Overclocking Proximity Checks in Contactless Smartcards

Dominic Celiano
Churchill College

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Department of Computer Science and Technology
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: dtc34@cl.cam.ac.uk

June 8, 2018

Declaration

I, Dominic Celiano, of Churchill College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14,646

Signed:

Date:

Acknowledgements

I would like to thank my supervisor, Dr. Markus Kuhn, for his valuable feedback, discussions, and suggestions. I would also like to thank Sam Ainsworth for reading a draft of my report and providing recommendations.

I would also like to thank the USAFA Francis E. Bennett Scholarship for funding my degree and the US Air Force for supporting me during my program.

Lastly, I would like to thank my family for their continuous support.

Abstract

Contactless smartcards are used in a variety of applications, from access control to payment systems. Various attacks exist against contactless smartcards, but one of the most difficult to defend against is the relay attack. By using two malicious devices, an attacker can relay communication between a valid smartcard and a valid reader across a large distance, thereby using the valid reader without the consent or knowledge of the card holder. This attack can be defended against by implementing a distance-bounding protocol, or proximity check. By using precise timing measurements of how long a card takes to respond, the speed of light, and cryptographic verification, the reader can verify a card is within a certain distance of the reader.

Because a distance-bounding protocol's effectiveness depends upon precise timing measurements, a card's internal timing implementation is of vital importance. However, because contactless smartcards are powered by an electromagnetic carrier, cards often derive their internal clock based solely on that carrier's frequency. If this is the case, overclocking is possible; the carrier frequency can be increased to speed up the smartcard's computation, forcing it to respond quicker during time-sensitive distance-bounding exchanges and allowing a relay attack to once again be carried out.

In this report, I describe experiments on overclocking two popular contactless smartcards: the Mifare Plus EV1 and Mifare DESFire EV2. When overclocked, these cards responded to their distance-bounding requests more than 20% faster than they should have, allowing a relay attack to be conducted at a one-way distance of over 40 km, despite the existence of a distance-bounding protocol.

The first part of this report describes precise timing measurements of the Mifare distance-bounding protocol. The low variance of the timing measurements collected shows an effective implementation of timing on both the Mifare Plus EV1 and Mifare DESFire EV2.

The second part of this report describes overclocking the Mifare Plus EV1 and Mifare DESFire EV2. Both cards were clocked at frequencies above 16.30 MHz – over 20% higher than the 13.56 MHz frequency which they are specified to run at. The consequences of overclocking at 16.30 MHz are demonstrated via an example relay attack that I describe in detail.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Aims and Contributions . . . 4
- 1.3 Report Organization . . . 5

2 Related Work **7**
- 2.1 Relay Attacks . . . 7
  - 2.1.1 Relay Attack Origins . . . 7
  - 2.1.2 Relay Attacks in Practice . . . 9
- 2.2 Distance Bounding . . . 10
  - 2.2.1 Distance-Bounding Protocols . . . 10
  - 2.2.2 Distance Bounding in Practice . . . 12
- 2.3 Miscellaneous Mifare Research . . . 14

3 Design and Implementation **17**
- 3.1 ISO 14443 . . . 17
  - 3.1.1 ISO 14443 Overview . . . 17
  - 3.1.2 ISO 14443A Physical Layer . . . 18
  - 3.1.3 ISO 14443A Timing . . . 20
- 3.2 Proxmark 3 . . . 22
  - 3.2.1 Proxmark 3 Overview . . . 22
  - 3.2.2 Proxmark 3 Software . . . 24
  - 3.2.3 Proxmark 3 Hardware . . . 26
- 3.3 Distance Bounding Timing Measurements . . . 28
  - 3.3.1 Mifare Proximity Check . . . 28
  - 3.3.2 Measuring Timing Using the Proxmark 3 . . . 30
- 3.4 Overclocking . . . 36
  - 3.4.1 Changing the Proxmark 3 Clock . . . 36
  - 3.4.2 Analog Considerations of Overclocking . . . 38

4 Results and Evaluation **45**
4.1 Timing Measurements 45
4.2 Overclocking Results 48
4.3 Overclocking Relay Attack Example 50
4.4 Recommendations 56

5 Conclusion **57**
5.1 Summary 57
5.2 Future Work 58

A Mifare Plus and Mifare DESFire Setup **65**

List of Figures

2.1 Visualization of a normal transaction compared to a relayed transaction. 8
2.2 Overclocking visualized. Image by Henzl et al. [1]. 13

3.1 Sequences for PCD to PICC communication. Image from ISO 14443 [2]. 19
3.2 REQA transaction at 13.56 MHz, with timing annotated. . . . 21
3.3 Proxmark 3 block diagram. 23
3.4 Proxmark 3 analog receive path and DSP at $f_c = 13.56$ MHz. 27
3.5 Mifare proximity check, used on the Mifare Plus EV1 and Mifare DESFire EV2. 29
3.6 Visualization of timing as it relates to PCD timestamps. t_{tot} is the timestamp recorded by the PCD. 31
3.7 Eavesdropping a Mifare DESFire REQA transaction using a near-field probe hooked up to a spectrum analyzer. 35
3.8 Antennas used with a frequency of 13.56 MHz other than the RyscCorp antenna. 39
3.9 Circuit diagram of the RyscCorp antenna. 39
3.10 Circuit diagram used for antenna tuning. 41
3.11 RyscCorp antenna tuning with C3 in parallel. Horizontal axis goes from 10 MHz–30 MHz. 42
3.12 Frequency limitations of the Mifare DESFire. Note the vertical scales are different. 43

4.1 The four relay attack processing delays. Original image by Clulow et al. [3]. 52
4.2 Delay between the eavesdropped PICC response and the mole's serial output. 53

List of Tables

4.1 Mifare Plus EV1 Proximity Check response times, in ms, with $f_c = 13.56$ MHz. 46
4.2 Mifare DESFire EV2 Proximity Check response times, in ms, with $f_c = 13.56$ MHz. 46
4.3 Mifare Plus EV1 Proximity Check response times, in ms, with $f_c = 16.50$ MHz. 49
4.4 Mifare DESFire EV2 Proximity Check response times, in ms, with $f_c = 16.30$ MHz. 50

Chapter 1

Introduction

1.1 Motivation

The flexibility of using smartcards for identification has led to their wide adoption. Contactless smartcards provide more convenience than their contact counterpart, only requiring the user to present his or her card to a reader. Because of this convenience, contactless smartcards are commonly used in access control and payment systems, having recently been integrated into the EMV (Europay, Mastercard, and Visa) standard alongside NFC (near-field communication) mobile phone payments [4]. Contactless smartcards can even be used to perform web authentication [5].

From a security perspective, there are three main attack vectors to compromise a contactless smartcard. The first is to steal a user's card, use physical tampering to recover its keys and memory contents, and produce clones of it. One clone can then be returned to the user without the user having any knowledge that his or her card was stolen in the first place. Defenses against this type of attack include using tamper-resistant smartcards [6] and ensuring a card does not get stolen.

The second is for an attacker to use a "rogue" reader. Unlike contact smartcards, contactless smartcards do not require the user's card to be physically

inserted into a terminal. Because of this, a rogue reader can be held up to a user's card, likely while the user is in a crowded space, and used to perform a transaction without the knowledge of the user. This attack vector can be defended against by a user keeping his or her smartcard in a Faraday cage such as a specially designed wallet [7]. Another defense is placing a sheet of metal geometrically parallel with a card to have the same effect as a Faraday cage and prevent the card from being powered up. The latter defense is implemented in US passports, which use an "anti-skimming" cover to prevent the passport's RFID tag from being read while the passport is closed [8].

The third attack vector is the most difficult to defend against. In it, a reader has either been internally reconfigured or had a malicious external component added to it by an attacker, making it a "malicious" reader. When a malicious reader is used, authentic smartcard transactions can be eavesdropped to collect the details of the transaction. Additionally, the malicious reader can be used to perform additional transactions with the authentic smartcard, either to relay the transactions to an external reader or to collect information from the card.

With the latter two attack vectors (rogue and malicious reader), two attacks can be carried out. The first is to compromise a smartcard non-invasively by performing transactions to recover the card's keys and memory contents. Later, those memory contents can be used to create a clone of the card which can be used as if it was authentic. Although this attack has previously been carried out on the Mifare Classic [9], it relies upon the use of insecure cryptography by the smartcard, a problem which has been fixed in newer models.

The second type of attack is a "relay attack", in which an attacker relays the communication from a valid card to a valid remote reader. The channel used for the relay can be wired or wireless, and can even be implemented using two smartphones. If implemented properly, the user has no way of knowing he or she was the victim of a relay attack.

Both contact and contactless smartcards are vulnerable to relay attacks, but

because contactless readers only send and receive radio signals (as compared to having physical access to the card they are reading, and perhaps requiring a PIN), contactless relay attacks are easier to carry out. Contactless relay attacks can also be less conspicuous than their contact counterpart, since a smartphone or a device hidden in a wallet can be used as relay devices. This compares to a custom contact smartcard, such as was designed by Bond et al. [10], which must be inserted into a terminal.

The main difficulty in preventing relay attacks on contactless smartcards, however, is the fact that contactless smartcards have no user interface (UI). When a user presents his card to a reader, he has no way of validating that his card was not used maliciously. This compares to an NFC-enabled smartphone, which can provide an interactive dialog to the user during the transaction, i.e. "Are you sure want to open Door 5 in Building A?". If a protocol which features such a UI is implemented securely, a user can ensure his device has not been used maliciously.

To prevent relay attacks on a device without a UI, however, a reader can take steps to ensure a card is within a certain distance, such as 20 cm. Doing so can prevent a transaction from being relayed over a one-way distance larger than 20 cm. To perform such distance bounding, a distance-bounding protocol must be used. Distance-bounding protocols work by the reader checking how long a card takes to respond to certain commands, using the speed of light to calculate how far away a card is. Such protocols depend upon cryptographic keys and signatures, and therefore must be implemented on both the reader and card.

A successful implementation of a distance-bounding protocol relies upon precise timing measurements. A contactless smartcard must reply to requests from the reader in an expected amount of time and with minimum variance. If the card takes longer to respond than expected, the transaction is presumed to have been relayed, and the user is denied access. If the card responds quicker than expected, however, an attack vector opens up; because the card responded quicker, there is a time gap which can be used to relay a signal. Given that the speed of light is 3×10^8 m/s, even the slightest time

gap (i.e. a few microseconds) can allow a transaction to be relayed over a distance of a few kilometers.

One way of forcing a time gap during a distance-bounding exchange is by overclocking the smartcard. Even if the smartcard provides precise timestamps with minimum variance, it might base those timestamps on a standardized carrier frequency, such as 13.56 MHz, and therefore derive its clock solely from the reader's carrier. If a higher carrier frequency is used by a reader, therefore, the card can be forced to respond more quickly than it should, increasing the time gap. This can be defended against by a card containing an independent time reference, such as an internal oscillator, or having a low-pass filter to filter out carrier frequencies above a certain cutoff frequency.

1.2 Aims and Contributions

In this report, I describe precise timing measurements and overclocking results on two popular smartcards: the Mifare Plus EV1 and the Mifare DESFire EV2, both of which implement distance-bounding protocols. First, I took precise timing measurements of these distance-bounding protocols. Next, I overclocked these cards with carrier frequencies of over 16.30 MHz, over 20% higher than the 13.56 MHz specified by ISO 14443, allowing a signal to be relayed over a round-trip distance of over 80 km. This is the first known work which overclocks these cards at such a high frequency, and the first which overclocks the Mifare Plus EV1. This is also the first known work which takes precise timing measurements of the Mifare Plus EV1's distance-bounding protocol.

I also implemented the Mifare distance-bounding protocol on an open-source reader, the Proxmark 3, including the precise timing measurements necessary for the protocol. With the granularity of timestamps I used, a contactless smartcard can be bounded to a one-way distance of 9.38 m. Additionally, I describe, in detail, a proof-of-concept relay attack using overclocking, includ-

ing a discussion of the hardware which would be necessary to perform such an attack.

All Proxmark 3 code used can be found on my fork of the Proxmark 3 repository[1], where my `mainwork` branch includes the code used for taking precise timing measurements as well as collecting statistics.

1.3 Report Organization

Related work about relay attacks, distance bounding, and Mifare smartcards is presented in detail in Chapter 2. The design and implementation of precise distance-bounding timing measurements and overclocking is then discussed in Chapter 3. Timing measurements, results of overclocking, and an example relay attack proof-of-concept are presented in Chapter 4. Finally, conclusions and future directions of research are discussed in Chapter 5. Details of the setup done on the Mifare Plus and Mifare DESFire are described in Appendix A.

[1]My `mainwork` branch can be found on Github at: `https://github.com/dceliano/proxmark3/tree/mainwork`

Chapter 2

Related Work

2.1 Relay Attacks

2.1.1 Relay Attack Origins

The concept of relaying information between two parties was introduced as early as Conway's chess grandmaster problem in 1976 [11]. Conway pointed out that a middleman can relay chess moves between two chess grandmasters via post, thereby appearing to be a skilled chess player to both his opponents and gaining either one win or two draws, even without knowing anything about chess.

When smartcards began to become popular in the 1980s, Fiat and Shamir presented a protocol in 1986 which could be used for communication between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ [12]. Their protocol allowed a user to identify him or herself without using any public keys or shared private keys. Fiat and Shamir's strong claim that their protocol was secure even in a mafia-owned store was challenged by Desmedt et al. in 1987 [13], in what Desmedt et al. termed "mafia fraud" based on a newspaper article quote by James Gleick quoting Shamir [14].

