectors, such as Wireshark [16].
- *Clone mode.* The clone mode allows the emulation of static tag data for NFC-A, NFC-B, and NFC-F technologies.
- *Relay mode.* The relay mode features a server software with a Python plugin system for analyzing and modifying traffic on-the-fly.
- *Replay mode.* This mode replays previously recorded or imported traffic in either reader or tag role locally without a relay setup, or remotely with on-the-fly traffic modifications.
- *On-device capture mode.* Capturing system-wide NFC traffic from other apps running on the Android device allows the analysis of HCE-based applications as well as NFC reader applications.

4.1 Logging and Interoperability

When NFCGate processes NFC traffic, it is logged to the app's private storage in a database. The app provides an overview of these logs, their timestamps, and the mode they have been recorded in. A single log shows APDUs of the NFC traffic with their timestamp, a hexadecimal binary dump of data, and an indicator of the originator device (reader or tag).

NFCGate allows exporting and importing logs in the *pcapng* file format, providing interoperability with other tools such as the protocol analyzer Wireshark [16]. In order to encode captured APDUs, ISO/IEC 14443 framing is used with the predefined `DLT_ISO_14443` link type. Since no predefined link type exists for storing static tag data, the user-defined link type `DLT_USER_0` is used instead. Because APDUs are captured without their associated header or checksum, NFCGate creates an artificial header of type `I_BLOCK` with a direction indicator of either PICC→PCD or PCD→PICC and a flag bit

Table 2: Comparing original PoC to current NFCGate version.

	PoC	Current version of NFCGate
OS version	max. 6	max. 10
Architecture	ARMv7	Added ARM64
Chipsets	Broadcom	Any NCI
Technologies	A	Added B, F
Modes	Clone, relay	Added replay, on-device capture
Interoperability	-	Added logging, import/export, Python plugin system

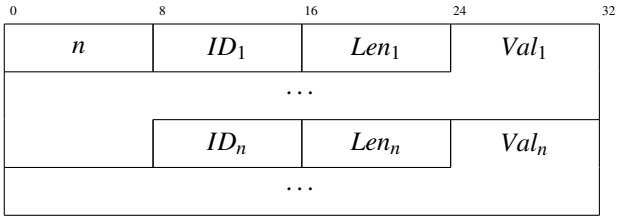

Figure 2: Format of NCI configuration parameters.

Table 3: Available NCI config parameters per technology.

Tech.	Prefix	Parameter *IDs*
NFC-A	`LA_`	`NFCID1`, `SEL_INFO`, `BIT_FRAME_SDD`, `PLATFORM_CONFIG`, `HIST_BY`
NFC-B	`LB_`	`NFCID0`, `APPLICATION_DATA`, `SFGI`, `SENSB_INFO`, `ADC_FO`, `H_INFO_RSP`
NFC-F	`LF_`	`T3T_IDENTIFIERS_1`, `T3T_FLAGS`, `T3T_PMM`

to ignore the missing checksum. This framing of APDUs with ISO/IEC 14443 headers enables the use of Wireshark's protocol dissector [35] and its sub-dissectors such as ISO/IEC 7816 [36] for analysis.

4.2 Clone Mode

Despite the HCE API on Android not supporting static tag data emulation, the lower-level NCI stack allows setting arbitrary tag data. Using symbol hooking, NFCGate gains access to the lower-level NCI stack, which allows the *clone mode* to emulate any captured static tag data. Note that while even some dedicated hardware restricts the range of values allowed to be manually set, no such restrictions were encountered using the Android NCI stack.

The NCI standard defines the `CORE_SET_CONFIG_CMD` command to configure the NFC discovery in poll and listen mode. When emulating a tag (listen mode), the NFCC uses the supplied configuration stream to initialize itself. In the NCI implementation [5], the `NFC_SetConfig` function is used for sending a configuration stream to the NFCC. Using this function with custom configuration options allows NFCGate to set the static tag data.

Figure 2 shows the format of the configuration stream. n is the total number of options in the stream, while ID_i, Len_i, and Val_i specify the identifier, length in octets, and value of the configuration parameter. Table 3 lists the relevant parameter *IDs* for the emulation of static tag data in any technology supported by Android HCE.

The NCI library also invokes `NFC_SetConfig` at different times, such as when starting RF discovery, overwriting the configuration set by NFCGate. In order to protect its custom set configuration and increase reliability, this function is altered using symbol hooking. The hook removes configuration parameters that would overwrite the custom values before passing them to the original function. This not only ensures that the custom configuration stays active at all times, but also allows NFCGate to save the rejected values and apply them after closing *clone mode* to restore the NFCC to its normal operating state.

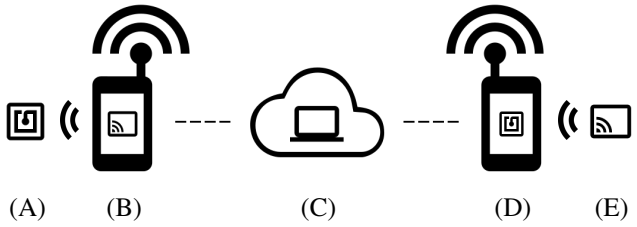

Figure 3: NFCGate relay setup with external server.

While this mode allows the emulation of static tag data, it does not support responding to any APDU commands. Despite this restriction, *clone mode* is useful when static tag data (i.e., the NFCID) is used for access control as seen in [18], since it only requires one device running NFCGate.

