Performance Variation in Host-Based Card Emulation Compared to a Hardware Security Element

Assad Umar*, Keith Mayes†, and Konstantinos Markantonakis‡
Information Security Group, Smart Card Centre, Royal Holloway, University of London, Egham Hill, Egham, Surrey. United Kingdom
*Email: * Assad.Umar.2011@rhul.ac.uk, † Keith.Mayes@rhul.ac.uk, ‡ K.Markantonakis@rhul.ac.uk*

Abstract

Traditionally, card emulation mode in Near Field Communication devices makes use of a hardware Secure Element (SE) as a secure storage and execution environment for applications. However, a different way of card emulation that bypasses the SE has emerged, referred to as Host-based Card Emulation (HCE). HCE relies on the phone CPU for processing power, sharing it with other running processes. This produces variable readings in terms of response times from the phone. This paper investigates this variability in HCE implementation as compared to an SE implementation. We also discuss how our findings may call into question the use of HCE in time critical scenarios.

1. INTRODUCTION

Near Field Communication (NFC), has showed a lot of promise in a number of markets since its emergence on phones and other mobile devices. NFC devices operate in three modes; *reader/writer mode*, *peer-to-peer mode*, and the *card emulation mode* (1). The card emulation mode is arguably the most interesting because it allows an NFC device to communicate with a contactless terminal as if it were a smart card. This means that NFC devices can be compatible with existing contactless infrastructure. As card emulation is security sensitive, NFC devices incorporate a Secure Element (SE). The SE was originally envisaged as a small tamper-resistant chip (or part of an existing security chip) conforming to high security standards such as the Common Criteria (2).

While card emulation has proven to be attractive for applications such as mobile payments and transport ticketing, access to the SE is tightly controlled by the Mobile Network Operators (MNO) and/or the Original Equipment Manufacturer (OEM). This tight control on the SE means small companies and mobile application developers have no access to the services of the SE and indeed the ability to use the NFC card emulation functionality. This has hindered the use of NFC phones and prevented usage from reaching its full potential. There have been moves to resolve this by using a Trusted Service Manager (TSM), although this has not been universally adopted by businesses and standards. The TSM provides secure application provision and personalization services, key management services as well as post issuance life cycle management.

However, in recent times, a different way of card emulation has emerged, which bypasses the hardware SEs. The approach is referred to as the Host-based Card Emulation (HCE) (3) which is sometimes referred to as the soft SE (4). In HCE, an application running on the Operating System (OS) of the host device can emulate a smart card and interact with an external reader directly; we will refer to these applications as HCE-apps in this paper. Research In Motion (RIM), on the Blackberry platform (4), were the first to incorporate this functionality in their phones. Subsequently Cyanogenmod integrated some patches (5) to the Android OS which permitted NFC enabled mobile phones to perform card emulation from the host. However, HCE attracted most attention when Google incorporated it within Android 4.4 (KitKat).

Unlike the case of the hardware SE where there is a tamper-resistant secure microprocessor, HCE relies on the phone CPU for processing. As the phone OS is multi-threaded, the HCE-app will share resources with other processes requiring CPU usage. This may have direct implication on the performance of HCE-apps, as CPU load of the phone is very variable. This potentially variable characteristic of HCE-apps with regards to processing times adds something rather new to the dynamics. Traditional contactless smart cards have consistent timing and often critical constraints on absolute duration, especially in transport ticketing.

In this paper we investigate variations in HCE execution times as compared to a similar chip-based SE implementation. It should be noted that the comparative security and attack-resistance of HCE and hardware SEs is a hot research topic, although out of scope for this paper

In Section 2, we give some background on the NFC technology, we briefly explain the CPU power management in Android as well as the role played by CPU governors. The way the test was set up and specifications of the devices used are represented in Section 3, followed by the actual testing carried out in Section 4. In Section 5, we put our tests together for analysis and comparisons. We also discuss how our findings may have an impact in a number of areas. We discuss future directions of our research and finally conclude the paper in Section 6 .

2. BACKGROUND

In this Section, we give some background information on NFC. and the card emulation options. We conclude the section by briefly explaining the way in which Android manages CPU power and also the role played by the CPU governors in achieving this.

