Precise and Mechanised Models and Proofs for Distance-Bounding

and an Application to Contactless Payments

Ioana Boureanu[1], Constantin Catalin Dragan[1], François Dupressoir[2], David Gerault[3], and Pascal Lafourcade[4]

[1] University of Surrey, UK i.boureanu@surrey.ac.uk, c.dragan@surrey.ac.uk
[2] University of Bristol, UK f.dupressoir@bristol.ac.uk
[3] Nanyang Technological University, Singapore dagerault@gmail.com
[4] University Clermont Auvergne, France pascal.lafourcade@uca.fr

Abstract. Distance-bounding protocols provide a means for a verifier (e.g., an RFID reader) to estimate his relative distance to a prover (e.g., a bankcard), in order to counteract relay attacks. We propose *FlexiDB*, a new cryptographic model for DB, parameterised over fine-grained corruptions. It allows to consider trivial cases, classical cases but also new, generalised scenarios in which we show manipulating differently-corrupt provers at once leads to new attacks. We propose a proof-of-concept mechanisation of *FlexiDB* in the interactive cryptographic prover EasyCrypt, and use it to prove a flavour of man-in-the-middle security on a variant of MasterCard's contactless-payment protocol.

1 Introduction

Not only do we use contactless payments, but we use them more and more. Across the UK alone, "contactless payments have grown in recent years, with a record 34% of card payments using contactless[5] in June 2017. And, we use contactless systems because of their high usability and convenience. Yet contactless communications, such as tap-and-pay and remote keyless ignition system (RKI), due to them requiring no active user-input, are particularly vulnerable to *relay attacks*. In these, a man-in-the-middle (MiM) ferries the communication back and forth between two parties P and V, unbeknown to them. The setting is one where P and V are not within the right communication range to start communicating as intended, but the relaying adversary forces a stealth out-of-band communication by impersonating V to P and vice-versa. The aim of the relaying MiM is to get some illicit gain, that normally is attributed to P and/or V. Indeed, relay attacks working successfully across distances as wide as from the US to the UK have been shown on contactless payments [17]; in this case, the attacker pays fraudulently by a payment terminal V by stealing the funds associated to a bankcard P, without any untoward evidence in the communications seen by P and V.

[5] https://www.visa.co.uk/about-visa/newsroom/press-releases.2130476.html

Distance Bounding (DB). To counteract relay attacks, one classical means is to add a *distance-bounding (DB)* or *proximity-checking* mechanism on top of contactless protocols, be them authentication, payments schemes or RKI. In its simplest form, in distance-bounding, the verifier party V (i.e., the car, the payment-terminal) measures the round trip times (RTT) of an exchange with the prover party P (i.e., the key-fob, the bankcard) and compares this measurement to a given bound; thus, if the measurement is within the bound, then the verifier concludes that the prover should be physically within some given, acceptable range. Nowadays, distance bounding is not just in the realms of theory, it is very much adopted in real-life systems, in various applications. For instance, since 2016, Mastercard has augmented its original, contactless payment scheme called PayPass with a *relay protection protocol (RRP)* which is indeed a distance-bounding procedure.

Incomparable DB Security Models. The security of DB constructs has been studied for two decades [2], not just as a RTT-measuring mechanism but generally as an authentication protocol. Semi-formal and formal models of its security appeared from 2011 onwards [2]. However, the particular security-modelling details of the threat models vary from formalism to formalism: (a) should we have multiple provers be exploitable in an attacks or consider that just the victim prover is present? (b) should device corruption be considered black-box or white-box? (c) should the attacker have powerful control over the network (e.g., use signal amplification, flip bits) or just do pure relaying? Not having a consensus on such matters leads to incomparable (in)security results.

No Mechanised Cryptographic Proofs for DB. In formal methods for security analysis, there are two main schools of thought: symbolic and computational [9]. The tools used by each of these two methodologies view general-purpose security/cryptography, having no built-in capabilities to facilitate the reasoning about physical aspects such as time-measurements or distance bounds. In the last two years, symbolic verification made steps towards the mechanisation of verification of distance-bounding protocols, including looking at those used in payments such as PayPass. However, there is no computational mechanisation of distance-bounding models.

Contributions. We will address the two main shortcomings described above. Our two main contributions are:

1. We develop a new DB security formalism, called FlexiDB, which is in fact a hierarchy of threat-models for DB, parameterised over the capabilities of the adversary w.r.t. its corruption and at the network-manipulation abilities. This means that each application (i.e., authentication, payments, RKI) can pick the adversary or sets thereof that fit their domain and security requirements.
   - Given this generic/flexible nature, the security properties included in our FlexiDB model capture and generalise existing DB-security properties,

as well as being –in the strongest attacker-model possible– totally new additions to this area.
  - Indeed, the latter leads to us also exhibiting new attacks on DB protocols, including on contactless payments.
2. We mechanise, in EasyCrypt, both the core of FlexiDB and a proof that – against one of the threat models in FlexiDB– one of MasterCard's contactless-payment protocol does indeed realise distance-bounding against MiM attackers.

Finally, the take-home message of our contributions is also two-fold. First, our model permits to prove the security of a protocol within a specific corruption "scenario", such as adversaries being able to obtain several secret keys, but having limited channel manipulation abilities. The need for such granularity arises directly from practical applications. For instance, in the plastic-card contactless payments, communications are assumed to only be possible within limited range, and cards are assumed to be resistant to tampering. Conversely, in smartphone contactless payments, key extraction can become feasible. Second, by providing and demonstrating support for the model in EasyCrypt, we aim to allow future proofs in the model to be machine-checked. Beyond ensuring that (computational) proofs are indeed correct—a worthy goal in and of itself, this in turn will allow evaluators and decision makers to focus their critical efforts on evaluating the security claims themselves to ensure they are appropriate for the use case considered, without having to also evaluate the claims' truth.

2 Related Work

Distance-Bounding Notions. Distance-bounding protocols are subject to four main threats. 1. *Mafia fraud (MF)* is an attack whereby a MiM, present in the range of the verifier V, tries to authenticate as a legitimate prover P, whilst P is out of V's range. 2. In a *distance fraud (DF)*, a malicious prover located passed the acceptable bound from the verifier attempts to authenticate. 3. *Distance hijacking (DH)* is generalising DF, as the far-away, corrupt prover is abusing honest provers found close to the verifier. 4. In a *terrorist fraud (TF)*, the DF-mounting far-away prover P has an accomplice located near the verifier and this accomplice is supposed to authenticate as P under special conditions (e.g., the accomplice does not learn P's cryptographic secrets). Variations and generalisations of the above descriptions of DF, DH, MiM exist [2]. In our model we consider the strongest generalisation of these and even strengthen them further. However, due to the lack of consensus on TF, we leave this threat out of our model.

Main Cryptographic Models for DB. In 2009, Avoine *et al.* put forward the first a semi-formal, computational framework for DB security [4]. They considered a single prover and verifier present in all attack scenarios. Possibly the

most relevant contribution of [4] was to explicitly distinguish black-box provers from white-box provers[6].

In 2011, Dürholz *et al.* proposed a cryptographic model more formal than [4], where the DB setups (DF positioning, MF positioning) where modelled via allowed/disallowed protocol-session interleaving. This formalisms allows for concurrency, all dishonest provers are corrupted just in the white-box manner, and the formalisation of DF, MF is not generalised.

In 2013, Boureanu *et al.* published the so-called *BMV* model [14,11], which formalises DB security as interactive proofs. This model allows for concurrency, all dishonest provers are corrupted just in the white-box manner, but unlike [4], the BMV model generalises the DF, DH and MF definitions, allowing for e.g., learning phases before the attacks and multiple provers being present alongside during the attacks. In 2017, Ahmadhi *et al.* [1] slightly extended the BMV model by allowing the adversary to send unicast messages (as opposed to the traditional broadcast-only); this gave rise to new attacks.

Main Symbolic Models for DB. In 2011, the first symbolic verification of DB was proposed [7]. Building on these ideas, in 2017 – 2018, three symbolic formalism for DB-security verification emerged: *Mauw et al.* [31], *Chothia et al.* [16], and *Debant et al.* [20]. Very recent and very specific extensions (i.e., around TF verification) are [32] (of 2018's *Mauw et al.*), [21] (of 2018's *Debant et al.*). On the one hand, these, being symbolic, are less precise than their aforementioned cryptographic counterparts. On the other hand, they make different assumptions on the capacities of the adversary and/or the communication patterns. For instance, in all these symbolic formalisms, the adversary has the "new" ability to block or replace messages in flight, even from afar. In fact, this capability is realistic and grounded in well-known signal manipulation techniques [35].

Symbolic models are particularly well-suited to mechanised verification. We note, however, that security claims in the computational model are often not comparable to their symbolic counterparts, and view support for mechanisation in both worlds as complementary, with symbolic verification particularly useful for rapid design-time iteration, and computational mechanisation useful to make precise concrete security claims, often at considerably greater expense.

Note. There are more variations on the cryptographic models mentioned above (i.e., on Dürholz *et al.*, on the BMV model), yet for the purpose of this work, these are not essential. For a summary of these, please see [2]. We also do not further review work on the symbolic verification of DB: we place ourselves squarely in a computational model of cryptography, and view our efforts are complementary to—rather than competing with—those of symbolic verification. We direct the reader interested in a review of symbolic verification work for DB to Pozo's very thorough survey [36]. We discuss and compare specific detail of prior work where relevant in the manuscript's developments. For example, we

[6] The user of a white-box device has access to its secret key, while black-box provers operate in a manner that it totally opaque to their users.

compare and contrast with other formulations of the threats we consider after giving our security definitions after giving them.

Our FlexiDB Model vs. Existing Models for DB. FlexiDB is a new cryptographic model for distance bounding, which additionally includes variations on the types of device-corruptions and illicit network-manipulation, as well as advanced levels of concurrency. All these adversarial capabilities are given as parameters inside our FlexiDB model, therefore enabling a fine-grained analysis in line with the practical realities of protocol implementation.

Our FlexiDB model runs closest in the style of definitions to the BMV model (e.g., we consider learning/attack phase, full concurrency). However, we operate in an oracle-based model rather than interactive-Turing-machines setting. Further, by allowing the variation in adversaries, we offer a hierarchy of definitions for each security property, as well as a strengthening the definitions in the BMV model. Concretely, in our DF and MF definitions, it is possible to have corrupt provers, and also corrupted in various ways, present in the security experiments, which was not the case in the BMV model. Indeed, we adopt the white/black-box corruption idea from [4] as Insider,Outsider adversaries but take it further. That is, whilst our *weak-insider* adversaries correspond to Avoine's white-box attackers, we additionally formalise the new and even stronger notion of *strong insiders* adversaries, who can pick their own secret keys. We further allow multiple, concurrent presences thereof, as well as combine this in various ways with other attacker's capabilities (e.g., network manipulation); to this end, like [1], we allow both broadcast and unicast messages. Finally, we also allow, like symbolic-verification works do, that messages can be modified by the adversary from afar.

3 FlexiDB: Formalising Refined Threats in DB

3.1 Distance-Bounding Protocols

Def. 1 gives our formally formulation of a DB protocol.

Definition 1 (Distance-Bounding Protocols). *A* distance-bounding protocol *is a tuple* $\Pi=(\mathbb{P},\mathbb{V},Setup,\mathbb{B})$, *such that:*
– $\mathbb{P}$ *and* $\mathbb{V}$ *are the prover and, respectively, the verifier algorithm,*
– $\mathbb{B}$ *is a fixed distance bound considered in a context of a metric space and a metric/distance,*
– $\mathbb{V}$ *is such that it has a binary output* out_V*, denoting success/failure of his execution,*
– *Setup is an algorithm used to initialise the DB system.*

All algorithms are polynomial probabilistic time (ppt) in a security parameter[7] *s.*

[7] Computational measures such as polynomial probabilistic time (ppt), negligible, etc., vary with the security parameter. We consider these and associated notions, e.g., Interactive Turing Machines (ITMs) [40], commonplace.

We assume the existence of an infrastructure that supports DB protocols to be run, i.e., the authentication material generation algorithms and cryptographic primitives relevant to a given protocol. We call this infrastructure a *DB system*.

Setup, Algorithms & Parties The main purpose of the *Setup* algorithm in Def. 1 is to generate the authentication material of provers and verifiers, as new provers and verifiers are registered onto the DB system. During this *Setup* phase, the $\mathbb{P}$ and $\mathbb{V}$ algorithms will also be loaded onto physical *devices* (e.g., cards, phones, terminals). Moreover, at *Setup*, each device is assigned a unique identifier, *i.e.*, no two devices have the same authentication material and/or identifier. A party P with public identifier i is denoted P_i.

There are a polynomial number of devices present in the system. Some of these devices are corrupted by the adversary, as it will become clearer later. We use the words *parties* and *devices* interchangeably; concretely, a party is a uniquely identifiable device running any algorithm as part of a protocol execution: be it a prover, a verifier or adversarial algorithm.

3.2 Physical & Communication Model

Now, we present our modelling of distance and timing.

Positions & Distances. As usual in distance bounding, we operate in a metric space. Each party U occupies a position/place in this metric space. We denote the position of party U as $place_U$. Also, in this metric space, we consider d to be the distance-function. And, as normal in DB, in security requirements we will also ascertain whether for some given parties U, V, we have that $d(place_U, place_V) \leq \mathbb{B}$. I.e., if this holds, we say that party U is *close* to party V, otherwise we say that U is *far* from V.

