Relay Attacks on Secure Element-Enabled Mobile Devices*

Virtual Pickpocketing Revisited

Michael Roland[1], Josef Langer[1], and Josef Scharinger[2]

[1] NFC Research Lab Hagenberg, University of Applied Sciences Upper Austria
{michael.roland,josef.langer}@fh-hagenberg.at
[2] Department of Computational Perception, Johannes Kepler University Linz
josef.scharinger@jku.at

Abstract. Near Field Communication's card emulation mode is a way to combine smartcards with a mobile phone. Relay attack scenarios are well-known for contactless smartcards. In the past, relay attacks have only been considered for the case, where an attacker has physical proximity to an NFC-enabled mobile phone. However, a mobile phone introduces a significantly different threat vector. A mobile phone's permanent connectivity to a global network and the possibility to install arbitrary applications permit a significantly improved relay scenario. This paper presents a relay attack scenario where the attacker no longer needs physical proximity to the phone. Instead, simple relay software needs to be distributed to victims' mobile devices. This publication describes this relay attack scenario in detail and assesses its feasibility based on measurement results.

1 Introduction

Near Field Communication (NFC) is an advancement of inductively coupled proximity Radio Frequency Identification (RFID) technology and smartcard technology. NFC has three operating modes: peer-to-peer mode, reader/writer mode and card emulation mode. Peer-to-peer mode is an operating mode specific to NFC and allows two NFC devices to communicate directly with each other. In reader/writer mode, NFC devices can access contactless smartcards, RFID transponders and NFC tags. In card emulation mode, an NFC device emulates a contactless smartcard and, thus, is able to communicate with existing RFID readers.

Three communication standards are available for card emulation mode: ISO/IEC 14443 Type A, Type B and FeliCa (JIS X 6319-4). Which mode is actually used depends on the NFC chipset and the geographic region. With current NFC-enabled mobile phones, card emulation is usually based on Type A.

* This work is part of the project "4EMOBILITY" within the EU programme "Regionale Wettbewerbsfähigkeit OÖ 2007–2013 (Regio 13)" funded by the European regional development fund (ERDF) and the Province of Upper Austria (Land Oberösterreich).

D. Gritzalis, S. Furnell, and M. Theoharidou (Eds.): SEC 2012, IFIP AICT 376, pp. 1–12, 2012.

Besides the communication standard, card emulation may also vary in the way how card emulation is performed. On the one hand, a card can be emulated in software (e.g. on the device's application processor). On the other hand, card emulation can be performed by a dedicated smartcard chip – a so-called secure element (SE). Such a chip can be a dedicated SE IC (integrated circuit) that is embedded into the NFC device. Another possibility is the combination of the SE functionality with another smartcard/security device that is used within the NFC device – like a UICC (universal integrated circuit card; often referred to as Subscriber Identity Module/SIM card) or an SD (secure digital) memory card.

Typical use-cases for card emulation are security critical applications such as access control and payment. Therefore, emulation by software on a non-secure application processor is not widely used. As of today, only some dedicated NFC reader devices – like ACS's ACR 122U – and only a small number of NFC-enabled mobile phones – specifically those equipped with RIM's BlackBerry 7 operating system [1, 12] – support software card emulation.

The majority of mobile NFC devices use dedicated smartcard chips for card emulation. Examples are the Nokia 6131, the Nokia 6212, the Samsung GT-S5230N ("Player One") and the Samsung Nexus S. The Nokia 6131 and the Nokia 6212 have an embedded SE. The Samsung GT-S5230N uses an SWP-enabled (Single Wire Protocol) UICC as SE and the Samsung Nexus S has both, an embedded SE and support for an SWP-enabled UICC. Only some recent NFC phones developed by Nokia have no support for card emulation at all [11].

Typical secure elements are standard smartcard ICs as used for regular contact and contactless smartcards. The only difference is the interface they provide: Instead (or in addition) to a classic smartcard interface, the secure element has a direct interface[1] for the connection to the NFC controller.