4.3 Relay Mode

Extending clone mode, the *relay mode* forwards received APDUs and static tag data to a server, as well as sending APDUs received from the server back to the tag or reader. Figure 3 shows the interaction of a legitimate tag (A), a legitimate reader (E), and two devices running NFCGate in reader (B) and tag mode (D) connected to the server (C). When the server receives data from NFCGate, it now processes the data by filtering it through all enabled plugins before broadcasting the potentially modified data to all other connected NFCGate devices in the same session. Plugins are invoked in a user-specified order and able to fully modify data, in accordance with the Dolev-Yao model [19]. The server is written in the Python programming language and does not require external dependencies, which enables hosting the server directly on smartphones with any Python interpreter.

In relay mode, NFCGate devices choose between the two roles *reader* and *tag*, while logging all data in the *pcapng* format (see Section 4.1). Using two connected NFCGate devices, one as the reader and one as the tag, in a setup similar to Figure 3, enables a user to capture NFC traffic between a legitimate tag and reader in NFC-enabled systems.

4.4 Replay Mode

In *replay mode*, NFCGate uses previously captured logs to communicate with an NFC device. As in relay mode, it supports the two roles *reader* and *tag*. The role specifies which side of the recorded NFC traffic to replay to the device. For

example, when using the tag role, NFCGate emulates a tag, similar to clone mode. When the reader sends APDUs to NFCGate, it responds with the corresponding tag APDU taken from the selected log. Multiple options exist for the selection of the APDU to replay: In *index-based* mode, the selection is only based on the position in the log, while the *data-based* replay selects responses based on the contents of the request APDU. All replay traffic is recorded in a new log for further analysis. In addition to replaying unmodified traffic from the log, NFCGate supports *advanced replay* over the network. This setting routes all traffic over the server, allowing for modifications or replacements of the replayed traffic by server plugins before sending it to the NFC device. By not requiring a second device running NFCGate, active attacks on an NFC protocol using replay mode have a lower response latency than attacks in relay mode, which could potentially bypass relay countermeasures based on timing.

4.5 On-Device Capture

The *on-device capture mode* supports capturing low-level NFC traffic of the Android device while reading or emulating an NFC tag. NFCGate hooks central functions in the NFC service, allowing it to capture the entire traffic without interfering with other apps. This enables undetected capturing of the NFC traffic from other apps without requiring any relay setup, thus avoiding problems with latencies introduced by a relay.

We hook the `transceive` method of the `NFCService` [7] after its execution to capture NFC traffic of the device while reading a tag. This method receives data to be sent to the currently active tag, with the tag's response in the return value. When the device discovers a tag, it invokes the `dispatchTag` method of `NfcDispatcher` [6], which is hooked to enable capturing the static tag data.

In HCE mode, the `HostEmulationManager` [3] manages the routing of data to apps registered with different AIDs. On discovery of the tag device by an external reader, `onHostEmulationActivated` is called. Whenever the reader transmits data, `onHostEmulationData` receives this data and routes it to the correct application. Hooking this method captures NFC traffic from a reader to the device in HCE mode, even if no AID is registered or the APDU is not compatible with ISO/IEC 7816-4. This circumvents the Android system restriction on the initial reader APDU. Any response of the HCE application is routed through the `NFCService.sendData` function, hence hooking this function also enables capturing NFC traffic from the device to the reader.

All hooks collect the captured data locally within the context of the NFC service. The NFCGate app requests these captures from the service through broadcasts and intents. In order to be exported and used in any other NFCGate mode, the capture is logged in the context of the app.

5 Case Study: Smart Door Lock

In order to demonstrate the capabilities of NFCGate in analyzing NFC protocols using off-the-shelf smartphones, we conduct a case study on an NFC-based locking system.

5.1 Overview

The subject of our case study is an enterprise-level electronic access control system manufactured by a well-known European security vendor[1]. We choose this product based on the existing reputation of the vendor for physical locking mechanisms, the security award the product had received, and the fact that it was available as an off-the-shelf solution unlike, for example, NFC-based hotel door locking systems. The system consists of a base station and cylinder locks, with permissions for each component bound to user accounts manageable via a web interface. A user issues actions such as unlocking doors with a transponder or an app compatible with the Android and iOS operating systems. In this paper, we focus on the NFC-based cylinder unlocking mechanisms, leaving aside communication with the base station. Our hardware setup consists of the base station, one cylinder lock, and two DESFire transponders. In addition to the hardware, we use the reader software, which is capable of adding transponders to the deployment.

The base station contains the web-based user, cylinder, and permission management. The lock connects to the base station wirelessly with a proprietary protocol in the 868.3 MHz band, requesting authorization and configuration parameters. The transponder presents itself to the lock using the NFC protocol described in Section 5.2 when in range. Then, the lock checks the unique identifier UID obtained from the transponder by sending this information to the base station. If this UID is authorized to access the lock, it accepts the transponder and unlocks the cylinder.

5.2 Unlocking Procedure

Using the relay mode of NFCGate, several communications between lock and transponder were recorded and exported in the *pcapng* format. Even though the locking protocol does not conform to ICS, NFCGate circumvents the Android limitation and allows us to receive arbitrary APDUs. Analyzing the captured traffic using a combination of Wireshark and the libfreefare source code [17] as a reference, we were able to reverse-engineer the protocol used by the locking system.

The lock uses a modified DESFire AES authentication protocol to derive a session key and to establish an authenticated and encrypted channel to the transponder. It is not immediately clear why the protocol was modified. Using the established channel, the transponder transmits its UID to the lock securely.