Messages sent by all parties, including adversaries, are subject to a time of flight. The time-of-flight is measured w.r.t. a global counter called $Clock$. This $Clock$ is incremented as messages travel, and the communication model will later explain this in detail. We assume that messages that are normally computed in polynomial time take 0 ticks of the $Clock$.

The distance d between two parties in fact measures the time-of-flight of messages between two positions, considering the messages travel uniformly at a speed c of one distance-unit per time-unit. No message, honest or adversarial, can travel faster that at the given speed c.

All messages sent by prover and verifier parties are broadcast. Adversarial parties can send unicast messages.

There is a system-recall of the position of parties, as well as of all sent and received messages, including their timestamps; we will later develop on the system-recall.

3.3 Threat Model

The adversary $\mathcal{A}$ has full control over two *adversarial parties* operating as ITMs. We distinguish these adversarial parties from the *honest parties* (provers and

verifiers), and we denote the latter as $\mathcal{A}_P$ and $\mathcal{A}_V$. This mirrors the classical mafia fraud, where two adversary parties are involved. Following this convention, $\mathcal{A}_V$ and $\mathcal{A}_P$ respectively represent the adversary near a verifier and the one near a prover.

We consider a hierarchy of adversaries, determined by *two different types of adversarial abilities*:
(1) corrupting/controlling parties;
(2) corrupting/controlling the network.

Party Corruption We distinguish three levels of this:
• Outsider (O), if the adversary is only given access to the public identifiers of all parties;
• Insider (I), if the adversary is also given access to the authentication material of some parties. Particularly, an *Insider* adversary can be of two kinds:

- Weak-Insider (WI), if the adversary is just given access to the authentication material of prover-devices of his choice;
- Strong-Insider (SI), if the adversary is also allowed to register prover-devices with his own choice of authentication material and identifiers.

Moreover, *Insider* adversaries are divided as follows:

- *1*-Weak-Insider (*1*-WI) or *1*-Strong-Insider (*1*-SI), when the adversary can only corrupt one prover;
- n-Weak-Insider (n-WI) or n-Strong-Insider (n-SI), when the adversary can corrupt several provers.

When the distinction between *Weak-Insider* and *Strong-Insider* is not important, we simply write "*Insider*".

These corruption abilities mean the adversary can register one or more provers for which he knows (*Weak-Insider*) or, alternatively, choses (*Strong-Insider*) the secret material. Table 1 recaps our party-corruption capabilities.

Nb. of Prover-Keys	**Adv. knowing keys**	**Adv. choosing Keys**
0	O	O
1	1-WI	1-SI
n >1	n-WI	n-SI

Table 1. Party-corrupting Adversaries by Nb. of Provers' Keys Known/Chosen.

Network Corruption We distinguish three types of adversarial communication capability, mainly determined by the physical-layer implementation of the protocol. This entails the following types of adversary:
• Dummy (*Dum*), if the adversary cannot control the whole network, in that

it can only send and receive messages to/from honest parties within a distance smaller than or equal to the bound $\mathbb{B}$;
• Amplifier (*Amp*), if the adversary can do signal-amplification, meaning that it can receive and send messages to/from honest parties across distances larger than the bound $\mathbb{B}$;
• Injector (*Inj*), if the adversary can block messages, or overwrite them with its own, when the message is originated from a point found no further than the bound $\mathbb{B}$;
• Full (*Full*), if the adversary can do all of the above, *i.e.*, send, receive, block and overwrite messages even if they originate from point found further than the bound $\mathbb{B}$.

Note: An *Injector* adversary may seem strong or even unrealistic. However, this is in line with practice: it is known as overshadowing, see [35] for more details. Moreover, whilst prior cryptographic models fo not capture this ability, all symbolic verification mechanisms (e.g., [21,32]) do model adversarial overshadowing/injection (almost by implicit virtue of symbolic formalisms).

Table 2 recaps our adversaries partitioned by network-corruption capabilities, and next we propose to focus only on the categories defined below, while noting that other combinations may be relevant.

Adversary Type	**Capability to replace msgs.**	**Capability to amplify msgs.**
Dum	×	×
Amp	×	✓
Inj	✓	×
Full	✓	✓

Table 2. Network-corruption Capabilities Across Our Adversaries.

The entire threat model and the adversarial communications aforementioned are formalised via a set of oracles presented in Subsection 3.4.

3.4 Execution Model

Sessions. As usual, a party's execution of (a part of) a DB protocol is called a *session*. If one execution is run on a prover-device or verifier-device, then it is a *prover session* or a *verifier session*, respectively. We write X^i for the *i-th session of a party* X.

Each prover and verifier party has a *status*, active or inactive, meaning that it is running at least one session or none.

As customary, the chronologically-ordered list of the messages sent and received by a party in a session is called the *transcript* of the session. All sessions are attributed a unique identifier (e.g., via the application of the pseudorandom function to the transcript). A session is *full* if its transcript contains all messages of the specification. As per our DB definition (Def. 1), the verifier-transcripts show whether the authentication is accepted or not. Moreover, we consider that

from a successful, full verifier-transcript one can extract the public identifier of the prover-party that was authenticated[8].

Execution environment. We consider the concurrent sessions of a polynomial number of honest and corrupted parties, including potentially the adversarial devices as well as their concurrent sessions. We refer to such a setting as an *execution environment.*

Challenger ($\mathcal{C}h$). To mechanise the execution environment and to arbitrate the adversarial actions within it, we introduce (as expected) a *challenger*. The main abilities of $\mathcal{C}h$ are:

1. The challenger $\mathcal{C}h$ is aware of the global clock $Clock$.
2. The challenger $\mathcal{C}h$ keeps a list `Pts` of all parties[9] in the system, indexed by their id.
   Also, $\mathcal{C}h$ deals with all adversarial actions via a set of oracles presented later; as such, challenger $\mathcal{C}h$ knows if a given party has been corrupted by $\mathcal{A}$ and his list `Pts` is kept up-to-date accordingly.
3. The challenger $\mathcal{C}h$ keeps track of every session opened by every party in a list called `Sess`. This list is indexed by the unique session identifier, and it registers the time the session started, if it is a prover session, a verifier session, as well as the up-to-date status of a session(i.e., finished or running), and a transcript of the session.
4. The challenger $\mathcal{C}h$ keeps a list ***`Sends`*** of timed, sent messages. This contains: the id of the session (of the sender party) to which this is message belongs to, the sender party, the aimed receiver party (which is optional), the message, and the time of send. Recall that most messages are sent in broadcast mode, and only the adversary can send messages in unicast mode; so, the latter is the only case in which is there is an aimed receiver.
5. The challenger $\mathcal{C}h$ keeps a list ***`Reads`*** of read messages at given times. This contains: the id of the session (of the reading party) in which this message is being read, the (apparent) sender party, the (real) sender party, the receiver party, the message, the time of the receipt.

We underline one **time-keeping aspect** here. Firstly, if the "read" is from/to a sender and receiver, then an entry in the `Reads` list is possible only if the message appears in the sent-messages list `Sends` and if the message had the time to travel from the sender to the receiver. I.e., $d(sender, receiver) \leq (current\_time - t_{sent}) \times c$ where the challenger $\mathcal{C}h$ finds the positions of $sender$, $receiver$ in the `Pts` list , the time t_{sent} in the `Sends` list, the $current\_time$ by using the global Clock, and c is the speed of messages. If this inequality holds, then the time of receipt inside `Reads` is recorded as the $current\_time$.

The points above show that the challenger $\mathcal{C}h$ is an arbiter for the setup of the system, of honest and corrupt behaviours, and of the communication rules. Specifically, w.r.t. point (5) above, the challenger $\mathcal{C}h$ uses his "communication

[8] This is realistic (as such public identifiers are often sent in clear) and poses no problem herein, as we do not treat provers' anonymity or privacy.

[9] "Parties" include all adversarial parties, as aforesaid.

logs" kept via the lists `Pts`, `Sess`, `Sends` and `Reads`, so that he does not allow our communication rules to be broken.

Adversarial Oracles. The challenger $\mathcal{C}h$ in fact generates the execution environment, and permits the adversary to interact with the environment through a polynomial number of calls to *oracles*. These oracles permit the adversary to populate the environment with provers and verifiers at positions of his choice, and enforce the communication and corruption rules.

All our oracle calls are done by an adversary party $\mathcal{A}_{id}$ and each call takes account of $\mathcal{A}_{id}$'s position in the metric space. For instance, all parties can read a message sent by $\mathcal{A}_{id}$ only at a time proportional to the distance between $\mathcal{A}_{id}$ and themselves. Similarly, creation of parties at a given position are only effective after the time proportional to the distance between the party created and $\mathcal{A}_{id}$. For simplicity, we often omit the $\mathcal{A}_{id}$ parameter in the description of our oracles.

To describe each oracle, we generally write $\texttt{oracle-name}^{\texttt{adversary}}_{\texttt{max}}$, where "adversary" denotes the kind of adversary (e.g., *Amplifier*, *Weak-Insider*, etc) allowed to call the oracle in cause, and "max" denotes the maximum value of a counter internal to the oracle. If the superscript is missing, this denotes that the oracle at hand can be called by any type of adversary. If the subscript is missing from the description of an oracle, this denotes that it is the challenger $\mathcal{C}h$ who keeps the tally of numbers of calls for this oracle, as opposed to the oracle itself. Our oracles are as follows.

`join`(*type*, *pos*): This oracle simulates the registration of a new honest party of a given *type* (i.e., prover or verifier) at a position *pos* in the metric space. To $\mathcal{A}_{id}$ calling `join`, the oracle returns the public identifier of the new party.

$\texttt{join}_k{}^{WI}$(*pos*): This oracle simulates a *Weak-Insider* adversary registering a corrupted prover at a position *pos* in the metric space. To $\mathcal{A}_{id}$ calling $\texttt{join}^{WI}$, the oracle returns the public identifier and authentication material of the new prover.

$\texttt{join}_k{}^{SI}$($id, auth, pos$): This oracle simulates a *Strong-Insider* adversary choosing a public identifier id and authentication material $auth$, and then registering a corrupted prover with this id and $auth$, at a position *pos* in the metric space. It aborts without creating the prover and returns $\perp$ if another prover with the same identifier or authentication material already exists. Otherwise, to $\mathcal{A}_{id}$ calling $\texttt{join}_k{}^{SI}$, the oracle returns $\top$.

For the oracles `join`, $\texttt{join}_k{}^{WI}$, $\texttt{join}_k{}^{SI}$, we also have that:

- the challenger $\mathcal{C}h$ adds the registered party to the `Pts` list and it also specifies its type: honest for `join`, corrupted for $\texttt{join}_k{}^{WI}$ and $\texttt{join}_k{}^{SI}$;
- at each call, an internal counter is incremented; after k calls, the oracle is disabled, i.e., returns $\perp$.

`enable-broadcast`(): This oracle activates a communication mode in which, by default, all messages by prover and verifier parties are sent to all parties even if they are found far apart from where the message originates. The challenger $\mathcal{C}h$ stores and s sets a flag *broadcast*, once this is called.

`init`($[P, V]$): This oracle simulates the start of new executions of their respective algorithms by a prover-party with id P and/or for a verifier-party with the

id V. Either P or V can be omitted, in which case the adversary is running a session with the party invoked.

If the *broadcast* flag is not set, then this oracle can only be called on provers and verifiers within the distance bound from the position of $\mathcal{A}_{id}$ who calls this.

From the point of the call, the $\mathcal{C}h$ delays the start of session by the time proportional to the distance the parties in the session (P and/or V and/or $\mathcal{A}$).

The session identified is returned to the adversary and it is stored by $\mathcal{C}h$ in the `Sess` list. All other relevant aspects (e.g., status of P, V in the `Pts` list) are updated by $\mathcal{C}h$ at its end.

`move`$([P]$, $pos)$: This oracle moves a party with the identifier P from its current position to pos. If P is omitted, the party being moved is the adversary party $\mathcal{A}_{id}$calling this oracle.

The challenger updates its `Pts` list accordingly.

`send`$^{Dum}([X, sid], m)$: This oracle simulates the sending of a message m from $\mathcal{A}_{id}$ to the session sid of the party with the id X. If X is a prover.verifier party far from $\mathcal{A}_{id}$ who call this, and *broadcast* is not set, then the oracle aborts and returns $\perp$.

The parameters X and sid are optional. If omitted, then the message m is broadcasted to all parties, either within the distance-bound from $\mathcal{A}_{id}$ if *broadcast* is not set, or otherwise broadcast even past the distance bound from $\mathcal{A}_{id}$.

The challenger records this in the `Sess` list (updating transcripts), the `Sends` list (updating time, etc.).

`replace`$(X, sid, \mathcal{B}, \mathcal{P}, \mathcal{S}, m')$: Let $M = M_0 \dots M_k$ denote all the bits of the next message to be sent by the party X in the session sid, $\mathcal{P}$ be a (possibly empty) set of parties, $\mathcal{S}$ be a (possibly empty) set of sessions, and m' be a message. This oracle replaces the message bits $\{M_i | i \in \mathcal{B}\}$ with m', so that the sessions in $\mathcal{S}$ and the parties in $\mathcal{P}$ receive the modified message; this modification can result also in deleting bits from the message. If $\mathcal{B} = \star$, then the whole message is replaced.

