Here and There at Once, with my Mobile Phone!

Ioana Boureanu[1], David Gerault[2] and James Lewis[3]
[1]*University of Surrey, Surrey Centre for Cyber Security, U.K.*
[2]*Nanyang Technological University, Singapore*
[3]*Sky UK Ltd, U.K.*

Keywords: NFC Security, Proximity Checking, Proximity Attacks.

Abstract: Whilst proximity-checking mechanisms are on the rise, proximity-based attacks other than relaying have not been studied from a practical viewpoint, not even in academia. Are the simplest proximity-based attacks, namely distance frauds, a practical danger? Can an attacker make it look like they are here and there at the same time? In this paper, we first distinguish "credible" vs. impractical distance frauds, in a quantifiable, formal manner. Second, we implement two "credible" distance frauds on off-the-shelf NFC-enabled Android phones. We present an initial evaluation focused on their feasibility.

1 INTRODUCTION

Consider two devices, a *prover* and a *verifier*, executing a communication protocol. In relay attacks, a man-in-the-middle adversary forwards the messages from the far-away *prover* to the *verifier* and vice-versa, unbeknown to the two devices, with the aim to gain a privilege illicitly: e.g., authenticate as the prover. *Distance-bounding (DB)* protocols were introduced in (Brands and Chaum, 1993) as a countermeasure to relay attacks. Distance bounding counteracts relaying by having the verifier measure the round-trip time in its exchanges with the prover; if these take longer than a pre-established bound, then the verifier concludes that a relay from a distant prover may be occurring. But, there are other threats that distance bounding opens to. Maybe the simplest DB-specific attack (other than relaying) is called *distance fraud (DF)*. In simple words, *distance frauds are about a dishonest prover making it look like they are in the permitted range of the verifier when in fact they are far-away.*

With the threat of relaying increasing, protection via distance bounding is being incorporated in everyday products, such as contactless payments by Mastercard. Thuswise, malicious relaying is slowly getting less feasible in practice, and arguably the next step for fraudsters is to mount other distance-bounding-specific attacks. But are distance-bounding threats other than relaying relevant? For instance, as contactless payments may soon log the proximity-checking measurements (Chothia et al., 2019), we should be incentivised to prevent distance frauds; otherwise, dishonest, remote payees could get cryptographically-backed alibis of being by a payment terminal when they were not, or issue reimbursements claims to the card-issuing bank. Yet, unlike relaying, distance frauds have not been tried in practice, not even in academia, despite the fact that they are much studied theoretically (Avoine et al., 2018). To this end, we tackle a series of aspects as follows.

Contributions:

1. Within existing white-box corruption models introduced in (Avoine et al., 2011) and recently revisited by (Boureanu et al., 2018), we draw a **new, fine-grained distinction relevant to distance-fraud threats: *strong* white-box corruption vs *weak* white-box corruption**. Intuitively, " *weak* white-box distance fraud" involves the dishonest prover learning his secret key and using this knowledge to increase his advantage in mounting a distance fraud in a trivial manner. By contrast, a " *strong* white-box distance fraud" involves the dishonest prover further in intricate cryptographic attacks. Arguably, weak white-box distance fraud are much more plausible: it simply requires the prover to infer its key by statistical or side-channel attacks onto its own protocol executions, or read it off the device with existing tools. Yet, it takes an expert and potential collusion with other parties to mount strong white-box distance frauds, e.g., to use the inferred key in complicated cryptographic attacks a la "PRF programming" in (Boureanu et al., 2012).

2. In the "weak white-box corruption-model", we **im-**

Boureanu, I., Gerault, D. and Lewis, J.
Here and There at Once, with my Mobile Phone!.
DOI: 10.5220/0007953004780484
In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019)*, pages 478-484
ISBN: 978-989-758-378-0

plement two DF attacks onto the Swiss Knife protocol (Kim et al., 2008) executing on NFC-enabled Android phones. In line with the threat model, we use little resources besides the knowledge of the secret key and simple techniques. This demonstrates that someone with low-to-middle level of experience in Android and NFC programming can implement a distance fraud attack, if they can get to the secret material on their device. We report on our experimental results, on what we believe to be **the first DF attacks implemented in practice**.