2.1. Near Field Communication (NFC)

Near Field Communication (NFC) (6) is a short range contactless communication technology (sometimes referred to as Radio Frequency Identification (RFID)) which enables the exchange of data between devices, it typically works within the range of less than 10 cm and at a Radio Frequency (RF) of 13.56 MHz. An NFC device may either be passive or active. An active NFC device (known as the initiator) is capable of producing its own electromagnetic field and directly transmitting data, while a passive NFC device (sometimes referred to as a target or a tag) relies on the initiator's electromagnetic field for power and clock, and uses load modulation for data transmission. The communication between the two entities is halfduplex, meaning only one entity can talk at a time. NFC is standardised by the NFC Forum (1) and is based on legacy standards such as ISO/IEC 14443 (7) as well as the ECMA (8) standards. NFC is therefore compatible with existing and widespread contactless/RFID infrastructure, such as found in payment, transport, identity and access control systems, making it a favourable technology for a lot of industries. NFC works in three different modes; *Reader/Writer Mode*, *Peer-to-Peer mode*, and *Card Emulation mode*, although the first two modes are beyond the scope of this paper.

2.1.1. Card Emulation Mode. In card emulation mode, the NFC device, typically behaves like an ISO14443 compliant smart card. Card emulation can be done in a number of ways. Traditionally it was done using the hardware SE, the idea been that the SE provides a safe and secure environment to house applications, sensitive data and cryptographic credentials, as well as a secure execution environment. These SEs are very similar to conventional smart cards in terms of hardware although the interface differs, with the SE connected to the NFC controller through the NFC wired interface (NFC-WI) or the single wire protocol (SWP) (9). In a conventional SE-based approach, when a user taps his device on a terminal, by default the NFC controller routes all messages from the terminal to the corresponding application residing in the SE. The applications are identified through their Application Identifier (AID). The hardware SE comes in three different forms:

Embedded SE. The SE in this case is soldered into the hardware of the mobile device at the production phase, and is not intended to be removable. The embedded SE is owned and managed (at least initially) by the OEMs.

Universal Integrated Circuit Card (UICC) SE. The Subscriber identity Module (SIM) in the mobile phone can also be used as the SE and the SWP acts as the interface for connection to the NFC controller. The SE in this form factor is owned and managed by the MNOs.

MicroSD-based SE. The SE could also be in the form of a memory card inserted into the mobile device. While this form factor offers the most flexibility in terms of SE ownership and control, it has practical drawbacks. Each Service Provider (SP) will have to issue a microSD card to all its customers, who will then have to swap memory cards when using a service from a different SP.

2.2. Host Card Emulation

HCE is a radical change to card emulation that to some extent bypasses the MNO and OEM control of SEs. Prior to HCE, all messages from a card terminal are routed directly to the SE. However with HCE, the host OS decides what should handle the messages, which could be an SE, but is more likely to be an application running on the host CPU. HCE, as far as Android OS is concerned, is implemented as a service, i.e. it can run in the background without a user interface (3). This is a feature very important to a number of NFC scenarios such as transport ticketing where a user does not have to launch the application before using it. A simple tap of the phone on the reader invokes the correct application to handle the transaction.

An application that wishes to emulate a smart card registers its AID with the NFC controller, which maintains a routing table containing the routing rules. Each rule is a mapping of an AID to an application. When a user taps a phone on a terminal, the first APDU the phone receives is a SELECT command, this command contains the AID of the application it wants to talk to. The NFC controller uses this AID to apply a necessary routing rule, and all subsequent commands are sent to the selected application until another application is selected, or the current one is deselected.

2.3. Android CPU Policy

Android devices typically have a multi-core processor design. This means that a single unit can consist of two or more more independent CPUs referred to as cores. Android devices use both the ARM and x86 architectures. Android also uses the Symmetric Multiprocessing (SMP) design for managing the different CPUs (10). Normally, all CPUs share the same CPU frequency policy, i.e. all CPUs are online at the same time and any process can run on any of them. Nevertheless, Android devices switch between different frequency levels in response to variable CPU load, this is enforced and regulated by a driver known as the CPU governor (11). There are several types of CPU governors with different characteristics. In this paper, we used the "ondemand" governor which is the factory default for most Android phones, it increases and decreases CPU frequency according to demand. We also used the "userspace" governor: it permits a user to choose which frequency state the CPU should run in.

3. TEST SETUP

In this Section we explain how the testing was set-up and conducted. To test the performance with regards to variable delays, we developed two card emulating applications. The first one is a Java card applet based on Java card framework v2.2.1 (12). We refer to this applet as the SE-app. We loaded the SE-app into the SE of a Nokia 6131 (one of the first NFC phones). Nokia, in (13) explains how to use the Nokia NFC Unlock Service MIDlet to unlock the SE of the Nokia 6131. The second application is an Android application (HCE-app), running on Android platform version 4.4 with Android framework Application Programming Interface (API) level 19 as the target API. This application is deployed within a Nexus 5 mobile device. Table 1 provides a summary of the devices used to conduct the testing.

