Master research Internship

Master Thesis

Implementing Distance-bounding protocols on Android smartphones

Domains : Cryptography and Security - Mobile Computing

Author:
Carlos E. R. K. LASSANCE

Supervisors:
Sébastien GAMBS
Cristina ONETE
Research Team: CIDre

Abstract

In authentication protocols, a relay attack allows an adversary to impersonate a legitimate prover, possibly located far away from a verifier, by simply forwarding messages between these two entities. The effectiveness of such attacks has been demonstrated in practice against ISO 14443-compliant smartcards and car-locking mechanisms. Distance-bounding (DB) protocols, which enable the verifier to use a clock to verify its proximity to the prover, are a promising countermeasure against relay attacks. In such protocols, the verifier measures the time elapsed between sending a challenge and receiving the associated response of the prover to estimate their proximity. So far, distance-bounding has remained mainly a theoretical concept. In practice, only three ISO 14443-compliant implementations exist: two proprietary smartcard ones and one on highly-customized hardware. During the internship we developed proofs-of-concept implementations of the Swiss-Knife DB protocol on smartphones running in RFID-emulation mode. To the best of our knowledge, this is the first time that such an implementation has been performed. Our experimental results are encouraging, as they show that relay attacks introducing more than 1.5 ms can be detected in our scenarios (in general off-the-shelf relay attacks introduce at least 10 ms of delay). We also leverage on the full power of the ISO-DEP specification to implement the same protocol with 8-bit challenges and responses, thus reaching a better security level per execution without increasing the possibility of relay attacks. The analysis of our results leads to new promising research directions in the area of distance-bounding.

Contents

Glossary

MET , Minimum Experimental Time: the minimum time found for a round during all our executions of the protocol and considered the minimum accepted time by the system. Each implementation has its own MPT.

MTT , Minimum Transmission Time: is the minimum time needed by the the physical devices to handle the data as required by the protocols and the standards the communication is object to. Defined in 3.5.

R , Range: it is the amount of time our system tolerates before considering a round invalid. This range goes from the MET to MET+R. Any round outside this range is considered invalid.

E_{MAX} , Maximum number of errors: how many incorrect/overlong rounds are accepted out of the total number of N_c of fast challenge responses rounds. Also called tolerance threshold.

$\mathcal{A}$, Adversary: this is a model of one or multiple (colluding) adversaries, whose goal is to attack the protocol or commit fraud.

$\mathcal{P}$, $\mathcal{P}^*$, Prover: device that wants to authenticate himself to the system. This denotes in practice a phone in NFC mode, used to authenticate the owner with respect to a verifier. If the prover (and its owner) behave honestly, we use the notation $\mathcal{P}$. If the prover is adversarially controlled or colluding with an adversary, we denote it as $\mathcal{P}^*$:

$\mathcal{V}$, $\mathcal{V}^*$, Verifier: device responsible to authenticate users. In our experiments this is an NFC smartphone, which can authenticate legitimate provers within its proximity. We denote "honest" verifiers by $\mathcal{V}\dot{I}$f an adversary impersonates a verifier we denote that entity as $\mathcal{V}^*$.

N_c , the number of challenge-response rounds in each protocol execution.

t_{max} , Proximity bound/threshold: total amount of time (or distance converted time) accepted by the protocol as a valid round.

execution : a full run of the entire distance-bounding protocol, between two designated participants, typically an honest prover and an honest verifier.

round : an exchange of two messages, by default, the verifier is assumed to first send a message to which the prover sends a response.

1 Introduction

Radio Frequency IDentification (RFID) is a widespread, cost-efficient technology, which is currently used for applications ranging from contactless payment to public transport and machine-readable identification. A fundamental cryptographic primitive that must be supported and implemented on RFID technology is *authentication*. During an authentication protocol, the device equipped with an RFID chip (also called the *prover*) interacts with a reader (called the *verifier*) to prove its legitimacy. To protect the confidentiality of the data stored and manipulated by RFID tags for sensitive applications, these devices must be equipped with a processor capable of performing cryptographic operations, and they must provide a secure environment for the processing and transmission of data.

RFID chips may be embedded in devices such as passive tags securing items in supermarket, public transport cards, as well as contactless-payment smartcards. Thus, the term "RFID prover" covers a wide range of hardware, which differs in terms of characteristics such as memory, surface area, functional independence with respect to the verifier (*i.e.*, passive versus active) and cost. However, traditional RFID provers always answer requests from the reader without asking for the user's consent. This property makes RFID technology prone to *relay attacks*. These attacks, also called mafia fraud [7] in the context of authentication, or wormhole attacks in the field of neighborhood discovery, allow an adversary to impersonate a legitimate prover by forwarding messages between him and the legitimate verifier. Relay attacks have been successfully implemented against Bluetooth [14], ISO 14443 smartcards [6, 15], electronic passports [20], electronic voting schemes [28], and even access mechanisms for cars such as Passive Keyless Entry and Start (PKES) [10].

Essentially relay attacks work by extending the intended transmission range for which the system was designed and ensuring that messages from a far-away prover are used to authenticate to a verifier. One way to prevent this type of attack is by limiting the time a transmission may take, thus essentially giving the verifier an upper-bound of the distance between him and the prover.

This idea lies at the core of *distance-bounding* (DB) protocols [5], introduced by Brands and Chaum to counter relay attacks. Thus, a DB protocol extends classical authentication schemes by additionally enabling the verifier to check his proximity to the prover. Most RFID tags operate according to the ISO-14443 standard, operating at a proximity range of about 10 cm. In a DB protocol, the verifier is equipped with a clock, which measures the round trip time of fast challenge/response rounds. As RFID communication takes place at the speed of light, such time measurements theoretically reflect the communication distance accurately, and can be used for proximity checking.

While the literature abounds with DB protocols (see for instance [2, 3, 19, 5, 17, 24, 8]), to the best of our knowledge only three practical implementations of distance-bounding for RFID currently exist.

The first one is a highly-customized (and expensive) proof-of-concept implementation by Ranganathan, Tippenhauer, Singelée, Škorić, and Čapkun [29]. The two other solutions, namely the Proximity Check option for NXP's Mifare Plus cards and the solution of 3DB Technologies, come from the industry, and their specifications are not public with the protocols being implemented on a proprietary (*i.e.*, customized) hardware. To our knowledge, the proximity check distance-bounding protocol is the first RFID DB-implementation that does not rely on using analog data during the fast challenge/response transmissions. However, no security analysis, nor protocol details are publicly available for this protocol. By contrast, the publicly-available implementation to [29] avoids converting the analog to digital data in the challenge-response phase to avoid the extra delays caused by the conversion. This solution is almost pin-point accurate detecting any delay greater than 2.75 nanoseconds.

Modern smartphones can use Near Field Communication (NFC) technology to *emulate* RFID tags. In this mode, the phone can behave either as an RFID prover or as a verifier, and its communication is subject to the ISO-DEP protocol, defined in the ISO14443-4 standard [22, 27]. Smartphones have already been used by Francis, Hancke, Mayes, and Markantonakis to conduct relay attacks [11]. Their work demonstrates that any application ranging from credit cards to electronic passports can be attacked by using NFC phones. More precisely, they showed that the authentication protocol was still running even when the relay attack introduced a delay of several seconds, in contrast to the implicit proximity requirement of ISO 14443. Expressed in terms of distance, this is equivalent to the prover being thousands of kilometers away from the verifier.

Using NFC smartphones to prove and verify legitimacy can be useful as they are widely available and highly accept and used. Indeed 275 million units were shipped in 2013 and more than 1 billion units are forecast for 2018 [21] as shown in Figure 1. Thus, an authentication system only relying in on smartphones is an interesting potential application, which can have many practical uses. However, if the authentication is not well secured, it can cause massive damage, ranging from illegitimate access to privileges to impersonation attacks. Therefore, implementing distance-bounding protocols on smartphones is an essential prerequisite to preventing relays and other attacks extending the legitimate communication range in authentication.

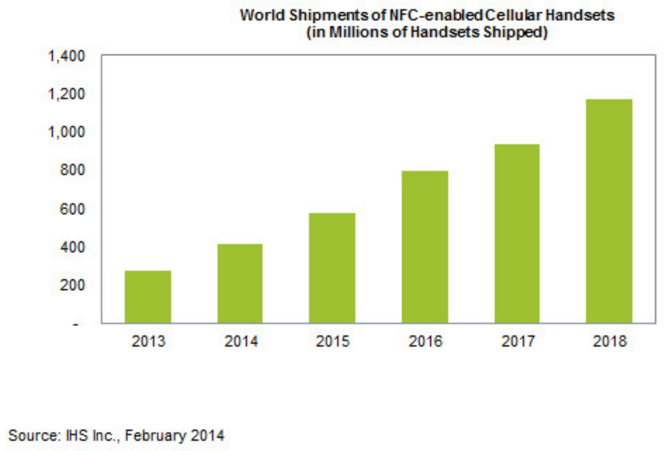


Figure 1: World shipments of NFC-enabled smartphones [21].

Main contributions. During this internship, we investigated the implementability of DB protocols on smartphones running in a tag-emulation mode. More specifically, we explored how DB can be implemented on smartphones that use the Android operational system, *without* changing the design of the hardware (including the SIM card, the phone's processor, or the phone itself). In contrast to the proximity check solutions of Mifare Plus and 3DB Technologies, our implementations are public and do not require proprietary hardware. The main objective was to evaluate how far one can implement DB protocols modyfing only the software of an existing smartphone. Our implementation does not rely on the SIM card as a cryptographic processor, nor customize its properties. Rather, we use the processor of the smartphone directly. As a consequence, the implementation is easier to adopt though prone to side-channel and malware attacks.

We furthermore evaluate our results, we first assessing the theoretical limits of our implementation. In particular, without tampering with the hardware/protocols involved in the communication the minimum possible communication time for an honest run of the DB protocol is 3.62 ms. This limit is due to the overhead induced by the communication standards between the smartphones and also inside the smartphone, between their processors and the NFC chip, being inherent to the underlying communication protocols. Converted in kilometers, this represents approximately a distance of 543 km.

We show three proof-of-concept Android implementations of increasing complexity always relying on the Swiss-Knife DB protocol [24]. All these three implementations use 32 challenge-response rounds. This number of rounds was chosen as it allowed us to show that there is a variance in the roundtrip times of each execution and also allowed us to experiment different values of accepted rounds while still keeping the protection against malicious prover high and also to have a reasonable total execution time of the protocol. We seek an implementation that does not have a great variance in its roundtrip values while having most of the results clustered close to the minimal value, because if our results show a wide-spread of values that are centered far away from the minimal value it either reduces our security (our protocol has to accept a higher range of values) or it will lead to a inefficient implementation (instead of rejecting only the illegitimate executions it rejects almost every execution).

1. **The basic implementation.** This first implementation works at the application layer and does not require root access. While it can be used straightforwardly, our results show that the roundtrip time measurement presents a lot of variation. More precisely, the measured standard deviation was 4.4 ms and the distribution of roundtrip times contained featured multiple clusters of value (i.e the values were grouped around multiple times instead of being mostly composed of the same values).

2. **The customized implementation.** In this second implementation, the operating system of Android is modified to make the protocol run at the Hardware Abstraction Layer (HAL). More precisely, both the verifier and the prover are customized to decrease the running time introduced by him during the DB protocol. With this implementation, the results shown a normal distribution of measured roundtrip values, which display almost no false negatives (i.e legitimate executions being diagnosed as illegitimate) when the protocol is implemented with an error threshold of 18 rounds and an allowed variation of 1.5 ms from the minimal observed measurement.

3. **The 8-bit implementation.** The phone communicates with its NFC chip via a serial port, which encodes the 1-bit challenges of the Swiss-Knife protocol as an entire byte. For this third implementation the objective was to explore how the customized implementation behaves when the transmitted byte is used to encode 8-bit challenges and responses. The results were surprisingly encouraging, Since the measured times present little variation we already have a better security level per execution of this 8-bit implementation than for previous one bit implementations for a proximity threshold within 1 ms of the minimum time. This last result indicates that designing DB protocols using larger challenges and response is a promising and viable future research direction.

For each implementation we analyzed the data in order to choose the parameters for the proximity bound and for the threshold value of erroneous transmissions. In particular, choosing a proximity bound within 1.5 ms of the observed minimum is an optimal value for our second implementation and a good one for our third implementation (by contrast, the first implementation requires a much larger allowed variation). In terms of fault tolerance and the resulting security level, the third implementation largely outperforms the other two. Indeed, even if 20 out of 32 rounds are faulty, this implementation still ensures a better security level than the 1 bit lower level implementation with 32 out of 32 rounds required to be correct. In our experiments we detected that requiring all rounds to be correct tends to cause a high occurrence of false negatives, an optimal value seems to be 12 out of 32 rounds.

However a somewhat negative result of our investigation is that our implementations cannot detect relays at less than 1.5 ms if we aim for a security level higher than 2^{-90} with a false negative level smaller than 10%. Nonetheless, the variation of 1.5 ms would still allow us to detect most relays with off-the-shelf hardware, which introduce delays of at least 10 ms [11, 18]. If an adversary uses custom hardware, e.g that of Hancke [15], this delay can be reduced to several microseconds becoming undetectable. These results are encouraging in the sense that our Swiss-Knife implementations can only be defected by an adversary using more sophisticated technology than off-the-shelf mechanisms.

The outline of this thesis is the following. In Chapter 2, we introduce the state-of-the-art of relay attacks and distance-bounding protocols. Then in Chapter 3, we explain how NFC communication and RFID Tag-emulation work in Android assessing the limitations of the roundtrip times resulting from all the standards involved and define a theoretical minimal time. Afterwards, in Chapter 4, we describe our methodology, outline our results analyzing them in Chapter 5. Finally, we conclude in Chapter 6 with a discussion and future research directions.

2 Relay attacks and distance-bounding protocols

In this Chapter, we will first introduce relay attacks and then review a possible counter-measure against them, which is the use of distance-bounding protocols.

2.1 Authentication protocols and relay attacks

The basic symmetric-key authentication protocol described in Figure 2 relies on a challenge-response exchange between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$. Both participants share an identifier of the prover denoted $\mathcal{ID}$ and the secret key sk. After establishing the connection, the verifier sends a random nonce $N_{\mathcal{V}}$, chosen from a large space (depending on a security parameter), and receives a response a such that $a \leftarrow \mathsf{PRF}(\mathsf{sk}, N_{\mathcal{V}})$, for a pseudorandom function PRF (typically implemented as H-MAC or a light weight block cipher). Most authentication protocols, including this one, can resist active attacks, even if the attacker can interact with the prover *and* the verifier as long as no relaying of messages is involved. The security level depends on the size of $N_{\mathcal{V}}$ and on the security of the PRF.

Symmetric key authentication protocols are a basic building block (primitive) in the cryptographic literature. They enable a verifier to check the legitimacy of a prover in the case that the two protocol participants share a secret key. Authentication protocols must have two basic properties:

1. Correctness: a legitimate prover is always authenticated.
2. Soundness: an illegitimate entry (an adversary) is never authenticated.

Relay attacks differ from traditional MiM[1] attacks in the sense that the adversary does not break a cryptographic aspect of the protocol, he just needs to route the data between a prover and a verifier. An example of relaying is illustrated in Figure 3. The goal of the adversary is to impersonate the prover $\mathcal{P}$ authenticating to the verifier as $\mathcal{ID}$ to do so it simply transmits the data exchanged between $\mathcal{P}$ and $\mathcal{V}$. In a MiM the adversary can fully control the data flow between the honest entities, being able to introduce,replay, or remove transmissions for example.