2 PRELIMINARIES

2.1 The Swiss-Knife Protocol

In the Swiss-Knife protocol (Kim et al., 2008) (depicted in Fig. 1), the prover and the verifier share a secret key x. During the initialisation phase, they respectively generate random nonces N_P and N_V and exchange them. Furthermore, both of them generate a session key a as: $a := f_x(cte, N_P)$, where cte denotes a constant. Then, they compute the values Z^0 and Z^1 such that: $Z^0 := a$ $Z^1 := a \oplus x$. In each distance-bounding round, of which there are n, the verifier selects a random challenge c_i (where $i \in \{1 \dots n\}$) and the prover responds with r_i such that: if $c'_i = 0$ then $r_i := Z^0_i$, and if $c'_i = 1$ then $r_i := Z^1_i$, where c'_i is the challenge that the prover actually received in the i-th round. That is, c'_i will be c_i itself, if the transmission was correct, or c'_i will be the negation $\overline{c_i}$ of the sent value c_i, if c_i was perturbed by noise. The verifier measures the time each challenge-response exchange took and stores this in a variable δ_i, with $i \in \{1, \dots, n\}$. After the end of the distance-bounding phase, the prover transmits a message t_B such that $t_B := f_x(C, ID, N_P, N_V)$ where $C = c'_1, \dots, c'_n$.The verification phase is self-explained, where $t_{max}, err_c, err_r, err_t, err, T$ are all fixed tolerance-parameters of the protocol denoting the upper bounds on the duration of timed round, the acceptable number of noise-perturbed challenges, the acceptable number of erroneous response, the acceptable number of late responses, the acceptable number of total errors.

.

3 REFINEMENT OF WHITE-BOX CORRUPTIONS IN DISTANCE BOUNDING

Existing Corruption Modes in DB. In DB security, provers can be corrupted in a *white-box (WB) manner*, or in a *black-box (BB) manner*; see (Avoine et al., 2018; Boureanu et al., 2018) for details. We summarise these now.

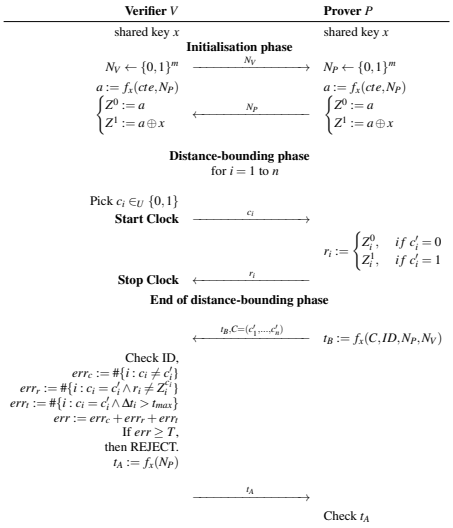

Figure 1: Swiss-Knife protocol (Kim et al., 2008).

Let P denote the algorithm of an honest prover, as specified by a distance-bounding protocol. Let P^* denote a malicious prover device/algorithm, mounting a distance-bounding attack. In the BB case, the dishonest prover P^* can use the algorithm P only as per the protocol specification to mount the attack. I.e., P^* sends inputs to and get outputs from the honest P as if P was a black-box. In the WB case, the dishonest prover P^* has read/write access to all the prover's components, including algorithms and cryptographic keys. As such, P^* can run any variation of P's specification or even use its secrets inside any algorithm to run the attack.

Our Refinement of Corruption Modes in DB. We identify 2 types of white-box attacks: i.e., Def. 3.1 and 3.2.