As secure elements have the same hardware and software platforms as regular smartcards, they also share the same security standards. A secure element provides secure storage, a secure execution environment and hardware-based support for cryptographic operations. The IC is protected against various attacks that aim at retrieval or manipulation of stored data and processed operations. Smartcard chips and their design process are usually evaluated and certified according to high security standards. The same applies to their operating systems. Thus, the secure element fulfills the requirements necessary for security critical applications like access control and even payment.

While smartcards by themselves are considered safe and secure, especially contactless cards are vulnerable to relay attacks. This issue is well-known (see [6, 7, 10]) but can be circumvented by isolating the card from the surrounding world when it is not in use. However, the integration of smartcard functionality into NFC-enabled mobile phones introduces a new and unexplored attack vector. For example, a fundamental difference between a regular smartcard and an NFC-enabled mobile phone is the network connectivity: A smartcard can easily be isolated from the surrounding world by means of shielding. In contrast to that, a mobile phone and its emulated smartcard are permanently connected to a global

[1] E.g. NFC Wired Interface (NFC-WI) or Single Wire Protocol (SWP).

network (cf. intrusion paths identified by Jeon et al. [9]). A further problem arises from the possibility to install arbitrary – possibly untrusted – applications on current mobile phones (specifically smart phones).

In this publication we investigate the potential of these new and unexplored weaknesses. We focus on a relay attack scenario which could be abused to remotely use a victim's emulated smartcard without the victim's knowledge. A system for relay attacks over the internet is introduced. Measurement results of the delays induced by this relay system are provided to verify this relay system. Finally, we evaluate the feasibility of relay attacks based on these measurement results.

1.1 Smartcard Communication

Low-level communication protocols for smartcards depend on the communication interface and are either character-based or frame-based. For contactless smartcards a frame-based protocol is standardized in ISO/IEC 14443. Application level protocols attach on top of these low-level protocols.

An application level communication protocol for smartcards is defined in ISO/IEC 7816-4. This protocol applies to both, contact and contactless smartcards. Command-and-response pairs are called APDUs (application protocol data units). Commands are always sent from the reader to the card while responses are always sent from the card to the reader.

1.2 Android's Secure Element API

While Android-based NFC-enabled mobile phones, like the Nexus S or the Galaxy Nexus, have an embedded secure element and also support UICC-based secure elements, there currently is no public API to access the secure element on the Android platform.

However, Google has already introduced its Google Wallet, which is available for the Nexus S in certain regions. For this wallet to work, Google secretly integrated an API called `com.android.nfc_extras` into their Android platform. This API can be used to access the embedded secure element and is available since Android 2.3.4. Yet, this interface is not included in the public software development kit (SDK) and, thus, is hidden from the average programmer.

The secure element API consists of two classes: `NfcAdapterExtras` and `NfcExecutionEnvironment`. `NfcAdapterExtras` is used to enable and disable external card emulation (`setCardEmulationRoute()`) and to retrieve an instance of the secure element's `NfcExecutionEnvironment` class (`getEmbeddedExecutionEnvironment()`). `NfcExecutionEnvironment` is used to exchange APDUs with the embedded secure element. This class provides methods to open and close the internal connection to the secure element (`open()`, `close()`) and to exchange APDU sequences with the secure element (`transceive()`).

In Android 2.3.4 this secure element API could be accessed by any application that held the permission to use NFC. In later versions this has been changed to a special permission named `com.android.nfc.permission.NFCEE_ADMIN`. This

special permission is only granted to applications which are signed with the same certificate as the NFC system service. Consequently, access to the secure element is restricted to applications trusted by the manufacturer/provider of the NFC system service.

2 Related Work

In [13], we evaluate APIs for SE access on various platforms. The level of protection for such APIs varies widely. A major weakness, that all access control schemes for SE APIs have in common, is that they all require the operating system and the mobile device hardware to be trusted. For instance, an attacker who has control over the operating system or the device hardware can either bypass security measures of the SE API or even bypass the whole SE API.