This type of attack is even more important in the RFID context because, many security applications of RFID implicitly assume a physical proximity between tags and readers, mostly due to the limited read range of RFID chips. This assumption fails, for instance, if a proximity card can be used to open a door located far away or if an RFID-based credit card can effect payment from a kilometer away [23].

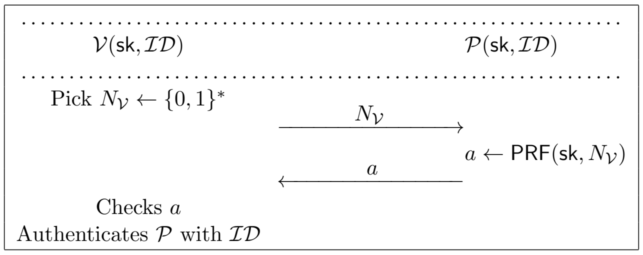

$\mathcal{V}(\mathsf{sk}, \mathcal{ID})$
$\mathcal{P}(\mathsf{sk}, \mathcal{ID})$
Pick $N_\mathcal{V} \leftarrow \{0,1\}^*$
$N_\mathcal{V}$
$a \leftarrow \mathsf{PRF}(\mathsf{sk}, N_\mathcal{V})$
a
Checks a
Authenticates $\mathcal{P}$ with $\mathcal{ID}$

Figure 2: Shared-key authentication protocol.

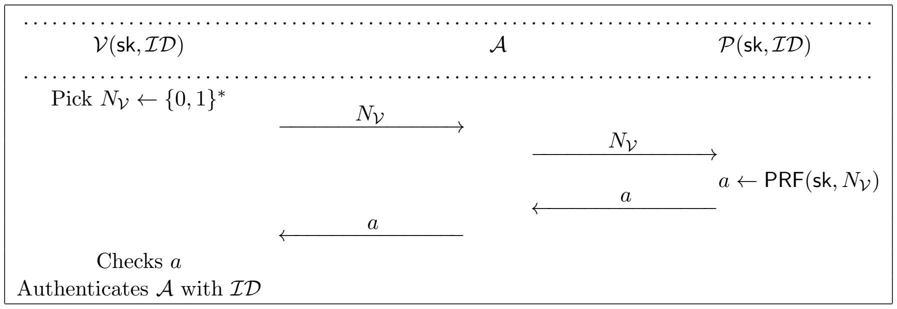

$\mathcal{V}(\mathsf{sk}, \mathcal{ID})$
$\mathcal{A}$
$\mathcal{P}(\mathsf{sk}, \mathcal{ID})$
Pick $N_\mathcal{V} \leftarrow \{0,1\}^*$
$N_\mathcal{V}$
$N_\mathcal{V}$
$a \leftarrow \mathsf{PRF}(\mathsf{sk}, N_\mathcal{V})$
a
a
Checks a
Authenticates $\mathcal{A}$ with $\mathcal{ID}$

Figure 3: Relaying example against the protocol from Figure 2.

2.1.1 Real instances of relay attacks

Relay attacks have been implemented in practice in many settings. In [15], Hancke conducted a relay attack against ISO 14443-compliant RFID tags. In his experiments, the honest prover and verifier were at a distance of 50 m from each other. Hancke created home-made devices to act as Leech and Ghost, at less then £100. The attack succeeded every time.

In [10], Francillon, Danev, and Čapkun demonstrated the possibility of using a relay attack against PKES (Passive Keyless Entry and Start) car systems. In this setting, a passive key is used for both unlocking and starting a car, and the device implicitly relies on a proximity. If the owner of

[1]Man in the Middle

the car is close to the car, carrying the passive key, e.g in their pocket, then the car should unlock. As soon as the token is placed close to the ignition, the car starts.However the security requirement is that the prover (and the key) are not close to the car, the latter should remain locked. This makes the PKES scenario highly vulnerable to relay attacks. The authors shown the possibility of creating the link between the honest prover and honest verifier by using an analogical coaxial cable and an antenna.

This attack demonstrated that out of more than 10 different PKES systems, installed in 9 cars (one system was an after market system not installed in any car), most of the models could be exploited even if the key was more than 50 meters from the car. One of the countermeasures proposed by the team was to incorporate distance bounding to the PKES protocol.

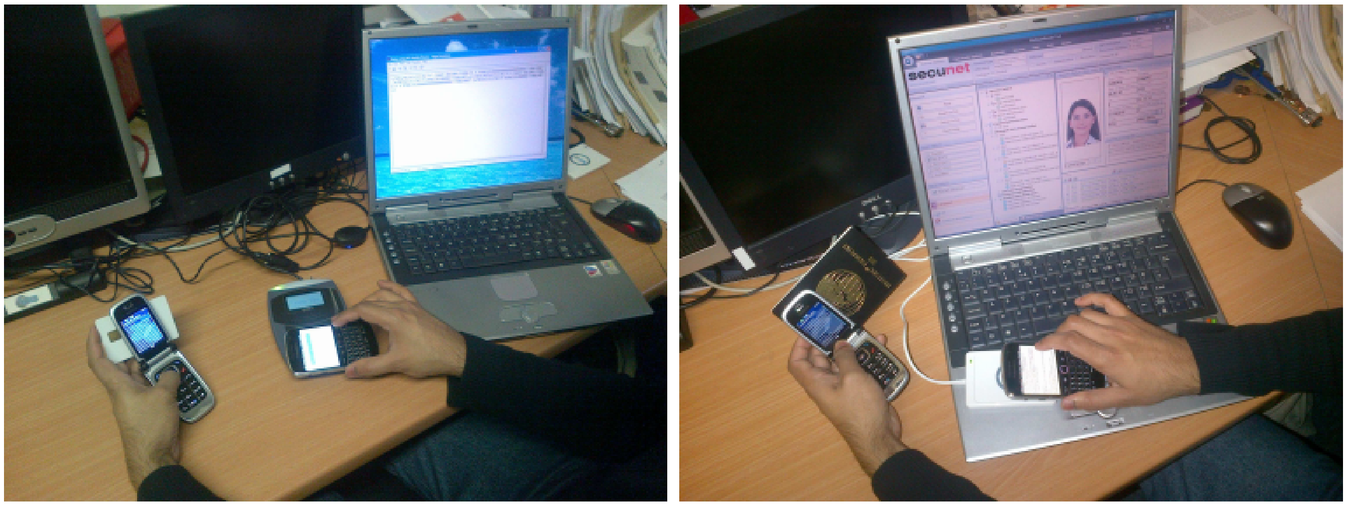


Figure 4: Smartphone attack on RFID tags presented in [11].

In another work, Francis, Hancke, Mayes, and Markantonakis [11] showed the possibility of using traditional hardware, in this case smartphones available in stores (a Nokia 6131 and a BlackBerry 9900 phone), to conduct a relay attack. In contrast to the previous attacks, there was no need to create or change any hardware; the attack relied only on the software.

More precisely, the authors developed an application in which the Nokia phone worked as a proxy-reader and the BlackBerry one as a proxy-token. This separation was made because the proxy-token part needed a card-emulation functionality only presented in more modern smartphones (for example it was only implemented in Android after version 4.4). The proxy-reader receives the information from the tag and sends it to the proxy-token, which relays this information to the real reader. The communication between proxy-reader and proxy-token was made via Bluetooth.

Francis et al. then proceeded by testing their attack against two types of systems, contactless credit cards and passports. Both systems were vulnerable to the attack and would accept a delay of up to 35 and 5.2 seconds respectively, while their attack added at least 50 ms to the . This minimal time is way lower than the maximum accepted time of the two systems.

In a more recent attack [18] Henzl, Hanáček, and Kačic showed another attack that used a off-the-shelf solution. Their minimal added delay was 27 ms, 23 ms quicker than the solution proposed by Francis et al.

2.2 Distance-bounding protocols

Distance-bounding protocols rely on fast roundtrip time measurements to establish the distance between the authenticated prover and the verifier. The latter is equipped with a clock, which is used to estimate the time-of-flight of a number N_c of challenge-response rounds. It is postulated in the distance-bounding literature [16] that both the challenges and the responses should be the size of a single bit, so as to ensure that measurement bases due to processing are minimal. Each response bit sent by $\mathcal{P}$ has to be emitted immediately after he receives the challenge from $\mathcal{V}$. The soundness of this approach relies on the fact that RFID transmissions are sent at the speed of light, which cannot be surpassed by an attacker.

With that in mind, distance-bounding protocols are based in two types of phases, slow and fast (sometimes called lazy and time-critical). In slow phases, the roundtrip time is not measured. Thus they may require a non-negligible processing time, allowing the use of correction and error-detection mechanisms. To fast phases the verifier measures the roundtrip time at each round. Thus only very simple processing tasks can be performed [16]. Many protocols use slow phases to exchange large setup parameters and to provide additional authentication. The fast phases are used to verify the prover's proximity, and sometimes provide further authentication.

2.2.1 Attacks in distance bounding

Distance-bounding protocols aim to defend against three different types of attacks: mafia, terrorist and distance fraud attacks.

Mafia fraud. Was first described by Desmedt, Goutier,and Bengio in [7]. In this attack, an adversary $\mathcal{A}$ consisting of a malicious prover $\mathcal{P}^*$ and a malicious verifier $\mathcal{V}^*$ by impersonating a legitimate prover $\mathcal{P}$ - see also Figure 5. Neither of the two honest parties is aware of the attack. The two adversarial devices, $\mathcal{P}^*$ and $\mathcal{V}^*$ are called Ghost and Leech respectively; "we assume that they communicate via a (secret), fast channel." secret communication link between the two that it also needs to approach $\mathcal{V}$. Once $\mathcal{P}^*$ is in the proximity of $\mathcal{V}$ and $\mathcal{V}^*$ is close to $\mathcal{P}$ the two adversarial devices mount a MiM attack to allow $\mathcal{P}^*$ to authenticate; however in the DB scenario, it is assumed that the verifier will always detect pure relaying of messages as depicted in figure 5.

RFID tags are especially vulnerable against this type of fraud as the tag generally does not know in advance when it will be activated, and it responds automatically without the user's concent.

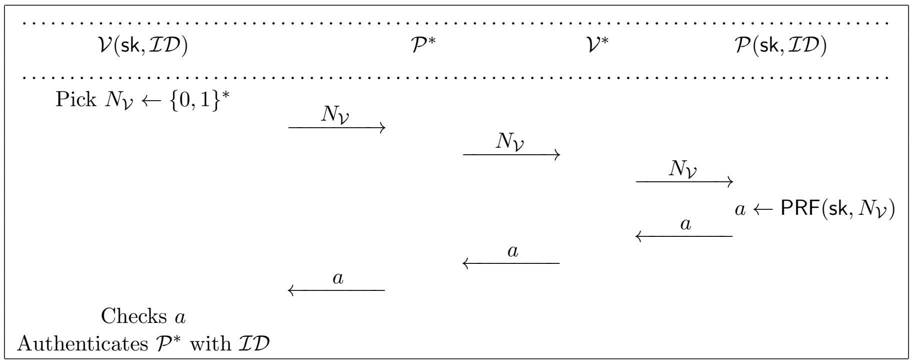

$\mathcal{V}(\mathsf{sk}, \mathcal{ID})$
$\mathcal{P}^*$
$\mathcal{V}^*$
$\mathcal{P}(\mathsf{sk}, \mathcal{ID})$
Pick $N_\mathcal{V} \leftarrow \{0,1\}^*$
$N_\mathcal{V}$
$N_\mathcal{V}$
$N_\mathcal{V}$
$a \leftarrow \mathsf{PRF}(\mathsf{sk}, N_\mathcal{V})$
a
a
a
Checks a
Authenticates $\mathcal{P}^*$ with $\mathcal{ID}$

Figure 5: Mafia fraud against the protocol from Figure 2.

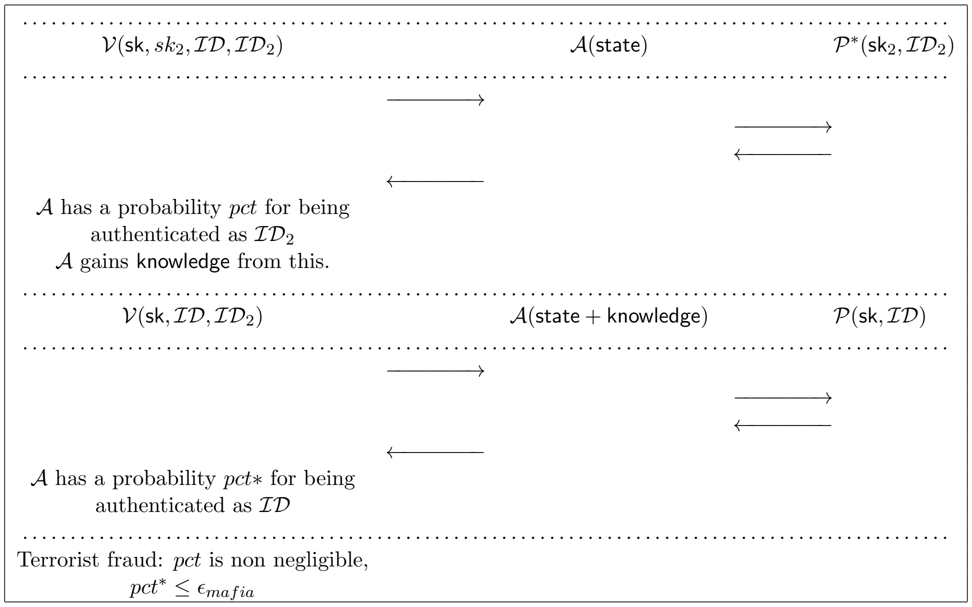
Terrorist fraud. In a terrorist fraud attack, a legitimate, but malicious, prover $\mathcal{P}^*$ collaborates with an adversary $\mathcal{A}$, now consisting of a single device. We assume that, while $\mathcal{P}^*$ is willing to help $\mathcal{A}$ authenticate, it is not willing to relinquish control over when $\mathcal{A}$ can impersonate it. Intuitively, the help $\mathcal{A}$ receives from $\mathcal{P}^*$ must:

1. allow $\mathcal{A}$ to authenticate with non-negligible probability as long as $\mathcal{P}^*$ helps $\mathcal{A}$;
2. not permit $\mathcal{A}$ to gain knowledge that would help it authenticate in later sessions.

In DB security models this second requirement means that $\mathcal{A}$ should not be able to, after being helped by $\mathcal{P}^*$ authenticate n times. he attacked the system the first time, to perform another attack, this time with an honest $\mathcal{P}$ and have a higher then mafia fraud success robability. This scenario is illustrated in Figure 6.

There exist several definitions of terrorist fraud resistance [9], which differs in how much the prover $\mathcal{P}^*$ can collaborate with the adversary. The two main differences between these definitions are:

1. how much the prover and adversary can communicate during a protocol execution;
2. the restriction on how much information an adversary might gain from $\mathcal{P}^*$



$\mathcal{V}(\mathsf{sk}, sk_2, \mathcal{ID}, \mathcal{ID}_2)$
$\mathcal{A}(\mathsf{state})$
$\mathcal{P}^*(\mathsf{sk}_2, \mathcal{ID}_2)$
$\mathcal{A}$ has a probability pct for being authenticated as $\mathcal{ID}_2$
$\mathcal{A}$ gains knowledge from this.
$\mathcal{V}(\mathsf{sk}, \mathcal{ID}, \mathcal{ID}_2)$
$\mathcal{A}(\mathsf{state} + \mathsf{knowledge})$
$\mathcal{P}(\mathsf{sk}, \mathcal{ID})$
$\mathcal{A}$ has a probability $pct*$ for being authenticated as $\mathcal{ID}$
Terrorist fraud: pct is non negligible, $pct^* \leq \epsilon_{mafia}$

Figure 6: Terrorist fraud definition from [9].