Definition 3.1. ***Strong White-Box (WB) Attacks.*** *Strong WB attacks in DB are formed via active actions by the attacker which use knowledge of the algorithm of the prover P and of P's secrets, to subvert cryptographic primitives.*

Here is an example of a strong WB attack. A prover P^* corrupted in a WB manner has access to his long-term secrets. Based on these secrets, P^* can –for instance– adaptively choose inputs for one of the protocol's pseudorandom function instances keyed on his secrets, to yield a special output. For everyone else other than P^* (who does not know his secrets), the pseudorandom function (PRF) still behaves pseudorandomly. That is, a P^* corrupted in a white-box manner can mount a "PRF programming attack" (Boureanu et al., 2012). Yet, breaking cryptographic mechanisms

in this way or, equivalently building the machinery to do so (e.g., relying on a PRF for which P's manufacturer/specifier allowed for a backdoor when the keys are known) are demanding settings. Synonymously, it is a strong threat-model.

Contrarily to this, the second class of WB attacks we define, we call *weak white-box (weak WB)*:

Definition 3.2. ***Weak White-Box Attacks.*** *Weak WB attacks in DB are formed via passive or active actions which may use knowledge of the algorithm of the prover P and of P's secrets, but do not subvert cryptographic primitives.*

In other words, a weak WB attacker uses his knowledge to simply to decide on inflicting forged messages but he obtains these messages by mechanisms that do not involve adversarial manipulation of the protocol's cryptography. That is, a weak WB adversary mounts his attack by non-cryptographic means: e.g., by guessing in an informed manner, by blocking messages at the right time, etc.

4 PRACTICAL DISTANCE-FRAUDS IN THE SWISS-KNIFE PROTOCOL

In Section 4.1, we show that, in the SwK protocol, polynomial attackers cannot mount strong white-box distance frauds, and that black-box distance frauds are not beneficial; in Section 4.2, we discuss weak white-box distance frauds. In all these attacks, we consider no communication noise, as our experiments in Section 5 show to be case.

4.1 Strong White Box & Black Box Distance Frauds in the SwK Protocol

If strong-WB DFs were mounted onto the SwK protocol, then they would entail the subversion of the computation $a := f_x(cte, N_P)$ and/or of the computation $t_B := f_x(C, ID, N_P, N_V)$. The first amounts to a PRF programming attack a la (Boureanu et al., 2012). That is, the dishonest P^* would adaptively choose N_P, on the bases of x, cte, such as to bias the output a (assuming the PRF f is such that its instance f_x is "programmable" in this way). The goal of the attacker is to lower $HW(Z^0 \oplus Z^1)$, or equivalently to maximise $\#\{i \in \{1,2,\ldots,n\} \,|\, (Z^0)_i = (Z^1)_i\} = \#\{i \in \{1,2,\ldots,n\} \,|\, a_i = a_i \oplus x_i\} = \#\{i \in \{1,2,\ldots,n\} \,|\, x_i = 0\}$, where by HW we mean Hamming weight. Hence, the attacker's advantage depends on $HW(x)$, which is independant of the output a from a programmed $f_x(cte, N_P)$. Similarly, programming f to adaptively choose t_B does not improve the attacker's advantage to a DF, since t_B is not involved in the timed exchanges. Hence, manipulating the output of the PRF does not help the adversary to mount a distance fraud against the SwK protocol.

In the case of black-box DFs, as the attacker cannot change the implementation of P, the attacker can simply not bias $HW(Z^0 \oplus Z^1)$ and as such he cannot increase his advantage.

4.2 Weak White Box Distance Frauds in the SwK Protocol

We showed that strong model did not help the adversary against the SwK protocol. We now describe two weak white-box distance frauds [1] next.

4.3 Early-reply in SwK: A Weak White Box DF with "Imperfect DB-timing"

In this attack, the dishonest, far-away prover P^* generally acts as follows. With his WB access to P, the attacker P^* gathers the indexes i of P's key x for which the corresponding key-bit is 0, i.e., P^* computes $S = \{i \in \{1,\ldots,n\} | x_i = 0\}$. For the rounds appearing in S, the answers are the same irrespective of value of the challenge and therefore the prover P^* can and will answer early. For the rounds not appearing in S, P^* will await V's challenges despite gathering DB-timing errors, and answer correctly as each arrives. A step-by-step description follows.