We further introduce two new attack scenarios against secure element-enabled mobile phones [13]:

- A denial of service (DoS) attack that can be abused to permanently lock an embedded SE and, consequently, render an NFC-enabled mobile phone unusable for card emulation applications.
- A relay attack that can be abused to access a SE from anywhere over an Internet connection.

Both attack scenarios can be applied to several existing NFC-enabled mobile phones, e.g. Nokia 6131, Nokia 6212 and Samsung Nexus S.

Hancke [6] first presented a successful relay attack against ISO/IEC 14443 smartcards in 2005. Compared to that scenario, where the demodulated RF signals are transmitted over an alternative carrier, the relay attack introduced in [13] relays application level commands (APDUs). This is necessary as SE APIs typically provide an APDU-based interface.

Application layer security protocols cannot prevent relay attacks (cf. *Mafia fraud* by Desmedt et al. [2] and conclusions about RFID and NFC protocols by Hancke [6], Kfir and Wool [10] and Francis et al. [4]). The reason is that the relay can be seen as an elongation of the communication medium. Messages are transferred through the relay channel without any modification. Thus, the only measureable difference is the propagation delay through the relay channel.

3 Attacking Mobile Phones

The application programming interface and the resources of mobile devices are usually restricted by access control policies. Access to critical parts of the system is typically shielded from untrusted applications. However, these are usually software-based restrictions enforced by the operating system. Thus, if an untrusted application can manipulate the behavior of the operating system or if it can elevate its privileges to the level of a trusted application, it can easily circumvent any access restrictions.

For many mobile phone platforms there exist methods to circumvent security measures. Popular techniques used on many smart phones are "jail breaking" and "rooting". Jail breaking refers to escaping the restrictions imposed by the operating system, so that an application can access resources it usually could not access. Rooting refers to an even sever scenario where the user or an application gains full access to the whole system. Both methods are often used intentionally by device owners/legitimate users to circumvent digital rights management or to gain "improved" control over their device.

However, intentional jail breaking and rooting imposes a significant security risk. Not only the legitimate user gains access to – otherwise restricted – resources but also an attacker gets these same possibilities. Thus, a jail broken or rooted phone is significantly more vulnerable to attacks.

On the one hand, rooting can be done by using vendor-supplied methods. Such methods typically exist for development phones (e.g. for the Google Nexus series of Android smart phones). They are usually implemented in a safe way that protects the user from malicious activities. For example, rooting a Google Nexus phone according to the official instructions will wipe all data on the phone. Thus, this method of rooting cannot be used to gain access to sensitive user data that resides on the device.

On the other hand, jail breaking and rooting can be done by exploiting vulnerabilities in software or hardware. These exploits are not only viable for intentional jail breaking and rooting by the device owners/legitimate users. The same exploits can be integrated in virtually any application. That way a malicious application could elevate its permissions without the (legitimate) user's knowledge.

Lately the topic of mobile phone security experiences significantly increasing awareness. Recent research activities include the assessment of vulnerabilities and threats and the uncovering of actual attack scenarios. According to Kaspersky Lab's monthly malware statistics [5] the trend towards threats and malware for Android (and mobile platforms in general) has dramatically increased within the last year. Therefore, it seems unlikely that this trend will be interrupted any time soon.

This is trend is confirmed by the most recent exploits for the Android platform:

- Levitator[2], published in Oct. 2011, works up to Android 2.3.5,
- zergRush[3], published in Oct. 2011, works up to Android 2.3.3,
- GingerBreak[4], published in Apr. 2011, works up to Android 2.3.3,
- ZimperLich, KillingInTheName, RageAgainstTheCage, Exploid and others for earlier versions.

Soon after a vulnerability gets fixed, a new exploit is published. If this trend continues, it's only a matter of time until exploits for the most recent versions of the Android platform become available.