Distance fraud. In the case of distance fraud, a malicious prover $\mathcal{P}^*$ tries to convince $\mathcal{V}$ that it is closer to $\mathcal{V}$ than $\mathcal{P}$ is in reality. In this setting, the adversary is not faking its identity as it was the case for mafia and terrorist fraud attacks; instead, it's goal is to cheat on the proximity test of the verifier, one example of this attack on the shared key protocol is shown in Figure 7.

If one assumes that the verifier's clock is accurate then the malicious prover must send its messages in advance in order to make them arrive before the proximity bound t_{max}. In fast DB challenge-response rounds, $\mathcal{A}$ must guess the challenge in advance and send the response early. This is a very different attack from the previous two because the adversary is actually the owner of the credentials it is using but is trying to cheat the distance-bounding aspect of the protocol. For example, an adversary could try to forge an alibi that he was in a specific place at a given time, while it was actually far from that location.

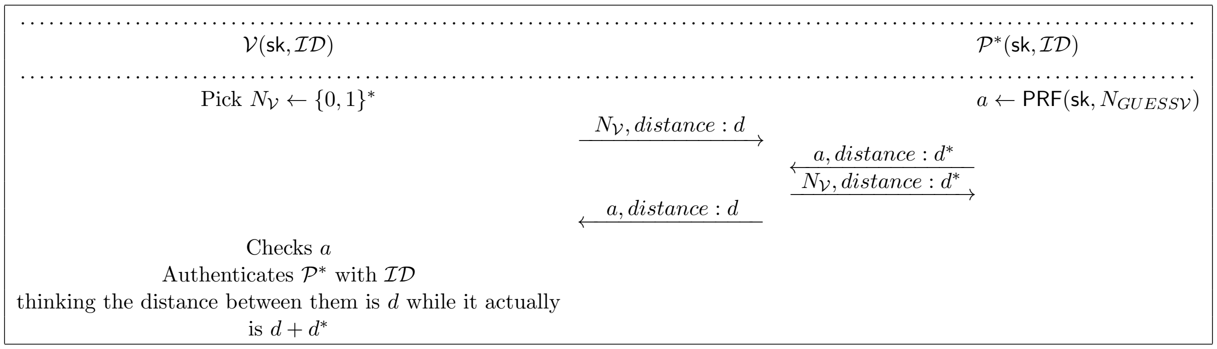

$\mathcal{V}(\mathsf{sk}, \mathcal{ID})$
$\mathcal{P}^*(\mathsf{sk}, \mathcal{ID})$
Pick $N_\mathcal{V} \leftarrow \{0,1\}^*$
$a \leftarrow \mathsf{PRF}(\mathsf{sk}, N_{GUESS\mathcal{V}})$
$N_\mathcal{V}, distance : d$
$a, distance : d^*$
$N_\mathcal{V}, distance : d^*$
$a, distance : d$
Checks a
Authenticates $\mathcal{P}^*$ with $\mathcal{ID}$
thinking the distance between them is d while it actually is $d + d^*$

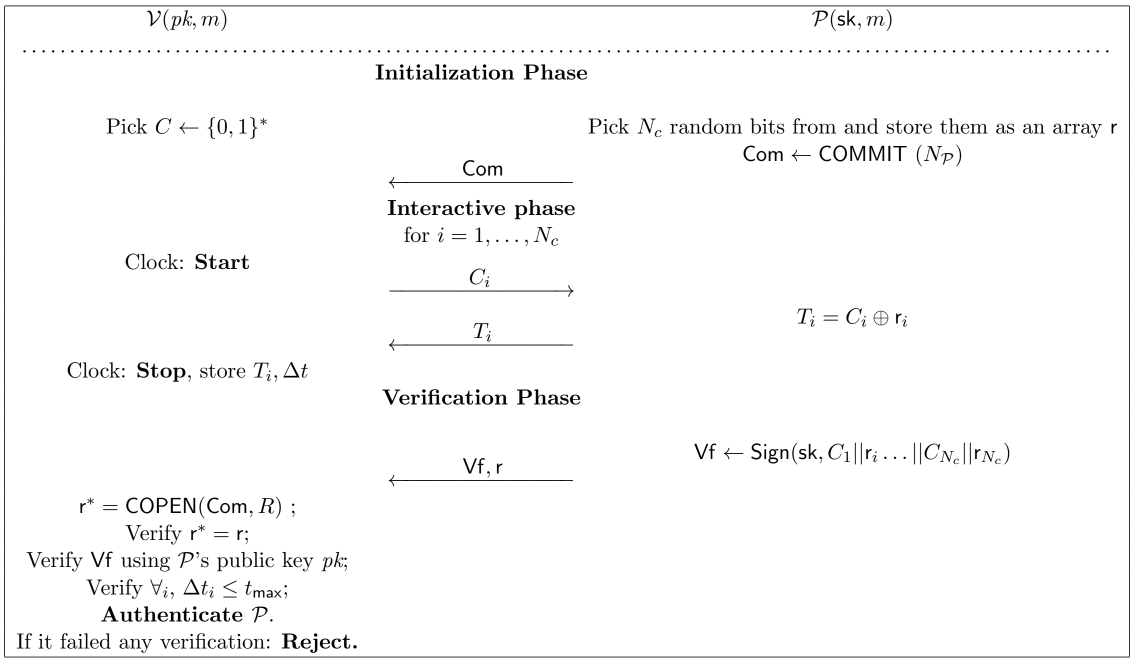
Figure 7: Distance fraud against the protocol from Figure 2.

2.2.2 Brands-Chaum protocol

Commitment schemes. The Brands and Chaum protocol uses commitment schemes as one of its primitives. A commitment scheme acts as an envelope with a special seal, which can be used to hide a given value. Typically, a user commits to some value, sending the commitment to another user; later, when the commitment is opened (using additional information provided by the first user), the envelope is unsealed and the same, unchanged value is revealed. Commitment schemes must have two properties: (1) hiding, i.e. the commitment leaks nothing about the value that was committed to; and (2) binding, i.e. the user who committed to a value cannot make the commitment open to a different value. A typical example is that of Pedersen commitments. allowing a user to commit to an integer x, by using a group $\mathbb{G}$ generated by an element g, with another element h such that $log_g(h)$ is unknown the commitment is $c = g^x h^y$, and it is statistically hiding (I.e. even unbounded adversaries learn nothing about x) and it is computationally binding under the discrete logarithm assumption in group $\mathbb{G}$.

The first proposed distance-bounding protocol was by Brands and Chaum [5]. It can be abstracted in two slow phases (initial processing and verification) and one fast phase (distance measuring challenges), a model that is used as a base for most DB protocols. We depict this protocol in 8. As we explained in the last subchapter, it can be abstracted in two slow phases (initial processing and verification) and one fast phase (distance measuring challenges). The prover is associated with a digital signatures pair (sk, pk). It also considers the pre-existence of a public array of bits called m, of size s. This array of bits is used to generate the responses for each challenge. The protocol works as follows:

- *Initialization/Setup.* $\mathcal{V}$ generates uniformly at random an array C of size $N_c < s$ bits and $\mathcal{P}$ chooses at random N_c bits from m to generate r. $\mathcal{P}$ commits the bits it has chosen for r as an array Com using a secure commitment scheme. A commitment scheme allows $\mathcal{P}$ to commit


$\mathcal{V}(pk, m)$ $\mathcal{P}(\mathsf{sk}, m)$

Initialization Phase

Pick $C \leftarrow \{0,1\}^*$

Pick N_c random bits from and store them as an array r

$\mathsf{Com} \leftarrow \mathsf{COMMIT}\ (N_{\mathcal{P}})$

$\xleftarrow{\mathsf{Com}}$

Interactive phase

for $i = 1, \ldots, N_c$

Clock: **Start**

$\xrightarrow{C_i}$

$T_i = C_i \oplus \mathsf{r}_i$

$\xleftarrow{T_i}$

Clock: **Stop**, store $T_i, \Delta t$

Verification Phase

$\mathsf{Vf} \leftarrow \mathsf{Sign}(\mathsf{sk}, C_1 || \mathsf{r}_i \ldots || C_{N_c} || \mathsf{r}_{N_c})$

$\xleftarrow{\mathsf{Vf}, \mathsf{r}}$

$\mathsf{r}^* = \mathsf{COPEN}(\mathsf{Com}, R)$;

Verify $\mathsf{r}^* = \mathsf{r}$;

Verify Vf using $\mathcal{P}$'s public key pk;

Verify $\forall_i$, $\Delta t_i \leq t_{\mathsf{max}}$;

Authenticate $\mathcal{P}$.

If it failed any verification: **Reject.**

Figure 8: The Brands-Chaum distance protocol

to a chosen array while only sending a commitment instead of the array. In addition, this keeps $N_{\mathcal{P}}$ hidden from $\mathcal{V}$ until the verification phase.

- *Interative phase.* This phase consists of N_c rounds. In each round $\mathcal{V}$ sends the bit i^{th} bit C_i, to $\mathcal{P}$ the latter receives C_i, and sends $T_i = C_i \oplus \mathsf{r}_i$.

- *Verification phase.* $\mathcal{P}$ sends the string r that allows $\mathcal{V}$ to open the commitment, then it concatenates the $2N_c$ bits C_i and r_i, signs the resulting message with its secret key and sends the resulting signature, Vf, to $\mathcal{V}$. With this information, $\mathcal{V}$ verifies the following:

  1. whether the bits $C_i \oplus T_i$ are indeed those committed in the initial processing phase;
  2. verifies Vf by using $\mathcal{P}$'s public key;
  3. that the proximity condition holds for $\mathcal{P}$ (i.e $\forall_i$, $\Delta t_i \leq t_{\mathsf{max}}$).

This protocol adds an distance-bounding proximity check to the security of a public key authentication scheme. The signature ensures authentication while the commitment scheme and randomness of the challenges ensure, mafia-and-distance-fraud resistance. More precisely, the hiding property of the commitment makes the responses random for mafia fraud attackers, while the binding property ensures distance-fraud resistance.

2.2.3 The Swiss-Knife protocol

The Swiss-Knife protocol was proposed by Kim, Avoine, Koeune, Standaert, and Pereira [24], and it addresses prover privacy issues, as well as mafia-fraud; terrorist-fraud; and distance-fraud attacks. In the initialization phase, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ first exchange nonces $N_{\mathcal{P}}$

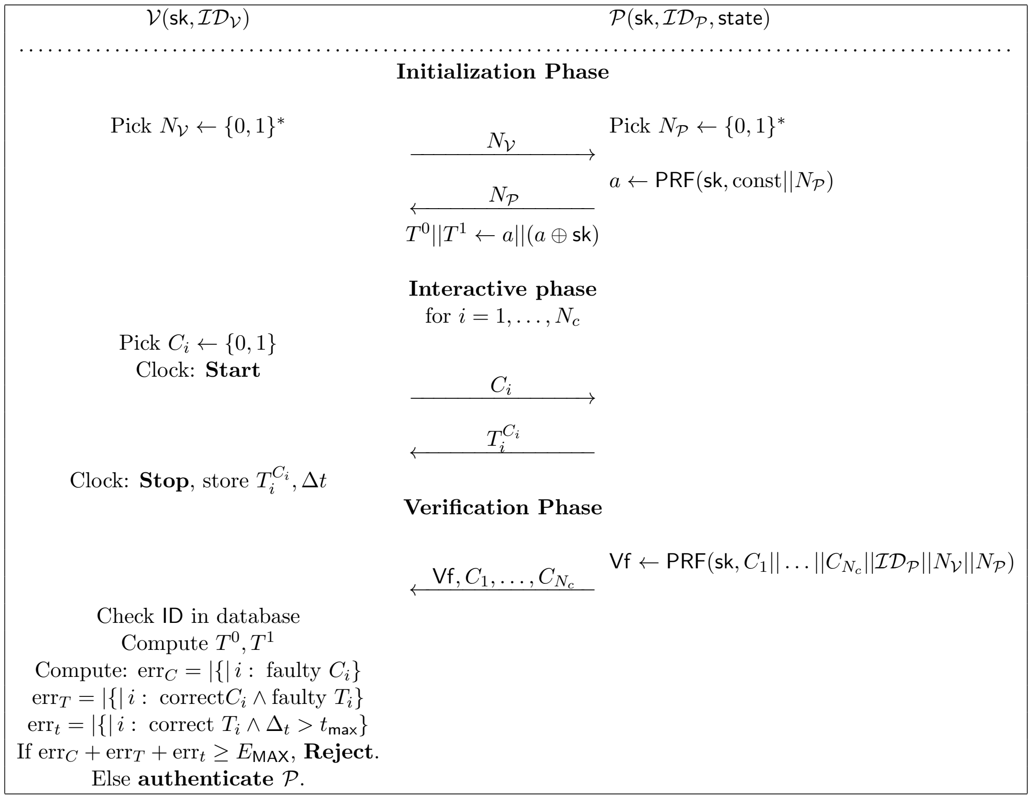

$\mathcal{V}(\mathsf{sk}, \mathcal{ID}_{\mathcal{V}})$
$\mathcal{P}(\mathsf{sk}, \mathcal{ID}_{\mathcal{P}}, \mathsf{state})$
Initialization Phase
Pick $N_{\mathcal{V}} \leftarrow \{0,1\}^*$
$N_{\mathcal{V}}$
Pick $N_{\mathcal{P}} \leftarrow \{0,1\}^*$
$a \leftarrow \mathsf{PRF}(\mathsf{sk}, \text{const}||N_{\mathcal{P}})$
$N_{\mathcal{P}}$
$T^0||T^1 \leftarrow a||(a \oplus \mathsf{sk})$
Interactive phase
for $i = 1, \ldots, N_c$
Pick $C_i \leftarrow \{0,1\}$
Clock: **Start**
C_i
$T_i^{C_i}$
Clock: **Stop**, store $T_i^{C_i}, \Delta t$
Verification Phase
$\mathsf{Vf} \leftarrow \mathsf{PRF}(\mathsf{sk}, C_1|| \ldots ||C_{N_c}||\mathcal{ID}_{\mathcal{P}}||N_{\mathcal{V}}||N_{\mathcal{P}})$
$\mathsf{Vf}, C_1, \ldots, C_{N_c}$
Check ID in database
Compute T^0, T^1
Compute: $\text{err}_C = |\{|i: \text{ faulty } C_i\}$
$\text{err}_T = |\{|i: \text{ correct} C_i \wedge \text{faulty } T_i\}$
$\text{err}_t = |\{|i: \text{ correct } T_i \wedge \Delta_t > t_{\mathsf{max}}\}$
If $\text{err}_C + \text{err}_T + \text{err}_t \geq E_{\mathsf{MAX}}$, **Reject**.
Else **authenticate** $\mathcal{P}$.

Figure 9: The Swiss-Knife protocol with unilateral authentication.

and $N_{\mathcal{V}}$ respectively. Afterwards $\mathcal{P}$ computes a bitstring a, output by running a pseudo-random function (PRF) keyed with the key sk shared by the prover and verifier, on input a system constant and the nonce $N_{\mathcal{P}}$. In practice the PRF is implemented as HMAC. The value a is then XORed with the same shared secret key sk, yielding the response vectors T^0 and T^1, each of length N_c. These two values are used during the fast phase, depending on the challenges sent at each round by the verifier. Note that the size of the secret key sk is assumed to be equal to the number of rapid-bit exchange rounds that the prover supports. Thus, the keys may be small.