Both applications run the same cryptographic protocol designed for the tests. The purpose of the protocol is simply to require the emulating processor to carry out some non-trivial and representative cryptographic processing We used the *javacardx.crypto* package for the SE-app, and the *java.crypto* package for the HCE-app.

We created a 1024-bit RSA algorithm keypair, and to simplify testing, we departed from cryptographic best practice not only by using this key-size, but also by using the same pair for both encryption and signing. For the encryption we used the *ALG_RSA_PKCS1* field of the *cipher* class from the *javacardx.crypto* package. *ALG_RSA_PKCS*1 uses the RSA cipher for encryption and the Public-key cryptography standards (PKCS1) (v1.5) for data padding. For the signature, we used *ALG_RSA_SHA_PKCS1* field of the *cipher* class from the *javacardx.crypto* package. *ALG_RSA_SHA_PKCS1* firstly computes a hash of the message resulting in a 20-byte SHA digest, which is then padded using the PKCS1 before signing using the RSA algorithm.

We recognise that SHA-1 is no longer recommended, however the goal was to test performance variation rather than establish a secure protocol. For performance testing, we first used the Java *timer* class in the terminal application to take measurements. To ensure accuracy of the values produced by the timer in our program, we then used the CLT Move- Contactless Spy tool from COMPRION (14) to double check the measurements. This tool monitors the flow of APDUs between the reader and the phone, giving a byte level view of messages and their corresponding delays.

4. EXPERIMENTAL RESULTS

The first phase of any communication between a card/phone and a smart card reader is the initialisation and anti-collision phase; this (6) is where they establish identities and agree on the parameters of communication for all subsequent messages. The parameters include the number of bits they can handle at a time and also the agreed amount of time the reader has to wait for a response from the phone before it times out, known as the Frame Waiting Time (FWT). The anti-collision phase exists to resolve the situation of multiple smart cards within range of a single terminal, which was not case for our tests.

Table 1. Devices used in the testing

Device	Manufacturer	Operating System	RAM	ROM
SE (Nokia)	NXP Semiconductors	SmartCafe expert 3.1(G&D)	4kb	160kb
Phone (Nexus 5)	LG Electronics	Android kit-kat 4.4	2GB	
Laptop	SONY	Ubuntu 14.04.1 trusty	4GB	
Card Reader	SCM Microsystems			

After a successful initialisation and anti-collision, the reader application selects the applet by sending a select APDU using the AID; the application sends back the 0x9000 status word if the select APDU was successful. The reader then sends a command APDU (ENCRYPT) with an 11-byte data to be encrypted by the card emulating application. The application encrypts the message and sends back a response APDU containing the corresponding cipher text together with the 0x9000 status word (or error code on failure). Subsequently another command APDU (SIGN) is sent to the phone in order to get a signature on the 8-byte data contained in the APDU. The phone signs the data and sends the result back to the reader. Timing is measured from the time the command is sent from the reader to the time it recieves the response back. For accuracy, we carried out 400 hundred runs of the protocol and computed the average. All timings were taken in milliseconds(ms).

4.1. SE-app TESTING

Table 2 shows a statistical analysis of the results from the SE-app on the Nokia hardware SE. It shows the statistics for individual commands as well the protocol in full over 400 runs.

Table 2. Readings from the SE-app testing

SE-app	SELECT	SIGN	ENCRYPT	FULL PROTOCOL
Average	18.6	1679	285	1982
Mode	18	1691	283	2002
Median	18	1680	284	1984
MAX	116	1746	300	2105
MIN	17	1618	279	1917

From the table we see that the SELECT command had an average execution time 18.6ms. SELECT is a fairly light weight command requiring no cryptographic processing and simply returning a status response. However, the first SELECT command of the 400 runs took 116ms. This is because of the initialisation and anti-collision phase which is not present in the remaining runs, which have very consistent execution times with the minimum and maximum times differing by at most 3ms. It is important to note that in a real-life scenario, for every tap of the phone unto a terminal by the user, there will be an initialisation and anti-collision phase. Overall our test protocol has an average execution time of 1982ms, with about 85 percent of the time spent on digitally signing the message. The absolute duration would be too long for timing critical applications (e.g. transport ticketing), however the measurements are very consistent, with some variation due to experimental tolerances.