[2] `http://jon.oberheide.org/files/levitator.c`
[3] `http://forum.xda-developers.com/showthread.php?t=1296916`
[4] `http://c-skills.blogspot.com/2011/04/yummy-yummy-gingerbreak.html`

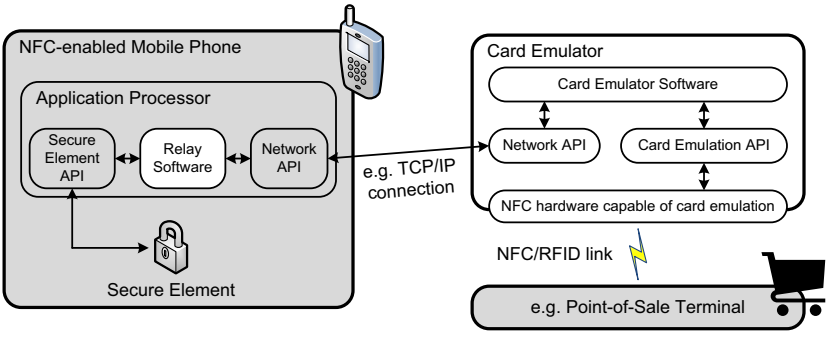


Fig. 1. Relay scenario: Relay software is installed on the victim's phone. The software relays APDUs between the secure element and the card emulator across a network (cellular network, WiFi, Bluetooth...) The card emulator emulates a contactless smartcard that interacts with a card reader (point-of-sale terminal, access control reader...) The card emulator routes all APDU commands received from the point-of-sale terminal through the network interface to the relay software on the victim's mobile phone. As soon as the response APDU is received from the relay software, it is forwarded to the reader.

As Höbarth and Mayrhofer [8] show, it is even possible to create frameworks for permanent on-device privilege escalation. Such a framework would use the most recent exploits for a certain platform to gain temporary super-user privileges. These elevated privileges would then be used to permanently root the device. Such a framework can be integrated by an attacker into any malicious application.

4 Relay Attack on the Secure Element

In [13], we initially proposed a relay scenario that allows an attacker to remotely use a victim's secure element over a network connection. At, for instance, a point-of-sale or an access control gate, nobody would suspect that the communication is actually relayed to a remote device.

The scenario of the relay attack is shown in Fig. 1. It consists of four parts:

- a mobile phone (under control of its owner/legitimate user),
- a relay software (under control of the attacker),
- a card emulator (under control of the attacker), and
- a reader device (e.g. at a point-of-sale terminal or at an access control gate).

The relay software is installed on the victim's mobile phone. This application is assumed to have the privileges necessary for access to the secure element and for communicating over a network. These privileges can be either explicitly granted to the application or acquired by means of a privilege escalation attack. The

relay application waits for APDU commands on a network socket and forwards these APDUs to the secure element. The responses are then sent back through the network socket.

The card emulator is a device that is capable of emulating a contactless smartcard in software. The emulator has RFID/NFC hardware that acts as a contactless smartcard when put in front of a smartcard reader. The emulator software forwards the APDU commands (and responses) between a network socket and the emulator's RFID/NFC hardware.

There are several different options when choosing a device for card emulation:

- *Building a new device from scratch:* This method gives full control over the whole design process. The card emulator can be put into any inconspicuous looking shape. The whole RFID protocol stack can be controlled starting from the lowest layer. Thus, even the protocol levels below APDU communication can be influenced. For instance the unique identifier (UID) that is used during the anti-collision sequence can be freely chosen. However, building a new emulator device from scratch also involves the highest design costs and effort.
- *Using a ready-made RFID card emulation device:* Card emulation devices – like the IAIK HF DemoTag[5] or the Proxmark[6] – already provide the hardware platform and a rudimentary software stack for card emulation. With this choice, it is still possible to control the whole RFID protocol stack. However, the ready-made devices cannot easily be fit into any desired shape.
- *Using an NFC reader:* Some NFC reader devices – like the ACS ACR 122U – can be put into software card emulation mode. In this mode, the device waits for APDU commands from an external reader device and forwards them to the computer. The computer then generates a response that is forwarded to the reader. The lower protocol layers are handled automatically by the reader firmware. One disadvantage of this approach is that typical NFC chipsets do not allow the user to freely choose all parameters for the lower protocol layers. For example, with the ACR 122U, the unique identifier for the anti-collision procedure can only start with `'08'`, which denotes a random UID. Another disadvantage is that this device can only emulate the ISO/IEC 14443 Type A communication protocol.
- *Using an NFC-enabled mobile phone:* Another alternative is the use of NFC-enabled mobile phones as software card emulation devices. Currently, only RIM's BlackBerry mobile phones are known to support software card emulation. Yet, other mobile phones' firmware could possibly be adapted to support software card emulation as well. Using a mobile phone as card emulation device has several advantages. First, the mobile phone already has the form-factor that is expected for NFC contactless transactions (i.e. as the device that actually carries the secure element). Second, the mobile phone has a network interface that can be used to connect to the relay software.

[5] `http://jce.iaik.tugraz.at/Products/RFID-Components/HF-RFID-Demo-Tag`
[6] `http://www.proxmark.org/`

Third, the mobile phone has all the processing capabilities to transfer APDU commands between its network interface and its card emulation hardware. The API documentation for the BlackBerry software card emulation API [12] suggests, that both, ISO/IEC 14443 Type A and Type B, communication protocols can be emulated and, that even low-level parameters – like the UID – can be freely chosen.

4.1 Limitations by the Communication Protocol

Hancke et al. [7] conclude that the timing constraints of ISO/IEC 14443 are too loose to provide adequate protection against relay attacks.

ISO/IEC 14443 specifies certain delays and timeouts. First, there is the *frame delay time* (ISO/IEC 14443-3 Type A) between commands sent by the reader and responses sent by the card. For commands that are used during anti-collision, the *frame delay time* defines a strict timing between requests and responses. This is necessary to detect collisions during the anti-collision procedure. For all other commands, the *frame delay time* specifies only a minimum delay. As our relay system operates on the APDU layer, these delays are handled by the card emulator and do not apply to the relay path.

Second, there is the *frame waiting time* (ISO/IEC 14443-4). The *frame waiting time* specifies the maximum timeout between a command frame sent by the reader and the response received from the card. This timeout is defined by the card and can range between about 302 us and 4949 ms. The timeout can be extended on a per-command basis by the card using *frame waiting time extension* commands. Thus, this timeout does not affect the APDU layer. The timeout extension can be handled by the card emulator and does not apply to the relay path. Even if *frame waiting time extension* would not be used, relaying APDUs may take almost 5 seconds without violating this timeout.

4.2 Implementation

We implemented a proof-of-concept of the relay system to verify our assumptions. Our relay system (see Fig. 2) consists of the following parts:

- Samsung Nexus S with Android 2.3.4,
- relay software (Android app) that accesses the hidden secure element API (`com.android.nfc_extras`) and relays commands over a TCP socket,
- card emulation software (Python script) that controls the card emulation hardware and relays commands over a TCP socket,
- ACS ACR 122U NFC reader in software card emulation mode,
- HID OMNIKEY 5321 USB contactless reader,
- reader application (Java SE).