The Detailed Attack. First, being in weak-WB mode, P^* runs the initialisation phase correctly.

Second, in the DB phase, for rounds not in S, the prover behaves honestly. For the rounds in S, P^* answers early but cannot do it in a manner that does not consider the NFC/RFID timing issues. That is, if P^* sends his predetermined responses too early, then the verifier will halt due to NFC timing issues. To avoid this, the dishonest prover P^* acts as follows: P^* approximates the duration between the end of the initialisation phase $t_{end\text{-}of\text{-}init\text{-}phase}$ and the start of the DB phase over a number of executions. Further, depending on P^*'s distance d from V and the speed v_{comm} of the communication medium, P^* approximates the times t_k at which to send the responses Z^0_k, for $k \in S$. These

[1] Note that these attacks were known in the literature. We add two things: (1) we identify them as being of our type called "weak WB" attacks; (2) we detail their workings, the exact parameters they affect in a way that allows us to measure, in the next sections, their effectiveness as DF attacks.

times can in fact be refined on the bases of the time $t_{arrival_{c_{k-1}}}$ of arrival at P^* of the challenge c_{k-1} which precedes the early-send r_k. In other words, the time t_k to send preemptively a predetermined answer r_k in the round $k \in S$ is a function F:

$$t_k = \begin{cases} F(t_{end\text{-}of\text{-}init\text{-}phase}, d, v_{comm}), & \text{if } k = 1 \\ F(t_{end\text{-}of\text{-}init\text{-}phase}, d, v_{comm}, [t_{arrival_{c_{k-1}}}]), & \text{if } k > 1 \end{cases}$$

Note a square bracket around the time $t_{arrival_{c_{k-1}}}$: it denotes that this parameter is optional. It would make the attack online and more accurate, but it would require more real-time computations and measurements by P^* in the timed phased. Such precision-driven, real-time computations themselves can inflict delayed response-times, hindering the feasibility of the attack in practice; see Section 5 for details.

Third, after the timed phase, the dishonest prover sends the values c'_i equal the values c_i that it received, both inside C and inside t_B.

High-level Success Analysis. In this attack, err_c=0, since the dishonest prover sends the correct challenges in the verification phase. Also, err_r =0, since the dishonest prover sends the correct responses throughout. However, $err_t = n - \#\{S\} = HW(x)$, where HW is the Hamming weight. Hence, we dubbed this attack "imperfect DB-timing". So, in this case, err= $err_t \simeq HW(x)$.

4.4 Pre-ask & Early-reply in SwK: A Weak White-box DF with "Perfect DB-timing"

The Detailed Attack. First, let us look at the pre-attack phase, when the pre-computed response-table is built. For rounds $k \in S$, the prover has already predetermined answers, as in the previous strategy. For each round $j \notin S$ for which the attacker does not have a predetermined answer, this dishonest, far-away prover will now guess, at random, a challenge c^*_j, and pre-compute the answer for this anticipated challenge c^*_j. In this way, the dishonest prover now builds a full table containing one answer ready for each round.

Second, in the DB, P^* will send all his answers early. To send these answers in "good" time, the prover P^* acts as per the previous strategy and pre-computes the times t_i of sending each answer r_i, $i \in \{1,\ldots,n\}$ using the function F.

Third, in the verification phase, P^* declares c'_j to be equal to c^*_j for all the guessed challenges and to be equal to the really-sent c_j otherwise.

High-level Success Analysis. In this attack, err_c=$\#\{c^*_j \neq c_j \,|\, j \in \{1,\ldots,n\} \wedge x_j = 1\}$, since the dishonest guesses some challenges in the DB phase and reports the facts as such in verification phase. Also, err_r =0, since the dishonest prover sends the correct responses for challenges where c'_j= c_j (which is what err_r checks for). However, err_t = 0, as for the challenges that coincide on both side the timing is right (in fact, the timing is right throughout). Hence, we dubbed this attack as "perfect DB-timing". So, in this case, err= $err_c \simeq \frac{HW(x)}{2}$.