Finally, in the verification phase, the prover authenticates by computing the PRF on all the received challenges, his identity, and the exchanged nonces (essentially "signing" the transcript). The verifier may also optionally authenticate by computing the PRF on the prover's nonce (preventing replays of the previous PRF value). The fact that the prover authenticates the session transcript effectively prevents both the recovery of the secret key in a mafia-fraud scenario, and it reduces the adversary's mafia-fraud success probability to about $(\frac{1}{2})$ per round. This protocol is depicted in Figure 9.

For the Swiss Knife scenario, each prover is associated with an identity ID(as opposed to similar PRF-based protocols), stored by the verifier together with the shared key sk belonging to that prover. The identity is never sent in clear, in order to protect user privacy. As opposed to other protocols in the literature, the Swiss-Knife scheme includes some fault/noise tolerance. Upon verification, the verifier allows for a total number of errors consisting of:

Reference/Attack	Mafia	Terrorist	Distance
Brands and Chaum [1] [5]	$(1/2)^{N_c}$	$\times$	$(1/2)^{N_c}$
Hancke and Kuhn [17]	$(3/4)^{N_c}$	$\times$	$\times$
Avoine and Tchamkerten [2] [2]	$(1/2) * (N_c + 2) * (1/2)^{N_c}$	$\times$	$(3/4)^{N_c}$
Reid and co-authors [3] [30]	$\times$	$\times$	$(3/4)^{(N_c - E_{\mathsf{MAX}})}$
Yang, Zhuang and Wong [32]	$(3/4)^{N_c}$	$\times$	$\times$
Swiss-Knife [24]	$(1/2)^{(N_c - E_{\mathsf{MAX}})}$	$\times$	$(3/4)^{(N_c - E_{\mathsf{MAX}})}$
SKI [3]	$2/3^{N_c}$	✓	$3/4^{N_c}$

Table 1: Comparison between different protocols based in [8] N_c *is the number of fast rounds,* E_{MAX} *is the error threshold and* $\times/\checkmark$ *represents that the protocol does not have/has a particular property.* [1] *The protocol uses expensive primitives.* [2] *The protocol requires exponential storage capacity.* [3] *The distance-fraud proof only holds for some implementations of the protocol.*

1. the number of incorrect challenges C_i that $\mathcal{P}$ receives;
2. the number of incorrect responses T_i received by $\mathcal{V}$;
3. the number of rounds in which the prover's response came late with respect to the threshold t_{max}.

The concept of fault/noise tolerance was explicitly introduced so as to account for the notorious unreliability of RF channels, which are prone to noise and collisions. Note that the value const in Figure 9 is a public system constant.

2.2.4 Comparison between protocols

Fischlin and Onete, presented in [8] a table comparing different distance bounding protocols and their protection against the frauds mentioned previously. A similar table was also presented by Boureanu and Vaudenay in [4]. We merged and summarized the results in the table below:

We chose to implement the Swiss-Knife Protocol, since it is well known and studied and follows the general layout of most other protocols in the literature. In particular we hope that our results can also be replicated for similar schemes. We also note that it seems to hard to achieve *provable* terrorist-fraud resistance even for schemes that intutively attain it. This also depends on the model used for that proof. Newer protocols e.g [12, 3] do have this property.

3 NFC communication on Android

NFC communication on Android smartphones is principally provided by two elements: the host (the Android operating system) and the controller, usually an NFC chip communicating internally with the host and externally with others chips. The core of NFC transmissions depends on the type of NFC controller in the phone. For older smartphones, the NFC controller was designed by NXP and its implementation uses the libnfc-nxp library of AOSP[1]. This library relies on two protocols: LLC (Logic Link Control) and HCI (Host Controller Interface). More recent controllers follow the NFC Controller Interface implementation (NCI) and use the library libnfc-nci [26]. The specific protocols used by each controller are highly relevant as they usually introduce significant delays and encode the transmissions in a way that decreases the useful information rate. Thus, in the context of distance bounding, such techniques render proximity-checking more difficult.

The LLC, HCI, and NCI protocols try to guarantee the success of an NFC communication and to define how both the controller and the host should behave in every situation; however this causes an overhead of data, lowering the useful information rate (number of actual information bits per number of sent/received bits) and so causing extra delays in the communication. These delays are normally unimportant in normal NFC communications, but essential if we want to accurately measure the round-trip time of a challenge-response round in distance bounding.

The NFC controller and host inside Android smartphones communicate via a serial port with a baud rate of 115200, using an 8-N-1 configuration. The latter is an abbreviation, denotes that for every eight information symbols that are transferred, there will be no parity symbol, but rather two extra symbols will be used, one for the end of the eight symbol sequence and one for the start, meaning that to send eight information symbols we have to send 10 symbols in total. This yields $11520 * 8$ information symbols per second, which is equivalent to a constant rate of 11520 bytes/s. As those bytes are transfered in a serial line, one at a time, transferring a byte of information takes approximately 0.0868 ms.

Additionally, serial ports deal with data in byte sized pieces. This is practical for dealing with ASCII characters [25], but inconvenient for many distance-bounding protocols in the literature, in which we normally want to send a bit rather than a byte.

LLC, HCI and NCI protocols are present used because the controller can have more than one host, for an example if the Android smartphone contains a smart-card (e.g a SIM card) acting as a secure element, the controller may need to direct messages only to the card or the phone. This was quite essential before Android KitKat (4.4) because host-card emulation was not yet enabled, leaving the secure element as the only way for the phone to emulate an NFC card/tag.

Two controllers can communicate via a peer-to-peer protocol or they can behave as reader and tag communicating via the ISO-DEP protocol [22, 27]. We chose to implement this last option in which the controller of the prover emulates an ISO/IEC 14443-4-compliant passive NFC-A tag. Both peer-to-peer and ISO-DEP transmissions would add a significant overhead to the transmission time. Additionally, the ISO/IEC 14443 standard defines a bit duration of $128/(fc * D)$ in which fc is the carrier frequency (13.56 Mhz in the case of NFC) and the allowed values for D are 1,2,4, and 8. The defined value of D in the ISO-DEP protocol is 1, leading to a bit duration of 9.439 microseconds (or $75,512$ microseconds per byte). This is highly relevant for our implementation as the bit-duration cannot be reduced during fast-phases (making the roun-trip times unusually high). This phone-to-phone communication is summarized in Figure 10.

3.1 The NFC stack on Android

On Android, any data transmitted by an application to the chip must pass through all the abstraction layers provided by the operating system, from the more abstract to the more concrete ones, as follows: (1) the application layer, (2) the Android NFC-Service layer, (3) the Android NFC Application layer, (4) the Libnfc NXP/NCI layer, (5) the Hardware Abstraction Layer (HAL), (6) the driver/serial port and finally (7) the NFC card layer. Originally, these different abstractions layers exist to facilitate the work of the developer and to enable an easy integration of different hardware within Android. However, these layers also add delays to the transmission time for both sending and receiving of data. In our experiments, this delay is at least 1.5 ms for each smartphone.

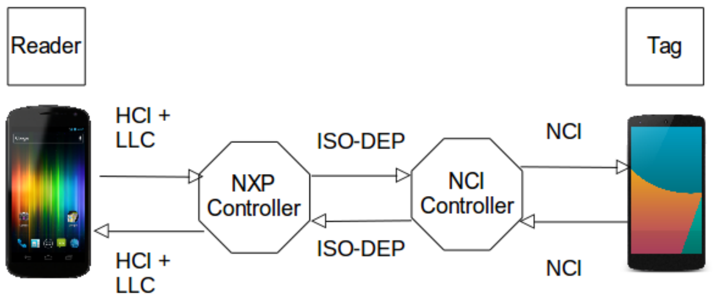


Figure 10: Phone-to-phone communication between an NFC prover and an NFC verifier.

Circumventing these layers requires root access in order to change the Android operating system or at least so as to install and use modified Android libraries. All the layers are presented in Figure 11.

Most abstract layer

Application
Android NFC-Service
Android NFC-Application
Libnfc NCI/NXP
HAL (Hardware Abstraction Layer)
Driver/Serial Port
NFC Card

Least abstract layer

Figure 11: Android abstraction stack for NFC communications.

3.2 NCI specification

The NCI specification follows the NCI protocol exclusively. The latter uses a credit-based data flow control mechanism for controlling the data sent from the host to the controller and may be invoked by the controller to eliminate buffer overflow conditions [26].

This credit-based data-flow controls the number of packets the host is allowed to send to the controller. In our case this number is always defined as 1. Each time the host wants to send a message it uses its (only) credit and has to wait for the controller to give it more credits. This creates an acknowledgment system, since the host has to wait for the controller to confirm it can send more messages.

In our implementation, we have to deal with two different types of messages, one for exchanging data and another for giving a credit to the host. Both messages use the same frame structure, the first three bytes compose a header, while the third one will be the payload size. The first three bits of the message indicates its type (000 for exchanging messages and 110 for the credit control), and the fourth bit informs if the message is split or not (in our case this will always be 0, indicating that the message is not split).

The last four bits have different meanings depending on the message. For instance, for the case of an information message it will represent the connection id (in our case it will be always 0, but it can change if more than one host is connected to the controller).For a control message, the last 4 bits represent it will represent a “GID” (Group Identifier). This value is also fixed for our domain, as the only control message we will be interested in is the credit control message, which is part of group “0” (NCI Core).

The second byte also differs in its meaning for each message. For instance, for the credit control message it shows the “OID” (Opcode identifier), a number that used together with the GID allows to identify which operation is being used. In our case it will always be 6 which means CORE_CONN_CREDITS_NTF. The second byte will always be 0 for exchanging message because this byte was reserved for future use (i.e., it is not currently in use in the last NCI specification).

In addition our experiments shown an additionnal byte that is not in the specification, which is added before the first byte as we could observe experimentally. There are only two allowed values for the byte. The code calls them HCIT_TYPE_NFC and HCIT_TYPE_EVENT. All of our messages are of the type HCIT_TYPE_NFC, which means that they start with 10.

The payload is also different. For the exchange message, it is the message we want to send/havereceived with/are receiving from the other controller. For the credit control, in our case, it will always be composed of three bytes: the number of entries (1), the connection identificator (0) and the number of credits that are given (1).

These frames can be seen in in Figures 12 and 13.

Credit Control Frame									
8	3	1	4	2	6	8	8	8	8
HCIT Type	MT	PBF	GID	RFU	OID	Payload Lenght	Entries	Connection Id	Number of Credits
1st byte	2nd byte			3rd byte		4th byte	5th byte	6th byte	7th byte

Figure 12: Credit control frame, most significant bit to the left.

Data Exchange Frame - NCI						
8	3	1	4	8	8	8+
HCIT Type	MT	PBF	Connection Id	RFU	Payload Lenght	Information
1st byte	2nd byte			3rd byte	4th byte	5th .. bytes

Figure 13: Data Exchange frame, most significant bit to the left.

To summarize, for this specification in the best case scenario we have a 7 (the credit frame) + 4 (message exchange) = 11 bytes of overhead due to the protocol when we are receiving data, and four bytes of overhead to send it. Considering the speed of transmission of 11520 information bytes/s, it will take $0,1736$ ms to transfer one byte from the host to the controller and back plus $1,302$ ms of overhead totalling $1,4756$ ms just to communicate between the host and controller in this specification.

3.3 NXP specification

In the NXP specification, the LLC protocol is used on the outer interface, and HCI for inner communication. In our context, we consider two types of messages: data exchange and Response Ready (RR). The latter signals that a message sent to the controller has already been treated, and no new messages were received after a specific delay. This message encumbers our transmissions because the information it conveys is not useful for the transmission, it adds an extra overhead. The LLC protocol adds 2 bytes at the beginning of the frame, one for the size of the rest of the frame and one for control. Two extra CRC bytes are appended at the end. This corresponds to a 7-byte overhead, which at the speed of 11520 bytes/s, yield a total round-trip delay of 1.3888 ms for the 8 bytes (7 overhead + 1 information) that are transmitted in the host-controller communications in the NXP specification.

The control byte (second bte of the message) depends on the value of the message LLC is helping to transmit. In the case of the data exchange, it will be composed of a one bit type (0 in this context), two three-bit message numbers N_r and N_s, and a one-bit P/F (Poll/Final) value. In our study, this last bit always has a value of 1. The Response Ready message will be composed of a two-bit type, a two-bit message, the one-bit of P/F value (still equal to 1), and a three-bit message number N_r.

The value N_s is used to number the messages and N_r is an acknowledgment that all messages $N_s < N_r - 1$ have already been received. These message numbers work modulo 8.

The HCI protocol will add three extra bytes:

- A control byte that specifies which communication pipe between host and controlled must be used and also if the message had to be split in more than one frame
- A type byte that specifies the message type. In our case, all our HCI messages are of type 0x10 -WRXCHGDATA.
- The last one depends if we are receiving or sending the message:
  - Messages sent will have a counter CTR, containing four bits reserved for future use and a four-bit value that defines the timeout as $(256 * 16/13,56Mhz) * 2^v$. This will vary from approximately 600 microsseconds to almost five seconds.
  - Messages received will show a response code that has four possible options. Normally this response code will be 00 that represents the message was well received.

The Response-Ready frames are always made of four bytes following the specification. The first byte will always have a value of 0x03, while the other three bytes may change. This frame is represented in Figure 14.

Response Ready (RR)		
Size	LLCP Control	CRC
1st byte	2nd byte	3rd and 4th bytes

Figure 14: Full view of the Receive Ready message.

The data exchange frames are a a bit different as they encapsulate both LLC and HCI protocols and the payload, see Figure 15.

Data Exchange Frame - NXP						
Size	LLCP Control	HCI Control	HCI Type	CTR/Receive Status	Information	CRC
1st byte	2nd byte	3rd byte	4th byte	5th byte	6th byte...	Last 2 bytes

Figure 15: Data Exchange frame, most significant bit to the left.

3.4 The ISO-DEP specification

The ISO-DEP protocol is responsible for defining the frames that are going to be sent wirelessly between the two controllers. This protocol frames are divided in three blocks called Start of Data (SoD), Payload, and End of Data (EoD). Each block is composed of two bytes.

The SoD block is composed of a Protocol Control Byte (PCB) and a Device Identification Number (DID). The EoD block is composed of a 16 bits CRC that is defined in [27]. These frames are described in Figure 16.

ISO-DEP Frame			
SoD		Information	EOD
PCB	DID		CRC
1st byte	2nd byte	3rd... bytes	Last 2 bytes

Figure 16: ISO-DEP frames

There are 3 types of messages that may be exchanged:

- I-blocks, which are used to exchange information between controllers. The important information is contained in the payload.
- R-blocks, which are used to convey positive or negative acknowledgments for the last block. These blocks never contained a payload.
- S-blocks, which are used to exchange control information. There are only two messages for S-Blocks: Waiting Time eXtension (WTX) that has 1 byte of payload and DESELECT that has no INF field.

This type of message is defined in the first two bits of the PCB (considering the most significant bits first). The specific values of each PCB can be found in [27].

In the best case scenario, we are going to have four bytes of overhead for transmitting information. Considering the byte duration of $75,512$ microsseconds, it will take $0,151$ ms to transfer one byte from the host to the controller and back plus $0,604$ ms of overhead totalling $0,755$ ms just to communicate between controllers.

3.4.1 Honest prover advantage over relay attacks due to ISO-DEP

It may seem unlikely, but these protocols can actually help us in deterring a relay attacker, as shown in the difference between Figures 10 and 17. Since the relay adversary has two devices (the Leech and the Ghost) which must both be ISO-DEP compliant at the interfaces communicating with the honest parties, the adversary has to use the ISO-DEP four times, twice for receiving messages (e.g. from the Prover to the Leech) and twice for sending them (e.g. from the Ghost to the verifier). This happens even if we assume a perfect, delay-free channel between the Leech and Ghost.