If X is a prover or verifier party located past the distance bound from $\mathcal{A}_{id}$ who is the called of this oracle, and if *broadcast* is not set, then the oracle aborts and returns $\perp$.

In addition to these adversary oracles, we define the following tool function, used by the challenger to determine the success of the adversary in a given attack. It is not accessible to the adversary, and merely used as a syntactic shortcut to express the result of a session.

`result`(sid, V): Retrieves the session with id sid of the verifier party V. If the session exists, and the V accepted the authentication of a prover $P.id$, returns $(\top, P.id)$. Otherwise, it returns $\perp$ – meaning the session was unsuccessful in authenticating a party.

Note: For simplicity, in this section, we included the level of detail necessary for the reader to understand the crux of our model, the security properties/games in Section 4 and the attacks in Section 5. To this end, we omitted the following: (a) a **read** oracle aligned to that of the **send** oracle; (b) details of the management by the challenger of the exact timing-keeping w.r.t. the **send**/ **read**

adversary type	$\mathsf{O}^{\mathsf{core}}$	$\mathsf{O}^{\mathsf{corr}}$	$\mathsf{O}^{\mathsf{com}}$
Outsider	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	$\emptyset$	NaN
1-*Weak-Insider*	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	$\{\texttt{join}_1^{WI}\}$	NaN
n-*Weak-Insider*	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	$\{\texttt{join}_n^{WI}\}$	NaN
1-*Strong-Insider*	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	$\{\texttt{join}_1^{SI}\}$	NaN
n-*Strong-Insider*	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	$\{\texttt{join}_n^{SI}\}$	NaN
Dummy	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	NaN	$\emptyset$
Amplifier	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	NaN	$\{\texttt{enable-broadcast}\}$
Injector	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	NaN	$\{\texttt{replace}\}$
Full	$\{\texttt{join}, \texttt{init}, \texttt{move}, \texttt{send}^{Dum}\}$	NaN	$\{\texttt{enable-broadcast}, \texttt{replace}\}$

Table 3. Oracles per Type of Adversary.

oracles; (c) details of exact book-keeping the `Sess` list, w.r.t. these two oracles; (d) the honest versions of the **send**/ **read** oracle, which the adversary would also have to call to emulate the execution environment. However, in Section 6, where we present the mechanisation of this model in EasyCrypt, such details are included.

4 DB Security Properties in FlexiDB

We first define a categorisation of the set of oracles in function of adversary type (i.e., *Amplifier* *Weak-Insider*, etc.).

4.1 Oracles, Adversary Positions and Attack Phases.

We write $\mathcal{A}^{\mathcal{O}}$ to mean that the adversary has access to a particular set $\mathcal{O}$ of oracles. Our oracles are split in three sets:

- $\mathsf{O}^{\mathsf{core}}$: set of oracles accessible to all adversaries.
- $\mathsf{O}^{\mathsf{com}}$: set of oracles related to network-corruption only;
- $\mathsf{O}^{\mathsf{corr}}$: set of oracles related to party-corruption only.

Our different adversaries are described in Table 3.

By integrating our advanced/fine-grained corruption capacities, we generalise the classical notion of mafia fraud with resistance to ***generalised mafia-fraud (GMF)*** and distance fraud with ***generalised distance-fraud (GDF)***.

In the GMF experiment, the adversary is considered as 2 entities: $\mathcal{A} = (\mathcal{A}_V, \mathcal{A}_P)$, one being close to the designated verifier, and the second being close to the designated prover.

In the GDF security experiment, a single adversary party $\mathcal{A}_P$ is located far away from the designated verifier.

In both cases, the designated prover is far from the designated verifier. These two settings are illustrated on Figure 1.

In our GMF and GDF, the adversary is allowed to perform a **learning phase**, in which he can freely interact with the environment, with no positioning restrictions w.r.t. to provers/verifiers. During this phase, $\mathcal{A}$ populates the environment

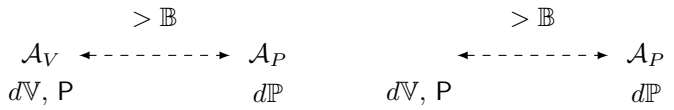

Fig. 1. Possible GMF Setting (on the LHS) and GDF Setting (on the RHS). $d\mathbb{V}$ is the designated verifier, $d\mathbb{P}$ is the designated prover authenticated/attacked, and P denotes an arbitrary set of provers.

with provers and verifiers, interacts with them and sets all positions as he wishes (all formally done via our oracles). Then, the adversary selects a designated prover $d\mathbb{P}$ and a designated verifier $d\mathbb{V}$, and gives their identifiers to the challenger. The challenger then disables some of the oracles (see formal definitions for details), verifies that the setting of the environment is correct with regards to the security property, and allows the adversary run the actual **attack phase**. During this phase, the adversary has access to a restricted set of oracles compared to the learning phase, and his position w.r.t. to the $d\mathbb{P}$ and $d\mathbb{V}$ is restricted (i.e., the adversary cannot be close to either of the two).

4.2 Security Properties Definitions

We formalise our definitions which lead to new attacks presented in Section 5.

Generalised Distance Fraud (GDF) It comprises in fact a class of distance frauds and distance-hijacking attacks, which vary with the strength of the corruption and network-manipulation of our attackers.

Our Fine-Grained GDF & Its Benefits. In the classical setting of distance fraud, a dishonest prover P tries to fraudulently authenticate from afar. As such, in our terminology, this would be an *Insider* adversary $\mathcal{A}$ who called (at least) $\texttt{join}^{WI}$(if not $\texttt{join}^{SI}$)on P (i.e, knows the authentication material of P) and who attempts to authenticate from afar. However, we do not restrict our $\mathcal{A}$ to be an *Insider* at all; this is possible in our generic framework, but so is the much stronger setting where $\mathcal{A}$ controls several provers and even their authenticating material (i.e., $\mathcal{A}$ is an n-SI adversary) as well as the benign case where $\mathcal{A}$ is an *Outsider*.

Such a n-WI adversary may not just act alone, but also exploit a series of corrupted provers located near the verifier. It opens for new attacks that we exhibit in Section 5. In particular, we show that under this adversary setting, most distance bounding protocols are vulnerable to a form of GDF.

Similarly, if our attacker is a SI adversary then he may perform distance frauds against certain protocols, that are not possible for classical WI adversaries (cf. Section 5).

We give our generalised distance-fraud in Definition 2: i.e., a class of attacks in which an adversary tries to make a designated verifier $d\mathbb{V}$ authenticate a prover $d\mathbb{P}$, even though no adversarial party nor $d\mathbb{P}$ is within a distance $\mathbb{B}$ of $d\mathbb{V}$.

Definition 2. Generalised Distance-fraud (GDF) & Security against GDF. *Let Π be the a DB protocol. A* generalised distance-fraud (GDF) game *$\mathcal{G}$ against the DB protocol Π is split in two phases: the* learning phase *and the* attack phase*.*

- *The* learning phase for GDF *is a multi-party execution of the protocol Γ in the presence of an adversary $\mathcal{A}= (\mathcal{A}_P, \mathcal{A}_V)$ such that the position $\mathsf{pos}_{\mathcal{A}_\mathsf{P}}$ of $\mathcal{A}_P$ and the position $\mathsf{pos}_{\mathcal{A}_\mathsf{V}}$ of $\mathcal{A}_V$ is arbitrary.*
- *In this phase, the challenger $\mathcal{C}h$ starts by setting up an execution environment and giving access to $\mathcal{A}$ to the set of oracles $\{\mathsf{O}^{\mathsf{core}}, \mathsf{O}^{\mathsf{com}}, \mathsf{O}^{\mathsf{corr}}\}$.*
- *The phase finishes with the adversary returning a designated prover and verifier pair $(d\mathbb{P}, d\mathbb{V})$, and the starting position of one adversarially controlled parties denoted $\mathcal{A}_P$, i.e.,: $(\mathsf{pos}_{\mathcal{A}_\mathsf{P}}, d\mathbb{P}, d\mathbb{V}) \leftarrow \mathcal{A}^{\{\mathsf{O}^{\mathsf{core}}, \mathsf{O}^{\mathsf{com}}, \mathsf{O}^{\mathsf{corr}}\}}$.*
- *The challenger $\mathcal{C}h$ disables all oracles, all the parties are remain fixed at the position at which they were when $\mathcal{A}$'s output was made, $\mathcal{A}_V$ is removed from the environment, and then $\mathcal{C}h$ checks whether the setting $(\mathsf{pos}_{\mathcal{A}_\mathsf{P}}, d\mathbb{P}, d\mathbb{V})$ returned by the adversary is* valid for GDF*.*
  A setting $(\mathsf{pos}_{\mathcal{A}_\mathsf{P}}, d\mathbb{P}, d\mathbb{V})$ is a valid setting for GDF *if (1) $d(\mathsf{pos}_{\mathcal{A}_\mathsf{P}}, d\mathbb{V}) \geq \mathbb{B}$ and (2) $d(d\mathbb{P}, d\mathbb{V}) \geq \mathbb{B}$.*
- *If $(\mathsf{pos}_{\mathcal{A}_\mathsf{P}}, d\mathbb{P}, d\mathbb{V})$ is not a valid setting for GDF, then the challenger aborts the game and $\mathcal{A}$ loses. Otherwise, the challenger begins the attack phase.*
- *The* attack phase for GDF *is a multi-party execution of the protocol Γ in the presence of an adversary $\mathcal{A}_P$ found at position $\mathsf{pos}_{\mathcal{A}_\mathsf{P}}$. In this, the challenger allows the adversary access to the set $\{\mathtt{init}, \mathtt{send}^{Dum}, \mathsf{O}^{\mathsf{com}}\}$ of oracles.*
- *The phase finishes with the adversary outputting a session identifier sid, i.e., $sid \leftarrow \mathcal{A}_P^{\{\mathtt{init}, \mathtt{send}^{Dum}, \mathsf{O}^{\mathsf{com}}\}}$.*
- *The adversary* wins the GDF game *if the session sid is a verifier-session started during the attack phase, such that $\mathtt{result}(sid) = (\top, d\mathbb{P})$, i.e., $d\mathbb{V}$ accepted the far-away prover $d\mathbb{P}$ during the attack phase.*
- advantage of an adversary $\mathcal{A}$ in the GDF game *is his success probability α.*
- *protocol Π is* GDF-secure *if the advantage of all adversaries $\mathcal{A}$ in winning in an un-aborted generalised distance-fraud game $\mathcal{G}$ is negligible.*

Generalised Mafia Fraud (GMF) In this setting, two adversary parties collaborate to authenticate as an uncorrupted prover located outside of the distance-bound of a designated verifier $d\mathbb{V}$. The goal of the adversary is to make $d\mathbb{V}$ accept the authentication of the prover $d\mathbb{P}$, while $d\mathbb{P}$ is at a distance greater than $\mathbb{B}$ of $d\mathbb{V}$.

We formalise generalised mafia-fraud in Definition 3.

Definition 3. Mafia-fraud (GMF) & Security against GMF. *Let Π be the a DB protocol. A* generalised distance-fraud (GDF) game *$\mathcal{G}$ against the DB protocol Π is split in two phases: the* learning phase *and the* attack phase*.*

- *The* learning phase for GMF *is a multi-party execution of the protocol Γ in the presence of an adversary $\mathcal{A}= (\mathcal{A}_P, \mathcal{A}_V)$ such that the position $\mathsf{pos}_{\mathcal{A}_\mathsf{P}}$ of $\mathcal{A}_P$ and the position $\mathsf{pos}_{\mathcal{A}_\mathsf{V}}$ of $\mathcal{A}_V$ can be arbitrary.*