As we did not have access to a mobile phone that had *real* applications (e.g. a credit card or an access control applet) on its secure element, we decided to access the GlobalPlatform card manager application (issuer security domain) for our tests of the relay system. Therefore, our reader application sent the following APDU commands:

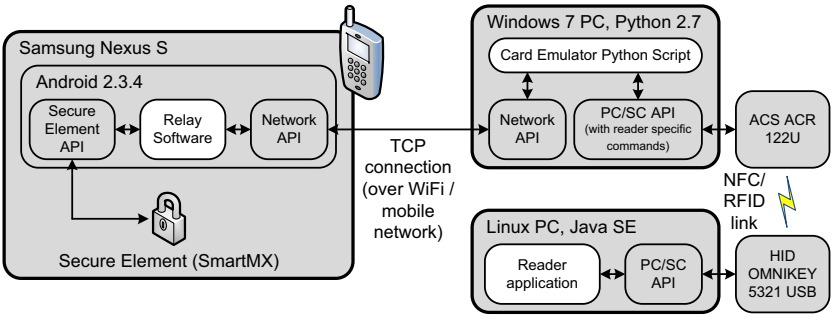


Fig. 2. Proof-of-concept relay system: Pre-existing components are drawn with gray background. Our customized components are drawn with white background.

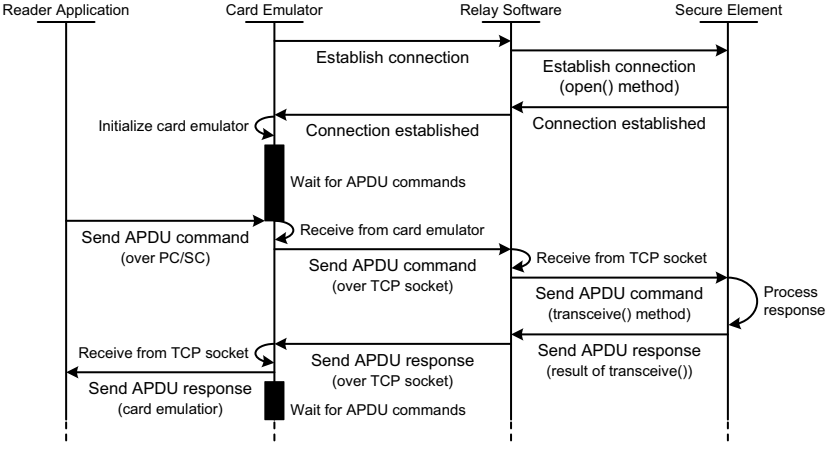
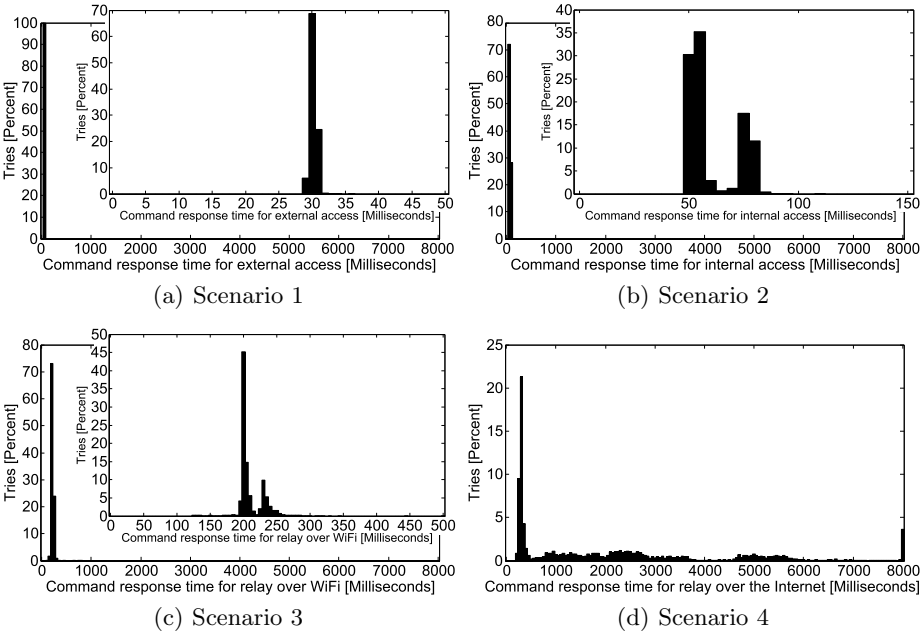