[1] Results published with consent of the vendor.

Figure 4 shows the protocol for the unlocking procedure. Deviations from the DESFire AES authentication protocol are marked in red. The key k is a pre-shared 128-bit AES key. The encryption and decryption routines, Enc and Dec, use AES-128 in CBC mode. The initialization vector (IV) for the cryptographic operations is stored in the variables IV_A and IV_B. The secure channel uses AES-128 in CBC mode with CMAC for authentication in the routines AEnc and ADec.

Initially, the PICC picks a random nonce r_B and sends it to the PCD encrypted with the pre-shared key k. Receiving this message, PCD decrypts r_B and sends its own nonce r_A and the rotated r_B^* to PICC. At this point, the protocol deviates from DESFire authentication by using a static value for r_A and not updating the IV used for encryption. This results in message $m_{5.1}$ being encrypted under $IV = 0$ while PICC expects $IV = m_4$. Therefore, PICC decrypts $m_{5.1}$ with $IV = m_4$ and obtains $r'_A = r_A \oplus m_4$ with $r'_A \neq r_A$. If the decrypted r_B^* matches the local rotation of r_B the PICC accepts the authentication and derives the session key k_s. Finally, it sends the rotated $r_A'^*$ to the PCD. In order to reconcile the difference between $r_A'^*$ and the expected r_A^*, the PCD needs to calculate $r'_A = r_A \oplus m_4$. After checking that the received $r_A'^*$ matches the local rotation of r'_A, the PCD accepts the authentication and derives the session key k_s. Since the PICC only knows r'_A, the session key must be derived from it instead of r_A.

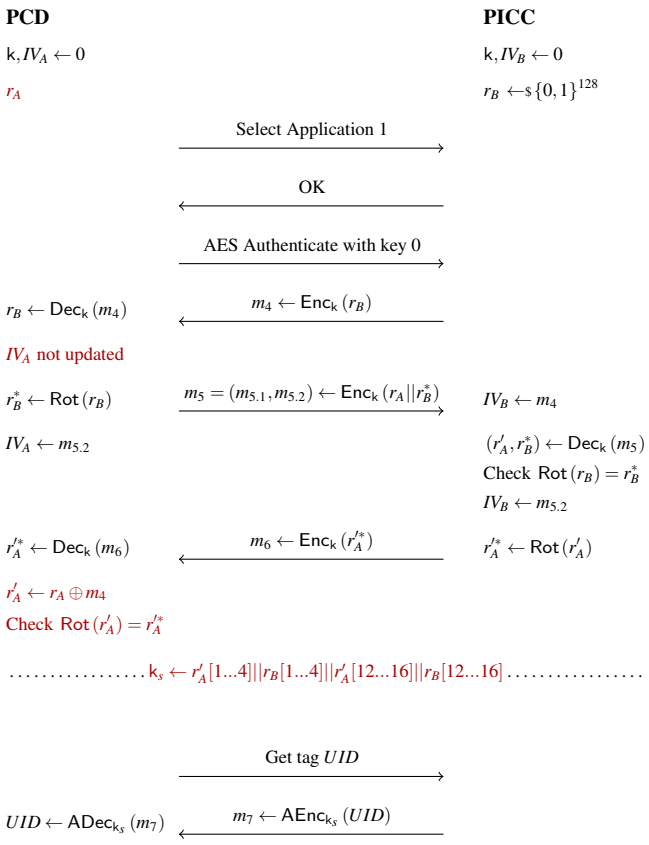

Figure 4: Cylinder unlocking procedure.

Concluding the interaction, the PICC sends its UID over the encrypted and authenticated channel using the established session key k_s. All encrypted channel communication contains a CMAC-based authentication tag alongside the plaintext, in accordance with the DESFire AES authentication protocol.

5.3 Security Issues

In this section, we discuss security-related issues in the design and implementation of the cylinder unlocking procedure. Proof of concept attacks facilitate NFCGate's features to demonstrate their applicability in real-world scenarios using off-the-shelf hardware. Figure 5 shows the different attack setups. We consider the adversary $\mathcal{A}$ in the Dolev-Yao model [19], who can capture, block, modify, and resend captured messages all without breaking the cryptographic primitives.

Relay Attack. With NFCGate's relay mode we are able to forward the NFC traffic between PICC and PCD over a network. $\mathcal{A}$ employs two smartphones running NFCGate connected to the server, one acting as the reader and the other as the tag. They communicate with the transponder and cylinder while transmitting NFC traffic to the server. As long as the unlocking procedure takes less than ≈ 1.8 s, it leads to a successful unlocking of the cylinder. Considering the average network delay around half the globe [Frankfurt/Main (Germany) to Sydney (Australia)] is ≈ 360 ms [57] we could successfully unlock the cylinder from around the world. This demonstrates that the time limit imposed by the cylinder is not an effective countermeasure to these kinds of attacks. In addition to tightening the timings, *distance bounding* techniques could be used as further mitigations (see Section 7).

Replay Attack. Using messages previously captured during an unlocking procedure between the cylinder and an authorized transponder, $\mathcal{A}$ replays the traffic employing NFCGate in the tag role. As a result of the protocol modifications outlined in Figure 4 and in contrast to the replay-protected DESFire authentication protocol, the unlocking procedure used by the cylinder is vulnerable to replaying the communication. Due to the static nonce r_A used by the PCD and the zero-initialized IV_A and IV_B, the randomness of the protocol entirely depends on the nonce r_B chosen by the PICC as shown in Figure 4.