- *In this phase, the challenger $\mathcal{C}h$ starts by setting up an execution environment and giving access to $\mathcal{A}$ to the set of oracles* $\{\mathsf{O}^{\mathsf{core}}, \mathsf{O}^{\mathsf{com}}, \mathsf{O}^{\mathsf{corr}}\}$.
- *The phase finishes with the adversary returning a designated prover and verifier pair ($d\mathbb{P}, d\mathbb{V}$), and the position of the two adversarially controlled parties $\mathcal{A}_P$ and $\mathcal{A}_V$, i.e.,:* $(\mathsf{pos}_{\mathcal{A}_\mathsf{P}}, \mathsf{pos}_{\mathcal{A}_\mathsf{V}}, d\mathbb{P}, d\mathbb{V}) \leftarrow \mathcal{A}^{\{\mathsf{O}^{\mathsf{core}}, \mathsf{O}^{\mathsf{com}}, \mathsf{O}^{\mathsf{corr}}\}}$.
- *The challenger $\mathcal{C}h$ then disables all oracles, and checks whether the setting defined by the adversary is* valid setting for GMF. *A setting* $(\mathsf{pos}_{\mathcal{A}_\mathsf{P}}, \mathsf{pos}_{\mathcal{A}_\mathsf{V}}, d\mathbb{P}, d\mathbb{V})$ *is a* valid setting for GMF *for GMF if (1) $d(d\mathbb{P}, d\mathbb{V}) \geq \mathbb{B}$, (2) $d\mathbb{P}$ is not marked as corrupted.*
- *If* $(\mathsf{pos}_{\mathcal{A}_\mathsf{P}}, \mathsf{pos}_{\mathcal{A}_\mathsf{V}}, d\mathbb{P}, d\mathbb{V})$ *is not a valid setting for GMF, then the challenger $\mathcal{C}h$ aborts the game and $\mathcal{A}$ loses. Otherwise, the challenger begins the* attack phase.
- *The* attack phase for GMF *is a multi-party execution of the protocol Γ in the presence of an adversary $(\mathcal{A}_P, \mathcal{A}_V)$ found at position* $\mathsf{pos}_{\mathcal{A}_\mathsf{P}}$ *and* $\mathsf{pos}_{\mathcal{A}_\mathsf{V}}$. *In this, the challenger allows the adversary access to the set* $\{\mathtt{init}, \mathtt{send}^{Dum}, \mathsf{O}^{\mathsf{com}}\}$ *of oracles.*
- *The phase finishes with the adversary outputting a session identifier sid, i.e.,* $sid \leftarrow \mathcal{A}_P^{\{\mathtt{init}, \mathtt{send}^{Dum}, \mathsf{O}^{\mathsf{com}}\}}$.
- *The adversary* wins the GMF game *if the session sid is a verifier-session started during the attack phase, such that* $\mathtt{result}(sid) = (\top, d\mathbb{P})$, *i.e., $d\mathbb{V}$ accepted the far-away prover $d\mathbb{P}$ during the attack phase.*
- *The* advantage of an adversary $\mathcal{A}$ in the GMF game *is his success probability β.*
- *The protocol Π is* GMF-secure *if the advantage of all adversaries $\mathcal{A}$ in winning in an un-aborted generalised mafia-fraud game $\mathcal{G}$ is negligible.*

Our Security Notions vs. Existing Ones

1. Our 1-*Weak-Insider* GDF corresponds to the classical distance fraud and distance hijacking that is, an adversary knowing one secret key in "white-box" manner or our attacker calling *join* as insider on one prover, attempts to authenticate from a distance, possibly with honest provers located near the verifier.
2. Our 1-*Outsider* GDF in terms of corruption corresponds to the more niche black-box distance fraud by Avoine et al. in [4]. Yet, our 1-*Outsider* GDF extends Avoine's black-box distance fraud in that 1-*Outsider* GDF allows additional honest provers to be present in the security experiment
3. Our $1-$ *Weak-Insider* GDF is equivalent to the generalised distance-fraud in the so-called BMV model [14].
4. Our *n-Insider* GDF is a completely new property, allowing the multiple provers to be corrupted fully and mount a type of distance fraud as a consequence of that. In Section 5, we show that this is indeed a feasible attack
5. Our GMF is new w.r.t. the fine-grained party-corruption it offers (i.e., *Insider* vs *Outsider*, as well as 1 vs n) , in that traditional mafia-fraud definitions only consider one *Outsider* adversary.
6. Our threat-model of *Strong-Insider* is new, in that this is an attacker who can not only get the authenticating material of a corrupted device (as in

Avoine's white-box) but can choose the authentication material. The latter is stronger than even the attackers by programming pseudorandom-functions in [10], as in that case the choice over authentication material is restricted, whereas an *Strong-Insider* can fully control this choice at the setup phase of a corrupted device.

7. In terms of network manipulation, the classical settings features limited adversaries that are mix of *Amplifier* and *Injector*, with variations in the latter. We consider the all-in-all *Full* as well as the fine-grained ability to send messages not in broadcast but in unicast mode.

5 Validating FlexiDB: Novel Proximity Attacks

We illustrate the applicability and (increased) expressivity of our *FlexiDB* model by exhibiting:
– a new vulnerability on the `EMV-RRP` protocol [22];
– new distance-hijacking attacks on previously proven secure protocols;
– a generic distance-hijacking strategy that enables attacks on most protocols of the literature.

5.1 New (1-*Weak-Insider*, *Full*)-Attack on Payments

In order to show our attacks on relay-protected contactless payments, we first need to present the `EMV-RRP` protocol, which is MasterCard's contactless-payment protocol with relay protection. This protocol is given in Figure 2, below.

High-level Description of `EMV-RRP` Mastercard's `EMV-RRP` (Figure 2 – without the UN msg. 8) is Mastercard's contactless-payment protocol with relay protection. For the latter, MasterCard added to their initial contactless-payment protocol, called PayPass, a special command *ERRD* ("Exchange Relay Resistance Data") which is sent by a reader to a card and explained below, with the rest of the protocol. In PayPass and in `EMV-RRP`, the card possesses a private key $Priv_C$, a symmetric key K_M shared with the bank, a certificate chain $Cert_{Priv_{CA}}(Pub_C)$ for the card's public key Pub_C. The card and the reader generate two nonces n_C and UN, respectively. After some generic setup messages, in `EMV-RRP`, the reader sends an *ERRD* command to the card, which contains the nonce UN. The card answers with nonce n_C. The reader measures the corresponding round trip time. The card also gives an estimation of the time of this exchange called "Timing_Info". The reader compares the two timings, and if the measured time is too large, then the reader suspects a relay attack and stops the communication. Otherwise, the reader requests that the card generates a "cryptogram" (a.k.a. AC). It is a MAC keyed with K_S of data including the ATC, the nonce UN, and the transaction information. The encryption with K_M of the counter ATC of times the card has been used by the card forms a session-key denoted K_S. The card also computes the card's signature on a message including UN, amount, currency, ATC, N_C, denoted "Signed Dynamic Application Data

$(SDAD)$". Finally, before accepting the payment, the reader checks the validity of the signature $SDAD$.

In [31,13], a slightly modified version of `EMV-RRP` is given, to which we refer as *`EMV-RRPv2`*; this is shown on Figure 2 with the UN in place in message 8. The `EMV-RRPv2` protocol differs from `EMV-RRP` only in that in the timed phase, the card adds the reader's nonce UN back into its timed-phase response. This protects against certain distance frauds in `EMV-RRP`, as [31,13] show.

Our Attack on `EMV-RRPv2` `EMV-RRPv2` was symbolically verified in [31,16,20,19] and found secure including against distance-hijacking attacks (in the security models considered by [31,16,20,19]). In fact, we show that `EMV-RRPv2` is in vulnerable to a type of distance hijacking, in the presence of a (1-$WI, Full$) adversary who substitutes messages in a legitimate authentication by an honest prover to succeed in his fraudulent authentication.

Our attack is executed in our generalised distance-fraud (GDF) type setting: a honest prover P and the designated verifier $d\mathbb{V}$ are within distance at most $\mathbb{B}$ of each other, and a (1-*Weak-Insider*, *Full*)-adversary $\mathcal{A}$ and the designated prover $d\mathbb{P}$ are both at a distance greater than $\mathbb{B}$ of $d\mathbb{V}$. We write $\mathsf{pos}_{\mathbb{P}}, \mathsf{pos}_{d\mathbb{V}}, \mathsf{pos}_{\mathcal{A}}$ and $\mathsf{pos}_{d\mathbb{P}}$ to denote their respective positions. Note that $d\mathbb{P}$ is not actually used in this attack, since the insider adversary knows $d\mathbb{P}$'s key and authenticates from a distance on $d\mathbb{P}$'s behalf.

The idea of attack is simple: the one value $\mathcal{A}$ cannot send on his own from a distance is UN in the message *(*UN*,* n_C*, TimingInfo)*. Therefore, $\mathcal{A}$ simply lets P reflect UN, and overwrites every other value sent by P with his own. Concretely, the attack is as follows:

1. During the learning phase, $\mathcal{A}$ registers P by calling `join`(prover,$\mathsf{pos}_{\mathbb{P}}$), $d\mathbb{V}$ by calling `join`(verifier,$\mathsf{pos}_{d\mathbb{V}}$), and $d\mathbb{P}$ by calling `join`(prover,$\mathsf{pos}_{d\mathbb{P}}$). He also calls `enable-broadcast`() oracle to enable full broadcast mode, and returns the setting ($\mathsf{pos}_{\mathcal{A}}$, $d\mathbb{P}, d\mathbb{V}$);
2. During the attack phase, $\mathcal{A}$ calls `init`($\mathsf{P}, d\mathbb{V}$), to start a session *sid* between P and $d\mathbb{V}$;
3. $\mathcal{A}$ uses the `replace` oracle to piggyback all of his messages on P's messages.
   (a) all messages are fully overwritten with $\mathcal{A}$'s own messages (computed with the secret key of $d\mathbb{P}$), except for *(*UN*,* n_C*, TimingInfo)*.
   (b) for this message, $\mathcal{A}$ uses `replace`(P, *sid*,{*bits(*n_C*, TimingInfo)*}, {$d\mathbb{V}$}, {*sid*}, *(*$n_{C\mathcal{A}}$*, TimingInfo*$_{\mathcal{A}}$*)*), where {*bits(*n_C*, TimingInfo)*} denotes the bit-positions corresponding to the values *(*n_C*, TimingInfo)*.
   This oracle call replaces the n_C and *TimingInfo* from P by the ones of $\mathcal{A}$, while not modifying the UN part of the message.
4. $\mathcal{A}$ returns *sid*.

The session *sid* authenticates $d\mathbb{P}$: all authenticating messages in the sessions are computed with the authentication material of $d\mathbb{P}$. Therefore, the prover $d\mathbb{P}$ is accepted by $d\mathbb{V}$, even though $d(\mathsf{pos}_{d\mathbb{P}}, \mathsf{pos}_{d\mathbb{V}}) > \mathbb{B}$ and $d(\mathsf{pos}_{\mathcal{A}}, \mathsf{pos}_{d\mathbb{V}}) > \mathbb{B}$.

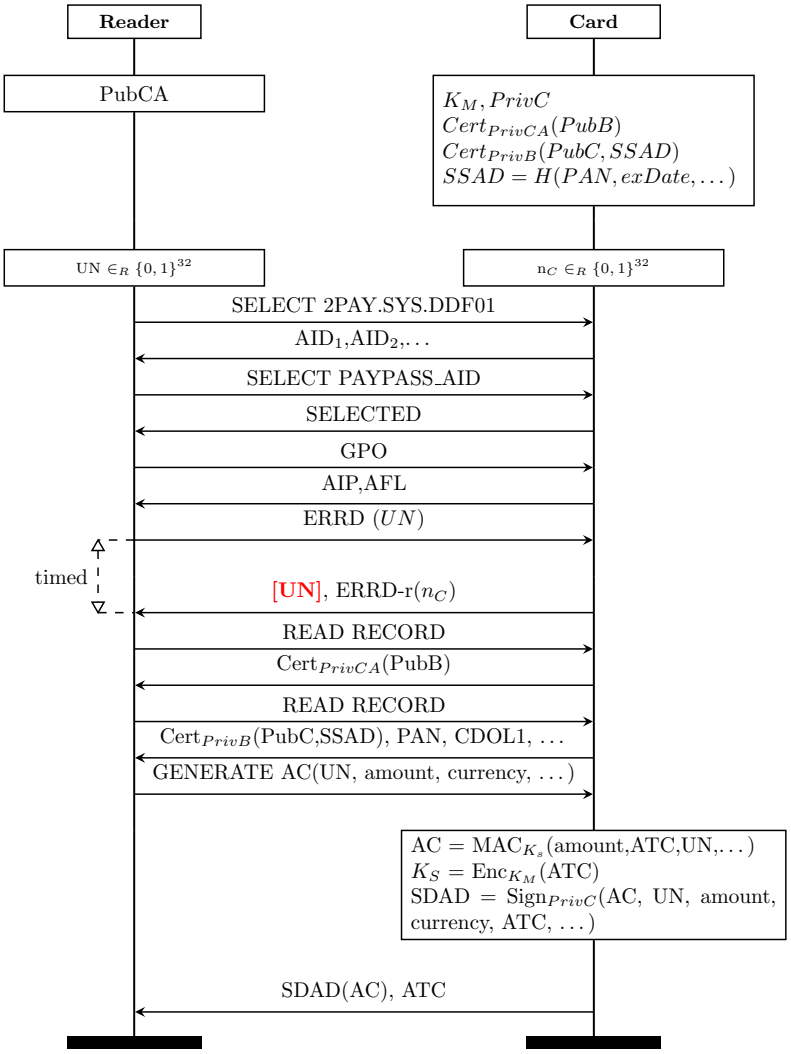

Fig. 2. MasterCard's `EMV-RRP` & `EMV-RRPv2`which is an `EMV-RRP` extension [31,13]; [UN] in msg.8 is only present in `EMV-RRPv2`.

Our Attack on PayBCR In [37], a new version of `EMV-RRP`, called PayBCR, is proposed. An attestation of the proximity-checking performed by the reader is sent to the card-issuing bank, who can further re-verify it. In this case, the transaction constitutes a strong proof that the card was within the range of

the verifier when the purchase was made. Yet, PayBCR is based [10] directly on `EMV-RRP`. And, our GDF against `EMV-RRPv2`, also applies to PayBCR.