4.2. HCE-app TESTING

For the HCE-app, we did things slightly differently; we ran the test in two different scenarios which we will refer to as CASE1 and CASE2 from now on. As suggested in (15) “While measuring CPU power, or holding CPU power constant in order to make other measurements, it may be best to hold the number of CPUs brought online constant, such as to have one CPU online and the rest offline (hotplugged out)”. In our case, the Nexus 5 has a multi core processor with four cores, so we hotplugged three of them, leaving only one online. This was to ensure that all processes ran on the same CPU giving us better control of the testing platform.

4.2.1. CASE1. Here, we used the “ondemand” governor running at 960Mhz (default). While running the program, we deliberately tried to simulate the day-to-day things a mobile phone user will usually do such as opening social media applications and answering phone calls. This was to mimic what might be the case/state when a user taps a turnstile at a train station. From the table 3 below, the first SELECT command took longer than the others (as was the case with the SE-app) i.e. 60ms as compared to the average of 12ms. The full protocol ran in an average of 213ms, however it is interesting to see that range between the MAX and MIN values of the full protocol is 708ms which is a very significant variation. Similar variations were measured for SIGN and ENCRYPT values individually, which ran on average of 110ms and 91ms respectively but with a range of 475ms and 404ms respectively.

Table 3 shows a statistical analysis of the results from the HCE-app (CASE1). It shows the statistics for individual commands as well the protocol in full over 400 runs.

Table 3. Readings from the HCE-app testing CASE1

CASE1	SELECT	SIGN	ENCRYPT	FULL PROTOCOL
Average	12	110	91	213
Mode	11	85	82	171
Median	12	95	81	188
MAX	60	544	465	853
MIN	10	69	61	145

4.2.2. CASE2. Table 4 shows a statistical analysis of the results from the HCE-app (CASE2). It shows the statistics for individual commands as well the protocol in full over 400 runs.

Table 4. Readings from the HCE-app testing CASE2

CASE2	SELECT	SIGN	ENCRYPT	FULL PROTOCOL
Average	20	301	250	572
Mode	20	271	235	529
Median	20	294	236	550
MAX	99	883	809	1146
MIN	16	265	222	514

After noticing the high variance in the results produced in CASE1, we decided to clock the CPU at its lowest possible frequency state of 300Mhz. To achieve this we changed the CPU governor from the default “ondemand” governor to the “userspace” governor, with this, we were able to set exactly the CPU frequency. We also set the minimum and maximum frequency to both be 300Mhz as suggested in (15), this gives us assurance that the CPU is fixed at 300Mhz. We ran the protocol under these conditions and table 4 shows the results. As in previous tests the first SELECT command is slower, taking 99ms as compared to the overall average of 20ms. The full protocol took an average of 572ms, but similar to CASE 1, there is a significant range of (632ms). The same applies to the SIGN and ENCRYPT values with an average of 301ms and 250ms respectively, and with a range of 618ms and 587ms respectively.

5. DISCUSSION

In this section, we put all three tests together to make analysis and comparisons.

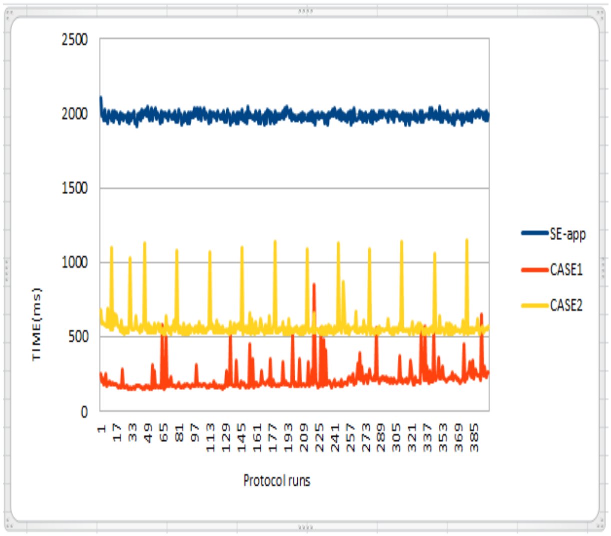

Figure 1. Graph Showing the Results of All the Tests

In Figure 1 above, we can see how the SE-app has a more clustered line graph (almost a constant), while the two HCE-app cases show marked variation in values. We can see how CPU clock affects average execution time for the protocol, however the wide variations exist even when the CPU core is allowed to run at full speed. The absolute execution time of the SE is notably slower than the HCE-apps although this is perhaps an unfair comparison as the Android phone is still current whereas the Nokia phone (and its SE) was produced in 2006. Furthermore, the HCE-app would likely be significantly slower in practice if software security measures were required to reduce side-channel leakage and improve attack resistance.