Replaying PICC messages to the PCD results in the same randomness, leading to the same session key k_s. Since the encrypted channel relies only on k_s and the content of the messages, which is always the same UID, an identical k_s leads to the exact same messages. This replay attack does not require knowledge of any encryption key or breaking any primitive in the protocol. Conforming to the DESFire protocol by choosing the nonce r_A randomly instead of statically mitigates replay attacks.

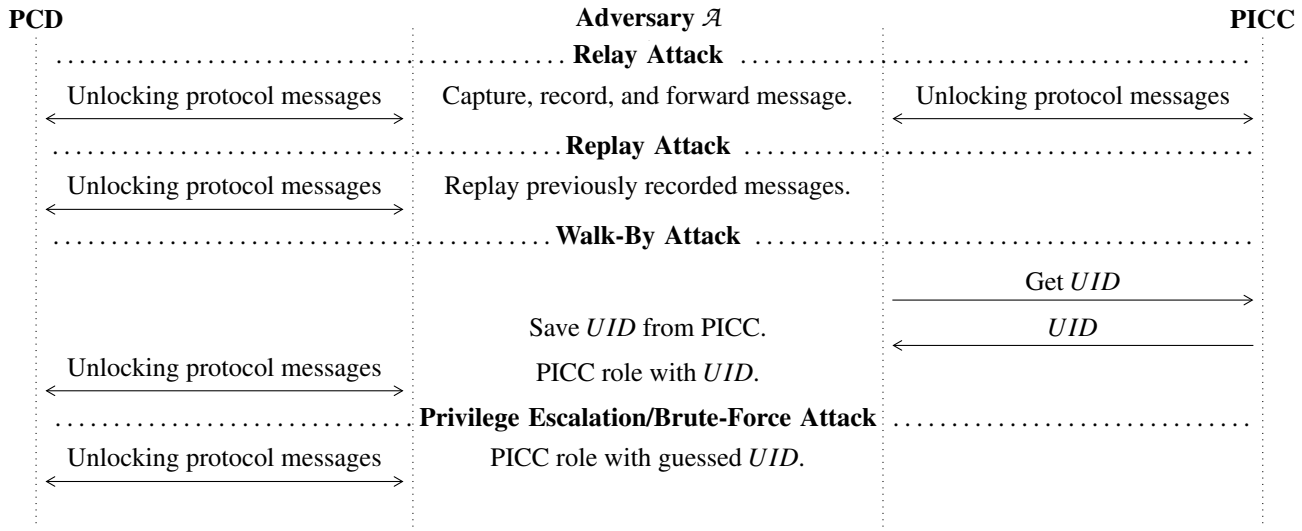

Figure 5: Attack setups with cylinder (PCD), transponder (PICC), and adversary $\mathcal{A}$.

Walk-By Attack. Because the secret key k used for encryption in the authentication phase is static for all cylinders and transponders, we can extract this key from the freely available utility software. This application allows the registration of new transponders to the system without requiring the physical presence of a cylinder by using a separately sold USB NFC reader. The slightly obfuscated key k can be extracted from the software by reverse-engineering the application binary. Therefore, we now consider k to be known to $\mathcal{A}$ and adapt our model to include the encryption and decryption of protocol messages.

The static key allows the implementation of the PCD role in the unlocking procedure as an NFCGate server plugin. Since the plugin reads the UID of transponders without requiring access to a cylinder, it simplifies the relay attack discussed earlier. In this walk-by attack, $\mathcal{A}$ can obtain the authorization by requesting the UID from a transponder, without needing an active connection to the cylinder. Additionally, this allows the creation of multiple copies of a single transponder, thus violating the software design where every transponder is uniquely mapped to exactly one user.

Walk-by attacks could be mitigated by using individual keys for each deployment, since the key is only available to users of this particular system. Even if an adversary $\mathcal{A}$ had access to the key k of their own system, no attacks on other systems would be possible. In addition, extracting k from a transponder or cylinder is not trivial because it requires destructive physical access to the hardware.

Privilege Escalation and Brute-Force Attack. Comparing $UIDs$ obtained from different transponders in consecutive production batches indicates the presence of a pattern. The $UIDs$ have 7 bytes, where the first byte specifies the tag's manufacturer, in our case `04` for NXP. The remaining 6 bytes do not seem to be chosen at random, instead tags from the same batch have similar byte sequences. This suggests a serial number pattern. The numerical difference between the examined $UIDs$ of our two transponders with the same production date and consecutive batches is 3596.

If $\mathcal{A}$ has access to a transponder in a system without authorization to unlock a specific cylinder, they can use the static key k to extract the UID of their transponder. This significantly reduces the number of guesses required for other $UIDs$ since tags of the system were likely produced in the same or close production batches. Guessing a UID correctly escalates privileges of $\mathcal{A}$ to other cylinders. A plugin for the NFCGate server implements brute-forcing $UIDs$, optionally starting from a known value. It achieves a throughput of approximately three tries per second since the cylinder implements no limit on the number of authorization tries. In our example, it would take around 20 minutes to execute such an attack.

These attacks can be mitigated by using a random number as authorization instead of the predictable UID. This random number would be stored on the tag protected by the existing authentication protocol. Furthermore, using individual keys per base station instead of a static key k hinders the execution of this attack. Adding brute-force countermeasures by limiting the number of authorization tries in a specific time frame significantly slows down any brute-force attempt.