Alternative Version. Depending on the distance d between the dishonest prover and the verifier (i.e., if the prover is just superficially outside of the distance-bounding or it is considerably further) and on the tolerance of the verifier w.r.t. NFC timing issues (i.e., if the verifier allows fewer or more NFC time errors), a variant of this attack is as follows. Instead of computing the times t_i to send the answers early as per the above, the prover will wait for the first challenge c_1 to send back the response r_1, and –after a time δ_1– will send the rest of the response table. (The value δ_1 is close to the RTT for the distance d in the medium given.) In this variant of the attack, err_c=$\#\{c^*_j \neq c_j \,|\, j \in \{2,\ldots,n\} \wedge x_j = 1\}$. Also, err_r =0, as before. However, err_t = 1, as the prover awaits for the first challenge to arrive at it. So, in this case, err= $err_c \simeq (\frac{HW(x_2 \ldots x_n)}{2} + 1)$.

Theoretically, in this attack, the guessing of the challenges c_i for $i \notin S$ and the computation of the corresponding r_i can be done in real time. In practice, this takes too long hindering the attack; see Section 5.

5 DISTANCE FRAUD ON ANDROID PHONES

In this section, we describe the implementation of DF attacks on the Swiss-Knife protocol, similar to the ones presented in the previous section, using two NFC-enabled Android phones. The attacks' implementation is given in Section 5.1, but first we present some background and preliminaries.

Existent SwK Implementation. We base our DF implementations on an existing proof-of-concept implementation of the Swiss-Knife protocol on NFC-enabled, Android phones presented in (Gambs et al., 2016). The implemented Swiss-Knife protocol follows the ISO/IEC 14443 standard and allows two smartphones to communicate via the ISO-DEP protocol. One phone acts as a prover which emulates a ISO/IEC 14443-4-compliant NFC-A tag through host-card emulation, and the other phone is a verifier behaving as

a card reader. The implementation is split out into two parts, one for the reader and one for the tag. The resulting applications can be installed on smartphones running Android 4.4 (KitKat) or higher. The execution of the SwK protocol is initiated by the reader, and triggered when a card-emulating phone is in range.

Pre-attack Time-measurements. A dishonest prover would first run a series of tests to see how a protocol works and discover its potential weak points. This amounts to the practical computation of the measures stated theoretically in Section 4: the values t_k (i.e., the moments in time to send the answer k early) and the function F (i.e., used to compute these t_k, based on different parameters and measurements), etc. From this viewpoint, a dishonest prover would measure or ascertain: 1. How long one single challenge takes to arrive from the verifier and/or how long it takes for the response to go the verifier. 2. The difference in communication time for a challenge or response containing a 0 or a 1. 3. How long it takes to process a challenge and send back a response (possibly again varying these upon the specific values of the challenge); an instance of this was denoted as δ_1 in Section 4. 4. The total duration of the whole timed phase. 5. The time it takes from the end of the initialisation phase to the beginning of the fast/time phase; we denoted this as t_1 in Section 4. 6. The time it takes from the end of the fast/timed phase to the beginning of the verification phase; this is important in case the dishonest prover has to process some additional aspects after the fast phase so that only after he becomes ready to pass the verification phase. 7. All of the above can (in a first instance) be done from several distinct distances (still) within the usual bound, to get the time differentials as the prover would gets further from the verifier. Using the commonly-known transmission-speed on the channel, the attacker would then extrapolate (with some error) what the communication times would be over larger distances d outside the bound.

A dishonest prover will/should do variations of the measurements 1-7 above in repeated executions and average the results. In the weak WB corruption-model, such dishonest provers would undertake most of these measurements from their side, during correct executions (i.e., with no direct access to the verifier's side or measurements, no intricate hardware attacks, etc).