Attacks' Significance Firstly, distance frauds would translate into financial loss for the banks. Assume a malicious card paying legitimately in store A. If this card can mount a distance fraud to pay in a far-away store B at the same time, then the card owner can then claim that their card was hacked/cloned, as it appears to be paying in two locations at the same time. This would most likely entail the bank having to reimburse both purchases.

Secondly, the above is even more problematic in the case of the DF we showed on top of PayBCR. Therein, any forgery of proximity-proof by a dishonest card is a forgery of a (hardware-attested) proof accepted by the bank. So, this can not only lead to reimbursement of fraudulent payments, but it can be used as a strong alibi by the card owner to show that he/she was by the payment terminal when they were not.

5.2 New (*n-Weak-Insider*, *Full*)-Attack on 40+ DB Protocols

We now demonstrate another type of generalised distance-frauds that works against against the vast majority of the existing distance-bounding protocols. In particular, it mostly works against 40+, traditional distance-bounding protocols with one-bit challenges and responses, where each round is independent from the previous rounds. To illustrate one such protocol, we chose the DB3 protocol [12].

The DB3 Protocol [12] The DB3 protocol (with its parameter q equal to 2) is a proven-secure protocol [12]. It works as follows. The verifier sends a nonce NV, the prover replies with a nonce NP. Both compute $a = f_x(NP, NV)$, where f_x is a PRF keyed on the shared key x. Then, in n timed rounds, the verifier sends a random bit c_i, expects a response $r_i = a_i \oplus c_i$. Finally, the prover sends $tag = f_x(NP, NV, c)$ (where c is the concatenation of all c_i values). The verifier accepts if the times, r_i and tag are correct. A complete description can be found in [12].

Our attack, to follow, is executed in our GDF setting: n honest provers $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and the designated verifier $d\mathbb{V}$ are within distance at most $\mathbb{B}$ of each other, and a n-WI, *Full* adversary $\mathcal{A}$ and the designated prover $d\mathbb{P}$ are both at a distance greater than $\mathbb{B}$ of $d\mathbb{V}$. We write $\mathsf{pos}_{\mathbb{P}_i}, \mathsf{pos}_{d\mathbb{V}}, \mathsf{pos}_{\mathcal{A}}$ and $\mathsf{pos}_{d\mathbb{P}}$ to denote their respective positions, without loss of generality. Note that $d\mathbb{P}$ is not actually used in this attack, as the insider adversary, knowing his key, authenticates from a distance on his behalf.

Let $R_j^i = (r0_j^i, r1_j^i)$ denote the responses of the prover P_i at round j for the challenge $c_j = 0$ (resp. $c_j = 1$). In our attack, at each round j, the adversary $\mathcal{A}$ selects a prover P_i such that $R_j^i = R_j^{\mathcal{A}}$, and blocks the responses of all provers but P_i.

[10] [37] does not aim for distance-fraud protection, yet if `EMV-RRP` has been analysed against DF and then so should PayBCR. This is even more so since the bank receives and verifies an attested "copy" of the proximity-check.

Our GDF Illustrated on DB3

1. During the learning phase, $\mathcal{A}$ registers P_i by calling $\texttt{join}^{WI}(\mathsf{pos}_{\mathbb{P}_i})$ (for i from 1 to n), $d\mathbb{V}$ by calling $\texttt{join}$(verifier,$\mathsf{pos}_{d\mathbb{V}}$), and $d\mathbb{P}$ by calling $\texttt{join}^{WI}(\mathsf{pos}_{d\mathbb{P}})$. He also calls $\texttt{enable-broadcast}()$ oracle to enable full broadcast mode, and returns the setting ($\mathsf{pos}_{\mathcal{A}}$, $d\mathbb{P}, d\mathbb{V}$);
2. During the attack phase, $\mathcal{A}$ calls $\texttt{init}(\mathsf{P}_i, d\mathbb{V})$ (for i from 1 to n), to start n sessions sid_i between P_i and $d\mathbb{V}$, and records the initial messages NP_i sent by each of the provers;
3. $\mathcal{A}$ selects a random nonce NV, and calls $\texttt{send}^{Dum}(\mathsf{P}_i, sid_i, NV)$ to send NV to the n provers (for i from 1 to n) ;
4. $\mathcal{A}$ calls $\texttt{init}(d\mathbb{V})$ to start a session sid with $d\mathbb{V}$, picks a random NP, and calls $\texttt{send}^{Dum}(d\mathbb{V}, sid, NP)$;
5. $\mathcal{A}$ uses the keys x_{P_i} to compute $a_i = f_{\mathsf{P}_i}(NP_i, NV)$ (for i from 1 to n) ;
6. At each round j, $\mathcal{A}$ selects a prover P_i, such that $R^i_j = R^{\mathcal{A}}_j$. If no such prover exists, the attack aborts.
7. $\mathcal{A}$ calls $\texttt{replace}(\mathsf{P}_z, sid_z, *, \emptyset, \emptyset, \emptyset)$ for $z \neq i$, to block the responses of all provers but P_i. $\mathcal{A}$ stores the corresponding challenge issued by $d\mathbb{V}$ in session sid_i as C_j (denoting the j^{th} bit of a string C);
8. $\mathcal{A}$ blocks the final messages of the provers with $\texttt{replace}(\mathsf{P}_i, sid_i, *, \emptyset, \emptyset, \emptyset)$ (i from 1 to n), and uses $\texttt{send}^{Dum}(d\mathbb{V}, sid, tag_{\mathcal{A}})$, where $tag_{\mathcal{A}} = f_{x_{d\mathbb{P}}}(NP, NV, C)$ to send his own final message;
9. $\mathcal{A}$ returns sid.

The session sid authenticates $d\mathbb{P}$: all authenticating messages in the session are computed with the authentication material of $d\mathbb{P}$. Therefore, the prover $d\mathbb{P}$ is accepted by $d\mathbb{V}$, even though $d(\mathsf{pos}_{d\mathbb{P}}, \mathsf{pos}_{d\mathbb{V}}) > \mathbb{B}$ and $d(\mathsf{pos}_{\mathcal{A}}, \mathsf{pos}_{d\mathbb{V}}) > \mathbb{B}$.

The pair $(r0^i_j, r1^i_j)$ can take 4 different values. At each challenge response round j, the probability to have $R^i_j = R^{\mathcal{A}}_j$, for any prover $d\mathbb{P}_i$, is therefore $\frac{1}{4}$. Hence, the probability that there exists no prover such that $R^i_j = R^{\mathcal{A}}_j$ is $1-(\frac{3}{4})^n$: over k rounds, the success probability of our attack is $(1-(\frac{3}{4})^n)^k$. For a large enough n, for instance $n = k$, the success probability P_S converges to 1.

Applicability of This Attack We described our attack with the parameter $q = 2$ in the DB3 protocol, meaning that the challenges and responses of each round can take 2 possible values. However, when q is greater than 2, our attack still applies, by having the adversary apply more granularity during the challenge response phase. In particular, the selective blocking of responses would be done bitwise, *i.e.*, $\mathcal{A}$ would select a different prover for each bit of the response at each round.

Some DB protocols resist this attack: e.g., those in [27], where the time between to consecutive challenges is randomised. Similarly, we only studied protocols where the response table can be efficiently stored: each response bit only depends on one challenge bit, so that it is sufficient to store the R vector to be able to respond. However, protocols where the challenge is $\mathsf{log}_2(q)$ bits long, and where each challenge bits influence all response bits, make our attack unpractical. For instance, let C be a $\mathsf{log}_2(q)$ bits long challenge issued, and let

$R^C = \mathsf{H}(x, C)$, where H is a cryptographic hash function and x is the secret key of the prover. In this case, there are q possible responses, and they cannot be considered bit by bit since the response bits are not independent. Therefore, the adversary needs to find a prover that has the same q responses as him, and the number of required provers grows exponentially with q.

5.3 More Attacks Using FlexiDB

Due to space constraints, it is only in a series of appendices that we show other attacks, of various types, using different strengths of adversaries on a number of DB protocols. This validates further the fine-grained nature of the threat-model that FlexiDB promotes, showing that considering tailoring the attackers in the ways we prescribed can lead to new vulnerabilities being exhibited. These new attacks are as follows:

1. In Appendix A, we show that a (2-*Weak-Insider*, *Full*)-attack applying to several distance-bounding protocols.
2. In Appendix B, we show that the famous Swiss-Knife protocol [29] is victim of new DF if attacker can fully control the network, i.e., a $(1-$*Strong-Insider*, *Full*$)$ generalised distance fraud applies to the Swiss-Knife protocol.

6 Easycrypt-Mechanised Proofs for `EMV-RRP`

We now discuss our mechanisation of the FlexiDB model given in Section 3 and its GMF security property in the EasyCrypt proof assistant. Based on the resulting formal models, we develop a machine-checked proof, in EasyCrypt, of the MiM-security of EMV's `EMV-RRP` against (*Outsider*,*Full*) adversaries with a slightly restricted `replace` oracle. In particular, we consider an attacker that corrupts cards as an outsider, can amplify and drop messages, but cannot strictly replace or modify messages after they have been sent. We discuss this more precisely in Section 6.5.

Beyond the security proof for EMV's `EMV-RRP` and the necessary cryptographic modelling, this mechanisation in EasyCrypt is the first attempt at capturing —in a formal model of computational security— the physical aspects linked to time and distance measuring in communication protocols. As such, our EasyCrypt models constitute a feasibility study for capturing distance-bounding in EasyCrypt, and carrying out machine-checked computational cryptographic proofs in such physicality-enhanced communication models. We later detail on choices made on modelling of interactive protocols vs. building a feasibility-study for DB proofs. In Section 6.7, we discuss the lessons learned on modelling physical aspects of communication, and potential modelling alternatives that could be usefully explored in further efforts.

6.1 A simplified `EMV-RRP` protocol and security model

We operate over a simplified version of the `EMV-RRP` protocol, displayed in Figure 3. This protocol differs from `EMV-RRP` in that the payment-issuing signature

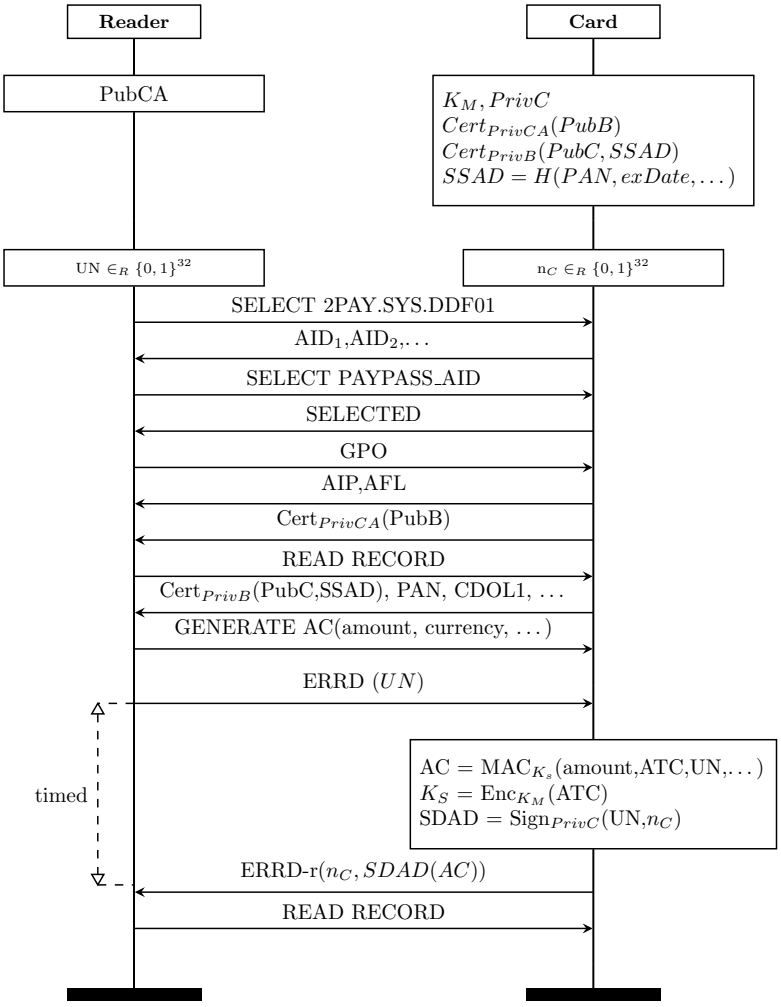

Fig. 3. `EMV-RRP` with a Modified *ERRD* Command.

SDAD issued by the card is sent at the same time as the response to the *ERRD* command, alongside the nonce n_c. The verifier checks the time over the *ERRD* command, as before. Like in `EMV-RRP`, the card is accepted by the reader if the *ERRD* passes the timing check and the *SDAD* signature verifies. We therefore also simplify other aspects of the protocol and model, which we consider to be orthogonal to this goal. These simplifications are made in line with existing mechanised models for distance-bounding. However, we note (and discuss in Sec-

tion 6.7) that—while they seem central to the feasibility of verification in other tools—they could be avoided in EasyCrypt. However, we take care to ensure our model could be—if desired—extended to include more of the protocol details. As we describe below, our protocol already includes a session ID, left abstract and controlled by the adversary, which could be used to carry some or all of the protocol context (considering the rest of the `EMV-RRP` protocol as an adversary against its authentication and distance-bounding component). More precisely, we simplify the following aspects:

- we focus the model and proof on the authentication and distance-bounding component of the `Core`-RRP protocol, noting that our model features an abstract and adversary-controlled session ID, which could extend the proof to the rest of the protocol, considered as an adversary against its authentication and distance-bounding component;
- we consider a single card and a single verifier, to avoid the burden of book-keeping credentials and corruption (which are both well-understood) taking focus away from our goal;
- we consider a weakened model of generalized mafia fraud where the adversary cannot interact with the card during its attack phase, in line with simplifying arguments first made by Chothia et al. [17].