In mafia fraud, a relay is set up between two malicious parties who have the

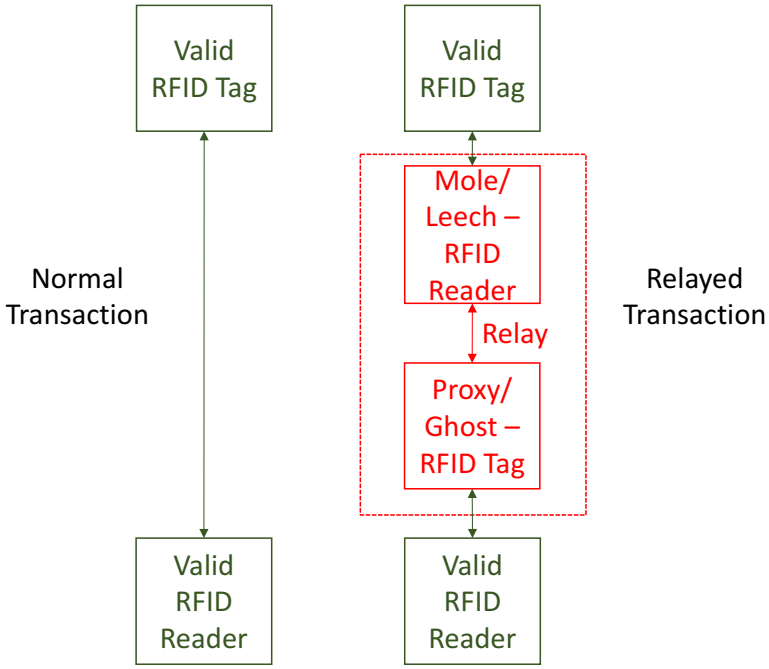


Figure 2.1: Visualization of a normal transaction compared to a relayed transaction.

goal of tricking two valid parties into performing a transaction. Nowadays, the term mafia fraud is synonymous with relay attack. A visualization of a relay attack with an RFID tag and reader is compared to a normal RFID transaction in Figure 2.1. In a relay attack, the two malicious parties which perform the roles of fake RFID reader and fake RFID tag are either known as the mole/proxy or the ghost/leech, as Figure 2.1 shows.

One year after presenting mafia fraud, Desmedt described another attack, "terrorist fraud", in which the contactless smartcard user is colluding with those conducting the mafia fraud [15]. This collusion makes the attack easier to conduct, but is also seen as a less dangerous attack vector. In 1991, Bengio et al. further critiqued the Fiat-Shamir protocol, expanding upon the principles of terrorist and mafia fraud [7].

2.1.2 Relay Attacks in Practice

Although the theory of relay attacks had been figured out by the end of the 1980s, it wasn't until the 2000s that practical relay attacks gained attention, likely due to the growing presence of RFID technology. The earliest practical relay attack was described by Hancke in 2005 [16]. Hancke showed that it is possible to successfully relay communication of an ISO 14443A proximity card up to a distance of 50 m. He included hardware details and showed that performing such an attack does not require expensive tools or extensive knowledge. He extended his work in 2006 by adding an FPGA to his design which reports the maximum attacking window available to conduct a relay attack [17]. In the same report, Hancke also demonstrated the threats behind and possibility of eavesdropping RFID transactions.

Kfir and Wool presented a low-cost practical relay attack similar to Hancke's later in 2005 [18]. They also used an ISO 14443 contactless smartcard as their test platform, and performed a relay at a distance of 50 m. Kfir and Wool also made an RFID reader function at a distance of 50 cm from the targeted smartcard.

Practical relay attacks have also been performed on contact smartcards, as was presented by Drimer and Murdoch in 2007 [19]. Drimer and Murdoch conducted a practical relay attack with the UK's "Chip & PIN" system and implemented a distance-bounding protocol in order to prevent such attacks.

Practical relay attacks gained more popularity in the 2010s when the ability to read and write to contactless smartcards became possible using smartphones. One of the first proofs of concept using a mobile phone as a relay was conducted by Francis et al. in 2010 [20]. In their paper, the authors show that peer-to-peer communication using NFC mobile devices can be used to perform a relay attack. For their test platform, they used two Nokia smartphones.

Since Francis et al.'s work in 2010, NFC has been built into more smartphones, and contactless payment has become integrated into the EMV pro-

tocol [4]. Additionally, the Android HCE (Host-based Card Emulation) API has been introduced, which allows Android applications to communicate directly with the NFC reader on the smartphone and thereby emulate a contactless smartcard. These developments have opened up the path for much more research into conducting relay attacks using mobile phones [21, 22, 23].

Relay attacks have also gained popularity because of the growth of passive keyless entry into cars and the realization that expensive cars can be stolen if a criminal gets hold of relay attack hardware. The West Midlands Police in the UK have released surveillance videos of criminals using such methodology to steal cars from owners [24]. Work which looks specifically at performing relay attacks on contactless entry into cars includes that conducted by Francillon et al. in 2011 [25]. In their paper, the authors describe how they are able to use inexpensive relay tools to a relay a contactless key transaction up to a distance of 50 m. They conduct this relay using both wireless and wired links, the latter allowing them to conduct relay attacks when the key is not within line-of-sight of the vehicle.

2.2 Distance Bounding

2.2.1 Distance-Bounding Protocols

As previously discussed in Section 1.1, a smartcard can be limited to a certain distance in order to defend against relay attacks. Gesture recognition can also be used to help prevent such attacks [26]. With distance bounding, the precise location of the user must be verified, done through the use of distance-bounding protocols. When discussing distance bounding, the verifier $\mathcal{V}$ is the party which is ensuring the prover $\mathcal{P}$ is within a certain distance. In the case of contactless smartcard transactions, $\mathcal{V}$ is the reader and the $\mathcal{P}$ is the smartcard.

When discussing distance-bounding protocols, it is necessary to distinguish between the four attack scenarios previously identified by Cremers et al. [27].

1. **Distance Fraud (Dishonest Prover)** – $\mathcal{P}$ is dishonest and will make every possible attempt to prove to $\mathcal{V}$ that he is closer than he actually is.

2. **Mafia Fraud** – $\mathcal{P}$ is honest, but is tricked into performing a malicious transaction, as previously discussed in Section 2.1.

3. **Terrorist Fraud** – $\mathcal{P}$ is dishonest and is voluntarily colluding with middlemen attackers to prove to an honest $\mathcal{V}$ that he is closer than he actually is.

4. **Distance Hijacking** – Multiple provers influence each other by a dishonest prover $\mathcal{P}$ exploiting the presence of an honest prover $\mathcal{P}'$ to convince $\mathcal{V}$ that he is closer than he actually is. One of the ways $\mathcal{P}$ can do this is by hijacking communication between $\mathcal{V}$ and $\mathcal{P}'$ during their distance measurement phase, as described by Cremers et al. [27].

The first distance-bounding protocol was developed by Brands and Chaum in 1993 [28], based on a previous suggestion of such a protocol by Beth and Desmedt in 1990 [29]. Brands and Chaum's protocol introduces the idea of a rapid bit exchange. In the rapid bit exchange, $\mathcal{V}$ times how long $\mathcal{P}$ takes to respond to $\mathcal{V}$ sending a series of single bits k to $\mathcal{P}$. Before the rapid bit exchange begins, both $\mathcal{V}$ and $\mathcal{P}$ draw random numbers, each of size k bits. Then, $\mathcal{V}$ sends out the bits of its random number one by one, to which $\mathcal{P}$ immediately responds with a single bit of its random number. The number of bits k in each random number is known as the *security parameter*.

At the end of the rapid bit exchange, $\mathcal{P}$ uses its private key to sign or produce a Message Authentication Code (MAC) of the concatenation of the $2k$ bits exchanged, the result of which is then sent back to $\mathcal{V}$ for verification. With the Brands-Chaum protocol, the probability of mafia fraud being successful is at most $1/2^k$.

With the Brands-Chaum protocol, $\mathcal{V}$ can send its bits at completely random times, thereby preventing $\mathcal{P}$ from anticipating when it should send its response. If $\mathcal{V}$ does decide to send out its bits at a predictable rate, however,

Brands and Chaum describe a protocol by which $\mathcal{P}$'s random bits can be dependent upon the bits it receives from $\mathcal{V}$, thereby preventing a fraudulent $\mathcal{P}$ from predicting when to send bits to $\mathcal{V}$. Brands and Chaum go on to show how to integrate their distance-bounding protocol into already-present public key identification schemes.

Since Brands and Chaum's protocol was presented, other distance-bounding protocols have emerged which take into account the computing limitations of RFID technology. These include the Hancke-Kuhn protocol described in 2005 [30]. In it, $\mathcal{V}$ sends a nonce N_v to $\mathcal{P}$ before the rapid bit-exchange period begins. Then, both $\mathcal{V}$ and $\mathcal{P}$ use a cryptographic function h to calculate two bitstrings of length k which are based on N_v and the shared private key K, i.e. $h(K, N_v)$. $\mathcal{V}$ then draws a nonce of k bits and the rapid bit exchange begins, with $\mathcal{V}$ sending its nonce bits to $\mathcal{P}$, while $\mathcal{P}$ responds with the bits of $h(K, N_v)$ calculated earlier.

However, the bit of $h(K, N_v)$ which $\mathcal{P}$ responds with needs to be decided based on the bit which $\mathcal{V}$ sent, since $h(K, N_v)$ is twice the length of N_v. The reason for this is so that $\mathcal{P}$ only ever reveals half the bits of $h(K, N_v)$, and therefore it becomes more difficult for an attacker to guess the value of $h(K, N_v)$, having a probability of guessing all the bits correctly of only $(\frac{3}{4})^k$. The Hancke-Kuhn protocol provides a faster authentication time than the Brands-Chaum protocol and also takes into account an environment in which there may be noise.

There have been many follow-ups to the Brands-Chaum and Hancke-Kuhn protocols. A thorough investigation of these different protocols and a framework for performing such a comparison is given by Avoine et al. [31, 32].

2.2.2 Distance Bounding in Practice

Understanding the theory of distance-bounding protocols is important, but where are such protocols actually implemented, and how can they be broken? Trying to break an implementation of a distance-bounding protocol was the

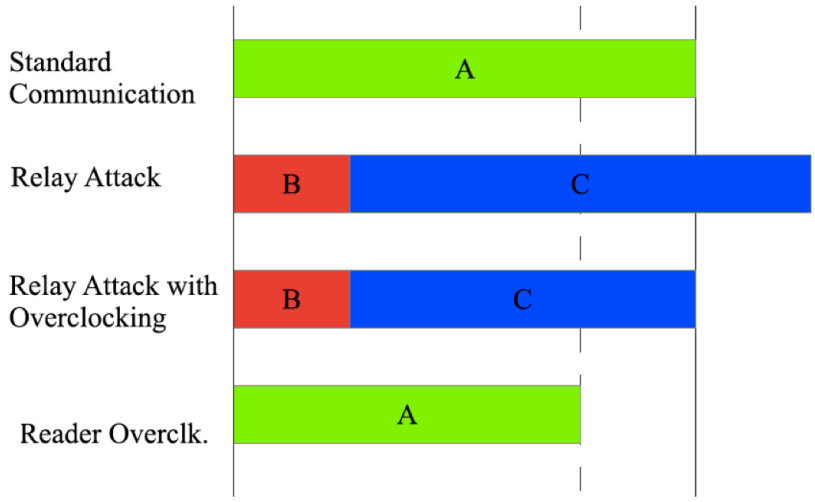


Figure 2.2: Overclocking visualized. Image by Henzl et al. [1].

goal of Hancke et al. in 2008, when the authors were able to successfully overclock a 13.56 MHz RFID tag with carrier frequencies of 14.56 MHz and 15.56 MHz [33]. A visualization of the consequences of overclocking can be seen in Figure 2.2.

In Figure 2.2, A and C are the amount of time needed to perform a transaction (A is without a relay, C is with a relay), while B is the delay caused by using a relay in a relay attack. A and C are sped up by overclocking a card, while B remains constant.

In 2014, Henzl et al. overclocked the Mifare DESFire at a frequency of 16.00 MHz [1], using the Proxmark 3 to get the DESFire to run at such a frequency. However, the authors did not provide extensive details of their methodology and did not conduct specific timing measurements of the DESFire's distance-bounding protocol.

Because of the threat of relay attacks, some researchers have implemented distance-bounding protocols using purely analog hardware. Examples include Rasmussen and Capkun's work in 2010, in which they presented a prototype prover which can receive, process, and transmit signals in less than 1 ns [34]. At this level of precision, an RFID tag can be verified to be within

a distance of 15 cm.

In 2008, Clulow et al. showed that it is possible to circumvent a distance-bounding protocol if it is not implemented properly [3]. Additionally, Reid et al. showed in 2007 that it is possible to detect relay attacks by using timing-based protocols [35]. In 2015, Gambs et al. implemented a distance-bounding protocol on a smartphone, giving the ability to detect relay attacks that introduce a latency of more than 1.5 ms [36].

In 2017, Soules et al. reverse engineered the distance-bounding protocol of the Mifare DESFire EV2, which uses the same distance-bounding protocol as the Mifare Plus [37]. Soules et al. also conducted initial timing tests with the Mifare DESFire distance-bounding protocol, which I compare to the timing measurements I collected in Chapter 4.

Distance-bounding protocols have also begun to be implemented in the EMV protocol in what is called a "Relay Resistance Protocol" [38]. Mastercard has been the first company to adapt such a protocol into their contactless smartcards [39].

2.3 Miscellaneous Mifare Research

Because of their popularity in the market, Mifare contactless smartcards have been a popular area of research. Much of this interest began in 2007, when Nohl uncovered initial results related to the cryptography used in the Mifare Classic [40]. Later results of reverse engineering the Mifare Classic's "Crypto1" protocol were presented in 2008 by Nohl et al. [41], and similar vulnerabilities were found by researchers at Radboud University in the same year [9, 42].

In later implementations of the Mifare Classic, NXP fixed the weak Pseudo-Random Number Generator (PRNG) in its cards, but in 2015 Meijer and Verdult showed that the improved PRNG does not fix the inherent flaws of the underlying Crypto1 cipher [43]. More recent version of Mifare cards,

such as the Mifare Plus and Mifare DESFire, use AES and Triple DES block ciphers rather than Crypto1.

Chapter 3

Design and Implementation

3.1 ISO 14443

3.1.1 ISO 14443 Overview

Contactless smartcards that work at a short range (0–10 cm) are known as proximity cards, and many of these cards use one of the two protocols defined in ISO 14443 [2]. This compares to vicinity cards, which operate at a longer range (0–1 m), and use the protocol defined in ISO 15693 [44]. Both proximity and vicinity cards derive their power from a reader which sends out an electromagnetic carrier which is specified to oscillate at a frequency f_c of 13.56 MHz. ISO 14443's four parts describe everything from the physical characteristics of proximity cards up to their transmission protocol. It uses the terms Proximity Coupling Device (PCD) to refer to a reader, and Proximity Integrated Circuit Card (PICC) to refer to a smartcard.