For the weak WB DFs' implementations presented next, we did not carry out all the measurements presented in the "real-world DF considerations" above. Notably, we did evaluate points 1, 5 and 6 above and a few other aspects (detailed further in Sec. 5.1). To this end, one important aspect for the DF attacks described in Section 3 is the time between two consecutive responses by the prover. Concretely, we were interested in the measurements shown in Fig. 2.

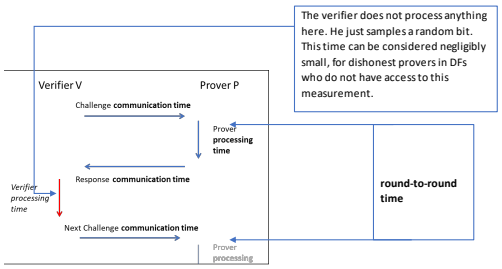

Figure 2: Communication & Computation Times in DB.

Using the notations in Fig. 2, we have that "`two-way communication time`" = ("`round to round time`" - "`prover processing time`" - "`verifier processing time`"). However, again, a weak WB dishonest prover does not have access the verifier processing time, but can approximate this to zero. So, "`response communication time` $\simeq$ "`two-way communication time`$''/2$. This gives the prover an estimate for the time-travel of a response from itself to the verifier.

Based on the SwK implementation, with the prover and verifier within the distance-bound (at 2cm apart), over 100 iterations, we measured the above from the perspective of a (dishonest) prover, i.e., keeping time logs only on the prover's side. The results of these are given in Table 3. By performing the same measurements from the verifier's viewpoint, we observed that the prover's estimates only differ by cca. 0.5ms, i.e., that a dishonest prover can reliably approximate these times, without having access to information on the time-measurements on the verifier's side.

Table 1: Relevant Timing Measurements Executed from the Prover's Side (avg. over 100 iterations).

round-to-round time	12.56746 ms
average round processing time	0.324687 ms
two-way communication time	12.24277 ms
response communication time	6.121385 ms

5.1 Implementations of DF Attacks

In all our experiments, we used two off-the-shelf Google Nexus 5 smartphones. Our implementation and tests are available at: people.itcarlson.com/ioana/df.zip.

5.1.1. The "Imperfect DB-timing" Attack

This is the DF in Section 3.1.1. Based on x, we construct a table of precomputed answers r_k for each $k \in S$.

Then, for any index $j \in \{1,\ldots,n\}$, the provers will send answers $k \geq j$ early if $k \in S$, until $k \notin S$. Otherwise, if still $k \notin S$, then P^* waits for the challenge by the verifier. Note that, depending on the fixed value of the key x, the prover P^* may send not one but several answers *at once* in advance. This can cause NFC timing issues: the larger the number of consecutive early responses, the larger the chance that the verifier may receive a (NFC) response before the (NFC) challenge, or simply the chance to produce NFC framing/timing issue. To avoid this, we did implement a small wait in between two such consecutive early sends. This wait was in line with pre-attack measurement 1 and the "response communication time" reported in Table 3, i.e., 6ms (which we later fine-tuned further as per the below). Also, this amounts to approximating the times t_k for the answers r_k in Section 3.1.1.

To describe the rest of the implementation (i.e., the verification phase), we first need to develop on an aspect linked to pre-attack measurement 6, as well as the APDU-based communications[2] and their treatment in NFC under Android. Namely, initially, we implemented the (rest of) attack exactly as per Section 3.1.1: i.e., in the verification phase, the dishonest prover would send C as it received it. However, despite the fact that we fine-tuned the sending of early responses with waiting times, we noticed that the attack resulted in $err_C \neq 0$. Upon inspection, it transpired that sending some answers early results in the prover's list of receiver challenges be no longer in order. This made us revisit the waiting time between consecutive early sends: we further fine-tuned the waiting time in between two consecutive early sendings at the 5ms mark. Also, helped by the fact that the indexes of the answers sent early are always the same (i.e., indexes $k \in S$, predetermined by x_k=0), we determined how the challenges' list was unordered. Moreover, our pre-attack measurement 6 turned useful. The standard implementation does allow enough time for us to have our dishonest P^* to reorder the list of challenges it receives and thus send out the verification-phase ector C such that err_C=0, as per Section 3.1.1.