5.4 Responsible Disclosure

We contacted the vendor to report our findings, and received an initial response within four hours. After providing the vendor with our report, we discussed the discovered issues in a telephone conference and an in-person meeting. The issues will be fixed in the next revision of the system, and the

vendor aims to provide patches to existing customers. The entire process was handled in a professional and collaborative manner.

6 Performance Evaluation

After demonstrating the capabilities of the software in this case study, we also evaluate the delays induced by NFCGate in relay and replay mode for more general cases. In this section, we investigate whether an upper bound on the response time imposed by the PCD could be a viable protection mechanism against relay attacks by determining the additional latency in comparison to a direct communication between a PCD and a PICC. One such upper bound could be imposed by the Frame Waiting Time (FWT) as defined in ISO/IEC 14443.

The evaluation consists of measuring the response time of a common command sequence for retrieving a value of a card in various PICC configurations. We use a PN532 chipset connected to a computer via USB/UART as the PCD and measure the command response time of the PICC. The original Mifare DESFire tag operates in ISO/IEC 7816 ICS mode. In particular, the measured commands are:

1. ICS `SELECT` file: DESFire AID (`0xA4`)
2. Select Application (`0x5A`)
3. Get FileSettings (`0xF5`)
4. Get Value (`0x6C`)

The NFCGate app runs on one Nexus 5X (Android 7) smartphone in the tag role and one OnePlus 6 (Android 9) smartphone in the reader role, while the NFCGate server is hosted directly on the Nexus 5X or a computer. In addition to a baseline measurement, where the PCD communicates with the original tag (TAG), we measure the latency in the following configurations:

RP *Local replay*. One smartphone replays previously captured NFC traffic locally to the PCD. No additional smartphone or server is used.

BT *Bluetooth relay*. Smartphones connected via a Bluetooth PAN. The server is hosted on one of the smartphones.

BW *Bluetooth tethering to wireless network*. One smartphone provides IEEE 802.11 wireless network access to the other via Bluetooth tethering. The server is hosted on a computer wired to the network.

WH *IEEE 802.11 wireless hotspot*. One smartphone offers a wireless network access point to the other one. The server is hosted on the smartphone providing the access point.

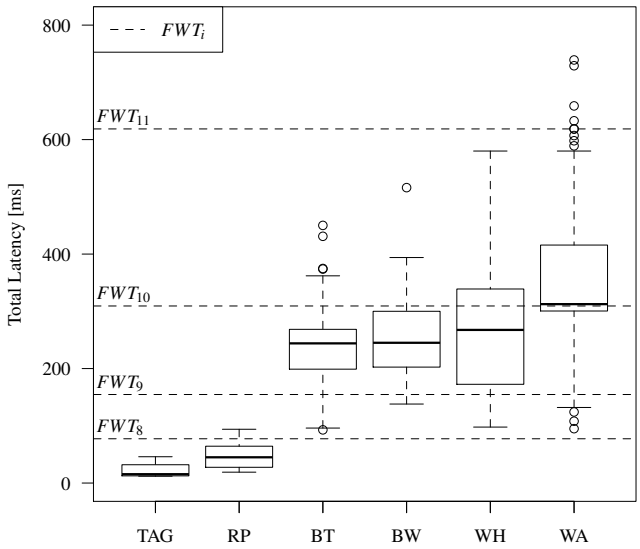

Figure 6: Latency measurements using NFCGate.

WA *IEEE 802.11 wireless network*. Both smartphones are connected to the same wireless network. The server is hosted on a computer wired to the network.

Every configuration using wireless technologies is conducted in an urban environment with multiple other wireless networks in active use.

Figure 6 depicts the total latency of command responses for the different configurations as a box plot ($n = 20$) with outliers, minima, maxima, median, and the lowest/highest values within 1.5 interquartile range. Dashed lines show values for the FWT, where FWT_j is only drawn for $8 \leq j \leq 11$, because FWT_8 is the minimum FWT specified by the original tag and any FWT_j with $j \geq 12$ is out of bounds. For $FWT \geq FWT_{11} \approx 619\,\text{ms}$, the latency stays within FWT bounds for all tested configurations, excluding some outliers. We note that the replay mostly stays within the FWT of the original tag and is in some cases indistinguishable from it.

All network configurations using the IEEE 802.11 wireless network standard with both smartphones exhibit a high variance in latency measurements while Bluetooth-based configurations appear to be more stable. We attribute the high variance to interference in the IEEE 802.11 wireless network and the stable Bluetooth measurements to the low data throughput in our evaluation. Therefore, we recommend a Bluetooth PAN configuration for close proximity relays.

7 Countermeasures

The FWT has been proposed as a potential countermeasure to relay attacks [21, 55], although it was primarily designed as a safety measure. Despite its standardization, the FWT has not been enforced in our experiments, which seems to be

very common [10]. Even if it was enforced, the FWT could be increased by the PICC up to ≈ 5 s. In the event of a loose mandatory FWT enforced by the PCD, our measurements show that it is possible to relay the communication even if FWT_j for $j \geq 10$ is in use. Assuming the use of the original tag's FWT is mandatory, a replay could be performed nevertheless. Especially in the case of a cryptographically intensive operation, a replay could be indistinguishable in timing from the original tag, since it does not use cryptography. Despite some configurations exceeding the FWT, a relay could still be performed, because the FWT was designed as a safety measure, not a security feature. The PCD simply retransmits a block after the FWT expires, allowing even more time for a relay to respond [30, Section 7.2]. This shows that FWT is not just an ineffective countermeasure but no countermeasure at all.