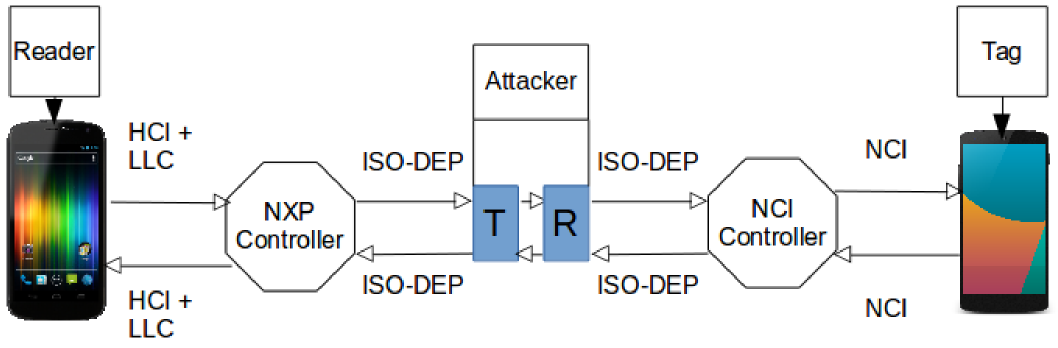


Figure 17: Communication example showing all the protocols envolved in a mafia fraud

Thus, a na´ive adversary will then add 0.755 ms per transmitted message just due the ISO-DEP protocol, while a sophisticated adversary may try to take advantage that the last two bytes can be calculated and only add add 0.453 ms (three bytes in each added communication). In terms of distance,the 0.453 ms correspond to a distance of more than 50 km at the speed of light.

3.5 Minimal transmission time

As all the different protocols (NCI,HCI,LLC, ISO-DEP) add a minimal time due to their overheads and transmission speed, we can indicate a minimal transmission time for each message exchange.

The minimal transmission time (MTT) consists of 1.4756 ms from the NCI specification (on the prover side) plus 1.3888 ms from the NXP communication (in the verifier side), which needs to be added to the 0.755 ms of the ISO-DEP communication, yielding $MTT=$ 3.6194. This added time is a crucial element in all our proofs-of-concept implementations, for the cases that the prover and verifier are hones (relay/mafia fraud). If the adversary uses a prover that does *not* use the NCI specification this can lower the MTT to 2.1438 ms, an enormous amount of time for distance bounding, where each added or removed millisecond is equivalent to a 150 km difference in the distance.

This is an important practical issue: on the one hand, honest prover-verifier communication always suffers from $MTT=$ 3.6 ms; on the other hand modifying the prover may reduce the MTT, giving distance-fraud and terrorist-fraud adversaries a chance to use relays/cheat the clock. Additionally, the 3.6194 ms must be taken into account when calculating t_{max}, as otherwise correctness would suffer.

4 Implementing distance bounding on Android

In our implementations, we use as a verifier a Samsung Galaxy Nexus smartphone following NXP specifications. The prover (*i.e.*, tag) is implemented on an LG Google Nexus 5 running Android Lollipop (5.1) and using NCI specifications for phone-to-chip communication. We implemented the Swiss-Knife protocol with 32 challenge/response fast rounds, for a variable tolerance threshold

E_{MAX} maximum number of errors it can accept. An error can be a faulty challenge or response, as well as a round that took longer than t_{max}. In Chapter 5, we discuss the reason for which the spread of the responses forces us to allow for a number of extended transmissions times, which correspond to rounds in which the time of flight is greater than the threshold t_{max}. This trade-off seems to be necessary since even honest prover-verifier sessions display important variations when measuring the transmission time.

Most of our experiments were run in 25 batches of 20 executions per batch. For some experiments, we only ran 10 batches as explained later. The batches themselves were separated by a random interval of time, since time-sensitive smartphone applications behave differently depending on what the phone is doing at the same time as well as on the load in the stack. For the computation of the pseudorandom function PRF, we relied on the implementations of HMAC_SHA1 or HMAC_SHA256 as explained below. We implemented three versions of the reader (*i.e.*, verifier) and tag (*i.e.*, prover). The first implementation runs the protocol with no modification to the Android operation system (*i.e.*, the phone is used in a "normal" mode). The second one executes the protocol by changing the Android operation system so as to bypass the stack layers in the roundtrip time estimation, thus we call it the "customized" implementation. Finally, the third version modifies the structure of the Swiss-Knife protocol by using challenges and responses of 8 bits. Hereafter, we use the following notation for the different implementations: $\mathsf{challengesize} : \mathsf{verifiermode} \leftrightarrow \mathsf{provermode}$, in which the prover and verifier modes are always either normal or custom, and the challenge size in bit varies from 1 bit to 8 bits (*i.e.*, 1 byte). Thus, $1 : \mathsf{normal} \leftrightarrow \mathsf{custom}$ denotes the implementation with 1-bit challenges (as specified by the Swiss-Knife protocol), with the prover is in normal mode and the verifier in custom mode.

For the custom mode, we modify the Android operating system by using the sources available on the AOSP (Android Open Source Project)[2]. For the reader protocol we used the branch called android-4.2.2_r1, while the tag protocol uses the branch called android-5.1.0_r3. For our normal mode implementations, the tag uses the stock version called *"LMY47I"* and the reader one is called *"JDQ39"*[3].

When the reader runs in the custom mode, it means that it uses a fully customized Android image. However, we tried to limit the customization for the tag; once the latter is used by regular users; More precisely, we rooted the prover phone and substituted the default library nfc_nci.bcm2079x.default.so with our own customized version it, thus effectively adding the Swiss-Knife implementation to the logic of the HAL library. To simulate a regular user environment for the prover, the smartphone implementing the tag has a number of applications installed on it and is always connected to a 3G/4G network.

The fast challenge/response roundtrip time is measured starting from the moment the sending function of the reader is activated until the time at which the size of the response packet is received (i.e the first byte). If this information was sent as soon as it is computed, then measuring only the time of arrival of the first part of the response should provide an accurate estimate. However, the driver of the phone acting as a reader actually buffers all the response data before sending it to the HAL, which leads to an overhead corresponding to the full processing and computation time for the response. An additional limit is due to the granularity of the reader's clock, which can only measure with a precision of up to 30.518 microseconds. In terms of distance, this is equivalent

[2]These sources can be downloaded by following the instructions at `https://source.android.com/source/downloading.html`

[3]These versions are available at `https://developers.google.com/android/nexus/images`

to 5 km, which makes this granularity incompatible with a precise accuracy for the DB protocol. However, in our setting this granularity is sufficient, since the prover and verifier take a processing time of the order of ms.

Note that if the experiments were run with other models of smartphones, the results obtained could vary slightly in terms of the measured time. For instance, changing the reader phone to a newer model would produce better results than the ones we have, while changing the tag could decrease the range of distance of the adversary for the relay attack if the new tag is faster in terms of processing or for its communication with the NFC chip. Our decision of implementing the DB protocol of the phone itself rather than on the SIM card makes it easier to install and deploy at large scale; however, implementing the computation on the SIM could directly provide more security and efficiency.

For each of our implementations, we consider viability in terms of the variation roundtrip-time measurements for honest executions. This variation enables us to choose "good" t_{max} and E_{MAX} bounds, allowing most honest executions to pass. What we typically do is to set t_{max} to the minimal execution time (MET) plus a bias (1 ms, 1.5 ms, etc); the bias depends on the spread of the data. We then choose E_{MAX} to ensure most executions pass.

4.1 1 bit sized challenges

In this subchapter we are going to discuss our different implementations that try to send the smallest quantity of information possible as our challenges and responses (1 bit).

4.1.1 Application layer implementation

For this first implementation, we created a pair of Android applications (one for the prover and the other for the verifier), which execute the Swiss-Knife protocol using the development tools found at `https://developer.android.com/sdk/index.html`. Here, our objective is to implement the protocol in the simplest way possible. Thus, the implementation exchanges the necessary bytes via NFC using the standard Android API for NFC communications. This application also stores the measurement obtained for each challenge/response round. The protocol was run with N_c= 32 rounds, and with $N_{\mathcal{V}}$, $N_{\mathcal{P}}$, sk, the constant const and the verification string Vf all having bits. The challenge string, as well as each of the two response strings, is also of the same size. For our PRF, we rely on the HMAC_SHA1 implementation from the package $\mathsf{javax.crypto}$ that outputs a string of 160 bits; we truncated the output to 32 bits. The values of sk and the constant const are generated at random and saved on the phone.

The results of the measurements are summarized in Table 4.1.2. A first important observation is that the variation in the measured times is important, ranging from 9.34 to 84.44 ms. In contrast, the standard deviation of the data was 5.36 ms, which is more than half our minimum time. In practice, if we want to prevent to prevent false negatives (which correspond to honest provers being considered as malicious), we would need a very high value of t_{max}, which allows practical relays to be mounted. Our median time for this batch was 18.77 ms, which means that at least half the measurements are more than twice our minimal value. In particular, even having an error margin E_{MAX} as high as $\frac{1}{2}N_c$= 16 rounds still results in the adversary having a 9 ms window for mounting his attack.

Implementation	Min time	Max time	STDEV[1]	Median	Mean	# Measurements
1 : normal $\leftrightarrow$ normal	9.34	84.44	5.36	18.77	17.94	500
1 : normal $\leftrightarrow$ custom	7.78	48.19	3.94	9.37	11.13	200
6 : normal $\leftrightarrow$ normal	9.37	63.38	4.28	12.12	14.14	200
7 : normal $\leftrightarrow$ normal	9.46	64.61	3.88	12.79	14.44	200
8 : normal $\leftrightarrow$ normal	9.40	53.01	3.06	13.28	13.95	500
1 : normal $\leftrightarrow$ normal (restart)	9.61	77.18	2.51	12.85	13.43	500
8 : normal $\leftrightarrow$ normal (restart)	9.52	121.67	3.55	12.91	13.96	500
1 : custom $\leftrightarrow$ custom	6.29	75.81	2.02	7.02	7.55	500
6 : custom $\leftrightarrow$ custom	6.35	63.08	1.64	7.60	7.53	200
7 : custom $\leftrightarrow$ custom	6.26	69.70	2.45	7.66	7.79	200
8 : custom $\leftrightarrow$ custom	6.26	113.19	2.29	7.66	7.65	500

Table 2: Implementation results and statistics(the measurement values are in milliseconds).[1] *Standard deviation.*

4.1.2 Lower layer implementation

For our second implementation, we considered the possibility of short-circuiting the abstraction layers of the smartphone. This customized implementation required to alter the Android operating system, which creates a trust issue. In particular, one must ensure that only the NFC transmissions are spoofed during the protocol. While this modification is not difficult to imagine in practice for the reader phone, regular users would also be required to install a modified Android library for the prover (which means that they also have a root access). Alternatively since Android is open-source, there is hope that our modification could be integrated in the default library. In this case, no root access would be required for the prover.

For this customized version, the size of the variables remains the same and we still use HMAC_SHA1, whose implementation was taken from the openssl library available on Android. The customization shortens the time measurement, as the clock runs from the HAL library layer rather than on the application layer. In addition, the measurements are also more uniform, with a much smaller standard deviation. In particular, the minimal time measured was reduced from 9.34 to 6.29 ms. Furthermore, while we still have outliers in the measurements (as reflected in the maximal measured time), the median and mean values are much closer to the minimum of 6.29 ms, i.e. the values are 7.02 ms for the median and 7.55 ms for the mean. Finally, the fact that the standard deviation is approximately 2 ms indicates that the measurements are concentrated closer to the minimal value. In this scenario, the adversary would have around 1 ms to mount the attack if the error threshold E_{MAX} is taken to be $\frac{1}{2}N_c$= 16 rounds (i.e. setting $t_{\mathsf{max}} = 6.26 + 1$ ms and E_{MAX}= 16 would allow 57.8% of our executions to pass). This version still uses the application layer to inform the user of the result, sending the latter to the application as a received NFC message.

4.1.3 Mixed implementation

: As both types of implementations are available, we also experimented by running the protocol between a custom-mode reader and a normal-mode tag using a standard Android (1 : normal $\leftrightarrow$ custom). These results are also shown in Table 4.1.2. As expected the minimum value is almost the average of the 1 : normal $\leftrightarrow$ normal and 1 : custom $\leftrightarrow$ custom modes (7.78 ms). However, the standard deviation remains very large as the one obtained in the 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ case. This shows that most of the variation in the measured time is actually caused by the prover's response times, and not by the manner in which the reader measures the elapsed time.

4.2 1 byte sized challenges

Now we analyze what happens when we change our challenge to better use the space available to our challenges and responses (1 byte) as seen in /refprelim

4.2.1 Lower layer implementation

The Swiss-Knife protocol was designed (as many other DB protocols in the literature) for challenges and responses of 1 bit each. So for our 8 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$ implementation, we modified the schema to carry challenges and responses using 8 bits instead of only one, essentially "batching" 8 fast rounds in 1. In particular, the manner of choosing the responses for each bit of the one byte challenge remains unchanged, we also run the protocol with the same number ($N_c = 32$) of fast rounds. We did this modification based on the observation that the encoding of the Swiss-Knife protocol in the ISO14443-4 standard/ISO-DEP protocol requires that the two parties exchange bytes instead of bits anyway. Thus, we wanted to leverage on this particularity to improve the security of the DB protocol.

Note that for 32 this new Swiss-Knife variant should have a mafia fraud resistance of (loosely) $2^{-256-E_{\mathsf{MAX}}} + \epsilon_{\mathsf{PRF}}$, in which the last term stands for the security of the used pseudo-random function[4]. In particular, we need to ensure that both the output size of the pseudo-random function and the size of the secret key sk are at least of 256 bits.

For the implementation, we made the following changes. We instantiate PRF as HMAC_SHA256, which returns 32 bytes. The variables const, sk a, T^0, T^1, Vf, C, $N_{\mathcal{V}}$, and $N_{\mathcal{P}}$ are now 32 bytes long. As the challenges are 8 bits long, $\mathcal{P}$ has to compute the 8 different responses (choosing from T^0 or T^1) before concatenating them into a byte before the sending. To avoid increasing the processing time, the responses are precomputed during the initialization phase so that during the fast challenge-response rounds, $\mathcal{P}$ only has to do a lookup for the precomputed result.

This may not be an optimal choice for a smartcard implementation due to the required storage; however in the case of the smartphone this presents no difficulty. We observe that the minimal time and standard deviation observed this time, see Table 4.1.2, are almost the same as those of 1 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$ implementation, while being significantly more secure. This open the door to the possibility of using less rounds in the future, which could be useful as environments in which the user does not want to wait before his accession permissions can be checked.

4.2.2 Application layer implementation

Finally, we also implemented the 8 bits version in the application-only version, using the same principles as the 8 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$. To our surprise, this version had a better performance than the 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ variant. We are not sure of the causes of this difference but is very noticeable as its median is 6 ms smaller and the mean 3 ms smaller. (see Table 4.1.2).

We provide a more thorough analysis of this phenomenon in Chapter 5.7.

[4]The security of a (family of) pseudorandom functions is measured in terms of the indistinguishability from random of the output of the function; for a secure PRF, ϵ_{PRF} is negligible

4.3 Varying challenge/response length

To better understand the effect of using larger challenges and responses, we also tested the same template as the one for 8 bits for a number of other variants, changing each time only the size of the C array and of the precalculated responses. We vary challenge-response from 2 to 7 bits. Thus, the C array only had values with the quantity of bits desired we did for each byte $C_i = C \wedge 2^{N_b}$ in which N_b is the number of bits used for the version and $\wedge$ is the logical AND. For the precomputed responses, instead of calculating 256 possible combinations per round (due to the 8 choices from T^0 or T^1), we only have to compute the first 2^{N_b} per round.