Similarly, as per Section 3.1.1, we did try a real-time adaptation of the time t_k to send out an early r_k based on the time $t_{arrival_{c_{k-1}}}$ arrival of the previous challenge c_{k-1}, but –due to the way in which the Android APDU service "stacks" challenges– this proved ineffective.

[2]APDU (Application Protocol Data Unit) is the "atomic" message sent/received between a reader and a card as per ISO 7816.

5.1.2. The "Perfect DB-timing" Attack

This is the DF in Section 3.1.2. Before the DB phase, dishonest prover P^* builds a table with answers r_j for the rounds $j \in S$, as well as for a fixed, guessed value for the challenges c_i for $i > 1$ and $i \notin S$. Then, there are two possible cases: (1) P^* calculates an approximation the time $t_{end\text{-}of\text{-}init\text{-}phase}$ from the end of the initialisation phase to the start of the DB phase, and based on this he starts sending his answers early, or (2) P^* waits for the first challenge to arrive and sends all his answers thereafter. The latter is the so-called "alternative" of the second attack in Section 3.1.2. We implemented both cases, but our experiments report on the first.

W.r.t. sending the answers early, we added a wait of 5ms before pre-emption; again, this is akin to computing the times t_k to emit early answers r_k in Section 3.1.1. An interesting note is the following: if we send the pre-computed response table at once, without the wait in between answers, then the APDU service in Android works in our favour[3]. I.e., for a prover that is not too distant from the verifier (as to not cause NFC timeouts), the NFC responses are "stacked" by the APDU service and treated by the verifier in the right order.

5.2 Experiments

We aimed to see the limits of the relatively simple DF attacks in Section 3, by using **just standard** NFC implementations, i.e., no amplification by special antennas, no modification of the NFC stack, etc. So, in this case, the first thing that matters is the distance at which the two phones can be from each other and still communicate via NFC; this is determined by the power of NFC antenna in smartphones. Limited by this and our constraints (i.e., no extra amplification), in our experiments the dishonest prover is not technically "far-away", as the tag and reader need to be at a maximum of 2cm apart (less than the ISO/IEC 14443 bound of 10cm) for an NFC connection to be established.

Due to the above, in fact our implementation of the attacks and our measurements only extrapolate how much distance could be gained via these DFs, i.e., are not performed with a distant prover. In other words, we are closer to studying distance-lowering attacks than mounting "fully-fleged" DFs.

In our experiments, we write a log to register the results of each attack of a series of runs, as well as

[3]This may not work if 2 devices are further apart than in our setup; see the "Experiments" section. Similarly, the approximation $t_{end\text{-}of\text{-}init\text{-}phase}$ may be improper.

the timings that each run produces. For this, we add code on the verifier side that measures the duration of the DB phase (in nanoseconds). No other modification besides this logging is done to the verifier. The verifier can run multiple executions uninterrupted. So, we set the verifier run 10 executions per batch, and we ran 20 batches. We used the same parameters as in (Gambs et al., 2016): n=32, err=3, err_T=70500000ns. Some results are given in Tables 2, 3.

Table 2: Measurements Averaged over 200 Iterations.

	No Attack	Imperfect DB-Timing DF	Perfect DB-Timing DF
Avg. DB-phase	425.18 ms	312.80 ms	292.58 ms
Avg. err_c	0	0	3
Avg. err_r	0	0	0
Avg. err_t	0	5	0

Table 3: Full-Runs' Durations (in ms, avg. over 200 iterations).

	Min	Max	Mean
No attack	315.08	591.39	425.18
Imperfect DB-Timing DF	254.4	447.87	312.80
Perfect DB-Timing DF	252.53	401.84	292.58

The "perfect DB-timing" DF shows a higher number of low execution times. Meanwhile, the "imperfect DB-timing" DF has a higher variance in times since the prover has to wait for a certain number challenges before being able to respond.