Mandatory Response Timeouts. One possible countermeasure against relay attacks is the implementation of a tight mandatory response timeout on the reader side. In contrast to the FWT, this cannot be influenced by the tag and constitutes a definitive communication timeout. Since this countermeasure requires knowing an upper limit on the response latency to any connectable tag beforehand, it only works in closed ecosystems, where all involved readers and tags are controllable. The duration of cryptographic operations in particular depends on hardware instruction support and performance, so low-end HCE devices might raise the upper limit on response latency, which would in turn reduce the effectiveness of this countermeasure.

Distance Bounding Protocols. First introduced by Brands and Chaum [13], distance bounding (DB) protocols solve the general issue of relay attacks. In a DB protocol, physical properties of a connection are used to prove that one endpoint is within a specified distance of the other. In addition to timing and freshness, some DB protocols employ cryptography to ensure the authenticity of endpoints [11, 12].

DB for NFC can be implemented on either the protocol or application layer. An implementation on the protocol layer requires an extension to the ISO/IEC 14443 standard and performs DB independently of the application [20, 25, 47]. While this simplifies the usage for many applications, it only measures the distance to some recipient, not necessarily the intended one. In contrast, an implementation on the application layer can ensure the correct recipient, but requires every application to implement their own, incompatible DB protocol [15, 26, 39].

One example for an application layer DB protocol is the DESFire EV2 Proximity Check [27]. The check uses cryptography to ensure mutual authentication at the end of the timing phase. Despite these engineering efforts, an attack against the DB protocol has already been discovered [14].

8 Conclusion

With NFCGate, we have enhanced the original PoC to an extensive NFC research toolkit that is compatible with any Android smartphones supporting the Xposed or EdXposed hooking framework and requires no changes to the system image. The interoperability proved invaluable in the case study, since it allowed us to further inspect the captured traffic in Wireshark with its existing NFC protocol dissectors. Using the toolkit, we analyzed a well-known NFC-based door locking system and uncovered several security issues of various severities. We launched a brute-force attack against the *UID* used as authorization employing the NFCGate server plugin feature. By informing the vendor of our case study, several security issues could be fixed, which helped secure a currently deployed product. The evaluation showed that the relay latency is low enough to bypass many countermeasures, while the negligible replay latency makes it almost indistinguishable from the original tag.

As future work, the Wireshark remote capturing protocol could be implemented, streaming the captured data from NFCGate to a running live capture to simplify the use of the *pcapng* export. Finally, we suggest using the on-device capture mode to further analyze HCE-based apps such as Google Pay, or inspect NFC reader apps, e.g. public transport services.

Availability

We make the source code of NFCGate available to the public under a free software license [53].

Acknowledgments

This work has been co-funded by the DFG as part of project C.1 within the RTG 2050 “Privacy and Trust for Mobile Users”, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE, and by the LOEWE initiative (Hesse, Germany) within the emergenCITY centre.

References

[1] Android Open Source Project. Halimpl - source. February 10, 2020. URL: https://cs.android.com/android/_/android/platform/hardware/nxp/nfc/+/65c90b3:halimpl (visited on February 20, 2020).

[2] Android Open Source Project. Host-based card emulation overview. Android developers. December 27, 2019. URL: https://developer.android.com/guide/topics/connectivity/nfc/hce (visited on February 24, 2020).

[3] Android Open Source Project. HostEmulationManager.java - source. May 15, 2019. URL: https://cs.android.com/android/_/android/platform/packages/apps/Nfc/+/ac65163:src/com/android/nfc/cardemulation/HostEmulationManager.java (visited on May 18, 2020).

[4] Android Open Source Project. Near field communication overview. Android developers. December 27, 2019. URL: https://developer.android.com/guide/topics/connectivity/nfc (visited on February 12, 2020).

[5] Android Open Source Project. Nfc - source. April 16, 2020. URL: https://cs.android.com/android/_/android/platform/system/nfc/+/91688f6:src/nfc/ (visited on May 18, 2020).

[6] Android Open Source Project. NfcDispatcher.java - source. March 24, 2020. URL: https://cs.android.com/android/_/android/platform/packages/apps/Nfc/+/68fe178:src/com/android/nfc/NfcDispatcher.java;l=270 (visited on May 18, 2020).

[7] Android Open Source Project. NfcService.java - source. April 9, 2020. URL: https://cs.android.com/android/_/android/platform/packages/apps/Nfc/+/b9d4425:src/com/android/nfc/NfcService.java;l=1470 (visited on May 18, 2020).

[8] Android Open Source Project. Tech - source. January 22, 2020. URL: https://cs.android.com/android/_/android/platform/frameworks/base/+/3eb265a:core/java/android/nfc/tech (visited on February 20, 2020).

[9] Apple Inc. Core NFC. Apple developer documentation. 2019. URL: https://developer.apple.com/documentation/corenfc (visited on February 12, 2020).

[10] Thomas Bocek, Christian Killer, Christos Tsiaras, and Burkhard Stiller. An NFC relay attack with off-the-shelf hardware and software. In Rémi Badonnel, Robert Koch, Aiko Pras, Martin Drašar, and Burkhard Stiller, editors, *Management and Security in the Age of Hyperconnectivity*, pages 71–83, Cham. Springer International Publishing, 2016. ISBN: 978-3-319-39814-3.

[11] Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Practical and provably secure distance-bounding. In Yvo Desmedt, editor, *Information Security*, pages 248–258, Cham. Springer International Publishing, 2015. ISBN: 978-3-319-27659-5.