ISO 14443 describes two types of communication: Type A (ISO 14443A) and Type B (ISO 14443B). This is due to the combination of the Philips Mifare standard (Type A) and the Innovatron standard (Type B), making ISO 14443 two standards combined into one document [45]. In this report, only Type A communication is considered as both the Mifare Plus and Mifare

DESFire are smartcards which adhere to ISO 14443A. In the remainder of Section 3.1, the details of ISO 14443A as it applies to distance bounding will be described.

3.1.2 ISO 14443A Physical Layer

Because PICCs are passive devices, they rely upon the PCD for their power. According to ISO 14443, the PCD should emit a carrier of 13.56 MHz ± 7 kHz. The alternating magnetic field of the carrier then powers the PICC using inductive coupling. The details of how this power transfer works can be found in Finkenzeller Chapter 4 [46].

The carrier which the PCD supplies is not only used to power the PICC, but is also a communication channel. This communication channel (1) allows the PCD to send messages to the PICC, and (2) allows the PICC to send messages to the PCD.

1. **PCD to PICC Communication** – Amplitude modulation is used by the PCD by turning on and off its carrier in order to send 0s and 1s. The carrier is turned off (known as a PauseA) for a minimum of 0.5 µs and a maximum of 3.0 µs (at a bitrate of 106 kbps), and the PICC remains powered on during this lack of a power source through energy stored in its inductors and capacitors. This process of communication is known as 100% ASK (Amplitude Shift Keying). How often the PCD turns its carrier off is specified by the type of encoding that it uses: Modified Miller encoding.

   ISO 14443-2 defines three sequences that can occur under Modified Miller encoding: sequence X, sequence Y, and sequence Z, shown in Figure 3.1. In sequence X, a PauseA of time t_1 occurs after half the bit duration t_x, where t_b is the bit duration. In sequence Z, a PauseA occurs at the beginning of the bit duration, while in sequence Y the carrier is left on for the entire bit period. A logic 1 is a sequence X, a logic 0 is a sequence Y, a start of communication is sequence Z, and

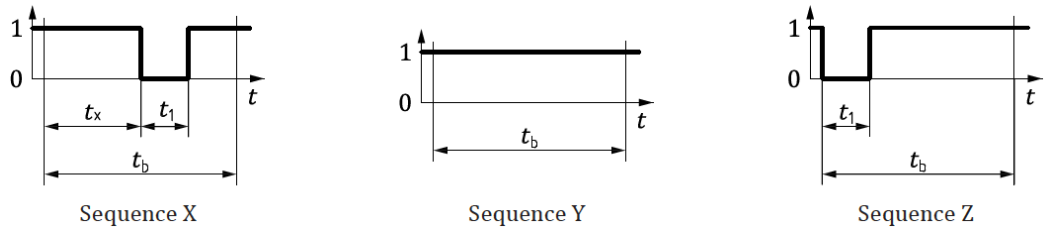


Figure 3.1: Sequences for PCD to PICC communication. Image from ISO 14443 [2].

an end of communication is a logic 0 followed by a sequence Y. These sequences of bits are detected and decoded by the PICC.

2. **PICC to PCD Communication** – The PICC communicates to the PCD by changing the strength of the carrier wave it has been supplied. It does this by switching a load resistor on and off, a strategy known as load modulation. By performing this switching at set times (On-Off Keying, or OOK), the PICC is able to send 0s and 1s to the PCD, which detects the change in the carrier strength. The PICC also uses load modulation to produce a subcarrier which operates at a fraction of the carrier frequency f_c. The PICC only generates this subcarrier when it is transmitting data, and the subcarrier itself does not contain any information.

   Similar to direction (1), the PICC sends its data using Manchester encoding. Three possible sequences exist for this encoding: sequence D, sequence E, and sequence F. In sequence D, the subcarrier is on for the first half of the bit period, while for sequence E, the subcarrier is on for the second half of the bit period. For sequence F, the subcarrier is off for the entire bit duration. This is similar to Figure 3.1, but can be fully visualized in Figure 3.2 in Section 3.1.3. Sequence D represents a logic 1 or start of communication, sequence E a logic 0, and sequence F an end of communication. These sequences are created using OOK, and the PCD then decodes them to receive the PICC's message. If there is a collision, the subcarrier would be on for an entire bit duration, an invalid sequence which signifies to the reader that there has been a

collision.

3.1.3 ISO 14443A Timing

In the first edition of ISO 14443 (2001), the bitrate for communication between the PCD and PICC was set at $\frac{f_c}{128}$, or 106 kbps. Two subsequent versions of the standard have been released, however, and in the most recent version (2016), much faster bitrates are possible for both directions of communication: $2f_c$ for PCD to PICC, and $\frac{f_c}{2}$ for PICC to PCD. During initialization and anticollision, however, a bitrate of $\frac{f_c}{128}$ is always used; afterwards, a higher bitrate can optionally be negotiated. For simplicity's sake, the scenario in which a higher bitrate is negotiated is not considered here.

A bitrate of 106 kbps means that the time it takes for a PCD and PICC to exchange a single bit of information is 9.4 µs. In those 9.4 µs, something different happens depending upon which direction the communication is going in. In the case of PCD to PICC communication, the PCD, if the encoding specified is sequence X or Z, turns off its carrier for a period of up to 3.0 µs. In the case of PICC to PCD communication, the PICC, if the encoding specified is sequence D or E, turns its load and subcarrier "on" for 4.7 µs, or half the bit duration. During the 4.7 µs where the load is on, the PICC subcarrier runs at a frequency of $\frac{f_c}{16} = 847.5$ kHz, or with a period of 1.18 µs. During those 4.7 µs, therefore, the subcarrier cycles 4 times. To help visualize these times, an example PCD-PICC transaction is shown in Figure 3.2.

In Figure 3.2, the PCD first sends a REQuest for type A (REQA) command, which queries any ISO 14443A PICCs which are within range and is the first command (other than a WUPA – WakeUP type A command) that occurs during any ISO 14443A communication. The REQA command is always 0x26 encoded as 7 bits. Any PICC which receives a REQA is required to respond with 4 bytes of data known as the card's ATQA (Answer To reQuest type A). It should be noted that the PICC's subcarrier in Figure 3.2 is distorted because of the way the video output of the spectrum analyzer was set up to

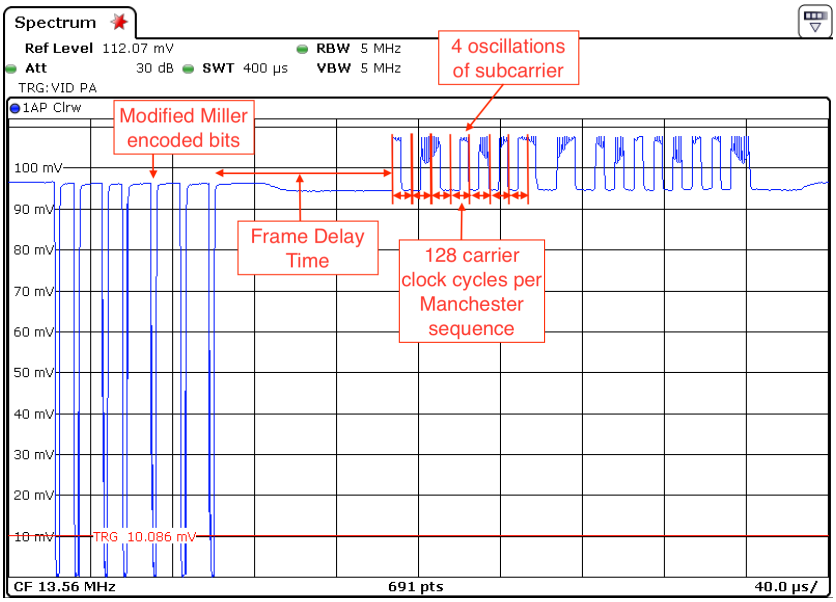


Figure 3.2: REQA transaction at 13.56 MHz, with timing annotated.

display the signal. If the image was zoomed in, the 4 cycles of the subcarrier during each bit period would be clearly visible.

Besides the timing of sending individual bits, another important timing consideration is the delay in the communication which occurs between a PCD and PICC. In ISO 14443-3, the minimum delay time in both directions is known as the Frame Delay Time (FDT), shown (in one direction) in Figure 3.2.

For a PICC responding to a PCD request, the FDT changes depending upon whether the last bit sent by the PCD was a 0 or a 1. If the last bit the PCD sent was a 0, FDT $= (n \times 128 + 84)$ / f_c. If the last bit of the PCD sent was a 1, FDT $= (n \times 128 + 20)$ / f_c. During the initialization and anticollision phases, $n = 9$, so that all PICCs respond synchronously and collisions can therefore be detected. With $n = 9$, FDT $= 1172$ / $f_c = 86.43$ µs if the last bit was a 0, and FDT $= 1236$ / $f_c = 91.15$ µs if the last bit was a 1. For all subsequent communication, $n \geq 9$, thereby giving the PICC extra time to respond if needed. However, the PICC's response must still be less than

the Frame Waiting Time (FWT), as discussed below.

For a PCD sending information to a PICC, the FDT is defined as the time between the last modulation transmitted by the PICC and the first PauseA transmitted by the PCD. In this scenario, the FDT must be at least 1172 / f_c. The PCD also has an additional timing constraint, known as the Request Guard Time, which is the minimum time between the start bits of two consecutive REQA commands. It is equal to 7000 / f_c.

Another important timing measurement, especially when considering relay attacks, is the Frame Waiting Time (FWT), or built-in timeout, specified in ISO 14443-4. The FWT is used by the PCD to "detect a protocol error or an unresponsive PICC" [2], but in practice it can also be used to prevent relay attacks that take longer than the FWT. The FWT is calculated using the formula FWT = $(256 \times 16 \ / \ f_c) \times 2^{\mathrm{FWI}}$. For Type A, the default FWI of 4 is normally used, giving a FWT of approximately 4.8 ms.

These timing details become important in Sections 3.3 and 3.4, where understanding how a reader both transmits and receives data is vital to taking timing measurements and overclocking.

3.2 Proxmark 3

3.2.1 Proxmark 3 Overview

A variety of off-the-shelf contactless smartcard readers are available. Examples include the ACR122 [47], the SCL3711 USB stick [48], NFC-enabled smartphones, and the Proxmark 3 [49]. Of these, the most flexible is the Proxmark 3, which contains both open-source hardware and software, allowing the user to have full control over the functionality it implements. Readers such as the ACR122 and the SCL3711 utilize open-source software libraries such as `libnfc`, as well as the PC/SC API through the `pcsc-lite` library. However, their hardware is unknown or based on proprietary reader chips,

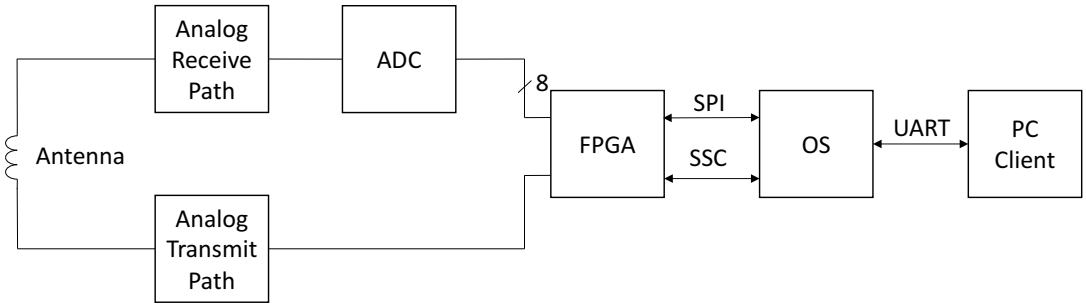


Figure 3.3: Proxmark 3 block diagram.

such as the NXP MFRC523 or MFRC631 chips. The case is similar for Android applications, which use custom software and NFC libraries, but don't give details of the underlying hardware.

This paradigm of readers with closed-source hardware contrasts with the Proxmark 3, which has both open-source hardware and software. By being open-source, the user has access to the schematics and circuit diagrams of the hardware as well as access to a development environment such as an interface to reflash its FPGA. The Proxmark 3 contains three main components: the PC client, the OS, and the transmit and receive path. These three components and their interfaces can be seen in Figure 3.3.

The Proxmark 3 OS runs on an AT91SAM ARM-based Atmel microcontroller. The OS communicates with a PC client over UART to update the OS firmware and to send and receive USB commands, many of which are forwarded through the transmit and receive path to communicate with a contactless smartcard. The Proxmark 3 contains an external antenna, the receive path of which gets fed into an 8-bit TLC5540 ADC after analog preprocessing. On both the transmit and receive paths, a Xilinx Spartan-II FPGA, an XC2S30, is used to perform DSP and implement time-critical PCD functionality.

Extensions to the Proxmark 3 over the years have implemented functions such as support for different RFID protocols such as those which use low frequencies (125–134 kHz). The Proxmark 3 also has the ability to eavesdrop

contactless smartcard transactions and emulate an ISO 14443A card. The Proxmark 3 remains an active project, and I have made all code developed for this report available to the Proxmark 3 community via its Github repository[1].

It should be noted that the Proxmark 3 is an experimental device and is not the tool of choice for all contactless smartcard applications. For operations in which a smartcard needs to be personalized, the ACR122 makes implementing custom protocols much simpler than using the Proxmark 3. When trying to do cryptographic functions, such as CMACs for example, the lack of Lua cryptography support on the Proxmark 3 makes it much easier to use the ACR122 reader with Python and the PC/SC API. This is the methodology I used to setup the Mifare Plus EV1 and Mifare DESFire EV2, the details of which are described in Appendix A.

However, the Proxmark 3 is a powerful tool for time-critical experiments and the main tool which I used. In Section 3.2.2, the software of the Proxmark will be described, with emphasis given to the communication between the Proxmark's PC client and OS. In Section 3.2.3, the transmit and receive path of the Proxmark will be covered in detail, including the FPGA functionality and DSP (digital signal processing).

3.2.2 Proxmark 3 Software

The standard user interface for the Proxmark 3 is the PC client's command-line tool, `proxmark3`. The PC client is written in C and allows the user to send various commands to a contactless smartcard and receive the corresponding response. The most basic PC client ISO 14443A command is `hf 14a raw`, which allows the user to control exactly which bits get sent to the PICC. Typing `hf 14a raw -b 7 -a 26`, for example, makes the Proxmark 3 transmit a REQA command, or 0x26 as 7 bits.