5 Comparative analysis

In this chapter, we describe our adversarial models, and compare and analyze the results obtained for the implementations described in the previous chapter.

5.1 Adversary models

One way of evaluating our implementations is to assess their success with respect to relay attacks and to distance-fraud attemps, as described in Chapter 2 We describe essentially the two adversarial models: a "Ghost and Leech" scenario for relays, and a "Malicious prover" setting for distance fraud.

5.1.1 Ghost and Leeches

This adversary performs a relay attack against our system. We classify such adversaries in terms of two aspects: (1) the minimal time required by an honest protocol round (with the adversary playing a MiM role); and (2) the delay introduced by the relay. Thus "computationally perfect" adversaries encounter an ideal implementation, in which all honest-prover-honest-verifier fast rounds take the minimal possible time (MTT). By contrast "computationally realistic" adversaries run the attack in a scenario in which honest fast rounds take an amount of time consistent with our minimal experimental (MET) value. For our second criterion, "Sophisticated relay" attackers do not add any delay or they add delays that we cannot capture (i.e. in the dozens of microseconds) as the attack proposed in [15]; by contrast "Off-the-Shelf relay" adversaries use off-the-shelf resources in which the delays are in the dozens of milliseconds, e.g. attacks as [11] and [18].

For "Computationally perfect" adversaries, their probability of success depends on whether the relay takes less than $(MET - MTT) + R$ if their delay is lower than this bound they can attack it. For example, using the data from the 8 : `custom` $\leftrightarrow$ `custom` that we presented in table 4.1.2 this adversary would have $2.65 + R$ ms ($MET = 6.26$ and $MTT = 3.61$ as its maximum delay. If R is small enough (e.g. less than 10 ms), "Off-the-Shelf relay" adversaries would fail; however, "Sophisticated relay" attackers would be able to exploit the system with distances of at least 360 km.

"Computationally realistic" adversaries have a smaller probability of success because the delay they introduce must be inferior to R. In this case, "Off-the-Shelf" relays would fail if R is smaller than a dozen ms, but "Sophisticated" relays would work within distances depending on R.

5.1.2 Malicious provers

Malicious provers aim at conducting a distance-fraud attack against the system. This game the adversary can always win if the he is within $300 * (MET+R)/2$ km of the verifier. However its goal is to authenticate from outside this range thus it has to produce the response before receiving the challenge. Note that E_{MAX} treats errors and late responses alike. The malicious prover will have to send N_c guesses and be correct in $N_c - E_{\mathsf{MAX}}$ rounds, which happens with a loose probability of $2^{-N_b*(N_c-E_{\mathsf{MAX}})}$ of correctly guessing at least $N_c - E_{\mathsf{MAX}}$ where N_b is the number of bits of the exchanged challenges and responses of the protocol.

One way of decreasing the adversary's success probability is to change the protocol so that it does not accept any challenge and response errors. This is possible due to the use of the ISO-DEP protocol, which ensures the correctness of the data transmitted. This modification would greatly reduce the probability of the attack as it would only have $N_c - E_{\mathsf{MAX}}$ chances to guess instead of N_c.

5.2 Distribution of measured times

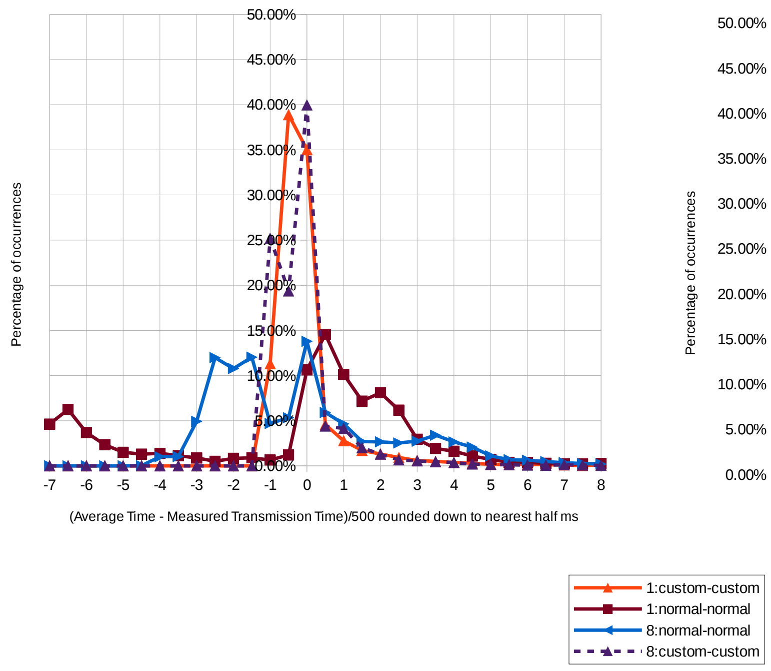


Figure 18: Distribution of measured times (right) and centered around the mean (left).

We also evaluate our results in terms of the obtained data. One first step is to investigate the distributions of the measured challenge/response times for our implementations. These distributions are depicted in Figure 18 (left), while the density of those measurements are depicted in the same figure, on the right. The left graph was obtained by taking the average measurement $\tilde{t}$ of each implementation before computing the difference between each value and $\tilde{t}$. The differences were rounded down to the smaller half a millisecond. In both fully customized implementations (for 1 and 8 bits), we can observe that the values cluster around the mean measured time, mostly being smaller than this value. By contrast, for the 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ implementation, there are multiple clusters in the values. This means that many measured values fall outside the neighborhood of the mean measurement, and thus *even* honest prover-verifier authentication can only succeed if the threshold t_{max} allows for a high variation in the challenge/response roundtrip time.

The average measurement is quite far from the minimal one for the 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ distribution, as can be seen in Figure 18 (right). In this figure, the density of occurrences is shown as a percentage of the total number of measurements for each measured value (in halves of milliseconds). Indeed, for the 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ implementation, the minimum observed time is (in absolute value) higher than that of the other two implementations. More importantly, most of the measured values of the 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ experiment are clustered far from the minimum measured time. In contrast, for the 1 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$ and 8 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$ implementations, the values are located close to the minimal value. For these two implementations, there is a subtle difference between the clusters. More precisely, the 8-bit custom version shows a true peak in the measurements exactly around the mean, with very few values clustered around the minimal value; by contrast the 1-bit version there are multiple values close to the observed minimum. This also demonstrates that for 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ the range R needed is higher than for the other versions as the times are not as well clustered near the minimal time.

Our analysis allows us to identify the size of the required bias R which is needed to avoid a high number of false negatives (i.e. rejected legitimate and honest provers) while still using a range that is small enough to counter relay attacks. We can already see that a range value R around 1 or 2 ms is needed by the custom implementations, while the 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ one requires R in the order of 10 ms. Our graphs show can already see that 8 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$ performed better than 1 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$. Our tests shown that a R between 3 and 5 would be optimal for 8 : $\mathsf{normal} \leftrightarrow \mathsf{normal}$.

5.3 Choosing optimal implementation parameters

The Swiss-Knife protocol allows us some flexibility in the choice of parameters. In particular, a prover authenticates successfully if and only if a number $(N_c - E_{\mathsf{MAX}})$ of challenge-response rounds are correct and within the threshold t_{max}. Thus, the higher E_{MAX}, the lower the global security, but the tighter one can set the bound t_{max} to avoid false negatives. We explore this relationship in more detail in the following. Figure 19 displays the percentage of honest-prover/honest-verifier custom executions passing for $E_{\mathsf{MAX}} = 16$ rounds (recall that the protocol runs for a total number of $N_c = 32$ rounds), as a function of the distance between t_{max} and the minimum observed measurement. In the 1 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$ implementation, almost 45% of the executions succeed if t_{max} is within 0.5 ms of the minimum measured time, as opposed to almost 40% for the 8 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$ case. Thus, an average prover must run the 8 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$ protocol 2.5 times on average before it can authenticate. This difference between implementations is much smaller if t_{max} is within 1.5 ms of the minimum time (86.8% honest executions pass for 1 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$ and 85.8% for 8 : $\mathsf{custom} \leftrightarrow \mathsf{custom}$). However, the farther t_{max} is from the minimum observed times the higher the risk of success probability for distance fraud, and relay attacks. Thus, there is a tradeoff between false negatives, the resistance to relay attacks, as well as the resistance to distance fraud. In particular, reducing R and E_{MAX} allow us to better protect against relays and malicious provers, but it also increases the rate of false negatives

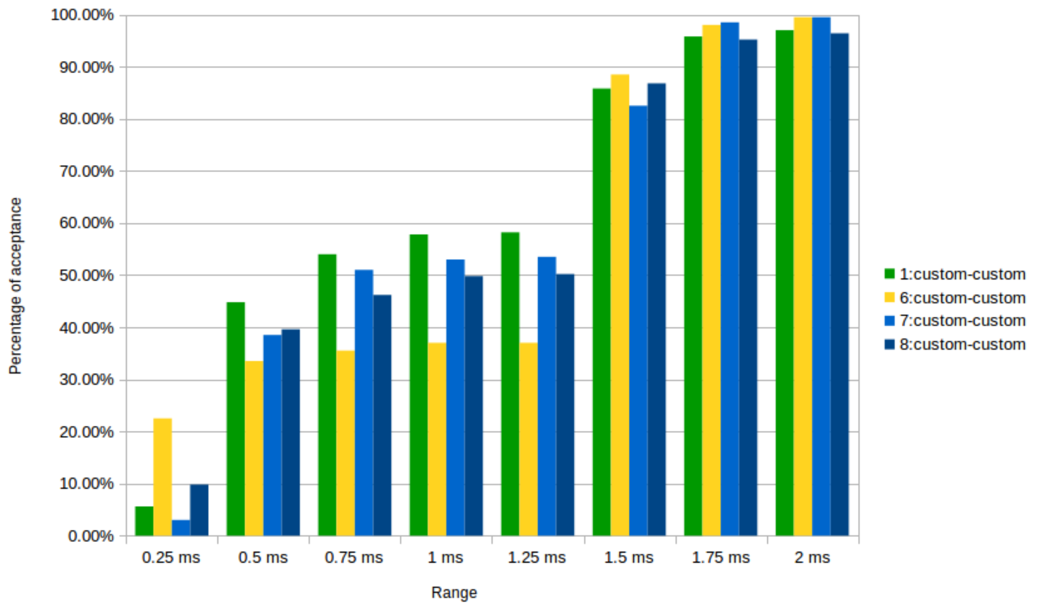


Figure 19: Percentage of successful executions for custom versions of the protocol when considering $E_{\mathsf{MAX}} = 16$ rounds.

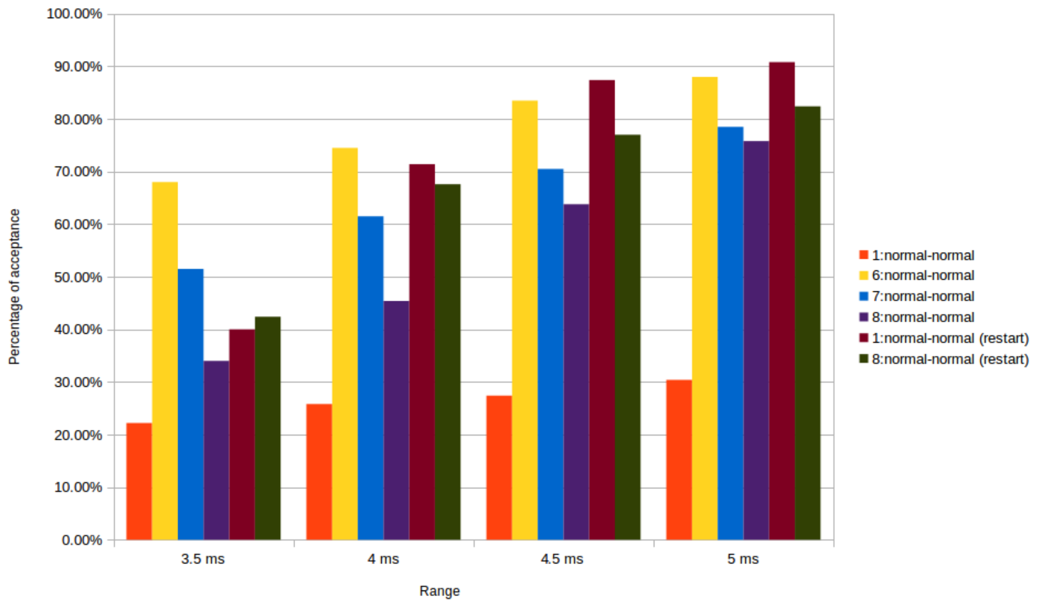


Figure 20: Percentage of successful executions for normal versions of the protocol when considering $E_{\mathsf{MAX}} = 16$ rounds.

Figure 20 shows the results obtained for different versions of the "normal" protocol. Clearly according to these results, the $1 : \mathsf{normal} \leftrightarrow \mathsf{normal}$ variant performed the worst, while we can see

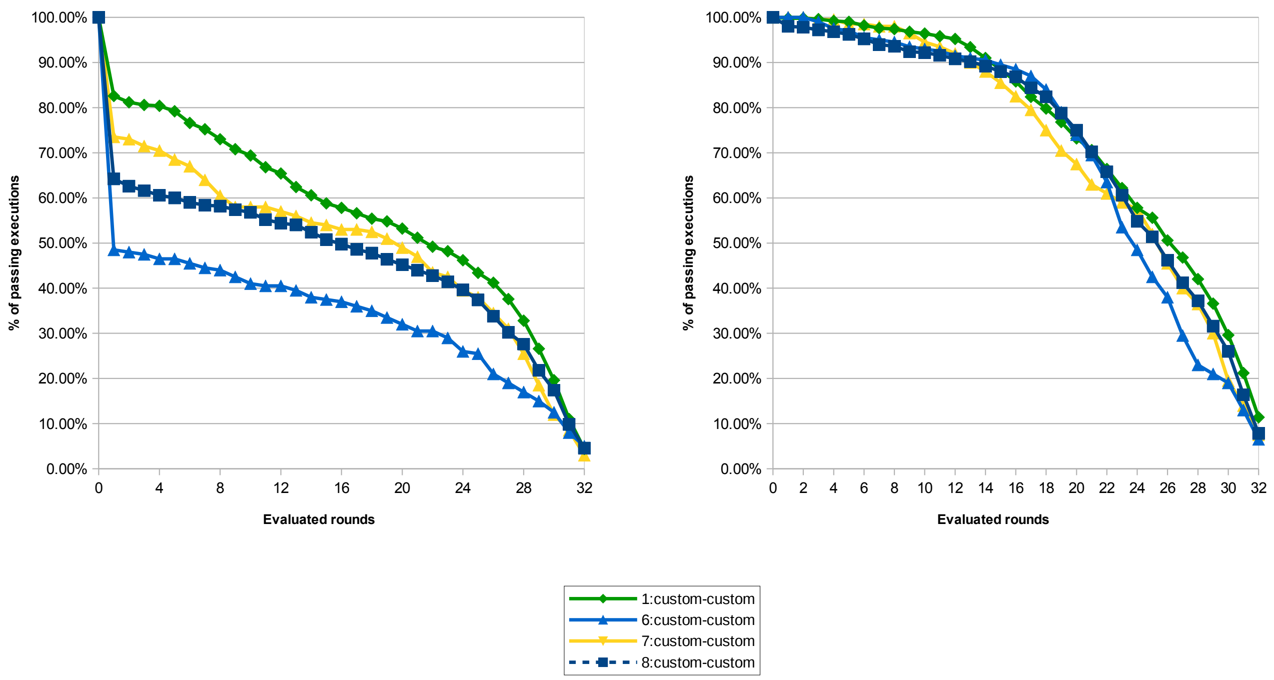


Figure 21: Percentage of successful executions against (N_c-E_{MAX}), for t_{max} within 1 ms (left) and 1.5 ms of the minimum (right).

that the 6-bit version is better for smaller values, followed by the 7-bit version but in the end it is overtook by the 8-bit version. This version had approximately 75% of acceptance with 5 ms as the R.