5.3 Discussions

The success of the prover depends on the total *err* parameter. An accepted *err* of 10% is common (Boureanu and Vaudenay, 2015), and *err* fixed at 3 in the implementation. Hence, according to our experiments, in a third of the cases the dishonest prover would be accepted. Also, when the illicit prover passes, the "Perfect DB-Timing" DF displayed a 31% reduction in execution time and the "Imperfect DB-Timing" DF exhibited a 26% reduction. If we also look at the gain in time, this could translate in DFs over thousands of kilometres. Of course, if an amplification antenna were used and thus NFC timing/framing aspects would become more difficult to tune, it is not guaranteed that the rate of increasing the distance compared to the distance-bound would be maintained at around 30%. Also, this would likely not translate to "pure" NFC communication, which do not need to interface with Android. Moreover, if err_T was smaller, then the success rate would also drop. Also, these results are for a fixed key and a full assessment for DF in SwK should be averaged over uniformly sampled keys on the prover's side. We leave this for future experiments.

The "perfect DB-timing" DF appears more suited for use further away from a verifier, due to the reduced execution times, and the ability to perfectly send back responses without timing errors. Though, this is at the expense of yielding response errors. "Imperfect DB-timing" DF results in no challenge or response errors, but the dishonest prover needs to be closer to the verifier, because in some rounds, a wait is required before the prover can respond. There is a chance of the verifier timing out when waiting for a response if the prover is too far away. All these also indicate, once more, than the error-tolerance parameters in DB need to be carefully chosen and possibly not compound into a single error factor.

6 CONCLUSIONS

Most distance-frauds (DFs) attacks are white-box: i.e., they require a dishonest modify its implementation. We singled out feasible, "weak" white-box DFs: a dishonest prover will not subvert cryptographic primitives in the attack. We implemented two such DFs and tested their feasibility on mobile phones, with no extra hardware resources, showing that with a DF-attacker can make it look like they are here and there at the same time! Much work remains to be done to use, e.g., amplification antennas in stronger versions of these attacks, run over large distances.

REFERENCES

Avoine, G., Bingöl, M., Kardas, S., Lauradoux, C., and Martin, B. (2011). A Framework for Analyzing RFID Distance Bounding Protocols. *Journal of Computer Security*, 19(2):289–317.

Avoine, G., Bingöl, M. A., and Boureanu, I. e. (2018). Security of distance-bounding: A survey. *ACM Comput. Surv.*, 51(5):94:1–94:33.

Boureanu, I., Gerault, D., and Lafourcade, P. (2018). Implementation-level corruptions in distance bounding. Cryptology ePrint Archive, Report 2018/1243. https://eprint.iacr.org/2018/1243.

Boureanu, I., Mitrokotsa, A., and Vaudenay, S. (2012). On the pseudorandom function assumption in (secure) distance-bounding protocols. In *LATINCRYPT*, pages 100–120. Springer.

Boureanu, I. and Vaudenay, S. (2015). Challenges in distance bounding. *IEEE Security Privacy*, 13(1):41–48.

Brands, S. and Chaum, D. (1993). Distance-Bounding Protocols. In *Proc. of EUROCRYPT'93*, pages 344–359, Lofthus, Norway.

Chothia, T., Boureanu, I., and Chen, L. (2019). Making contactless EMV payments robust against rogue readers colluding with relay attackers. In *Financial Crypto*.

Gambs, S., Lassance, C. E. R. K., and Onete, C. (2016). The Not-so-Distant Future: Distance-Bounding Protocols on Smartphones. In *CARDIS*, pages 209–224. Springer.

Kim, C. H., Avoine, G., Koeune, F., Standaert, F., and Pereira, O. (2008). The Swiss-Knife RFID Distance Bounding Protocol. In *ICISC*, LNCS. Springer.