The PC client includes more complicated functionality, such as getting the

[1] As an example, the accepted pull request for my Mifare Plus script can be seen here: `https://github.com/Proxmark/proxmark3/pull/593`

keys off a Mifare Classic, much of which is implemented in C, and some of which is implemented via Lua scripts. I used this functionality to replicate old Mifare Classic attacks implemented on the Proxmark 3, such as extracting the private keys from my University of Cambridge ID card (a Mifare Classic 4k) and dumping the card's memory.

So, how do the PC client and OS interact? As was previously shown in Figure 3.3, once the client has sent its packets over the UART, they are read and processed by the Proxmark 3 OS. The communication channel between the Proxmark 3 client and the Proxmark 3 OS is a custom protocol in which `UsbCommand` structs are exchanged. In a `UsbCommand` struct, `arg[0]` has the flags which hold information about what is being sent. For example, if the PC client wants the OS to send a sequence of raw bytes out over its interface, `ISO14A_RAW` is integrated into `arg[0]`. Similarly, if the OS should keep providing a smartcard with power even after sending it raw bytes, `ISO14A_NO_DISCONNECT` is integrated into `arg[0]`.

For implementing any large amount of functionality, it is desirable to chain Proxmark 3 commands together using some sort of scripting language. I attempted to integrate Python with the Proxmark 3's C code, but after encountering difficulty decided to use the built-in Lua functionality instead. The glue which connects the Lua and C code together is contained in the `scripting.c` file, which maps Lua commands to native C commands which then get sent to the Proxmark 3 OS.

In order to communicate with the Mifare Plus EV1 and Mifare DESFire EV2 and implement their distance-bounding protocols, I created Lua scripts and integrated them into the Proxmark 3 project. I have made both the Mifare DESFire and Mifare Plus scripts available to the Proxmark 3 community via Github (see above footnotes). Both scripts used the Lua library `read14a.lua` in order to set up an ISO 14443A connection and perform all necessary anticollision steps. I also made it possible to send raw bytes to the smartcard in Lua, using a similar user interface to that of the `hf 14a raw` command. The card setup I performed to be able to do the Proximity Check on the Mifare Plus and Mifare DESFire is described in detail in Appendix A.

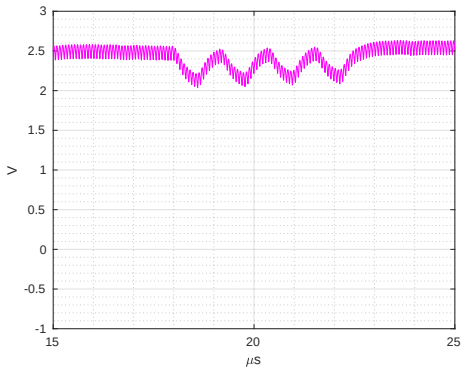
3.2.3 Proxmark 3 Hardware

The Proxmark 3 OS has two ways of communicating with the Spartan-II FPGA. The first is over a SPI (Serial Peripheral Interface), and the second is over an SSP interface (Synchronous Serial Port), as was previously shown in Figure 3.3. The SPI is normally used only to send configuration information from the OS to the FPGA. The FPGA can be told to, for example, act as a reader or emulate a card. Depending upon what configuration information the FPGA receives from the OS, it changes the section of Verilog code which it uses via a multiplexer. Normally, the SPI is not used to send information from the FPGA to the OS. However, I added functionality for sending precise timestamps from the FPGA to the OS over SPI, the implementation of which will be described in Section 3.3.2.

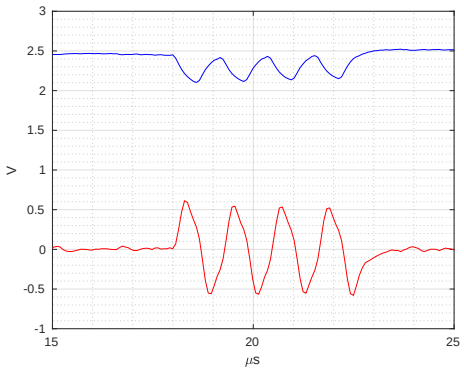
Data that actually gets sent to and received from the contactless smartcard is sent over the SSP. This data gets transferred at a rate determined by `ssp_clk`, which is a clock produced by the FPGA and is equal to $\frac{f_c}{16}$, where f_c is the frequency of the oscillator used by the FPGA, normally 13.56 MHz. This makes it so that `ssp_clk`'s period of $\frac{f_c}{16}$ is 8 times the ISO 14443 bitrate, $\frac{f_c}{128}$, previously described in Section 3.1.3, allowing 8 bits of data to be sent over the SSP for each ISO 14443 bit period.

As an example, the OS would send `00001100` to the FPGA to send out a sequence X (active low – turn off the carrier after half a bit period). For receiving data, the OS would take 8 ticks of `ssp_clk` to receive `00001111`, which would represent a sequence E (modulation with subcarrier during second half of the bit period). These bits of data are already in their encoded form because the encoding and decoding is done by the OS. The OS first encodes data using Modified Miller encoding, and whenever it receives data over SSP, immediately decodes that data using Manchester decoding.

If the bitrate was to be increased to $\frac{f_c}{64}$, $\frac{f_c}{32}$, or any of the other speeds possible in the latest version of ISO 14443, the frequency of `ssp_clk` would have to be increased as well. This would be possible, but the bitrate would eventually be limited by the processing limitations of the AT91SAM. As was previously



(a) Signal after analog peak detection. The 13.56 MHz carrier is still present.



(b) Output of the ADC (top) and output of the Gaussian derivative filter input (bottom).

Figure 3.4: Proxmark 3 analog receive path and DSP at $f_c = 13.56$ MHz.

mentioned in Section 3.1.3, I did not extend the Proxmark 3 to implement bitrates higher than $\frac{f_c}{128}$.

The analog hardware which is included on the Proxmark 3 is also of vital importance when considering communication with contactless smartcards. The analog circuitry of the Proxmark is available as open-source Eagle files, and its operation can be visualized using a Spice simulation. To conduct this Spice simulation, I used LTSpice with the Proxmark 3 front end circuitry[2].

The input given to the circuit simulates a PICC turning on its subcarrier for 4.7 µs, or half a bit period (at 106 kbps). During those 4.7 µs, the subcarrier is turned on and off 4 times to simulate the 4 cycles of the 847.5 kHz signal which would occur during half a bit period. The output of the Proxmark 3's peak detection analog circuitry from the Spice simulation can be seen in Figure 3.4a. In this plot, the 4 cycles of the subcarrier can clearly be seen.

To further visualize the Proxmark front end, I used 1-dimensional interpolation in Matlab to simulate sampling the output shown in Figure 3.4a at the carrier frequency f_c. By doing so, I was able to replicate the step the

[2]The `.asc` file used for the LTSpice simulation was taken from the Proxmark 3 forums: `http://www.proxmark.org/forum/viewtopic.php?id=1797`

Proxmark 3 ADC takes to get rid of the 13.56 MHz carrier. The result of this output is shown in the top plot of Figure 3.4b.

Another important DSP step the Proxmark 3 takes before performing edge detection is digital filtering. The FPGA's DSP filter, which has five taps and an impulse response which is the derivative of a Gaussian normal distribution, prepares the signal to be edge detected in order to detect whether a load modulation has occurred or not. The output of the digital filter can be seen in the bottom plot of Figure 3.4b. Applying the digital filter centers the subcarrier at 0 and makes it a sine wave which cycles four times.

Next, the FPGA uses the sine wave to look for both a significant rising and falling edge (based on the `EDGE_DETECT_THRESHOLD`) during each $\frac{1}{8}$ of the bit period. If both a significant rising and falling edge occur during the $\frac{1}{8}$ of the bit period, the FPGA detects a load modulation and sets the `curbit` signal high. Then, `curbit` is sent back to the OS on the next tick of `ssp_clk` to communicate that a PICC modulation was detected. For the example shown in Figure 3.4, the FPGA would detect `curbit` as `...00111100...`, which would then get sent to the OS to decode.

3.3 Distance Bounding Timing Measurements

3.3.1 Mifare Proximity Check

The details of the Mifare DESFire EV2 proximity check were previously reverse engineered by Soules et al. in 2017 [37]. Additionally, NXP research [50] and patents [51] have pointed towards the use of a similar distance-bounding protocol to that which Soules et al. discovered. Given this information, the details of the Mifare Plus EV1 and Mifare DESFire EV2 distance-bounding protocol ("Proximity Check") are presented in Figure 3.5.

In Figure 3.5, the Mifare Proximity Check is broken down into three phases: PreparePC, Proximity Check, and VerifyPC. During PreparePC, the PCD sends the byte 0xF0 to the PICC, to which the PICC responds with a status

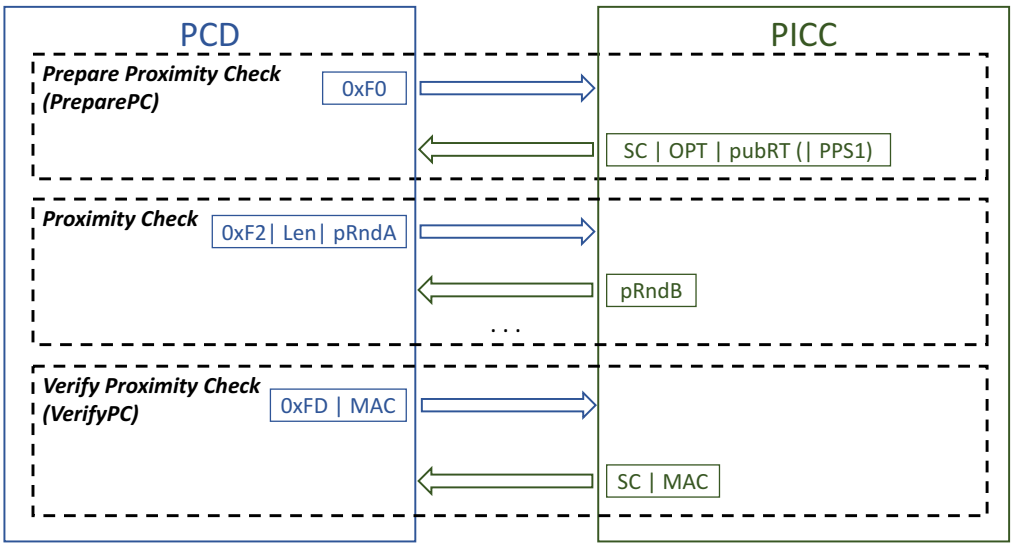


Figure 3.5: Mifare proximity check, used on the Mifare Plus EV1 and Mifare DESFire EV2.

code (SC), an option byte (OPT), and its published response time, t_{adv}. This advertized time is two bytes, sent MSB first, which specify in µs the amount of time the PICC claims it will take to respond to a command sent during the Proximity Check phase. For the three different Mifare Plus EV1 cards and the single Mifare DESFire EV2 tested, I found that $t_{\text{adv}} = 1696$ µs (at a bitrate of 106 kbps).

Next, the Proximity Check phase begins. At this point, both the PCD and PICC have drawn 8-byte random numbers RndA and RndB respectively. During the "rapid exchange" phase, the PCD sends its 8 bytes of RndA in chunks of whichever size it decides, sending 0xF2 followed by the length of pRndA, followed by pRndA. To more easily implement the Proximity Check, the size of pRndA will usually be a constant 1, 2, 4, or 8 bytes, although it does not have to be.

The PICC then responds with pRndB, which is the same number of bytes of RndB as the length of pRndA. The amount of time the PICC takes to respond is supposed to be as close as possible to t_{adv}, and is recorded by the PCD in order to calculate how far away the PICC is from the PCD. It does

this using the speed of light and factoring in the PICC's delay time, t_{dly}.

The final phase is VerifyPC, where the bytes exchanged during the Proximity Check phase are verified by both the PCD and PICC. Message Authentication Codes (MACs) are exchanged which are calculated using the VCProximityKey of the card and the CMAC mode of MAC calculation with the AES block cipher.

In relation to other distance-bounding protocols, the Mifare Proximity check is most similar to the Brands-Chaum protocol described in Section 2.2 [28], where verification occurs at the end of the rapid bit exchange. However, instead of having a "rapid bit exchange" where individual bits are exchanged, the Mifare Proximity Check exchanges individual bytes.

3.3.2 Measuring Timing Using the Proxmark 3

As was previously discussed in Section 2.2, precise timing measurements are vital for implementing an effective distance-bounding protocol. Therefore, this section will explain how I took precise timing measurements on the Proxmark 3 for both the Mifare Plus and the Mifare DESFire. The results of measuring timing during the distance-bounding protocol will be presented in Section 4.1.

From the PICC's perspective, as soon as it receives the last expected bit of pRndA or the last bit of pRndA's ISO 14443-4 CRC epilogue, it waits t_{dly} before sending out pRndB, as shown in Figure 3.6. It is unclear which of these two events the PICC uses as a time reference. In theory, $t_{\text{dly}} = t_{\text{adv}}$. However, as long as t_{dly} has a low variance, its relation to t_{adv} is unimportant. As long as the PICC is consistent in the amount of time it takes to respond to a request sent during the "rapid exchange" phase, the PCD can adjust its timing thresholds accordingly, learning off of previous exchanges used for calibration.

The one-way delay time it takes to send a signal between a PCD and PICC, t_{air}, shown in Figure 3.6, also needs to be taken into account. In a successful

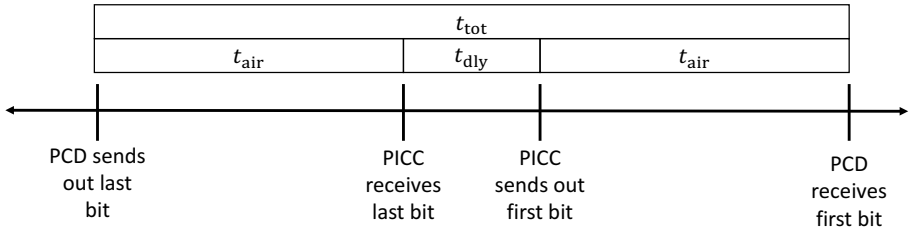


Figure 3.6: Visualization of timing as it relates to PCD timestamps. t_{tot} is the timestamp recorded by the PCD.

implementation of a distance-bounding protocol, the distance d, where $d = t_{\text{air}} \cdot \text{c}$, should be limited to, for example, 1 m, where c is the speed of light in a vacuum. Depending upon the precision of the PCD timing measurement, however, t_{air} may not be capable of bounding a user to such a small distance as 1 m.