Afterwards, we investigated the impact that the error threshold E_{MAX} has on the percentage of false negatives. More precisely, Figure 21 shows the results obtained for custom implementations when t_{max} is chosen within 1 ms and 1.5 ms respectively from the measured minimum. In this figure, the x-axis represents the number of *evaluated* rounds, i.e. ($N_c$$-$$E_{\mathsf{MAX}}$). If we consider the bias R to be 1 ms, the 1 : custom $\leftrightarrow$ custom implementation always tolerates a lower value of E_{MAX} than the other versions. However, for t_{max} within 1.5 ms of the minimum measurement, the curves are almost identical. If we aim at reaching an acceptance rate of 90%, we should choose $E_{\mathsf{MAX}}= 18$ for 1 : custom $\leftrightarrow$ custom and $E_{\mathsf{MAX}}= 20$ for 8 : custom $\leftrightarrow$ custom. Overall the security level against a malicious prover of a 1-bit challenge execution with $E_{\mathsf{MAX}}= 18$ is only around 2^{-14}, whereas the security level of an 8-bit challenge execution with $E_{\mathsf{MAX}}= 20$ is approximately $2^{-12*8} = 2^{-96}$. These results are also shown in Figure 23. Thus in the long run, even with the higher E_{MAX} threshold, the 8 : custom $\leftrightarrow$ custom version tends to give better security.

To be able to conduct the same type of analysis for the normal implementations, we chose t_{max} of 4.5 ms and 5 ms. Figure 22 shows our results considering the E_{MAX}. For most of the implementations, we can use R=4.5 ms and E_{MAX}=20 to achieve an acceptance rate of 90%. This is the same value we used for 8 : custom $\leftrightarrow$ custom, meaning that for our parameters the 8 bits version of the normal protocol is going to have a security level against malicious provers equal to

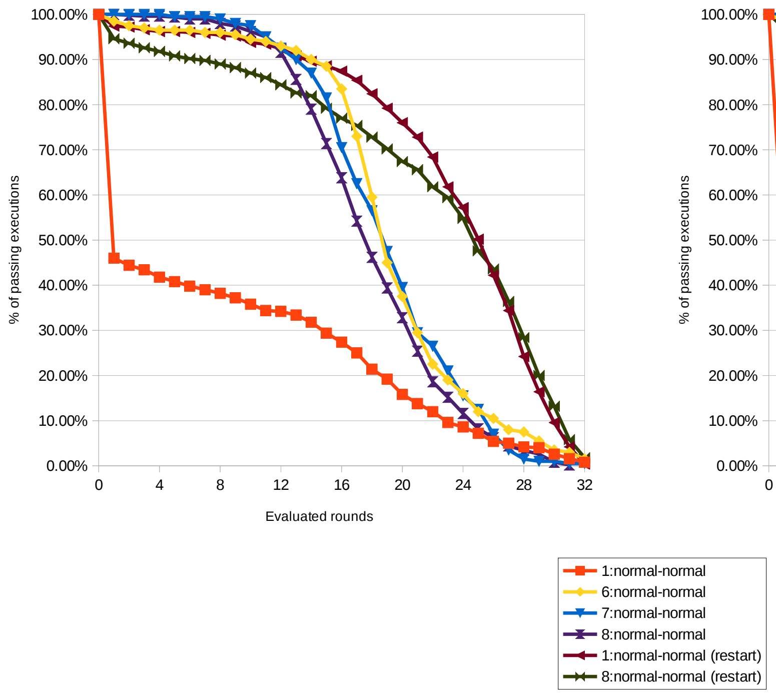


Figure 22: Percentage of passing executions against (N_c-E_{MAX}), for t_{max} within 4.5 ms (left) and 5 ms of the minimum (right).

Implementation	R	E_{MAX}	Security	Accept %	Time	Distance	Distance - MTT
1 : custom ↔ custom	1.5 ms	18	2^{-14}	91%	7.75 ms	1162.5 km	620 km
8 : custom ↔ custom	1.5 ms	20	2^{-96}	90.8%	7.75 ms	1162.5 km	620 km
8 : normal ↔ normal	4.5 ms	20	2^{-96}	92.8%	13.62 ms	2043 km	1501 km

Table 3: Synthesis of the chosen parameters

the 8 bits version of the custom protocol. This seems to indicate that, from the point of view of the security level, the 8 : custom ↔ custom and 8 : normal ↔ normal are both optimal. By contrast the 1 : custom ↔ custom is also highly interesting since it allow us to choose a smaller range R, thus effectively forcing relay adversaries to introduce smaller delays to be detected. A synthesis of the optimal parameters can be seen in Table 5.3.

5.4 Optimal vs. Experimental values

If we compare our experimental values to the optimal ones (MET vs. MTT), we can see that there is still an important gap and a possibility for progress. Our best MET is around 6.25 while the MTT is approximately 3.6 ms. This is due to several reasons, such as the fact that our implementations do not run on the least abstract layer. In particular, it takes approximately one additional millisecond for the data to arrive from the NFC card to the Android processor than the calculated theoretical time. Note that our MTT idealizes a prover that has a zero/negligible

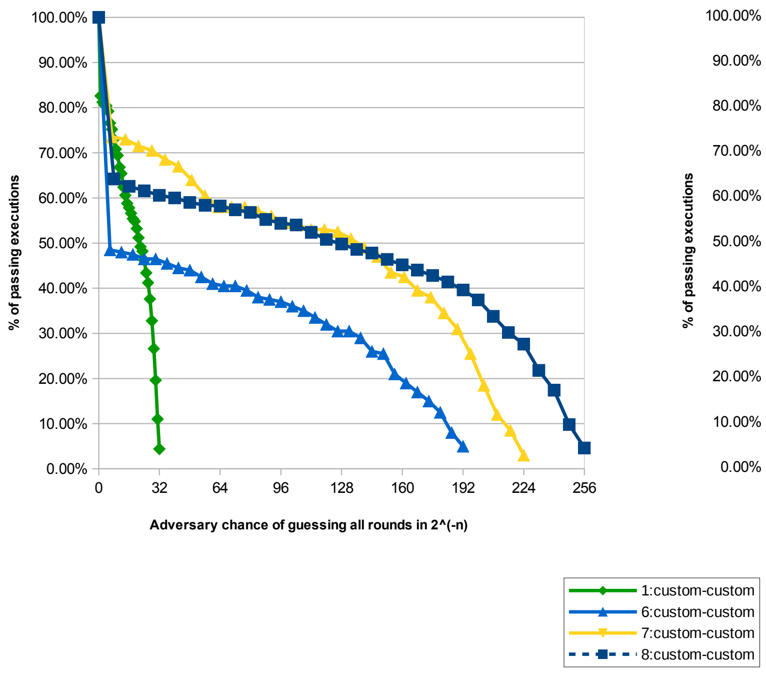


Figure 23: Percentage of passing executions against the security level (in $2^{(-n)}$), for t_{max} within 1 ms of the minimum (left) and within 1.5 ms of the minimum (right).

processing time

Reducing the gap between MET and MTT would for a better resistance to attacks, and approaching "Computationally perfect" and "Computationally realistic" attacks. It would also enable us to have a better measure of the real distance, also reducing the exploitable bias for a malicious prover.

Another experimental problem is the variation in the measured times. Minimizing this variation would allow us to reduce R while still maintaining a low false negative percentage. One solution to reducing the computational variation is a low level implementation possibly with dedicated hardware.

5.5 Chosen parameters vs. Adversaries

We analyze our chosen optimal parameters against the adversarial models presented in chapter 5.1. In terms of delays introduced by the relay attacker, note that the requirement of using the ISO-DEP protocol "adds" 0.755 ms to the relay time for relay attacks, while if the prover is adapted by the adversary to bypass some of the ISO-DEP protocol, this adds only 0.453 ms (see 3.4.1). We consider here that "sophisticated" relays are considered to be able to have the smaller added delay (0.453 ms) , while the "off-the-shelf" relays are going to have the higher delays (0.755 ms). In table 5.5 it is possible to see that our implementations defend well against attackers using "off-the-shelf" relays, while still guaranteeing a certain range against the sophisticated relays.

In fact, we can affirm that the strongest possible adversary that uses "computationally realistic"

Attack	Ghost and Leech				MP[1]
Computation	"Instantaneous"		"Experimental"		
Relay	Sophisticated	Off-the-Shelf	Sophisticated	Off-the-Shelf	
1 : custom $\leftrightarrow$ custom	4.15 ms 555 km	Impossible for relays slower than 3.4 ms	1.05 ms 158 km	Impossible for relays slower than 0.75 ms	2^{-14}
8 : custom $\leftrightarrow$ custom	4.15 ms 555 km	Impossible for relays slower than 3.4 ms	1.05 ms 158 km	Impossible for relays slower than 0.75 ms	2^{-96}
8 : normal $\leftrightarrow$ normal	7.15 ms 1005 km	Impossible for relays slower than 6.4 ms	4.05 ms 608 km	Impossible for relays slower than 3.75 ms	2^{-96}

Table 4: Comparison between implementations and its parameters and adversaries. All times, distances and probabilities are approximate. [1] *Malicious Prover*

values would be capable of increasing his distance by approximately 160 km while one that has "computationally perfect" values could extend that range to a maximum of 1005 km, depending on the modifications it makes. In other words, we can be pretty sure that the former would be in the same country/continent of the system (with some exceptions); however for the latter it is more extreme, one possible use is in the French Polynesia that given its isolation from the other groups of islands, we could possibly guarantee that the adversary is within one of the French Polynesia islands.

5.6 Methodology bias

During our tests, we observed that the execution of the protocol was impacted by different factors. While these factors influence our experiments, we do not exactly know their causes, nor the correlations between them. The main experimental factor is the status of the smartphone (mainly if it was recently booted or not).

In the process of analyzing why the 8 : normal $\leftrightarrow$ normal version performs better than the 1 : normal $\leftrightarrow$ normal, we detected a methodology bias caused by the state of the smartphone. More precisely, if the smartphone was recently reinitialized the results tended to have less variation, while taking on the average a little more time (a difference in the order of hundreds of microseconds). We believe that this is due to the smartphone having more available memory after the reboot, which reduces the variation (caused probably by the garbage collector), while adding some extra time due to a reboot procedure running in background. These procedures have less variation than the garbage collector but also have a higher minimal time or are more frequent.

Changing the methodology to restart the phone after each execution and running the protocol as soon as the phones finished rebooting, we could observe a decrease in the differences between the 8 : normal $\leftrightarrow$ normal and 1 : normal $\leftrightarrow$ normal as depicted in Table 4.1.2 and in Figure 24.

We leave as future work to investigate the reasons why that a recently rebooted phone performs better than a phone that has been running for a while, as that could lead us to a way to improve our implementations. The same kind of issues about the unpredictability of the garbage collector of Android have already been observed and discussed in other works such as [13] and [31] for example.

5.7 Understanding the differences between bytes vs bits in the application layer version

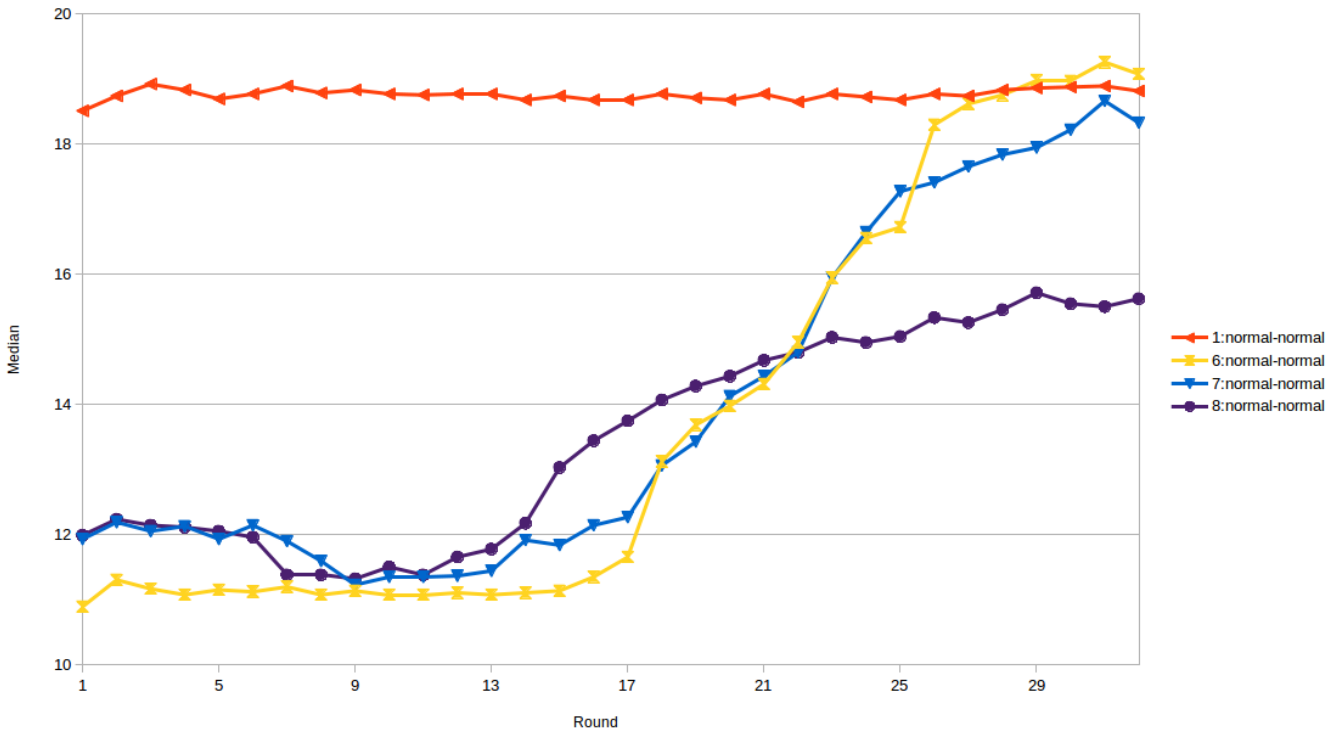


Figure 24: Median time for each round per implementation.

To analyze why 8 : normal $\leftrightarrow$ normal performs better than 1 : normal $\leftrightarrow$ normal we used the same methodology we used to program 8 : normal $\leftrightarrow$ normal to create other versions that used 7 and 6 bits respectively for the challenge/response rounds.

First, as shown in Figures 24 and 25 we observed that there is a correlation between the time needed to initialize the protocol (measured from the moment the $\mathcal{V}$ sends the $N_{\mathcal{P}}$ to the time it sends C_1) and the times measured. Figure 24, in which the median value measured for each round is plotted for each implementation variant, shows that protocols with a higher initialization time had better measurements in the starting rounds and worse in the finishing rounds. Further studying this relation between the initialization times and advantages in the earlier rounds could lead us to either an optimal value for the initialization phase or for the number of evaluated rounds.

Batching executions seems furthermore to have little effect: most batches perform uniformly (the first executions showing the same patterns as the last), this can be seen in Figure 26.