What is most interesting is the variation in the HCE-app response times, which may call into question HCE use in time critical applications. For example, in transport ticketing applications there is a rule of thumb that a gate transaction should complete in less than 500ms, however we are seeing variations that are greater than this, regardless of the expected execution time.

This timing variation may also prevent the use of some security measures to detect fake cards or attacks in progress. For example, there has been a lot of research and proposals on distance bounding protocols (16) as a way of detecting RFID relay attacks. These protocols establish an upper bound on the distance of the proving party. This is done by taking into consideration the delay introduced into the channel from the time a challenge is sent to the time a response is received. This is only possible when there is a reasonable benchmark for an acceptable delay. In the case of HCE the tolerance around the benchmark will be extremely large making it very difficult to distinguish between a relay attack and variation due to normal phone operation.

6. CONCLUSION AND FUTURE WORK

We have conducted comparative testing on hardware SE-based card emulation and also on an HCE-based card emulation. The main finding was the significant variation in the HCE execution times that could call into question its use in time critical applications (such as transport ticketing) and prevent the use of distance bounding security measures for combating relay attacks. We also measured significant variation in average response times with controlled CPU clock speed, used to mimic loading affects. To build upon these initial findings, further work is planned to consider additional phone platforms and SEs, and the performance analysis of real world transport and payment protocols.

References

[1] NFC-Forum, *NFC Forum Specification Architecture*, NFC-Forum Std., 2006. [Online]. Available: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/,

[2] K. London and K. Markantonakis, *Smart Cards, Tokens, Security and Applications*. Springer, 2007. [Online]. Available: http://books.google.co.uk/books?id=9iwFQarJdBMC

[3] A. D. Guide, "Host-based card emulation," https://developer.android.com/guide/topics/connectivity/nfc/hce.html.

[4] M. Roland, "Software card emulation in nfc-enabled mobile phones: Great advantage or security nightmare," in *Fourth International Workshop on Security and Privacy in spontaneous Interaction and Mobile Phone Use*, Newcastle UK, June 2012.

[5] D. Yeager, "Added nfc reader support for two new tag types: Iso pcd type a and iso pcd type b." https://github.com/CyanogenMod/android_packages_apps_Nfc/commit/d41edfd794d4d0fedd91d561114308f0d5f83878,.

[6] W. Rankl and W. Effing, *Smart Card Handbook*. Wiley, 2010. [Online]. Available: http://books.google.co.uk/books?id=C55-4kVUQ14C

[7] I. O. for Standardization (ISO), *Identification cards – Contactless integrated circuit cards – Proximity cards*, ISO/IEC Std. 14 443, 2008. [Online]. Available: http://www.iso.org/,

[8] ECMA-340, *Near Field Communication - Protocol and Interface (NFCIP-1)*, European Computer Manufacturers Association Std., 2004.

[9] M. Roland and J. Scharinger, "Practical attack scenarios on secure element-enabled mobile devices," in *Fourth International Workshop Near Field Communication*, Helsinki Finland, March 2012.

[10] A. D. Guide, "Smp primer for android," https://developer.android.com/training/articles/smp.html.

[11] S. Tarkoma, M. Siekkinen, E. Lagerspetz, and Y. Xiao, *Smartphone Energy Consumption: Modeling and Optimization*, ser. Smartphone Energy Consumption: Modeling and Optimization. Cambridge University Press, 2014. [Online]. Available: http://books.google.co.uk/books?id=ai0DBAAAQBAJ

[12] S. Microsystems, "Java card platform specification v2.2.1," http://java.sun.com/products/javacard/specs.html,.

[13] Nokia, "Nokia nfc unlock service midlet," Nokia, Tech. Rep., 2007. [Online]. Available: http://developer.nokia.com/community/wiki/images/0/0f/Nokia_NFC_Unlock_MIDlet,

[14] COMPRION, "Clt move - contactless spy tool," http://www.comprion.com/en/products/monitoring/clt_move/overview.

[15] A. D. Guide, "Power profiles for android," https://source.android.com/devices/tech/power.html.

[16] S. Brands and D.Chaum, "Distance-bounding protocols," in *Advances in Cryptology EUROCRYPT '93*, Norway, September 1993.