From the PCD's perspective, as soon as it sends out the last bit of its CRC on its pRndA, it starts a timer. It stops that timer when it receives the first bit of pRndB from the PICC. The resulting difference in time is t_{tot}, as shown in Figure 3.6. t_{tot} is given by the equation:

$$t_{\text{tot}} = 2 \cdot t_{\text{air}} + t_{\text{dly}} \tag{3.1}$$

Because the distance at which a relay attack can be conducted is $d = t_{\text{air}} \cdot \text{c}$, $t_{\text{air}} = \frac{d}{\text{c}}$. Substituting this into and rearranging Equation 3.1, we get:

$$d = \frac{\text{c} \cdot (t_{\text{tot}} - t_{\text{dly}})}{2} \tag{3.2}$$

The Proxmark 3 was used to record t_{tot}. On the Proxmark 3, accurate timing measurements can be taken using two methodologies.

The first is to use the AT91SAM in the Proxmark 3 OS to take a start timestamp immediately after the last bit of data gets sent out over the SSP to the FPGA. A stop timestamp can then be taken as soon as the first bit of data is received from the PICC over the SSP from the FPGA. Timing

measurements using this methodology are simple to implement, but are not ideal because they depend upon the underlying software details of the OS, as well as microarchitectural considerations of the AT91SAM's ARM7TDMI processor such as pipelining.

The second methodology is to take the timestamps directly on the FPGA, and later use either the SPI or SSP interfaces to send the data back serially to the OS. This method is preferable to the first because it depends solely upon hardware, and is therefore easier to assure to be accurate. However, using an FPGA to collect timestamps does not provide a level of precision high enough to perform extremely sensitive distance-bounding, especially given that the clock used for time measurements was only 16 MHz. Despite this limitation, this methodology was the most accurate, and is the methodology I used to take the measurements in Section 4.1.

For implementing the second methodology, I needed to change the Verilog code on the Spartan-II. For generating the Spartan-II bitfile, I used Xilinx ISE Version 10.1 because of lack of Spartan-II backwards compatibility in newer versions of ISE. I added three pieces of functionality to the FPGA: (1) I created a 16 MHz clock signal, derived from a 48 MHz clock input, (2) I added the ability to send data from the FPGA to the OS over SPI, and (3) I added the ability to count clock cycles which correspond to t_{tot}. Once I took the measurements, I calculated the actual t_{tot} by multiplying the number of clock cycles collected with the period of the corresponding clock.

1. **Clock Generation** – The 16 MHz clock was derived from programmable clock 0, `pck0`, the AT91SAM's programmable clock I programmed in the OS to run at a frequency of 48 MHz by writing to the AT91SAM's programmable clock register (details will be provided in Section 3.4). I then divided `pck0` by 3 in the Verilog code using a counter. Changing the clock source on the FPGA to be the 16 MHz clock rather than the normal 13.56 MHz clock ended up being difficult because of obscure Xilinx errors, so I consolidated all the high frequency (HF) Verilog code, which is normally divided amongst multiple files, into a single file, `fpga_hf.v`. In this file, I only kept the code which

was necessary to implement an ISO 14443A reader and take precise timing measurements.

2. **FPGA to OS SPI Communication** – Normally, the Proxmark SPI interface is only used for sending configuration data to the FPGA via the OS. Therefore, the SPI interface had already been configured to have the AT91SAM act as the SPI master and the FPGA as the SPI slave. The Proxmark SPI channel uses four pins: NCS, SPCK, MISO, and MOSI. The MISO pin to send data back to the AT91SAM, therefore, is normally unused on the Proxmark. However, there needed to be a way for the FPGA to send the precise timestamps collected in (3) to the OS to be recorded.

   The Proxmark SPI connection gets set up in `fpgaloader.c`, and the `FpgaWriteConfWord()` method is used to send data over SPI which contains the FPGA setup word as 16 bits. This setup word determines, amongst other things, what major mode the FPGA is in (reader, emulate, or eavesdrop). SPCK is the clock used for the SPI communication generated by the AT91SAM, and has a baud rate of MCK / 6 = 4 Mbps, where MCK is the AT91SAM's 24 MHz master clock. NCS is a chip select/slave select pin for the SPI, and is active low. That is, whenever it is low, SPI data is being transferred.

   The AT91SAM SPI has two holding registers: the Transmit Data Register `SPI_TDR` and the Receive Data Register `SPI_RDR`. It also contains a single Shift Register which actually does the sending and receiving over the SPI interface. All of these registers are 32 bits, but only 16 of those bits are used for data transfer on the Proxmark. On the SPI, transmission and reception occur at the same time; whenever new data is written into `SPI_TDR`, data is also received in `SPI_RDR`. `SPI_RDR` can then be read by the OS to receive the timestamp collected by the FPGA.

3. **Collecting Timestamps** – On the FPGA, it was desirable to collect timestamps at the highest resolution possible to get the highest pre-

cision. For this purpose, the fastest clock available was `pck0`, which runs at 48 MHz and would give a precision of about 20 ns. The actual clock I used, however, was the 16 MHz division of `pck0` generated in (1). I used this clock in place of the 48 MHz clock in order to avoid overflowing the 16-bit counter used to collect the timestamp. In order to prevent overflowing vulnerabilities which this implementation might introduce, an operational reader should trigger an interrupt whenever the 16-bit counter overflows.

In order to collect timestamps on the FPGA, I did three things: (a) I collected a start timestamp when the Proxmark finished sending its data to the PICC, (b) I collected an end timestamp when the Proxmark received the first bit of data from the PICC, and (c) I sent that timestamp back to the OS over the SPI interface to later be reported to the PC client.

(a) **Start Timestamp** – I reset the start timestamp every time the data received over the SSP from the OS, `ssp_dout`, went high. Because `ssp_dout` was configured to be active low (i.e. a 0 meant the carrier was on), whenever `ssp_dout` was 1, that meant the carrier was being turned off in order to perform ASK modulation. By detecting this event, the start timestamp was reset to the last time the carrier signal sent by the PCD went low, allowing the timestamp to be started at the last bit of the ASK communication.

(b) **End Timestamp** – I collected the end timestamp as soon as a load modulation was detected in the data received from the PICC. The details of how the PCD detects this load modulation were previously covered in Section 3.2.3, but whenever a modulation is detected, the FPGA's `curbit` signal goes high. Therefore, the first time `curbit` went high, the running timestamp started in (a) was stopped. The delay of `curbit` as compared to the analog PICC response will be shown in Figure 4.2 in Section 4.3.

(c) **Timestamp Reporting** – I used the OS to poll the timestamp

Figure 3.7: Eavesdropping a Mifare DESFire REQA transaction using a near-field probe hooked up to a spectrum analyzer.

collected in steps (a) and (b) by writing to the SPI and consequently reading and printing the 16-bit timestamp (SPI reads and writes need to occur together, as described in (2)).

In order to verify that the timing measurements taken on the FPGA were accurate, I wirelessly eavesdropped transactions using the available FSV7 spectrum analyzer using a near-field probe. The spectrum analyzer was used in zero-span mode with a bandwidth of 5 MHz and a center frequency of 13.56 MHz, similar to the configuration previously shown in Figure 3.2. An example of eavesdropping a smartcard transaction using the spectrum analyzer can be seen in Figure 3.7.

I used the cursors on the spectrum analyzer for a few distance-bounding exchanges to compare the results with what was measured with the Proxmark 3. Seeing that the results were close to each other (<5% difference), I gained confidence in the accuracy of the FPGA timing measurements. More tests could have been conducted using this methodology, but manually taking timing measurements on the spectrum analyzer proved to be tedious.

3.4 Overclocking

3.4.1 Changing the Proxmark 3 Clock

In order to overclock a contactless smartcard using the Proxmark 3, I needed to change the Proxmark 3's carrier frequency. In the Proxmark 3 FPGA code, all the processing is synchronous to a single clock, `osc_clk`. This clock is used to control the speed at which the SSP transfers data between the OS and the FPGA, as well as the frequency at which the ADC is sampled. Normally, `osc_clk` is generated by a 13.5600 MHz external oscillator and connected to the Proxmark 3 through the FPGA's `ck_1356meg` signal. However, `osc_clk` can be set to any clock source.

The two ways I generated a clock to be used as `osc_clk` were to (1) use the already-present PLL on the AT91SAM, and to (2) use a function generator to generate an external clock. An easier option would have been to use some sort of phase-locked loop (PLL) on the FPGA. However, the Spartan-II lacks a PLL and only has a delay-locked loop (DLL) which is capable of multiplying a clock by 2.

1. **Using the AT91SAM PLL** – Using `pck0` to generate a 16 MHz clock on the FPGA was previously discussed in Section 3.3.2. To use 16 MHz as the carrier frequency, I set `osc_clk` equal to the division of the 48 MHz `pck0`, `pck_clkdiv`. However, I did not program `pck0` to be a frequency above 48 MHz due to limitations in the AT91SAM's clocking system. The AT91SAM's programmable clock is based on its PLL and corresponding PLL clock, `PLLCK`. `PLLCK` is based on `MAINCK`, the Proxmark 3's 16.00 MHz external oscillator. `PLLCK` is generated from `MAINCK` by passing a multiplier and divider to the `CKGR_PLLR` register.

   Normally, the Proxmark 3 code sets `PLLCK` as `MAINCK` $\times$ 6 = 96 MHz. Then, programmable clock `pck0` is set through the `PMC_PCKR` register to equal `PLLCK` / 4 = 24 MHz. By instead dividing `PLLCK` by 2, I set

pck0 equal to 48 MHz. pck0 can also be set to PLLCK / 1, which would have allowed a granularity fine enough to make pck_clkdiv equal to 19.2 MHz, rather than the 16.0 MHz frequency which was actually used.

If PLLCK is set to anything above 96 MHz, however, things become more complicated due to other modules which use PLLCK, such as USB. The AT91SAM's USB clock is always based on the PLL clock. Therefore, the PLL needs to run at a certain frequency (48 MHz, 96 MHz, or 192 MHz) in order for USB communication to work. For fear of irreversibly damaging USB communication with the Proxmark PC client, I kept PLLCK, and thereby the USB clock, at 96 MHz. If a frequency of 192 MHz had been fed to the FPGA over pck0, however, the FPGA would have been capable of achieving carrier frequencies such as 17.45 MHz.

2. **Using an External Clock** – A simpler way of generating a variable clock to be used as the FPGA's osc_clk was to use an external clock source. By using a function generator hooked up to Test Pin 7 (TP7) of the Proxmark 3, I was able to use a variable clock as an input to the FPGA, providing an easily tunable frequency. I used TP7 on the Proxmark 3 as a normal FPGA I/O pin, dbg, and then used that input for clocking all the FPGA registers by connecting dbg and osc_clk together.

   However, I needed to make certain considerations when using an external oscillator as the clock source for the Spartan-II. In the Spartan-II, the I/O logic levels are by default set to LVTTL, specifying the use of TTL logic levels. With LVTTL on the Spartan-II, a 0 is considered any voltage from −0.5–0.8 V, while a 1 is considered any voltage from 2.0–5.5 V. Given this, a function generator needs to be able to create a sine wave with a peak-to-peak voltage swing of at least 1.2 V to toggle a signal between a 0 and a 1.

   This required voltage swing made the available Agilent 81180A function

generator an ideal tool to use as it has a maximum peak-to-peak voltage of 2 V and a sampling rate of 10 MHz to 4.2 GHz. However, the 81180A has a maximum offset of 1 V, giving it, at most, a voltage swing of 0 V to 2 V. Because this is insufficient to be used as a clocking logic input to the Spartan-II, I put the 81180A in series with a 1.5 V DC power supply to provide a 1.5 V DC offset. This was safe because the DC power supply's earth ground was not connected to the circuit ground, preventing a short circuit between the power supplies from occurring. By setting the 81180A offset to 0 V and using its AC driver, the resulting sine wave produced and fed into the FPGA had a voltage swing of 0.5 V to 2.5 V.

3.4.2 Analog Considerations of Overclocking

When increasing the carrier frequency used with a contactless smartcard, I needed to consider the signal strength and characteristics of the antenna along with the Proxmark 3 front-end analog circuitry. Although the Proxmark 3 is open-source, there is no standardized antenna. However, certain manufacturers of the Proxmark 3 ship antennas with their particular version. The Proxmark 3 used for testing was purchased from RyscCorp, so the RyscCorp antenna was the primary antenna I used. However, I also successfully used other antennas, which are shown in Figure 3.8.

The difference between the RyscCorp antenna and the antennas shown in Figure 3.8 is that the RyscCorp antenna has capacitors on it to help it reach a certain tuning frequency. This tuning frequency can be adjusted via a switch on the RyscCorp antenna which either adds or takes away a parallel capacitor C3. The circuit diagram of the RyscCorp antenna can be seen in Figure 3.9.

When understanding why certain antennas work and others do not, the theory discussed in Finkenzeller Chapter 4 can help [46]. The simplest way to model a loop antenna is as an inductor. When a PCD's carrier is placed next to a PICC, the power transfer which occurs can be modeled as two in-

Figure 3.8: Antennas used with a frequency of 13.56 MHz other than the RyscCorp antenna.

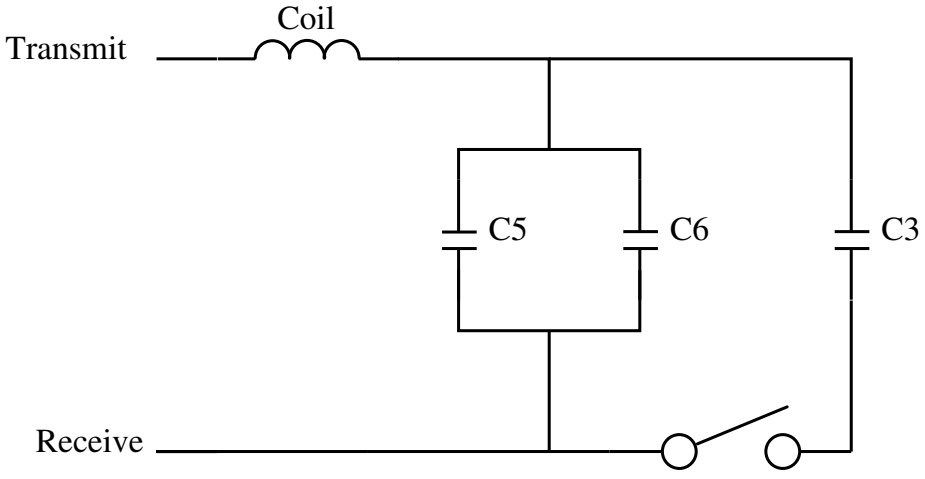


Figure 3.9: Circuit diagram of the RyscCorp antenna.

ductors transferring power through Faraday's law, similar to a transformer. These inductors have a certain coupling coefficient, k, which determines how efficiently the inductors transfer power to each other.