[12] Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Towards secure distance bounding. In *International Workshop on Fast Software Encryption*, pages 55–67. Springer, 2013.

[13] Stefan Brands and David Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 344–359. Springer, 1993.

[14] Dominic Celiano. *Overclocking Proximity Checks in Contactless Smartcards*. Master's thesis, University of Cambridge, 2018.

[15] Tom Chothia, Flavio D. Garcia, Joeri de Ruiter, Jordi van den Breekel, and Matthew Thompson. Relay cost bounding for contactless EMV payments. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security*, pages 189–206, Berlin, Heidelberg. Springer Berlin Heidelberg, 2015. ISBN: 978-3-662-47854-7.

[16] Gerald Combs, Gilbert Ramirez, Guy Harris, et al. Wireshark. 2020. URL: https://www.wireshark.org/ (visited on February 13, 2020).

[17] Romuald Conty, Romain Tartière, Philippe Teuwen, et al. Nfc-tools/libfreefare. A convenience API for NFC cards manipulations on top of libnfc. URL: https://github.com/nfc-tools/libfreefare/ (visited on March 6, 2020).

[18] Mathias Dalheimer. Warum das Laden eines Elektroautos unsicher ist, December 27, 2017. URL: https://media.ccc.de/v/34c3-9092-ladeinfrastruktur_fur_elektroautos_ausbau_statt_sicherheit (visited on February 17, 2020).

[19] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1983. ISSN: 1557-9654. DOI: 10.1109/TIT.1983.1056650.

[20] Saar Drimer and Steven J. Murdoch. Keep your enemies close: distance bounding against smartcard relay attacks. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, Boston, MA. USENIX Association, 2007. ISBN: 1113335555779.

[21] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical relay attack on contactless transactions by using NFC mobile phones. English. In *Radio Frequency Identification System Security*, volume 8 of *Cryptology and Information Security Series*, pages 21–32, 2012. ISBN: 9781614991427. DOI: 10.3233/978-1-61499-143-4-21.

[22] Leigh-Anne Galloway, Tim Yunusov, and Aleksei Stennikov. First contact: new vulnerabilities in contactless payments, December 4, 2019. URL: https://leigh-annegalloway.com/presentation-materials/ (visited on February 13, 2020).

[23] Dennis Giese, Kevin Liu, Michael Sun, Tahin Syed, and Linda Zhang. Security analysis of Near-Field Communication (NFC) payments, 2019. arXiv: 1904.10623 [cs.CR].

[24] Google LLC. Set up Google Pay – Android. Google Pay help. 2020. URL: https://support.google.com/pay/answer/7625055?co=GENIE.Platform%3DAndroid (visited on February 13, 2020).

[25] Gerhard P. Hancke and Markus G. Kuhn. An RFID distance bounding protocol. In *Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks*, SECURECOMM '05, pages 67–73, USA. IEEE Computer Society, 2005. ISBN: 0769523692. DOI: 10.1109/SECURECOMM.2005.56. URL: https://doi.org/10.1109/SECURECOMM.2005.56.

[26] Jens Hermans, Roel Peeters, and Cristina Onete. Efficient, secure, private distance bounding without key updates. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 207–218, Budapest, Hungary. Association for Computing Machinery, 2013. ISBN: 9781450319980. DOI: 10.1145/2462096.2462129. URL: https://doi.org/10.1145/2462096.2462129.

[27] Darren Hurley-Smith and Julio Hernandez-Castro. Measuring the distance: investigating the DESFire EV2 distance bounding protocol. In *Cryptacus 2017*, 2017.

[28] iQIYI Inc. xHook. A PLT hook library for Android native ELF. URL: https://github.com/iqiyi/xHook (visited on May 24, 2020).

[29] ISO/IEC 14443-1:2000. Identification cards – Contactless integrated circuit(s) cards – Proximity cards – Part 1: Physical characteristics. Standard, International Organization for Standardization, Geneva, CH, April 2000. 5 pages.

[30] ISO/IEC 14443-4:2001. Identification cards – Contactless integrated circuit(s) cards – Proximity cards – Part 4: Transmission protocol. Standard, International Organization for Standardization, Geneva, CH, February 2001. 34 pages.

[31] ISO/IEC 15693-1:2010. Identification cards – Contactless integrated circuit cards – Vicinity cards – Part 1: Physical characteristics. Standard, International Organization for Standardization, Geneva, CH, October 2010. 5 pages.

[32] ISO/IEC 7816-3:2006. Identification cards – Integrated circuit cards – Part 3: Cards with contacts – Electrical interface and transmission protocols. Standard, International Organization for Standardization, Geneva, CH, November 2006. 58 pages.

[33] ISO/IEC 7816-4:2005. Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange. Standard, International Organization for Standardization, Geneva, CH, January 2005. 90 pages.

[34] JSA - JIS X 6319-4. Specification of implementation for integrated circuit(s) cards – Part 4: High speed proximity cards. Standard, Japanese Standards Association, Tokyo, JP, March 2016. 129 pages.

[35] Martin Kaiser, Guy Harris, Gerald Combs, et al. Wireshark/packet-iso14443.c at wireshark-3.2.1. Wireshark/wireshark. 2019. URL: https://github.com/wireshark/wireshark/blob/wireshark-3.2.1/epan/dissectors/packet-iso14443.c (visited on January 27, 2020).