6.2 The EasyCrypt proof assistant

EasyCrypt is an interactive proof assistant designed for analysing cryptographic primitives or protocols in the computational model. Theorem statements proved in EasyCrypt can be interpreted as exact security statements when combined with some (unverified) complexity analysis.

EasyCrypt can be used to prove concrete bounds on the advantage of a black-box reduction, constructed as concrete programmes that make use of abstract, universally-quantified *modules*. The same mechanism can also be used to prove general statements on universally quantified modules (which serve as abstractions), and later instantiate these requiring any assumptions made in the abstract proof to be discharged to concrete values without re-doing the entire proof.

This methodology aligns particularly well with game-based notions of security. The challenger is represented as a module parameterised by a protocol and an adversary—also modules, mediates the interactions between the adversary and the protocol. This is done via *oracles* which are accessed by the adversary as part of an *experiment* (or game). These oracles are usually simple wrappers around the protocol operations, that ensure that only interactions allowed by the threat model can occur, and keeping any state required to decide whether security was broken in a particular execution.

Modules have procedures, which are written in a small imperative probabilistic language, pWhile, which supports standard control-flow (**if** statements and **while** loops), procedure calls, deterministic assignments (denoted with $\leftarrow$) and sampling in discrete distributions (denoted with $\leftarrow_{\$}$). In order to simplify the code presented, and more specifically to simplify error handling, we also make use of an "error-checking assignment" (denoted with $\leftarrow_{\perp}$) that stops execution

and returns a distinguished error symbol $\perp$ if its right-hand side evaluates to $\perp$, and otherwise lets execution carry on as specified. (In code, $v \leftarrow_{\perp} e$ is syntactic sugar for **if** $e = \perp$ **then return** $\perp$ **else** $v \leftarrow e$.)

Procedures within a module can share state, declared as global variables. Such variables are given a type, which we denote using set membership in module specifications. In practice, the initial value of such global variables must be explicitly specified as part of the model. To simplify presentation here, we omit this initialisation. Unless otherwise specified, numeric-type variables are initialised with 0, and global variables that model partial maps are initially everywhere undefined. Variables of other types are always explicitly initialised in the modules discussed here.

6.3 Modelling Environments with Physicalities

As a *general* proof assistant, EasyCrypt does not cater for domain-specific modelling of time, positions, distances, or of systems with such "physicalities" or generalisation thereof. Further, the EasyCrypt semantics are purely sequential. It is therefore not possible to model a ticking, global clock that keeps time during the execution of a protocol.

High-level Choices from FlexiDB We develop a formal framework within which our proof for Core-RRP is carried out. Our formal EasyCrypt framework captures the essential aspects of the FlexiDB model, that is time, space, and asynchronous broadcast communication. Our framework also gives the protocol and adversary certain (controlled) abilities to monitor and act on the physical environment it models. By design, we choose to only enforce simple constraints on the behaviour of clock, positions and communication in the framework. This should support, as and when needed, a layered imposition of additional constraints. Importantly, the correctness and security of mechanisms meant to provide or enforce such additional constraints could also be reasoned about in EasyCrypt.

Concrete Modelling of FlexiDB Our framework takes the form of a single module Env, parameterised by three types (or sets) name, position and message, which respectively capture the names of parties, the set of positions (in particular, we assume a notion of distance d over type position), and the set of messages that will be exchanged. Figure 4 displays this module, whose details we now discuss.

Time is captured as a global variable clock taking values in $\mathbb{R}$. The Environment[11] exposes a getter procedure denoted *get_clock*, and a *controlled* setter procedure denoted *set_clock*, which prevents an update if it sets time back.

The actual **positions** of parties are captured as a partial mapping lmap from names to positions. The Environment allows anyone to retrieve the position

[11] This is the equivalent of the Challenger $\mathcal{C}h$ in FlexiDB.

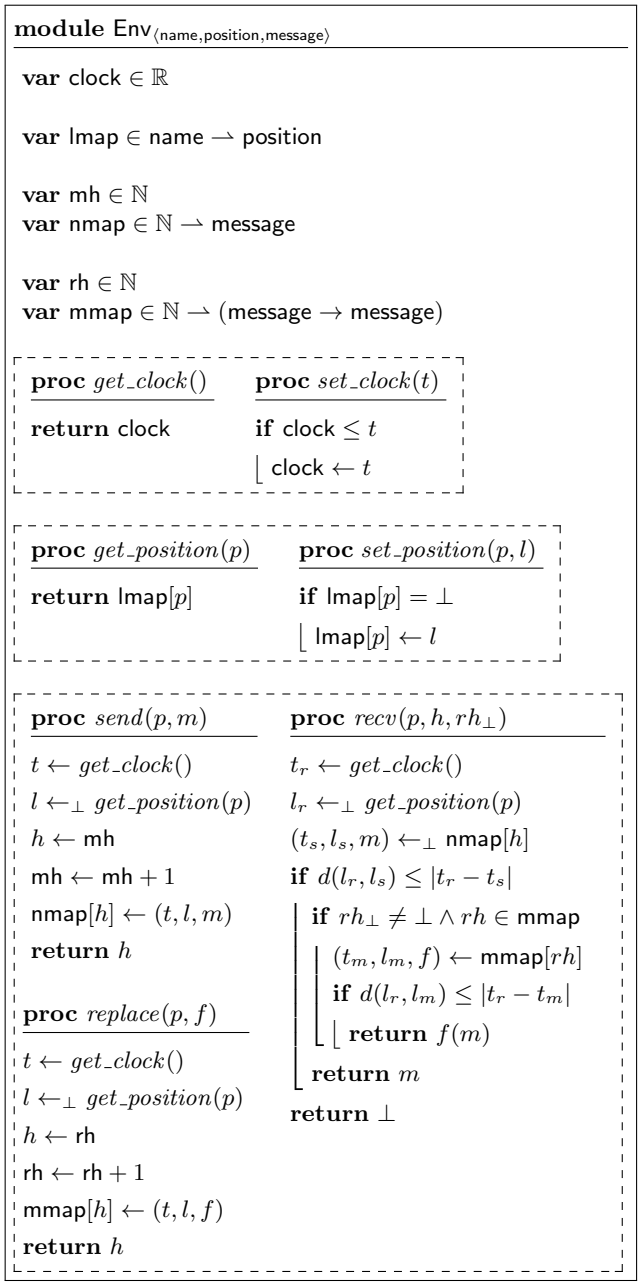

Fig. 4. Environments with physicalities.

of some party given its name (through procedure *get_position*). Further, the position of each party can be initialised once using *set_position*.

Finally, we capture **asynchronous broadcast communication** as a network map nmap that maps *message handles* to messages. Message handles are indices, here in $\mathbb{N}$, that are never reused.

As per FlexiDB, in our EasyCrypt framework, **sending a message** m on behalf of party p proceeds by retrieving the current clock value t and the current position l of p if it exists, and stores t, l and m against an unused message handle h. The message handle is returned to the caller.

We make **replacing messages** possible through a separate *modify map* mmap that maps *replace handles* to message transformations (functions from messages to messages). Replace handles are as before, indices in $\mathbb{N}$, which we ensure are never reused. Our replace oracle is slightly more powerful than FlexiDB's: indeed, the adversary does not need to specify *where* the replacement will take place when registering her intention to replace a message. Instead, this is delegated to network reads.

Reading a message was left underspecified in Section 3 saying that the challenger makes the necessary check. Concretely, in our EasyCrypt framework, to read a message from the network on behalf of party p given the corresponding message handle h, we simply recover the time t_s, position l_s and message m stored against h in the network map, recover the current time t_r and the position l_r of p from the Environment, and check that enough time has elapsed between t_s and t_r to allow the message's propagation from l_s to l_r.[12] If the message handle does not exist, or insufficient time has elapsed, we return a distinguished error symbol. Otherwise, the retrieved message m is returned to the caller.

In addition, an optional replace handle can be provided to the oracle. This is used to find a transformation in the modify map, which is applied to the message m before it is returned to the caller if enough time has elapsed since the transformation was registered. Otherwise, the unmodified message is returned to the caller.

6.4 Modelling Core-RRP in EasyCrypt

Modelling in EasyCrypt the operations in Core-RRP, as expected, rests on calling the Environment oracles to obtain the current time, to send and receive messages. Figure 5 shows the protocol code.

Modelling the verifier The code for the verifier allows an arbitrary number of parallel protocol executions, indexed by session identifiers that allows the adversary to control scheduling, and could be used to capture protocol context in a broader proof. Each session is a simple two-state machine, whose state is stored in a *state map* smap, indexed by session identifiers. Each session can be either *uninitialised* (when smap contains no entry against *sid*, or $\mathsf{smap}[sid] = \bot$), or *initialised* with a time in $\mathbb{R}$ and nonce in $\{0,1\}^\ell$—used to track which challenge was sent, and at what time.

[12] Our model assumes a constant message propagation speed of one "unit" of distance per "unit" of time. This could be generalised.

Each of the protocol oracles proceeds by retrieving the session state from the state map and checking whether the transition it captures applies to the current state (*send_challenge* only applies to an uninitialised session, whereas *recv_response* only applies to an initialised session). It then operates on the given state, and saves the resulting state back to the state map before returning any data needed to produce outputs to be emitted to the network, or used locally.

Apart from its state map, the reader also presents two variables: a local bound $\mathbb{B}$ on the distance it considers as "near", and a public key cpk for which it will receive/check signatures. Both are provided as arguments to a *setup* procedure. In our model, the cpk variable is a single public key, and will be set by the experiment to be the public key of the (single) honest card. In more complex models, it could be replaced with a dynamically-updatable set of keys whose signatures would be accepted (idealising a PKI), or even with a single root certificate if certificate validation were to be modelled.

Modelling the prover In contrast, modelling the prover is a much easier task, since its part of the protocol is entirely stateless. Module P in Figure 5 captures its operations.

The experiment is expected to initialise the prover by calling its *setup* procedure, which generates a fresh keypair for the signing scheme, storing the secret key on the card itself, and outputting the public key back to the experiment (for use, for example, in initialising the reader). The *recv_challenge* oracle captures the prover's step in the Core-RRP protocol: upon receiving a nonce N from the network, the card will sample a nonce N_2, then sign the pair (N, N_2) and output N_2 and the signature to the network.

6.5 Modelling MiM adversaries with physicality

We aim to prove a version of FlexiDB's GMF security for the Core-RRP protocol against a MiM (*Outsider*,*Full*)-adversary as per FlexiDB's hierarchy. To capture the desired (*Outsider*,*Full*)-capabilities (with limited replace functionality), we give our adversary control over: i. the initial position of protocol participants (including the adversary herself); ii. the clock; iii. the scheduling of messages, including the ability to drop or insert broadcast messages; and iv. the scheduling of protocol steps.

This is done, as is usual, through *oracles*. The oracles are displayed in Figure 6. They make use of a partially instantiated environment E, in which the sets of names and messages are defined concretely. The set of names is simply defined as $\mathsf{name} = \{\mathcal{A}, \mathsf{P}, \mathsf{V}\}$. In this paper, the set of messages is assumed to be some set that properly encodes requests and responses (such that we have functions $\mathsf{format\_challenge} \in \mathsf{nonce} \to \mathsf{message}$, and $\mathsf{format\_response} \in \mathsf{nonce} \times \mathsf{signature} \to \mathsf{message}$; and respective partial inverses $\mathsf{parse\_challenge} \in \mathsf{message} \to \mathsf{nonce}_\perp$ and $\mathsf{parse\_response} \in \mathsf{message} \to (\mathsf{nonce} \times \mathsf{signature})_\perp$). This implies additional, but reasonable, assumption on the protocol's wire format. Namely, we assume that formatting is invertible (and indeed fully inverted by the appropriate parsing

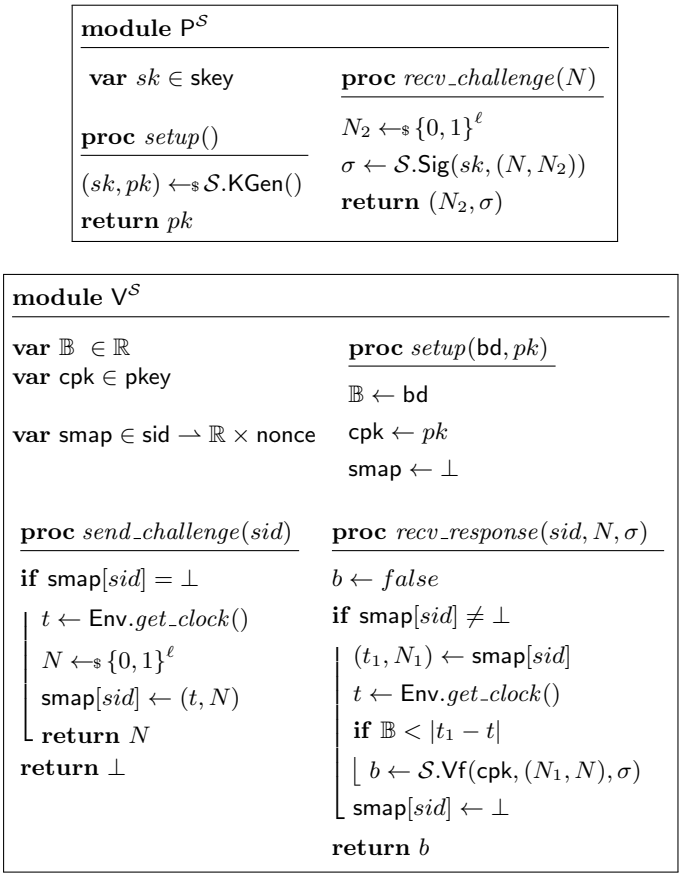

Fig. 5. The `EMV-RRP` protocol based on signature scheme $\mathcal{S}$.

function) and unambiguous (that is, if parsing succeeds, then the message is indeed a formatted value of the right kind).[13]

When triggering protocol operations, the adversary provides as input, where necessary, a session identifier, or a message handle (with an optional replace handle) used to retrieve network input from the Environment. Output from such oracles is most often output to the Environment through *send*, and the corresponding handle given out to the adversary for use in a subsequent oracle query. In the case of the reader's verification step, we choose instead to return the oracle's output directly to the adversary. This helps us capture the fact

[13] Our formal model makes similar assumptions, expressed slightly differently: our type of messages is a sum type, or tagged union, essentially leaving the adversary in charge of parsing and formatting, under the same practical assumptions on the messages' wire format. We note that these assumptions could be relaxed, but this is unrelated to this paper's objectives.