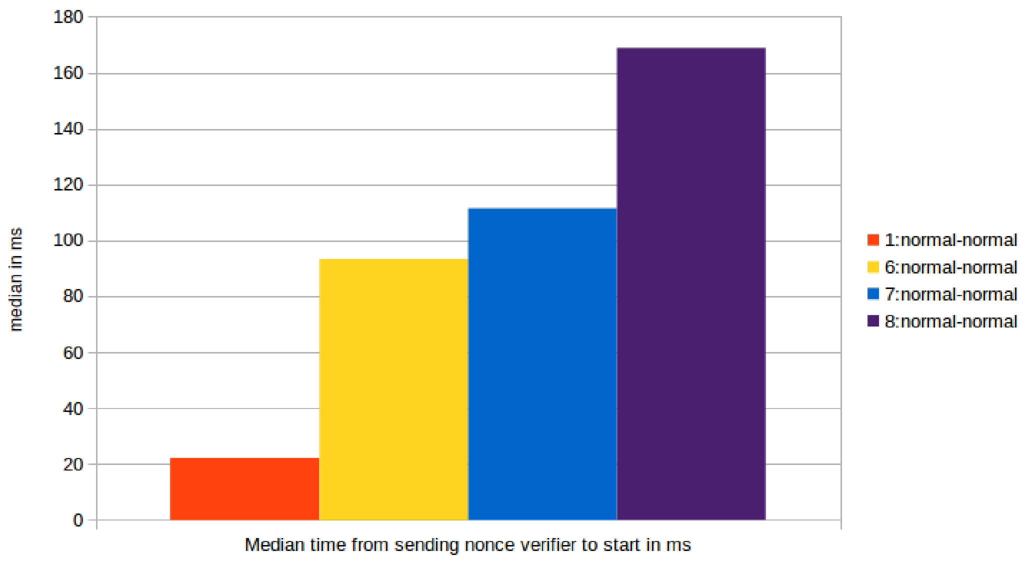


Figure 25: Median time between sending the nonce in the verification phase to sending the first C per implementation.

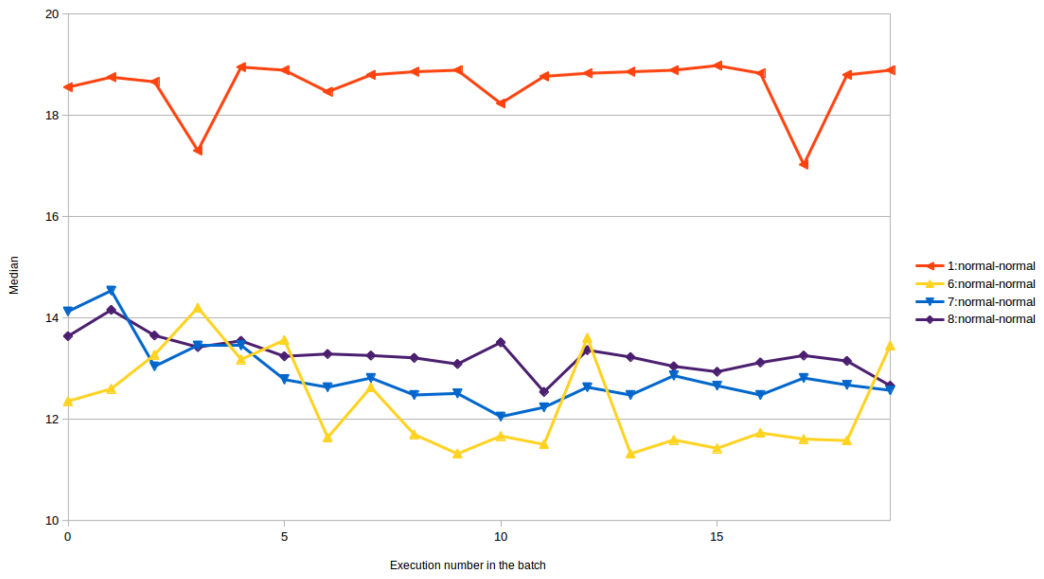


Figure 26: Median time of each execution in the batch (median time of all the first executions, the second, etc...)

5.8 Security Problems Caused by Android

Having the protocol run under the Android system means that we are also subject to all the possible fails and exploits that attack this type of system, e.g modifications in the software/hardware of the verifier or even malware.

One example of attack against the application-only versions is to create a clone of our application that accepts higher E_{MAX} or R, which is facilitated by the fact that our code is open-source and so anyone could download and create his own version. One way to prevent this attack would be to have a checksum of the "official" version, that could be verified prior to the installation. This cannot be added to the application as it would be trivial for the attacker to fake this check. The same principle of attack can be used against the "custom" versions.

Another type of attack would be to modify any of the abstraction layers that are between the HAL and the application layer, doing a Man-in-the-Middle attack inside the phone and controlling the messages that arrive at the application layer. This may be mitigated by an implementation that uses only the HAL layer.

6 Conclusions and future work

Overall, the results obtained by our experimental implementation of distance-bounding protocols on smartphones are highly encouraging. More precisely, we have demonstrated that operating the DB protocol in the HAL layer of the smartphone already provides sufficient constancy in the measurement of the challenge/response roundtrip times for distance-bounding to be effective as long as the adversary introduces a delay of more than 1.5ms during the relay attack. While customized relay mechanisms certainly take less than this [15], most off-the-shelf attacks seem to introduce delays of at least some tens of milliseconds [11, 18]. Our implementation allows us to avoid adversaries that use the currently available off-the-shelf attacks, and allows to detect even more sophisticated relays over large distances.

Another important conclusion of our work is that using 8-bit, rather than 1-bit challenges, increase the security of the protocol in the long run. In this case, one can leverage on the smartphone's memory to pre-compute the responses for the fast rounds, thus maintaining a much more stable processing time. This area of research has been very little explored in the design of DB protocols, at least in the context of smartphones.

The most important result is that introducing a threshold for the total number of allowed errors is crucial. Indeed the variations in the measurements, even for the best executions, usually conduct to a few rounds being outside the proximity threshold. More precisely, for t_{max} being within 1.5 ms from the minimal measured time of around 6.2ms (for both the 1-bit and the 8-bit versions), thresholds of $E_{\mathsf{MAX}}= 20$ and $E_{\mathsf{MAX}}= 18$ are optimal respectively for the 8-bit version and the 1-bit version.

We also conceptualized a minimal theoretical transmission time for our implementation and showed that it greatly impacts the implementation of distance-bounding protocols in Android smartphones. In the future, it is possible that this time can be reduced with the improvement of both the communication protocols and the technology that supports them.

Our three implementations only scratch the surface of the full potential of distance bounding over a smartphone. In particular, our implementations do not use the SIM card for the computations conducted. Investigating this direction would make the computation time more stable, while increasing security against side-channel attacks. We also leave a future work the investigation of the exact causes that lead to the time variations in the measurements. In particular, we observed that different factors may be involved from the time of the day (congestion of GSM communications can make the smartphone communicate more with the antennas), quantity of noise, experimental issues, etc.

In the long term if a better security guarantee could be achieved against a relay adversary, then the distance-bounding technology could have a fundamental impact in augmenting smartphone-payment possibilities or providing new functionality such as the construction of secure location proofs.

Bibliography

[1] Android: The Android open source project. `https://source.android.com/`

[2] Avoine, G., Tchamkerten, A.: An efficient distance bounding RFID authentication protocol: Balancing false-acceptance rate and memory requirement. In: Proceedings of ISC'09. LNCS, vol. 5735, pp. 250–261. Springer-Verlag, Berlin, Heidelberg (2009), `http://dx.doi.org/10.1007/978-3-642-04474-8_21`

[3] Boureanu, I., Mitrokotsa, A., Vaudenay, S.: Practical and provably secure distance-bounding. Journal of Computer Security 23(2), 229–257 (2015), `http://dx.doi.org/10.3233/JCS-140518`

[4] Boureanu, I., Vaudenay, S.: Challenges in distance bounding. IEEE Security & Privacy 13(1), 41–48 (2015), `http://dx.doi.org/10.1109/MSP.2015.2`

[5] Brands, S., Chaum, D.: Distance-bounding protocols. In: Proceedings of Eurocrypt'93. LNCS, vol. 765, pp. 344–359. Springer-Verlag, Secaucus, NJ, USA (1994), `http://dl.acm.org/citation.cfm?id=188307.188361`

[6] Carluccio, D., Kasper, T., Paar, C.: Implementation details of a multi purpose ISO 14443 rfidtool. In: Printed handout of RFIDSec 06 (2006)

[7] Desmedt, Y., Goutier, C., Bengio, S.: Special uses and abuses of the Fiat-Shamir passport protocol. In: Proceedings of CRYPTO'87. LNCS, vol. 293, pp. 21–39. Springer-Verlag, London, UK, UK (1988), `http://dl.acm.org/citation.cfm?id=646752.704723`

[8] Fischlin, M., Onete, C.: Subtle kinks in distance bounding: an analysis of prominent protocols. In: Proceedings of WiSec'13. pp. 195–206. ACM (2013)

[9] Fischlin, M., Onete, C.: Terrorism in distance bounding: Modeling terrorist-fraud resistance. In: Proceedings of the 11th International Conference on Applied Cryptography and Network Security. pp. 414–431. ACNS'13, Springer-Verlag, Berlin, Heidelberg (2013), `http://dx.doi.org/10.1007/978-3-642-38980-1_26`

[10] Francillon, A., Danev, B., Čapkun, S.: Relay attacks on passive keyless entry and start systems in modern cars. In: Proceedings of NDSS'11. pp. 7:1–7:16. USENIX Association (2011)

[11] Francis, L., Hancke, G., Mayes, K., Markantonakis, K.: Practical relay attack on contactless transactions by using NFC mobile phones. In: Proceedings of RFIDSec'10. pp. 35–49. USENIX Association (2010)

[12] Gambs, S., Killijian, M.O., Lauradoux, C., Onete, C., Roy, M., Traoré, M.: VSSDB: A verifiable secret-sharing and distance-bounding protocol. In: International Conference on

Cryptography and Information security (BalkanCryptSec'14). Istanbul, Turkey (Oct 2014), https://hal.inria.fr/hal-01090056

[13] Gerlitz, T., Kalkov, I., Schommer, J.F., Franke, D., Kowalewski, S.: Non-blocking garbage collection for real-time android. In: Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems. pp. 108–117. JTRES '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2512989.2512999

[14] Haataja, K., Toivanen, P.: Two practical man-in-the-middle attacks on bluetooth secure simple pairing and countermeasures. Transactions on Wireless Communications 9(1), 384–392 (2010)

[15] Hancke, G.: A practical relay attack on ISO 14443 proximity cards. http://www.rfidblog.org.uk/hancke-rfidrelay.pdf, accessed: 9th of january 2015

[16] Hancke, G.P.: Design of a secure distance-bounding channel for rfid. Journal of Network and Computer Applications 34(3), 877–887 (2011)

[17] Hancke, G., Kuhn, M.: An RFID distance bounding protocol. In: Proceedings of SECURECOMM'05. pp. 67–73. IEEE Computer Society, Washington, DC, USA (2005)

[18] Henzl, M., Hanáček, P., Kačic, M.: Preventing real-world relay attacks on contactless devices. In: Proceedings of IEEE ICCST'14. pp. 376–381. IEEE (2014)

[19] Hermans, J., Peeters, R., Onete, C.: Efficient, secure, private distance bounding without key updates. In: Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks. pp. 207–218. WiSec '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2462096.2462129

[20] Hlaváč, M., Tomáč, R.: A Note on the Relay Attacks on e-Passports (2007), http://eprint.iacr.org/2007/244.pdf

[21] IHS_Press: NFC-Enabled Cellphone Shipments to Soar Fourfold in Next Five Years. http://press.ihs.com/press-release/design-supply-chain/nfc-enabled-cellphone-shipments-soar-fourfold-next-five-years, accessed: 23th of july 2015

[22] ISO/IEC-14443: Identification cards - contactless integrated circuit(s) cards - proximity cards. Tech. rep., International Organization for Standardization (2008)

[23] Juels, A.: RFID security and privacy: A research survey. Selected Areas in Communications, IEEE Journal on 24(2), 381–394 (2006)

[24] Kim, C.H., Avoine, G., Koeune, F., Standaert, F.X., Pereira, O.: The swiss-knife rfid distance bounding protocol. In: Lee, P.J., Cheon, J.H. (eds.) Proceedings of ICISC'08. pp. 98–115. Springer-Verlag, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-00730-9_7

[25] Lawyer, D.S., Hankins, G.: The Linux Documentation Project (TLDP): Serial HOWTO. http://www.tldp.org/HOWTO/Serial-HOWTO.html, v2.27, February 2011. Accessed: 28th of july 2015,

[26] NFC Forum TM: NFC Controller Interface (NCI), version 1.1 edn. (2014)

[27] NFC Forum TM: NFC Digital Protocol, version 1.1 edn. (2014)

[28] Oren, Y., Wool, A.: Relay attacks on RFID-based electronic voting systems. Cryptology ePrint Archive, Report 2009/442 (2009), http://eprint.iacr.org/2009/422.pdf

[29] Ranganathan, A., Tippenhauer, N.O., Singelée, D., Škorić, B., Capkun, S.: Design and Implementation of a Terrorist Fraud Resilient Distance Bounding System. In: Foresti, S., Martinelli, F., Yung, M. (eds.) Proceedings of ESORICS'12. LNCS, vol. 7459, pp. 415–432. Springer-Verlag, Pisa,Italy (2012)

[30] Reid, J., Nieto, J.M.G., Tang, T., Senadji, B.: Detecting relay attacks with timing-based protocols. In: Proceedings of ASIACCS'07. pp. 204–213. ASIACCS '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1229285.1229314

[31] Yan, Y., Cosgrove, S., Anand, V., Kulkarni, A., Konduri, S.H., Ko, S.Y., Ziarek, L.: Real-time android with rtdroid. In: Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services. pp. 273–286. MobiSys '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2594368.2594381

[32] Yang, A., Zhuang, Y., Wong, D.S.: An efficient single-slow-phase mutually authenticated RFID distance bounding protocol with tag privacy. In: Proceedings of the 14th International Conference on Information and Communications Security. pp. 285–292. ICICS'12, Springer-Verlag, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-34129-8_25

Appendix

A Installing the application

For the moment, we made our code available as a repository at github (`https://github.com/cadurosar/swiss-knife-android`). Some parts of the code (essentially the modified Android parts) are protected under the Apache2 license [5]. The idea is to in the future create a proper distribution page with details about the project and links to results and documents. This manual supposes that your operational system is Linux, other versions may be available in the future.

After downloading the repository, either by using the GIT tool[6] or the direct download url `https://codeload.github.com/cadurosar/swiss-knife-android/zip/master` you can see that the main directory is divided in 2 folders, "Custom" and "Apps". Custom refers to the custom implementations and is separated in two sub-folders Reader and Tag, inside each sub-folder there is a file called install.sh that will download the Android source code and install the different drivers needed (it supposes that the phones you are going to install into were the same used in our implementation, if you want to use this code in models that differ from our implementation you are going to have to change the installation file), this process may take several hours. There's also 2 other folders called "Code" and "Others", the former contains all the code that we modified/added for this version; while the latter contains utility scripts (e.g. compilation and flashing for example), a manual file explaining how to choose which version, compile and install the modified operational system in your android smartphone, and also a file showing all the differences between the pure Android code for the version and our modifications.

For the applications and also the normal the folder Apps is divided in two sub-folders, "Tag" and "Reader", referent to which entity is implemented by the app. Then for each entity we have two sub-folders, one for which implementation (8-bit and 1-bit version). In each version folder we have a file named "app-debug.apk", the compiled version of our application, and a folder that contains an Android Studio project containing the code of our application.

[5] `http://www.apache.org/licenses/LICENSE-2.0`
[6] https://git-scm.com/