[36] Martin Kaiser, Guy Harris, Gerald Combs, et al. Wireshark/packet-iso7816.c at wireshark-3.2.1. Wireshark/wireshark. 2019. URL: https://github.com/wireshark/wireshark/blob/wireshark-3.2.1/epan/dissectors/packet-iso7816.c (visited on January 27, 2020).

[37] Kasper & Oswald GmbH. ChameleonMini. 2020. URL: https://kasper-oswald.de/gb/chameleonmini/ (visited on February 13, 2020).

[38] Sungkwan Kim, Junyoung Park, Kyungroul Lee, Ilsun You, and Kangbin Yim. A brief survey on rootkit techniques in malicious codes. *J. Internet Serv. Inf. Secur.*, 2(3/4):134–147, 2012.

[39] Handan Kılınç and Serge Vaudenay. Contactless access control based on distance bounding. In Phong Q. Nguyen and Jianying Zhou, editors, *Information Security*, pages 195–213, Cham. Springer International Publishing, 2017. ISBN: 978-3-319-69659-1.

[40] Eddie Lee. NFC hacking: the easy way. In *Defcon hacking conference*, volume 20, pages 63–74, 2012.

[41] Juan Lopez, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. A survey on function and system call hooking approaches. *Journal of Hardware and Systems Security*, 1(2):114–136, 2017. DOI: 10.1007/s41635-017-0013-2.

[42] Max Maass, Uwe Müller, Tom Schons, Daniel Wegemer, and Matthias Schulz. DEMO: NFCGate: an NFC relay application for Android. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, New York, New York. Association for Computing Machinery, 2015. ISBN: 9781450336239. DOI: 10.1145/2766498.2774984. URL: https://doi.org/10.1145/2766498.2774984.

[43] NFC Forum. NFC Controller Interface (NCI). Technical Specification, November 2016. 206 pages.

[44] NXP Semiconductors N.V. MF3ICDx21_41_81. MIFARE DESFire EV1 contactless multi-application IC. Product short data sheet, December 9, 2005. 18 pages. URL: https://www.nxp.com/docs/en/data-sheet/MF3ICDX21_41_81_SDS.pdf (visited on February 23, 2020).

[45] ProxGrind. Proxmark3 Rdv4.0 development. 2020. URL: https://proxgrind.com/prototyping/proxmark3-rdv4-0-development/ (visited on February 13, 2020).

[46] Quartz. Apple Pay is on pace to account for 10% of all global card transactions. February 11, 2020. URL: https://qz.com/1799912/apple-pay-on-pace-to-account-for-10-percent-of-global-card-transactions/ (visited on February 12, 2020).

[47] Jason Reid, Juan M. Gonzalez Nieto, Tee Tang, and Bouchra Senadji. Detecting relay attacks with timing-based protocols. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, ASIACCS '07, pages 204–213, Singapore. Association for Computing Machinery, 2007. ISBN: 1595935746. DOI: 10.1145/1229285.1229314. URL: https://doi.org/10.1145/1229285.1229314.

[48] Michael Roland and Josef Langer. Cloning credit cards: a combined pre-play and downgrade attack on EMV Contactless. In *Presented as part of the 7th USENIX Workshop on Offensive Technologies*, Washington, D.C. USENIX, 2013. URL: https://www.usenix.org/conference/woot13/workshop-program/presentation/Roland.

[49] Michael Roland, Josef Langer, and Josef Scharinger. Applying relay attacks to Google Wallet. In *2013 5th International Workshop on Near Field Communication (NFC)*, pages 1–6, February 2013. DOI: 10.1109/NFC.2013.6482441.

[50] rovo89 and Thungstwenty. Welcome to the Xposed module repository. Xposed module repository. 2020. URL: https://xposed.info (visited on February 24, 2020).

[51] solohsu and Jim Wu. EdXposed framework official website. MeowCat Studio. 2020. URL: https://edxp.meowcat.org/ (visited on February 24, 2020).

[52] Yuyi Sun, Swarun Kumar, Shibo He, Jiming Chen, and Zhiguo Shi. You foot the bill! Attacking NFC with passive relays, 2020. arXiv: 2001.08143 [cs.CR].

[53] The NFCGate Team. NFCGate. An NFC research toolkit application for Android. 2020. URL: https://github.com/nfcgate/nfcgate (visited on March 19, 2020).

[54] Michael Tuexen, Fulvio Risso, Jasper Bongertz, Gerald Combs, and Guy Harris. PCAP Next Generation (pcapng) capture file format. draft-tuexen-opsawg-pcapng. Internet-Draft, Network Working Group, January 26, 2020.

[55] José Vila and Ricardo J. Rodríguez. Practical experiences on NFC relay attacks with Android. In Stefan Mangard and Patrick Schaumont, editors, *Radio Frequency Identification*, pages 87–103, Cham. Springer International Publishing, 2015. ISBN: 978-3-319-24837-0.

[56] Sebastian Vogl, Robert Gawlik, Behrad Garmany, Thomas Kittel, Jonas Pfoh, Claudia Eckert, and Thorsten Holz. Dynamic hooks: hiding control flow changes within non-control data. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 813–328, San Diego, CA. USENIX Association, August 2014. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/vogl.

[57] WonderNetwork. Ping time between Frankfurt and other cities. Wondernetwork. 2020. URL: https://wondernetwork.com/pings/Frankfurt (visited on February 6, 2020).

[58] yStats GmbH & Co. KG. Mobile wallet profiles 2019: Apple Pay, Google Pay, Samsung Pay. March 18, 2019. URL: https://www.ystats.com/market-reports/google-pay-profile-2019 (visited on February 12, 2020).