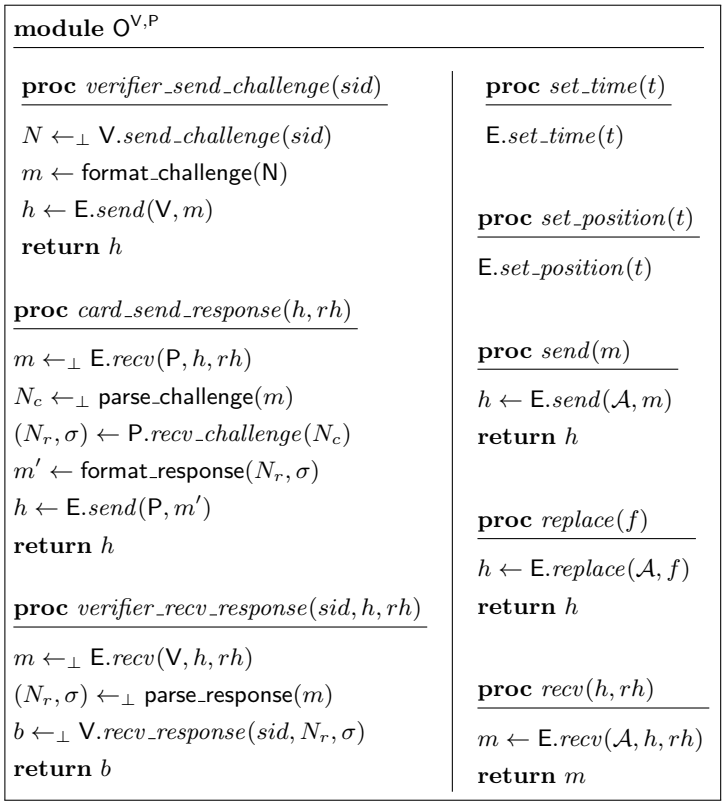

Fig. 6. Adversary oracles for a MiM adversary with control over scheduling and network.

that that output is meant to be used locally by the reader in some overarching application.

6.6 MiM security of `EMV-RRP`.

Figure 7 shows the GMF security property as we formalise it in EasyCrypt. The advantage of an adversary $\mathcal{A}$ in breaking this notion of GMF security is $\mathsf{Adv}^{\mathrm{bsec}}_{\mathcal{A},\mathsf{P},\mathsf{V}} = \Pr\left[\mathbf{Exp}^{bsec}_{\mathsf{P},\mathsf{V},\mathcal{A}}() = \mathsf{true}\right]$.

Theorem 4. ***GMF Security against (*Outsider,Full*)-Adversaries against `Core`-RRP.*** *For any (*Outsider,Full*)-adversary $\mathcal{A}$ that makes at most q queries to its prover_recv_challenge oracle, we construct a forger $\mathcal{B}(\mathcal{A})$ targeting the signature scheme $\mathcal{S}$ and such that:* $\mathsf{Adv}^{\mathrm{bsec}}_{\mathcal{A},\mathsf{P},\mathsf{V}} \leq q/2^{\ell} + \mathsf{Adv}^{\mathrm{euf}}_{\mathcal{B}(\mathcal{A}),\mathcal{S}}$.

Proof. We give a brief sketch in Appendix C. □

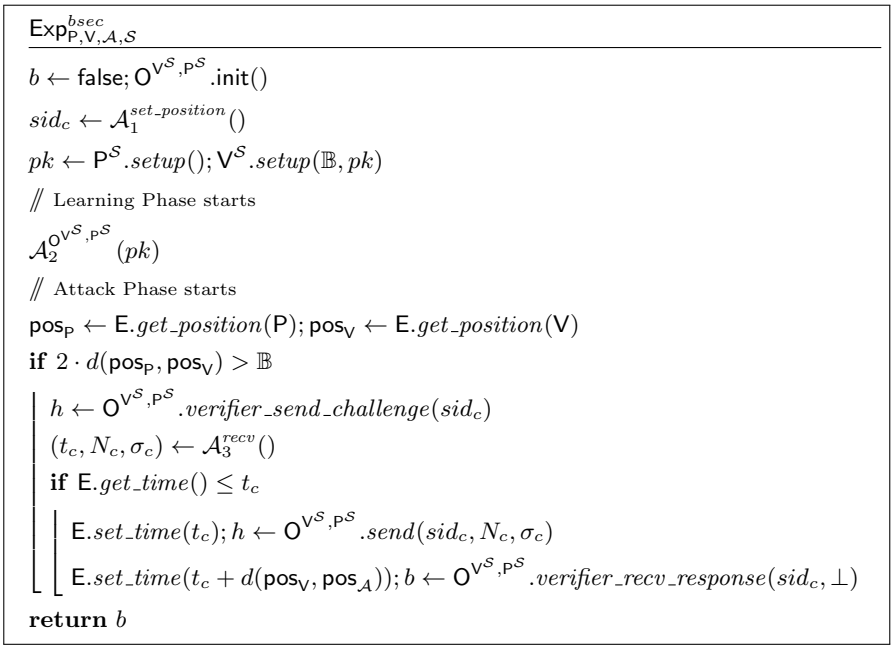

Fig. 7. Security against an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, with a single prover P, a single verifier V, and the set of oracles O defined in Figure 6.

Mechanised proof Our formalisation is publicly available from `https://gitlab.com/fdupress/ec-db`. It is composed of roughly 900 lines of model (including a significant amount of reusable framework code) and 200 lines of proof.

This proof involves a small example, but its definition to proof ratio is encouraging, and seems to indicate that our approach—based on a separate Environment that serves to mediate all interactions between the adversary and protocol participants—does not introduce a significant burden to the proof. In fact, most of the non-cryptographic proof burden is related to the management of verifier sessions. This is in line with previous efforts on formalising stateful protocol, where difficulties arise mainly from non-monotonic state (such as the verifier's session map smap, in our case).

In more complex proofs, the heavy use of maps to model state may also make it useful to manually express and prove framing variants for all oracles—expressing the fact that sections of the state disjoint from those used by a particular query are both irrelevant to the query's semantics, and left untouched by the oracle. Such invariants can be expressed and proved once and for all, and used as needed in combination with more direct proofs. Although we did not rely on this in our proof, our formalisation of the Environment does include statements and proofs to this effect.

6.7 Discussion

We now discuss some alternative modelling choices that could be considered for further extension.

Enforcement vs Assumption of Physical Constraints In our model, we choose to let the Environment *enforce* physical constraints on the propagation of messages and information. A popular alternative when discussing the violation of trust assumptions or other constraints is to explicitly include the advantage of an adversary in violating these constraints whilst still allowing them. We discuss this more in Appendix D.1.

Alignment with the FlexiDB Model The Environment-based framework presented here only captures those details necessary to an *Outsider* adversary with some strong control of the network. Our framework, however, captures all core aspects of FlexiDB, and is developed in such a way as to support extensions to cover all aspects of FlexiDB. We only discuss them briefly (here and in Appendices D.2 and D.3), as we do not yet know whether such extensions could be carried out in a way amenable to reasoning.

Positions are currently static. Implementing a *move* oracle, which updates the position map, is already possible, and would align the framework fully with FlexiDB with respect to adversarial control over participant positions. However, care needs to be taken to prevent teleportation of parties and the information they carry in their state. In practice, it would be sufficient to make *get_location* return $\perp$ or some time-dependent intermediate location for parties that are "in transit".

Our *replace* oracle is slightly more powerful than that speficied by FlexiDB. Indeed, our oracle allows the adversary to decide *where* and *when* the transformation will be applied and have these decisions propagate instantly, although the information contained in the transformation itself still propagates within the given physical constraints. Finer-grained modelling of the *replace* oracle is possible, but would require significantly more complexity in the *read* oracle. In particular, it would require the *read* oracle to *modify* the environment state (instead of just consuming it) to mark a transformation as having already taken effect. We do not add this complexity here, since it is unnecessary in our proof, but note that the ability for the adversary to use the (instant) control channel associated with its *replace* oracle may cause problems in proofs for more complex protocols, or in settings where multiple adversaries in different physical locations collaborate to break protocol security.

Finally, we chose in this paper not to generalise the management of **multiple instances of parties**, and **multiple protocol sessions**. This problem is known to be hard independently of physicalities [6,15], and should first be tackled separately. Our approach here was to capture session management as part of the protocol directly, rather than as part of the model. We were still disciplined in our modelling of session management: although we do not describe

these details here, our formal model separates the—entirely stateless—code for protocol steps from the stateful wrapper that manages the session state. We believe this discipline could be generalised into a framework and folded into the Environment, but note that this may not in fact be beneficial. Dealing with such scenarios in ad hoc ways may currently be the best approach until more tool support is available.

Towards full GMF security As discussed in Section 6.1, we formalise a weakened notion of security—already adopted by Chothia et al. [17], by explicitly preventing the adversary from interacting with the card while the challenge session is ongoing. This restriction is in fact *not* necessary in our model: we can prove systematically in EasyCrypt that the sampling and computation of data that is to be sent through the environment can equivalently be delayed until the sampled value, or the computation's result, affects the adversary's view—that is, either because the adversary queries its *recv* oracle on the corresponding message handle, or queries a final oracle with direct output (such as *verifier_recv_response*).

In the context of Core-RRP, allowing the adversary to interact with provers and verifier during the attack phase would add a case to the reduction, where the challenge nonce from the challenge session collides with one the adversary submitted to the card independently of the reader during the attack phase. Delaying the sampling of the challenge nonce until it becomes visible to the adversary would reduce this case to that of a freshly sampled value being equal to one picked by the adversary earlier—a low probability event.

7 Conclusion

In this paper, we introduce FlexiDB, a formal model for the hierarchical modelling of (existing and new) threats in distance-bounding protocols. FlexiDB is indexed by the different levels of an attacker's ability to manipulate the network and/or have control over active parties in the system. We extend the definitions of distance-fraud and mafia-fraud to incorporate and strengthen existing DB-security formulations, and add new ones. Thus, we find new attacks on most existing DB protocols, including on contactless payments. We also provide a feasibility-study in EasyCrypt by modelling core aspects of FlexiDB. This is the first time DB formal model have been modelled in EasyCrypt, or any cryptographic prover. We complete this study by a mechanised-proof for a version of MasterCard's contactless payment protocol in one of the threat models in FlexiDB. This current proof-of-concept can be used as basis for future work aiming to fully formalise DB, FlexiDB and contactless payments in EasyCrypt.