The geometric properties of the PICC, besides its orientation and distance from the PCD, are a constant. Contactless smartcards are PICCs which adhere to ISO 14443-1 Class 1, which means their antennas have a length of 64 to 81 mm and a width of 34 to 49 mm, working out to a perimeter of 196 to 260 mm. The winding of this antenna does not have to be in the shape of a perfect rectangle, and can vary depending upon the manufacturer's implementation. The inductance of a loop antenna is given by $L = N^2\mu_0 R \cdot \ln(\frac{2R}{d})$, where R is the radius of the conductor loop, d is the diameter of the wire used, N is the number of windings, and μ_0 is the magnetic field constant.

The PCD antenna, on the other hand, can be changed. Steps can be taken to change its dimensions, number of windings, and wire diameter. Additionally, capacitors can be added to the antenna circuit to change its resonant frequency, as is done with the RyscCorp antenna. The resonant frequency of an antenna is given by $\frac{1}{2\pi\sqrt{LC}}$, where L is the inductance of the loop and C is the parallel capacitance.

To test the tuning frequency of the antenna, I swept a sine across a range of different frequencies to see at which frequency the antenna was most resonant. This functionality is normally conducted by a vector network analyzer, but I manually implemented it using a function generator and an oscilloscope. I used this methodology to test the tuning of the RyscCorp antenna as well as the antennas in Figure 3.8. The circuit diagram used for tuning can be seen in Figure 3.10.

I performed the antenna tuning using the Agilent 81180A with a start frequency of 10 MHz, an end frequency of 30 MHz, and a sweep time of 10.0 µs. The output of this antenna tuning test can be seen in Figure 3.11. As the figure shows, the RyscCorp antenna with C3 in parallel is resonant at a frequency close to 13.56 MHz, which is expected given the desire to be compliant

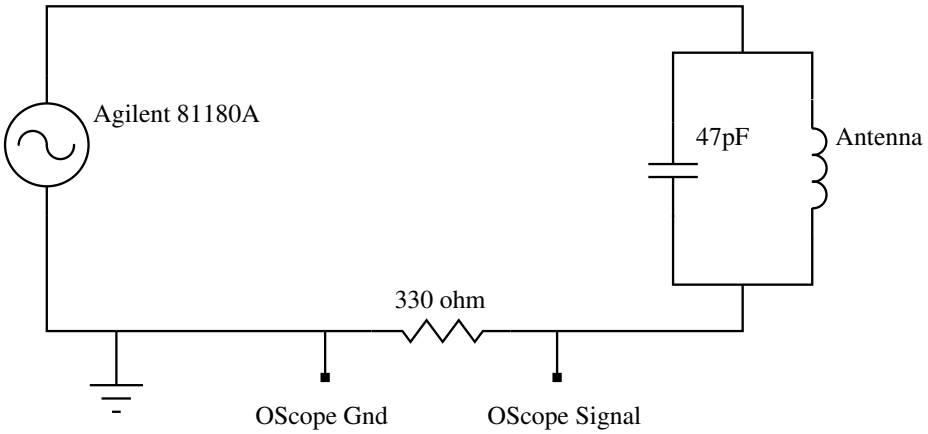


Figure 3.10: Circuit diagram used for antenna tuning.

with ISO 14443. The tuning frequency without C3 in parallel was slightly higher – around 14 MHz – because of the lower capacitance which means a higher resonant frequency by the equation $\frac{1}{2\pi\sqrt{LC}}$.

To experiment with overclocking a card, I finely tuned the carrier frequency of the PCD using a function generator and the external clock setup previously described in Section 3.4.1. To ensure the card was still responding, I sent REQAs and then eavesdropped the corresponding communication using a spectrum analyzer in zero-span mode connected to a near-field probe. If the PICC still performed load modulation to send its ATQA, similar to the output previously shown in Figure 3.2, the card was determined to be working properly.

Looking at the output on the spectrum analyzer, I saw that the Mifare Plus and Mifare DESFire were still responding at frequencies such as 16 MHz, but that the Proxmark 3 receive path was not properly decoding the PICC's load modulation. In order to fix this, I increased the `EDGE_DETECT_THRESHOLD` in the Proxmark 3's FPGA code to 40, rather than the value of 5 it is normally at. Making this change increased the sensitivity of the Proxmark 3's edge detection on the sine wave previously described in Section 3.2.3. Such a change made the Proxmark 3 more likely to have bit errors, but also allowed it to decode weaker signals, which was necessary for decoding higher frequency

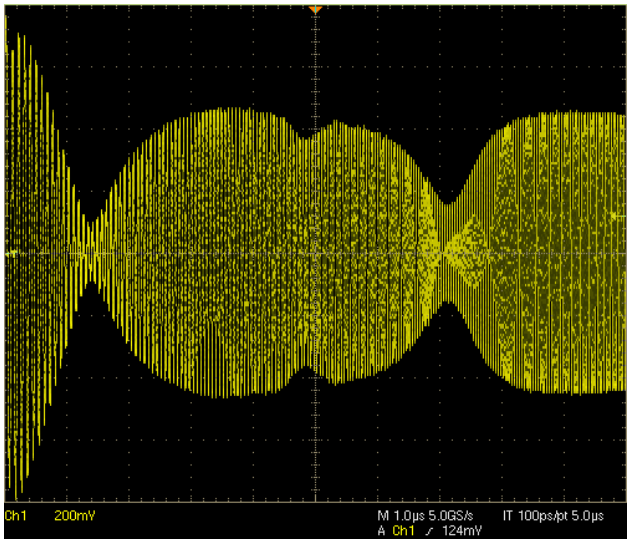
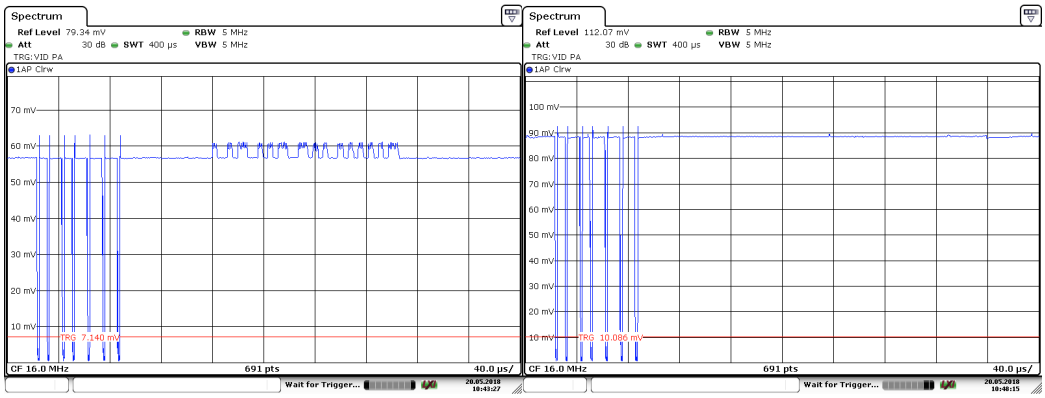


Figure 3.11: RyscCorp antenna tuning with C3 in parallel. Horizontal axis goes from 10 MHz–30 MHz.

signals which were weaker due to their frequency being different from the RyscCorp antenna's resonant frequency.

However, there was a certain frequency at which the spectrum analyzer clearly showed that the Mifare Plus and Mifare DESFire stopped responding to REQA commands. There are two possible reasons for such a limit. The first is that the PICC no longer receives a strong enough power supply from the PCD to properly respond. If this is the case, the properties of the antenna come into question. The second is that some sort of internal circuitry in the PICC prevents a higher frequency from being used. The captured traces in Figure 3.12 demonstrate that the latter possibility is most likely.

In Figure 3.12, the Mifare DESFire is run at two frequencies near its discovered limit of 16.30 MHz. At 16.30 MHz (Figure 3.12a), the DESFire successfully responds to a REQA command with a strength of 56 mV on the near-field probe used (relative, not absolute strength). At 16.40 MHz (Figure 3.12b), the RyscCorp capacitor C3 is taken out of parallel and the DESFire is sent the same request, this time with a relative strength of 88 mV. However, the card does not respond, even though the signal sent to the card



(a) Mifare DESFire REQA request and response at 16.30 MHz, with C3 in parallel. The card successfully responds with its ATQA.

(b) Mifare DESFire REQA request at 16.40 MHz, without C3 in parallel. Even with higher signal strength, the card no longer responds.

Figure 3.12: Frequency limitations of the Mifare DESFire. Note the vertical scales are different.

has a higher strength – a phenomenon which only occurs at frequencies above 16.30 MHz. A similar phenomenon occurred with the Mifare Plus at a frequency approximately 200 kHz higher, giving it a maximum frequency of 16.50 MHz. Implications of these results are discussed in Section 4.2.

Chapter 4

Results and Evaluation

4.1 Timing Measurements

Distance-bounding timing measurements were taken on both the Mifare Plus EV1 and the Mifare DESFire EV2. These results were taken using the FPGA timing method and the 16 MHz clock previously described in Section 3.3.2. Because the timing measurements were taken with a 16 MHz clock, their granularity was 62.5 ns. Therefore, all the times, reported in µs, are only reported to a precision of 100 ns. Given the speed of light in a vacuum, a granularity of 100 ns corresponds to a round-trip distance of 30 m. Similarly, a granularity of 62.5 ns corresponds to a round-trip distance of 18.75 m, which would allow a card to be bounded to a one-way distance of 9.38 m. Using a granularity of 100 ns, the Proxmark 3 can be used to prevent relay attacks which aim to send a signal over a one-way distance greater than 15 m. The results of the timing measurements for both the Mifare Plus and Mifare DESFire can be seen in Tables 4.1 and 4.2.

The experiments conducted to produce the results in Tables 4.1 and 4.2 were run with a RndA of `0001020304050607`, and the PICC was placed directly on top of the RyscCorp antenna. Additionally, trials were run with RndA sent as 1 round, 2 rounds, 4 rounds, and 8 rounds. Furthermore, the Mifare

Num Rnds	Rnd 1	Rnd 2	Rnd 3	Rnd 4
1	1.5855 ± 0.0000	–	–	–
2	1.5997 ± 0.0000	1.6091 ± 0.0000	–	–
4	1.6185 ± 0.0000	1.6233 ± 0.0000	1.6280 ± 0.0000	1.6327 ± 0.0000
8	1.6232 ± 0.0000	1.6327 ± 0.0000	1.6280 ± 0.0000	1.6280 ± 0.0000

Num Rnds	Rnd 5	Rnd 6	Rnd 7	Rnd 8
8	1.6280 ± 0.0000	1.6280 ± 0.0000	1.6421 ± 0.0000	1.6421 ± 0.0000

Table 4.1: Mifare Plus EV1 Proximity Check response times, in ms, with $f_c = 13.56$ MHz.

Num Rnds	Rnd 1	Rnd 2	Rnd 3	Rnd 4
1	1.6504 ± 0.0000	–	–	–
2	1.6646 ± 0.0003	1.6742 ± 0.0004	–	–
4	1.6742 ± 0.0005	1.6882 ± 0.0003	1.6837 ± 0.0005	1.6976 ± 0.0000
8	1.6886 ± 0.0006	1.6885 ± 0.0005	1.6837 ± 0.0005	1.6931 ± 0.0005

Num Rnds	Rnd 5	Rnd 6	Rnd 7	Rnd 8
8	1.6931 ± 0.0005	1.6931 ± 0.0005	1.6979 ± 0.0005	1.6977 ± 0.0003

Table 4.2: Mifare DESFire EV2 Proximity Check response times, in ms, with $f_c = 13.56$ MHz.

Proximity Check was run immediately after the initialization and anticollision phases of communication, which mimics an operational implementation of the Proximity Check. Means and standard deviations were calculated after running the distance-bounding protocol 15 times with each configuration. Experiments were run on a single sample of the Mifare Plus EV1, and a single sample of the Mifare DESFire EV2.

As can be seen in Tables 4.1 and 4.2, the times for the Mifare Plus and Mifare

DESFire differ quite a bit from the t_{adv} of 1.696 ms presented in Section 3.3.1. While both cards claim to have a t_{dly} of 1.696 ms, the results show that t_{dly} must in fact be lower.

Additionally, the response time of both cards for different configurations is quite different. If a single 8-byte chunk is sent to the Mifare Plus, for example, the card responds in 1.5855 ms. When the first 1-byte chunk in a series of chunks is sent, however, the card takes 1.6232 ms to respond. I have no logical explanation for this difference in timing other than a strange implementation detail in the card's firmware. I give the same explanation to the fact that the PICC's response time increases as it sends out sequential parts of RndB. When sending 8 bytes in 8 rounds with the Mifare Plus, for example, the PICC takes 1.6232 ms to respond to the first round, but takes 1.6421 ms to respond to the last round. This difference in timing provides no clear advantage in distance bounding.

It is also interesting that the Mifare DESFire takes longer to respond to distance-bounding requests than the Mifare Plus does. Soules et al. [37] also conducted round-trip time measurements of the Mifare DESFire EV2, sending RndA over 1 round and measuring a RTT of 1.64 ms. This comes extremely close to the 1.6504 ms recorded using the Proxmark 3, further validating the measurements presented here.

What is most important with the Mifare Plus measurements is the fact that the variance in the measurements is extremely low with the experimental setup used; zero for the level of precision at which these measurements were taken. The standard deviations in the Mifare DESFire were slightly higher than in the Mifare Plus, but can still be used to make the distance-bounding protocol effective enough to limit a card to a one-way distance of 15 m. As discussed in Section 3.3.2, this points to a good implementation by NXP.