Fig. 3. Command/APDU flow diagram

1. SELECT card manager by AID: `00A4040008A000000003000000`, expected response: File control information template (105 byte)
2. GET_DATA for data object `'65'`: `00CA006500`, expected response: Reference data not found error (2 byte)
3. GET_DATA for data object `'66'`: `00CA006600`, expected response: Card data/security domain management data (78 byte)

Fig. 3 shows the command/APDU flow diagram of the relay system. Initially the card emulator establishes a connection to the victim's secure element. When the card emulator is put in range of an RFID/NFC reader, the reader sends an APDU command. This command is forwarded to the secure element. The secure element generates a response and the relay software, then, returns this response back through the card emulator to the reader.



(a) Scenario 1 (b) Scenario 2

(c) Scenario 3 (d) Scenario 4

Fig. 4. Histograms of delay between command and response at the reader side for the APDU "SELECT card manager by AID" for 5000 repetitions. The histogram is divided into 160 bins. Each bin has a width of 50 ms. The last bin also contains all measurements above 8000 ms. (a) is zoomed from 0 to 50 ms with 1-ms-bins. (b) is zoomed from 0 to 150 ms with 5-ms-bins. (c) is zoomed from 0 to 500 ms with 5-ms-bins.

In our implementation we designed the card emulator as a server to which the relay software on any victim's mobile phone can connect. That way, the card emulator can choose one of the connected secure elements as soon as a reader is in range. This method also bypasses firewalls that protect the victim's device from incoming TCP connections.

4.3 Measurement Results

To verify the feasibility of our relay system, we compared four different scenarios:

1. Direct access to the secure element with an external reader (i.e. no relay),
2. direct access to the secure element with an app on the phone,
3. access through the relay system using a direct WiFi link between the phone and the card emulator,
4. access through the relay system using the mobile phone network and an Internet link between the phone and the card emulator.

Fig. 4 shows the histograms of the delay between command and response at the reader side for the APDU "SELECT card manager by AID" for 5000 repetitions of the command-response sequence. The other APDUs mentioned in section 4.2

lead to similar results that only differed in delays due to command lengths. Except for scenario 2, the phone/the emulator was isolated from the reader between each repetition.

The delay for scenario 1 centers on about 30 ms. On-device access to the secure element (scenario 2) already takes significantly longer (50 to 80 ms). The delay over a WiFi connection (scenario 3) ranges from 190 to 260 ms. Thus, the WiFi relay link adds a delay in the range of 100 and 210 ms. For scenario 4, the delays start at about 200 ms and have a significant peak around 300 ms. Yet, more than 55 percent of the measured delays are above 1000 ms, more than 19 percent are above 4000 ms, and more than 2 percent are above 10000 ms.

Typical limits for contactless transactions in transport ticketing and payment are between 300 to 500 ms (cf. [14]). These limits apply to overall transactions, which, typically, consist of multiple command-response pairs. The EMV specification for contactless payment systems [3] specifies a limit of 500 ms for a contactless payment transaction. However, a payment terminal is not required to interrupt a transaction if it takes longer than this limit. The limit is merely meant as a benchmark target to maintain user experience.

Consequently, both relay scenarios are likely to fail these timings if transactions consist of several command-response pairs. Nevertheless, as the limits are not meant as hard timeouts (after which transactions are canceled), we do not see any problematic side effects of failed timings. For the relay scenario across the mobile network and the Internet, most of the tests showed a delay below 4000 ms. While this is significantly longer than typical delays for contactless transactions, it is still below the timeout limits (without timeout extension) imposed by the ISO/IEC 14443 standard. As contactless transactions (especially with mobile phones) are quite new and users are still not used to them, we assume that even long delays (in the order of 20 to 30 seconds per transaction) will not raise suspicions.

5 Conclusion and Outlook

We presented a new relay attack scenario against secure elements. With this scenario an attacker can use a secure element in a remote mobile phone over the Internet. Due to limited security features of current mobile phone systems, attackers are likely to perform such attacks even if the secure element APIs have mechanisms to prevent unauthorized access.

We described a possible implementation based on Android devices. We conducted several measurements to prove the feasibility of our implementation. The results show that our attack scenario is technically possible due to the lack of strict timing requirements in the communication protocols. Attacks are possible even over long distances. Nevertheless, a relay attack induces significantly longer delays than with usual contactless transactions.

Future work is necessary to find adequate solutions for avoiding on-device relay attacks. A possible direction would be the adoption of trusted computing concepts for mobile phone systems. That way, the SE could recognize if the

application processor is in a trusted state. Another possibility is the introduction of strict timeouts in contactless transactions. Timeouts could also be based on a history of measured command-response delays from previous transactions to detect significant deviations in comparison to previous transactions. A third possibility would be the explicit activation of card emulation by user interaction (e.g. by pressing a button on the mobile phone that is directly connected to the SE.) However, this would significantly complicate over-the-air card management.

References

1. Clark, S.: RIM releases BlackBerry NFC APIs. Near Field Communications World (May 2011), http://www.nfcworld.com/2011/05/31/37778/rim-releases-blackberry-nfc-apis/
2. Desmedt, Y., Goutier, C., Bengio, S.: Special Uses and Abuses of the Fiat-Shamir Passport Protocol (extended abstract). In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 21–39. Springer, Heidelberg (1988)
3. EMVCo: EMV Contactless Specifications for Payment Systems – Book A: Architecture and General Requirements, Version 2.1 (March 2011)
4. Francis, L., Hancke, G., Mayes, K., Markantonakis, K.: Practical NFC Peer-to-Peer Relay Attack Using Mobile Phones. In: Ors Yalcin, S.B. (ed.) RFIDSec 2010. LNCS, vol. 6370, pp. 35–49. Springer, Heidelberg (2010)
5. Gostev, A.: Monthly Malware Statistics: August 2011 (September 2011), http://www.securelist.com/analysis/204792190
6. Hancke, G.P.: A Practical Relay Attack on ISO 14443 Proximity Cards (January 2005), http://www.rfidblog.org.uk/hancke-rfidrelay.pdf (retrieved September 20, 2011)
7. Hancke, G.P., Mayes, K.E., Markantonakis, K.: Confidence in smart token proximity: Relay attacks revisited. Computers & Security 28(7), 615–627 (2009)
8. Höbarth, S., Mayrhofer, R.: A framework for on-device privilege escalation exploit execution on Android. In: Proceedings of IWSSI/SPMU (June 2011)
9. Jeon, W., Kim, J., Lee, Y., Won, D.: A Practical Analysis of Smartphone Security. In: Smith, M.J., Salvendy, G. (eds.) HCII 2011, Part I. LNCS, vol. 6771, pp. 311–320. Springer, Heidelberg (2011)
10. Kfir, Z., Wool, A.: Picking Virtual Pockets using Relay Attacks on Contactless Smartcard. In: Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM 2005), pp. 47–58 (September 2005)
11. McLean, H.: Nokia: No mobile wallet support in current NFC phones. Near Field Communications World (July 2011), http://www.nfcworld.com/2011/07/21/38715/nokia-no-mobile-wallet-support-in-current-nfc-phones/
12. RIM: Blackberry API 7.0.0: Package net.rim.device.api.io.nfc.emulation (2011), http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/io/nfc/emulation/package-summary.html
13. Roland, M., Langer, J., Scharinger, J.: Practical Attack Scenarios on Secure Element-enabled Mobile Devices. In: Proceedings of the Fourth International Workshop on Near Field Communication (NFC 2012), Helsinki, Finland, p. 6 (March 2012)
14. Smart Card Alliance: Transit and Contactless Open Payments: An Emerging Approach for Fare Collection (November 2011), http://www.smartcardalliance.org/resources/pdf/Open_Payments_WP_110811.pdf