To deal with timing variations which might occur amongst different cards, an ideal way of implementing a distance-bounding protocol in a large system would be to make the timing constraints unique to the card with which the reader is communicating. That is, when a card is first registered into a sys-

tem, timing measurements can be taken on the card which record its expected response time to distance-bounding requests, similar to the results shown in Tables 4.1 and 4.2. Then, the reader can base distance-bounding measurements on the experimental t_{tot} of each card rather than the potentially inaccurate t_{adv}. Further tests would need to be conducted to see whether a card's timing results change over its lifetime, which is an important consideration when implementing such a scheme.

4.2 Overclocking Results

The Mifare Plus EV1 still responded to REQA commands at a frequency of up to 16.50 MHz, while the Mifare DESFire EV2 responded at a frequency of up to 16.30 MHz. Additionally, the Mifare Plus responded at a frequency as low as 10.56 MHz, while the Mifare DESFire responded at a frequency as low as 10.85 MHz. This existence of both a minimum and maximum operating frequency at almost exact distances from the assumed center frequency of 13.56 MHz suggests internal circuitry used in the cards which, for the Mifare Plus allows an f_{c} of 13.56 MHz $\pm$ 3.0 MHz, and for the Mifare DESFire allows 13.56 MHz $\pm$ 2.7 MHz (assuming the center frequency is 13.56 MHz).

It is possible that the center frequency is slightly different than 13.56 MHz, and also that the center frequency shifts due to manufacturing variability and center frequency shifting which occurs with card age or operating temperature and other environmental effects. This can also help explain the large range of frequencies which the Mifare DESFire and Mifare Plus allow; the manufacturer might leave some component tolerances to account for such variability.

Using the Mifare Plus EV1 maximum frequency of 16.50 MHz and the Mifare DESFire EV2 maximum frequency of 16.30 MHz, the distance-bounding protocol was rerun and the resulting timing measurements were recorded using the same experimental setup as Section 4.1. At these frequencies, the speedup of the Mifare Plus should be $\frac{16.50\ \text{MHz}}{13.56\ \text{MHz}} = 1.2168$, while the speedup

Num Rnds	Rnd 1	Rnd 2	Rnd 3	Rnd 4
1	1.3019 ± 0.0000	–	–	–
Speedup	1.2178	–	–	–
2	1.3135 ± 0.0000	1.3213 ± 0.0000	–	–
Speedup	1.2179	1.2178	–	–
4	1.3290 ± 0.0000	1.3329 ± 0.0000	1.3368 ± 0.0000	1.3407 ± 0.0000
Speedup	1.2178	1.2179	1.2178	1.2178
8	1.3329 ± 0.0000	1.3407 ± 0.0000	1.3368 ± 0.0000	1.3368 ± 0.0000
Speedup	1.2178	1.2178	1.2178	1.2178

Num Rnds	Rnd 5	Rnd 6	Rnd 7	Rnd 8
8	1.3368 ± 0.0000	1.3368 ± 0.0000	1.3484 ± 0.0000	1.3484 ± 0.0000
Speedup	1.2178	1.2178	1.2178	1.2178

Table 4.3: Mifare Plus EV1 Proximity Check response times, in ms, with $f_c = 16.50$ MHz.

of the Mifare DESFire should be $\frac{16.30 \text{ MHz}}{13.56 \text{ MHz}} = 1.2021$ The actual speedups, calculated as $\frac{f_{\text{overclocked}}}{f_{13.56 \text{ MHz}}}$, can be seen in Tables 4.3 and 4.4.

In Tables 4.3 and 4.4, the actual speedups matched the expected speedups very closely (<0.1% difference). For RndA sent as 1 round to the Mifare Plus, the speedup was 21.78%, or 0.2836 ms. That 0.2836 ms time gap would allow a signal to be relayed over a round-trip distance of 85.08 km. For RndA sent as 1 round to the Mifare DESFire, the speedup was 20.21%, or 0.2775 ms. In that 0.2775 ms, a signal can be relayed over a round-trip distance of 83.25 km. However, these distances are assuming no delay in the implementation of the relay devices. In Section 4.3, technical details will be presented on how a practical relay attack can be carried out.

Num Rnds	Rnd 1	Rnd 2	Rnd 3	Rnd 4
1	1.3729 ± 0.0000	–	–	–
Speedup	1.2021	–	–	–
2	1.3846 ± 0.0000	1.3925 ± 0.0000	–	–
Speedup	1.2022	1.2023	–	–
4	1.3925 ± 0.0000	1.4042 ± 0.0000	1.4003 ± 0.0000	1.4121 ± 0.0000
Speedup	1.2023	1.2023	1.2024	1.2022
8	1.4042 ± 0.0000	1.4042 ± 0.0000	1.4003 ± 0.0000	1.4082 ± 0.0000
Speedup	1.2025	1.2025	1.2024	1.2023

Num Rnds	Rnd 5	Rnd 6	Rnd 7	Rnd 8
8	1.4082 ± 0.0000	1.4082 ± 0.0000	1.4121 ± 0.0000	1.4121 ± 0.0000
Speedup	1.2023	1.2023	1.2024	1.2023

Table 4.4: Mifare DESFire EV2 Proximity Check response times, in ms, with $f_c = 16.30$ MHz.

4.3 Overclocking Relay Attack Example

A valid reader inside a top-secret building, Reader 1, is tampered with. Its standard hardware is left alone, but an inconspicuous module is added to it which has the ability to perform an overclocked transaction with a smartcard and wirelessly relay the communication to a proxy. The proxy desires to gain access to Reader 2, which is located in a similar top-secret building 25 km away and within line-of-sight of Reader 1. Both PCDs perform a distance-bounding protocol in order to prevent relay attacks by ensuring that the PICC is within a distance of 1 m of the PCD.

Whenever a user presents a smartcard to Reader 1, Reader 1's standard hardware performs the normal transaction to grant a user access, using a frequency of 13.56 MHz. The mole listens for when this transaction is complete, notifies the proxy to prepare a transaction, and then performs a transaction of its own at a frequency of 16.30 MHz (while, to avoid interference, ensuring the 13.56 MHz Reader 1 does not transmit).

The mole receives the requests originating at the proxy and sends the responses it receives from the valid PICC back to the proxy. Meanwhile, the proxy communicates with Reader 2, presenting data to the valid PCD which was received from the mole. Using this methodology, the proxy is able to circumvent Reader 2's distance-bounding protocol and successfully gain access to Reader 2 without the consent or knowledge of the smartcard holder.

So, how can such an attack be implemented? The Proxmark 3 has the ability to act as both a mole and a proxy. Then, hardware needs to be chosen to implement the relay between the mole and proxy, which can be wired or wireless. I present both wired and wireless relays, although wireless relays are more likely to be used for a distance of 25 km. The implementation of (1) the mole and proxy, and (2) the wireless relay are discussed in detail below.

1. **Mole and Proxy Implementation** – Two-way communication between the mole and proxy is required in order to implement the relay attack previously shown in Figure 2.1. A simple way of communicating between the two Proxmarks over a wired relay is using digital communication. With digital communication, the communication between the two Proxmarks would take place after the Proxmark 3 has demodulated the signal coming from the valid PCD or PICC, introducing four delays:

   (a) The time for the proxy to demodulate the signal received from the valid PCD.

   (b) The time for the mole to fully receive and send to the valid PICC, at 16.30 MHz, the message received from the proxy.

   (c) The time for the mole to demodulate the signal received from the valid PICC.

   (d) The time for the proxy to fully receive and send to the valid PCD, at 13.56 MHz, the response received from the mole.

   A timing diagram with these delays is shown in Figure 4.1, which shows each of the four timing delays that occur after each component in the

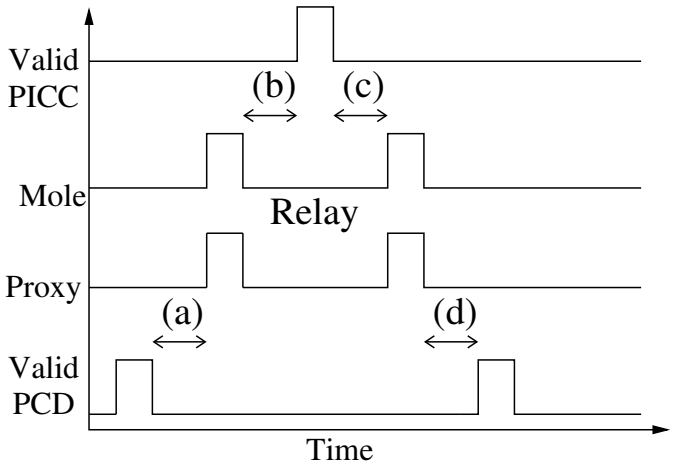


Figure 4.1: The four relay attack processing delays. Original image by Clulow et al. [3].

relay does its processing.

The two demodulation delays (delays (a) and (c)) can be calculated based on the number of clock cycles which occur after the valid PCD and valid PICC send their data. The Proxmark's ADC takes 3 `osc_clk` clock cycles (0.22 µs at 13.56 MHz) to convert analog data into its digital form, and the Proxmark 3's Gaussian derivative filter uses the 4 previous ADC outputs to perform edge detection. Therefore, the total demodulation delay is estimated to be 7 clock cycles. With a frequency of 13.56 MHz, this corresponds to a delay of 0.52 µs, or 0.43 µs with a frequency of 16.30 MHz. Assuming this delay occurs once per demodulation, the total demodulation delay time is estimated at about 1 µs.

To experimentally test the demodulation delay, I measured the delay between the overclocked reader (mole) receiving a response from the valid PICC and sending that response out on a serial channel (delay (c)). The results of conducting this test with a Mifare Plus EV1 clocked at 16.00 MHz are shown in Figure 4.2, where the top plot shows the output of `curbit` and the bottom plot shows the PICC's raw response as eavesdropped using a near-field probe.

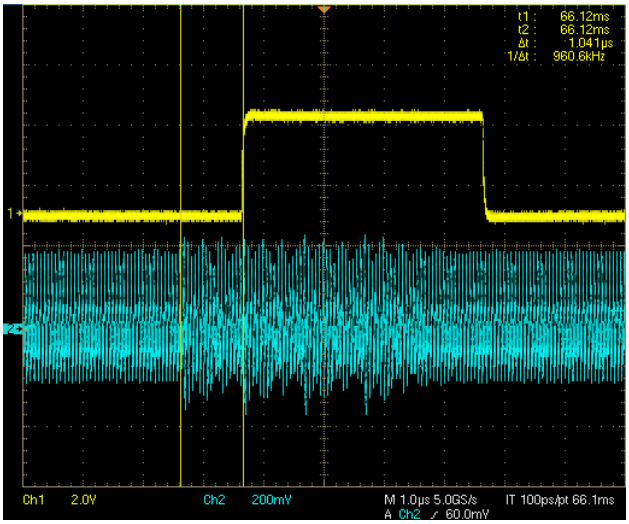


Figure 4.2: Delay between the eavesdropped PICC response and the mole's serial output.

The cursors in Figure 4.2 show that the delay between when the PICC turns on its 847.5 kHz subcarrier and when the mole's serial output (`curbit`) changes is about 1 µs. Given that the delay time of 7 clock cycles at 16.00 MHz would correspond to a predicted delay of 0.44 µs, the 7-cycle prediction is off by a factor of approximately 2. This is possibly due to the Proxmark 3's analog circuitry delay and the time the Proxmark 3 takes to perform edge detection.

The two remodulation delays (delays (b) and (d)) are partially due to the frequency difference between 13.56 MHz and 16.30 MHz. When the proxy sends data to the mole, it does so at a frequency of 13.56 MHz. However, the mole needs to communicate with the valid PICC at a frequency of 16.30 MHz, forcing it to wait to receive data from the 13.56 MHz proxy before it is able to forward all of that data to the PICC. The resulting delay time is estimated to be 1 µs.

With an estimated two 1 µs delays for remodulation and two 1 µs delays for demodulation, the total delay to perform a wired relay would be 4 µs. With such a delay, the time gap with the Mifare Plus would still be over 0.28 ms, and the time gap with the Mifare DESFire would

still be over 0.27 ms, allowing a relay attack to still be conducted at a round-trip distance of over 80 km.

For a wired relay, digital communication can take place over a dedicated wire used for half-duplex communication between the mole and proxy. This wire can use the TP7 `dbg` signal on the two Proxmarks, over which digital communication can occur between the two FPGAs. Given that the FPGA does no decoding, the 0s and 1s which get sent serially to the other Proxmark will still be encoded as either Modified Miller or Manchester data. A relay attack would work as follows:

(a) Reader 2 sends a request to the proxy Proxmark 3. That request is received by the proxy Proxmark 3 and demodulated using its FPGA (delay (a)). It immediately sends the bits over TP7 to the mole.

(b) The mole Proxmark 3 ensures the standard Reader 1's communication with the PICC is complete and that Reader 1 is not active. The mole tells the proxy to repeat step (a) until Reader 1 finishes its standard transaction.

(c) The mole remodulates the request received over TP7 at the overclocking frequency of 16.30 MHz and sends the request to the PICC (delay (b)).

(d) The mole demodulates the PICC's 16.30 MHz response (delay (c)) and sends the corresponding bits (using `curbit`, which contains Manchester encoded PICC data) to the proxy over TP7.

(e) The proxy receives the PICC's response and sends it at 13.56 MHz to the valid PCD (delay (d)).

(f) Reader 2 continues to send requests to the proxy and the relay continues until access is granted by Reader 2.

2. **Wireless Relay Implementation** – Implementing a wireless relay is more complicated than implementing a wired relay. A naïve way of wirelessly relaying the 13.56 MHz and 16.30 MHz signals being sent

back and forth is to directly amplify the signals which are transferred between the mole and valid PICC, or proxy and valid PCD. However, doing so is not practical because of the large antenna which would need to be used at such low frequencies (a half-wave dipole would be 11 m long at 13.56 MHz), as well as the problem of distinguishing between which direction the signals are going in.

Because of this, a higher frequency needs to be used for wireless communication, such as a UHF frequency (300 MHz–3 GHz). Hancke, for example, used the Micrel QwikRadio MICRF103 and MICRF005 to set up an 868 MHz OOK channel for mole to proxy communication and a 915 MHz OOK channel for proxy to mole communication [16]. The QwikRadio chips were capable of communication rates up to 115 kbps, allowing the 106 kbps information rate specified by ISO 14443 to be satisfied.

To do something similar on the Proxmark 3, the bits which are received from the valid PICC or valid PCD need to be demodulated and sent serially to a UHF transceiver module. Two separate pins need to be used by both the mole and proxy to talk to a wireless module – one for transmitting and one for receiving – due to the separation of Tx and Rx on most wireless communication chips.

The send and receive signals needed can be set up as I/O pins on the Proxmark 3 FPGA. However, the PCB currently only gives one explicit FPGA debug pin, TP7. `ANT_LO` (TP2), however, can be used through the `CROSS_LO` input into the FPGA, despite the path between `ANT_LO` and `CROSS_LO` having 2 resistors and a comparator. Ideally, the Proxmark 3 Eagle schematic would be changed to include more FPGA debug pins than just TP7.

Given that the normal ISO 14443 106 kbps data rate would become 20.2% higher, or 127.4 kbps, when overclocking at 16.30 MHz, using a higher bandwidth chip than the 115 kbps one used by Hancke would be necessary. An example of a chip which can achieve a data rate of

127.4 kbps is the Murata TRC103, which has a maximum data rate of 200 kbps. The TRC103 can be communicated with using separate SPI channels for sending and receiving, and can be run at frequency ranges of 863-870, 902-928 and 950-960 MHz. To be fully implemented, it needs an external 50 Ω antenna, an RF SAW filter (Surface Acoustic Wave), a 12.8 MHz crystal, and a few passive components. The DR-TRC103 development kit is readily available with these components integrated and offers a range of up to 1 km.

With a wireless relay such as the TRC103, steps (a) and (d) of the relay attack would involve, rather than communicating with the mole or proxy over a wire, sending data to the TRC103 over SPI. Additionally, all the delays present in the wired relay attack would still exist, plus an additional delay imposed by using two TRC103s. However, these delays would likely be on the order of microseconds, and therefore the time gaps for the Mifare DESFire and Mifare Plus would still be on the order of 0.27 ms and allow a relay attack to be conducted at a one-way distance of over 40 km.

4.4 Recommendations

Although the Mifare Plus and Mifare DESFire appear to have minimum and maximum frequencies at which they operate, they are still susceptible to overclocking relay attacks. In order to prevent overclocking, Reid et al. suggest implementing a low pass filter [35]. Clulow et al. recognize the same threat and suggest using an independent time reference [3]. Implementing extra circuitry to adjust the clock to 13.56 MHz might cost more money, but doing so is vital to successfully preventing relay attacks. If a company is serious about implementing a precise distance-bounding protocol, they need to take into account the importance of precise timing and implement their product accordingly.

Chapter 5

Conclusion

5.1 Summary

In this report, I have shown that it is possible to overclock two popular contactless smartcards, allowing a relay attack to be conducted on these cards at a one-way distance of over 40 km, despite the existence of a distance-bounding protocol. I have used these results to present a setup for conducting a relay attack in which open-source tools such as the Proxmark 3 are used. To prevent relay attacks, I suggest that contactless smartcards do not base their communication speeds on the externally supplied carrier frequency, but instead use an independent time reference such as a local oscillator.

Additionally, I have shown how to collect precise timing measurements for the distance-bounding protocols of both the Mifare Plus EV1 and the Mifare DESFire EV2. I described the methodology of how I collected these timing results using the Proxmark 3 and presented timing measurements of both cards at both 13.56 MHz and overclocking frequencies.

5.2 Future Work

As contactless smartcards and RFID technology grow in popularity, relay attacks and distance-bounding protocols will only continue to grow in importance. It is important to take precise timing measurements on other RFID tags that implement distance-bounding protocols as they become available. Because all the Proxmark 3 code used in this project is available on Github, tests can easily be conducted on such tags, especially if they conform to ISO 14443A. Extensions can also be made to the Proxmark 3 or other tools to measure time-sensitive NFC transactions on smartphones.

Another future direction of research is to use a tool other than the Proxmark 3 to increase the precision of the timestamps taken to a granularity finer than 62.5 ns. Timing measurements can also be taken on a larger number of Mifare Plus EV1 and Mifare DESFire EV2 cards, giving more insight into environmental effects on the cards' timing as well as manufacturing variables which might have an effect.

A further extension to this work is to carry out the proof-of-concept relay attack using the hardware and methodology described in Section 4.3. Doing so would help communicate the threats which overclocking a contactless smartcard poses as well as prove that conducting such an attack is both easy and inexpensive.

Bibliography

[1] Martin Henzl, Petr Hanacek, and Matej Kacic. Preventing real-world relay attacks on contactless devices. In *Security Technology (ICCST), 2014 International Carnahan Conference on*, pages 1–6. IEEE, 2014.

[2] Identification cards – Contactless integrated circuit cards – Proximity cards. Standard, International Organization for Standardization, Geneva, CH, 2016.

[3] Jolyon Clulow, Gerhard P Hancke, Markus G Kuhn, and Tyler Moore. So near and yet so far: Distance-bounding attacks in wireless networks. In *European Workshop on Security in Ad-hoc and Sensor Networks*, pages 83–97. Springer, 2006.

[4] EMVCo. EMV Level 1 Specifications for Payment Systems – EMV Contactless Interface Specification, Version 3.0. EMVCo, 2018.

[5] Vinay Deo, Robert B Seidensticker, and Daniel R Simon. Authentication system and method for smart card transactions, Feb 1998. US Patent 5,721,781.

[6] Oliver Kömmerling and Markus G Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. *Smartcard*, 99:9–20, 1999.

[7] Samy Bengio, Gilles Brassard, Yvo G Desmedt, Claude Goutier, and Jean-Jacques Quisquater. Secure implementation of identification systems. *Journal of Cryptology*, 4(3):175–183, 1991.

[8] Marci Meingast, Jennifer King, and Deirdre K Mulligan. Embedded RFID and everyday things: A case study of the security and privacy risks of the US e-passport. In *RFID, 2007. IEEE International Conference on*, pages 7–14. IEEE, 2007.

[9] Flavio D Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter Van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs.

Dismantling MIFARE Classic. In *European symposium on research in computer security*, pages 97–114. Springer, 2008.

[10] Mike Bond, Omar Choudary, Steven J Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and Skim: cloning EMV cards with the preplay attack. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 49–64. IEEE, 2014.

[11] John Horton Conway. On numbers and games. 1976.

[12] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[13] Yvo Desmedt, Claude Goutier, and Samy Bengio. Special uses and abuses of the Fiat-Shamir passport protocol. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 21–39. Springer, 1987.

[14] James Gleick. A new approach to protecting secrets is discovered. *The New York Times*, 18:C1, 1987.

[15] Yvo Desmedt. Major security problems with the "unforgeable" (Feige)-Fiat-Shamir proofs of identity and how to overcome them. In *Proceedings of SECURICOM*, volume 88, pages 15–17, 1988.

[16] Gerhard P Hancke. A practical relay attack on ISO 14443 proximity cards. *Technical report, University of Cambridge Computer Laboratory*, 59:382–385, 2005.

[17] Gerhard P Hancke. Practical attacks on proximity identification systems. In *Security and Privacy, 2006 IEEE Symposium on*, pages 328–333. IEEE, 2006.

[18] Ziv Kfir and Avishai Wool. Picking virtual pockets using relay attacks on contactless smartcard systems. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 47–58. IEEE, 2005.

[19] Saar Drimer and Steven J Murdoch. Keep Your Enemies Close: Distance Bounding Against Smartcard Relay Attacks. In *USENIX security symposium*, volume 312, 2007.

[20] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical NFC peer-to-peer relay attack using mobile

phones. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 35–49. Springer, 2010.

[21] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical relay attack on contactless transactions by using NFC mobile phones. In *Cryptology and Information Security Series*. 2012.

[22] Michael Roland. Applying recent secure element relay attack scenarios to the real world: Google Wallet Relay Attack. *arXiv preprint arXiv:1209.0875*, 2012.

[23] Jordi van den Breekel and BlackHat Asia. Relaying EMV contactless transactions using off-the-shelf android devices. *BlackHat Asia, Singapore*, 2015.

[24] West Midlands Police. WATCH: Police release footage of relay crime. `https://www.west-midlands.police.uk/news/4544/watch-police-release-footage-relay-crime`, 2017. Accessed: 09 May 2018.

[25] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science, 2011.

[26] Alexei Czeskis, Karl Koscher, Joshua R Smith, and Tadayoshi Kohno. RFIDs and secret handshakes: Defending against ghost-and-leech attacks and unauthorized reads with context-aware communications. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 479–490. ACM, 2008.

[27] Cas Cremers, Kasper B Rasmussen, Benedikt Schmidt, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 113–127. IEEE, 2012.

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

[29] Thomas Beth and Yvo Desmedt. Identification tokensor: Solving the chess grandmaster problem. In *Conference on the Theory and Application of Cryptography*, pages 169–176. Springer, 1990.

[30] Gerhard P Hancke and Markus G Kuhn. An RFID distance bounding protocol. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 67–73. IEEE, 2005.

[31] Gildas Avoine, Muhammed Ali Bingöl, Süleyman Kardaş, Cédric Lauradoux, and Benjamin Martin. A framework for analyzing RFID distance bounding protocols. *Journal of Computer Security*, 19(2):289–317, 2011.

[32] Gildas Avoine, Sjouke Mauw, and Rolando Trujillo-Rasua. Comparing distance bounding protocols: A critical mission supported by decision theory. *Computer Communications*, 67:92–102, 2015.

[33] Gerhard P Hancke and Markus G Kuhn. Attacks on time-of-flight distance bounding channels. In *Proceedings of the first ACM conference on Wireless network security*, pages 194–202. ACM, 2008.

[34] Kasper Bonne Rasmussen and Srdjan Capkun. Realization of RF Distance Bounding. In *USENIX Security Symposium*, pages 389–402, 2010.

[35] 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*, pages 204–213. ACM, 2007.

[36] Sébastien Gambs, Carlos Eduardo Rosar Kós Lassance, and Cristina Onete. The Not-so-Distant Future: Distance-Bounding Protocols on Smartphones. In *International Conference on Smart Card Research and Advanced Applications*, pages 209–224. Springer, 2015.

[37] Kevin Soules, Darren Hurley-Smith, and Julio Hernandez-Castro. Measuring the Distance: Investigating the DESFire EV2 Distance Bounding Protocol. *Cryptacus 2017*, 2017.

[38] EMVCo. EMV Contactless Specifications for Payment Systems, Book C-2, Kernel 2 Specification, Version 2.6. pages 97–98. EMVCo, 2016.

[39] Steven J. Murdoch. Do you know what you're paying for? How contactless cards are still vulnerable to relay attack. `https://www.benthamsgaze.org`, 2016. Accessed: 15 May 2018.

[40] Karsten Nohl. MIFARE, little security, despite obscurity. In *the 24th Congress of the Chaos Computer Club in Berlin, December 2007*, 2007.

[41] Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz. Reverse-Engineering a Cryptographic RFID Tag. In *USENIX security symposium*, volume 28, 2008.

[42] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D Garcia. A practical attack on the MIFARE Classic. In *International Conference on Smart Card Research and Advanced Applications*, pages 267–282. Springer, 2008.

[43] Carlo Meijer and Roel Verdult. Ciphertext-only Cryptanalysis on Hardened MIFARE Classic Cards. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 18–30, New York, NY, USA, 2015. ACM.

[44] Identification cards – Contactless integrated circuit cards – Vicinity Integrated Circuit(s) Card. Standard, International Organization for Standardization, Geneva, CH, 2010.

[45] John Wehr. Is the Debate Still Relevant? An in-depth look at ISO 14443 and its competing interface types. `https://www.secureidnews.com/news-item/`, 2003. Accessed: 31 May 2018.

[46] Klaus Finkenzeller. *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. John Wiley & Sons, 2010.

[47] Advanced Card Systems Ltd. ACR122U USB NFC Reader. `https://www.acs.com.hk/en/products/3/acr122u-usb-nfc-reader/`, 2018. Accessed: 15 May 2018.

[48] Identiv. SCL3711 Contactless USB Smart Card Reader. `https://support.identiv.com/scl3711/`, 2018. Accessed: 15 May 2018.

[49] Flavio D Garcia, Gerhard de Koning Gans, and Roel Verdult. Tutorial: Proxmark, the swiss army knife for RFID security research. *Technical Report, Radboud University Nijmegen*, 2012.

[50] Ventzislav Nikov and Marc Vauclair. Yet Another Secure Distance-Bounding Protocol. *SECRYPT*, 2008:218–221, 2008.

[51] Peter Thueringer, Hans De Jong, Bruce Murray, Heike Neumann, Paul Hubmer, and Susanne Stern. Decoupling of measuring the response time of a transponder and its authentication, 2011. US Patent App. 12/994,541.

Appendix A

Mifare Plus and Mifare DESFire Setup

Before the Mifare Plus EV1 and Mifare DESFire EV2 Proximity Checks could be performed, I had to initialize the cards.

On the Mifare Plus EV1, the card starts out in Security Level 0 (SL0). In order to perform the Proximity Check however, the card needs to be in either SL1 or SL3. I began my tests by using the Mifare Plus in SL1, but in SL1 the ACR122 had trouble communicating with the Mifare Plus due to the PC/SC library not completing the PPS (Protocol and Parameter Selection) check required by ISO 14443-4. Because of this, I ended up using the Mifare Plus in SL3. Before moving the Mifare Plus into SL1 or SL3, I explicitly set all the Mifare Plus keys, including the VCProximityKey, which is a 128-bit AES key used for MAC verification during the distance-bounding protocol. Setting keys on the Mifare Plus EV1 does not require an authentication while the card is in SL0.

On the Mifare DESFire EV2, the concept of security levels does not exist. However, I could not perform the Proximity Check on the DESFire until I explicitly set the VCProximityKey, which is disabled by default. Unlike the Mifare Plus EV1, setting keys on the Mifare DESFire EV2 requires an active

authentication to the card. To change keys on the DESFire, therefore, I used the ACR122 with the Python `pyscard` library because of the availability of cryptography libraries in Python such as `cryptography`. First, I performed a 3DES authentication to the DESFire using the PICCMasterKey, at which point I was able to set the VCConfigurationKey using the DESFire's ChangeKey command. Next, I performed an AES authentication to the DESFire using the VCConfigurationKey and used the ChangeKey command to set the VCProximityKey. After VCProximityKey had been explicitly set, the key was enabled and I was able to perform the Proximity Check using either the ACR122 or the Proxmark 3.