References

1. Hadi Ahmadi and Reihaneh Safavi-Naini. Secure distance bounding verification using physical-channel properties. *CoRR*, abs/1303.0346, 2013.
2. G Avoine, MA Bingol, Ioana Boureanu, S Capkun, G Hancke, S Kardas, CH Kim, C Lauradoux, B Martin, J Munilla, et al. Security of distance-bounding: A survey. *ACM Computing Surveys*, 2018.
3. G. Avoine, X. Bultel, S. Gambs, D. Gérault, P. Lafourcade, C. Onete, and J. Robert. A terrorist-fraud resistant and extractor-free anonymous distance-bounding protocol. In *Proc. of ASIA CCS '17*, pages 800–814. ACM, 2017.
4. Gildas Avoine, Muhammed Ali Bingol, Suleyman Karda, Cedric Lauradoux, and Benjamin Martin. A formal framework for analyzing RFID distance bounding protocols. In *Journal of Computer Security - Special Issue on RFID System Security, 2010*, 2010.
5. Gildas Avoine and Aslan Tchamkerten. An efficient distance bounding RFID authentication protocol: Balancing false-acceptance rate and memory requirement. In *Information Security, 12th International Conference, ISC 2009, Pisa, Italy, September 7-9, 2009. Proceedings*, pages 250–261, 2009.
6. Gilles Barthe, Juan Manuel Crespo, Yassine Lakhnech, and Benedikt Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 689–718, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
7. David Basin, Srdjan Capkun, Patrick Schaller, and Benedikt Schmidt. Formal reasoning about physical properties of security protocols. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):1–28, 2011.
8. Ahmed Benfarah, Benoit Miscopein, Jean-Marie Gorce, Cédric Lauradoux, and Bernard Roux. Distance bounding protocols on TH-UWB radios. In *Proceedings of the Global Communications Conference, 2010. GLOBECOM 2010, 6-10 December 2010, Miami, Florida, USA*, pages 1–6, 2010.
9. Bruno Blanchet. Security protocol verification: Symbolic and computational models. In *Principles of Security and Trust*, pages 3–29, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
10. I. Boureanu, A. Mitrokotsa, and S. Vaudenay. On the pseudorandom function assumption in (Secure) distance-bounding protocols. In *Progress in Cryptology – LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 100–120. Springer Verlag, 2012.
11. I. Boureanu, A. Mitrokotsa, and S. Vaudenay. Practical and Provably Secure Distance-Bounding. *Journal of Computer Security*, 23(2):229–257, 2015.
12. I. Boureanu and S. Vaudenay. Optimal proximity proofs. In *Proc. of Inscrypt*, pages 170–190. Springer, 2015.
13. Ioana Boureanu and Anda Anda. Another look at relay and distance-based attacks in contactless payments. Cryptology ePrint Archive, Report 2018/402, 2018. `https://eprint.iacr.org/2018/402`.
14. Ioana Boureanu, Aikaterini Mitrokotsa, and Serge Vaudenay. Practical and provably secure distance-bounding. In Yvo Desmedt, editor, *ISC 2013*, Cham, 2015. Springer.
15. Ran Canetti, Alley Stoughton, and Mayank Varia. Easyuc: Using easycrypt to mechanize proofs of universally composable security. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 167–183. IEEE, 2019.

16. Tom Chothia, Joeri de Ruiter, and Ben Smyth. Modelling and analysis of a hierarchy of distance bounding attacks. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 1563–1580. USENIX Association, 2018.
17. Tom Chothia, Flavio D. Garcia, Joeri de Ruiter, Jordi van den Breekel, and Matthew Thompson. Relay cost bounding for contactless EMV payments. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2015.
18. Jolyon Clulow, Gerhard P. Hancke, Markus G. Kuhn, and Tyler Moore. So near and yet so far: Distance-bounding attacks in wireless networks. In Levente Buttyán, Virgil D. Gligor, and Dirk Westhoff, editors, *Security and Privacy in Ad-Hoc and Sensor Networks, Third European Workshop, ESAS 2006, Hamburg, Germany, September 20-21, 2006, Revised Selected Papers*, volume 4357 of *Lecture Notes in Computer Science*, pages 83–97. Springer, 2006.
19. Alexandre Debant and Stéphanie Delaune. Symbolic verification of distance bounding protocols. Research report, Univ Rennes, CNRS, IRISA, France, February 2019.
20. Alexandre Debant, Stéphanie Delaune, and Cyrille Wiedling. Proving physical proximity using symbolic models. Research report, Univ Rennes, CNRS, IRISA, France, February 2018.
21. Alexandre Debant, Stéphanie Delaune, and Cyrille Wiedling. Symbolic analysis of terrorist fraud resistance. In *European Symposium on Research in Computer Security*, pages 383–403. Springer, 2019.
22. EMVCo. Book C-2 kernel 2 specification v2.7. EMV contactless specifications for payment system. `www.emvco.com/wp-content/plugins/pmpro-customizations/oy-getfile.php?u=/wp-content/uploads/documents/C-7_Kernel_7_V_2_7_Final.pdf`, Feb, 2018.
23. R. Entezari, H. Bahramgiri, and M. Tajamolian. A mafia and distance fraud high-resistance rfid distance bounding protocol. In *ISCISC*, pages 67–72, 2014.
24. Masoumeh Safkhani Fatemeh Baghernejad, Nasour Bagheri. Security analysis of the distance bounding protocol proposed by jannati and falahati. *Electrical and Computer Engineering Innovations*, 2(2):85–92, 2014.
25. Ali Özhan Gürel, Atakan Arslan, and Mete Akgün. Non-uniform stepping approach to rfid distance bounding problem. In *Proceedings of the 5th International Workshop on Data Privacy Management, and 3rd International Conference on Autonomous Spontaneous Security*, DPM'10/SETOP'10, pages 64–78, Berlin, Heidelberg, 2011. Springer-Verlag.
26. Gerhard P. Hancke and Markus G. Kuhn. An RFID distance bounding protocol. In *Proceedings of SecureComm 2005*, pages 67–73. IEEE, 2005.
27. Handan Kilinç and Serge Vaudenay. Optimal proximity proofs revisited. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 478–494. Springer, 2015.
28. Chong Hee Kim and Gildas Avoine. Rfid distance bounding protocol with mixed challenges to prevent relay attacks. In *Proceedings of the 8th International Conference on Cryptology and Network Security*, CANS '09, pages 119–133, Berlin, Heidelberg, 2009. Springer-Verlag.

29. Chong Hee Kim, Gildas Avoine, François Koeune, François-Xavier Standaert, and Olivier Pereira. The swiss-knife RFID distance bounding protocol. In *Information Security and Cryptology (ICISC) 2008*, LNCS, pages 98–115. Springer-Verlag, 2008.
30. Sangho Lee, Jin Seok Kim, Sung Je Hong, and Jong Kim. Distance bounding with delayed responses. *IEEE Communications Letters*, 16(9):1478–1481, 2012.
31. Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Distance-bounding protocols: Verification without time and location. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 549–566, 2018.
32. Sjouke Mauw, Zach Smith, Jorge Toro-Pozo, and Rolando Trujillo-Rasua. Post-collusion security and distance bounding. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 941–958. ACM, 2019.
33. Aikaterini Mitrokotsa, Cristina Onete, and Serge Vaudenay. Mafia fraud attack against the rč distance-bounding protocol. In *2012 IEEE International Conference on RFID-Technologies and Applications, RFID-TA 2012, Nice, France, November 5-7, 2012*, pages 74–79, 2012.
34. Jorge Munilla and Alberto Peinado. Distance bounding protocols for rfid enhanced by using void-challenges and analysis in noisy channels. *Wirel. Commun. Mob. Comput.*, 8(9):1227–1232, November 2008.
35. Christina Pöpper, Nils Ole Tippenhauer, Boris Danev, and Srdjan Capkun. Investigation of signal and message manipulations on the wireless channel. In *European Symposium on Research in Computer Security*, pages 40–59. Springer, 2011.
36. Jorge Luis Toro Pozo. *Computational and symbolic analysis of distance-bounding protocols*. PhD thesis, University of Luxembourg, Luxembourg City, Luxembourg, 2019.
37. Liqun Chen Tom Chothia, Ioana Boureanu. Making contactless emv payments robust against rogue readers colluding with relay attackers. In *the 23rd International Conference on Financial Cryptography and Data Security (Financial Crypto 2019)*, 2019, to appear.
38. Rolando Trujillo-Rasua, Benjamin Martin, and Gildas Avoine. The poulidor distance-bounding protocol. In *Proceedings of the 6th International Conference on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec'10, pages 239–257, Berlin, Heidelberg, 2010. Springer-Verlag.
39. Rolando Trujillo-Rasua, Benjamin Martin, and Gildas Avoine. Distance-bounding facing both mafia and distance frauds: Technical report. *CoRR*, abs/1405.5704, 2014.
40. Jan van Leeuwen and Jiří Wiedermann. The turing machine paradigm in contemporary computing. In Björn Engquist and Wilfried Schmid, editors, *Mathematics Unlimited — 2001 and Beyond*, pages 1139–1155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

A (2-*Weak-Insider Full*)-GDF against 13+ DB Protocols

The distance-fraud security of distance-bounding protocols is usually proven with regards to one dishonest prover knowing his own secret key. We now show

that our model FlexiDB allowing us to capture an adversary knowing not one but two secret keys, leads to new attacks on protocols that were proven secure in classical models. Concretely, the attacks rely on a weak insider controlling two provers and a total adversary w.r.t. communications, written $\mathcal{A}_{2\text{-}WI,Full}$.

To illustrate an example of these new vulnerabilities opened by FlexiDB, we present an attack on the proven-secure DB3 distance-bounding protocol [12], which we recalled in Section 5.2.

The DB3 Protocol [12] & Its Security The DB3 protocol (with its main parameter q equal to 2) is a proven-secure protocol [12]. It is a DB protocol that works as follows. The verifier sends a nonce NV, the prover replies with a nonce NP. Both compute $a = f_x(NP, NV)$, where f_x is a PRF keyed on the shared key x. Then, in n timed rounds, the verifier sends a random bit c_i, expects a response $r_i = a_i \oplus c_i$. Finally, the prover sends $tag = f_x(NP, NV, c)$ (where c is the concatenation of all c_i values). The verifier accepts if the times, r_i and tag are correct. A complete description can be found in [12].

Programmable PRF. In DB3, as in many distance bounding protocols, the response to the challenge c_i is computed as a function of a and c_i, where a is the output of a PRF f for some initially-exchanged nonces. Our attack assumes that the PRF used in the protocol is programmed, as defined in [10]. Specifically, the PRF returns a constant value R when one of its inputs has a certain form.

Let f be the PRF specified in DB3, f_z denote an instance of f keyed with a key z, and R be a constant. Let pf be the programmed version of f, such that:

$$\mathsf{pf}_z(NP, NV) = \begin{cases} R \text{ if } NP = g(z) \\ R \text{ if } NV = h(z) \\ f_z(NP, NV) \text{ otherwise,} \end{cases}$$

where g and h are functions from $\{0,1\}^{|z|}$ to $\{0,1\}^{|nonce|}$. For clarity, we use $g(z) = h(z) = z$. Therefore, for two different secret keys x_{P} and $x_{\mathcal{A}}$, we have (1) $pf_{x_{\mathsf{P}}}(NP, x_{\mathsf{P}}) = pf_{x_{\mathcal{A}}}(x_{\mathcal{A}}, NV) = R$. Our attack exploits this equality.

A.1 Our (2-*Weak-Insider Full*)-GDF against DB3 [12]

This attack is executed in our GDF setting: a honest prover P and the designated verifier $d\mathbb{V}$ are within distance at most $\mathbb{B}$ of each other, and an $\mathcal{A}_{2\text{-}WI,Full}$ adversary $\mathcal{A}$ and the designated prover $d\mathbb{P}$ are both at a distance greater than $\mathbb{B}$ of $d\mathbb{V}$. We write $\mathsf{pos}_{\mathbb{P}}, \mathsf{pos}_{d\mathbb{V}}, \mathsf{pos}_{\mathcal{A}}$ and $\mathsf{pos}_{d\mathbb{P}}$ to denote their respective positions. Note that $d\mathbb{P}$ is not actually active in this attack, as the insider adversary, knowing $d\mathbb{P}$'s key, authenticates from a distance on his behalf.

The idea of our attack is as follows:
– $\mathcal{A}$ injects nonces such that equality (1) above holds;
– therefore, the response vector of P matches the response vector of $\mathcal{A}$, and $\mathcal{A}$ does not need to run the timed phase of the protocol himself.

In the next, let x_{P} (resp. $x_{d\mathbb{P}}$) denote the secret keys of P and $d\mathbb{P}$. Concretely, the attack goes as follows:

1. During the learning phase, $\mathcal{A}$ registers P by calling `join`(prover,$\mathsf{pos}_{\mathbb{P}}$), $d\mathbb{V}$ by calling `join`(verifier,$\mathsf{pos}_{d\mathbb{V}}$), and $d\mathbb{P}$ by calling $\texttt{join}^{WI}(\mathsf{pos}_{d\mathbb{P}})$. He also calls the `enable-broadcast`() oracle to enable full broadcast mode, and returns the setting ($\mathsf{pos}_{\mathcal{A}}$, $d\mathbb{P}, d\mathbb{V}$);
2. During the attack phase, $\mathcal{A}$ calls `init`($\mathsf{P}, d\mathbb{V}$), to start a session sid between P and $d\mathbb{V}$;
3. $\mathcal{A}$ calls `replace`($\mathsf{P}, sid, *, \{d\mathbb{V}\}, \{sid\}, x_{d\mathbb{P}}$). This replaces the message NP from $d\mathbb{P}$ with the secret key of $d\mathbb{P}$.
4. $\mathcal{A}$ calls `replace`($d\mathbb{V}, sid, *, x_{\mathsf{P}}, \{\mathsf{P}\}, \{sid\}$). This replaces the message NV from $d\mathbb{V}$ with the secret key of P. Yet, $\mathcal{A}$ receives the unmodified message NV. At this stage, we have $a_{\mathsf{P}} = a_{\mathcal{A}} = R$.
5. During the challenge response phase of the protocol, $\mathcal{A}$ does not interact with the parties, but records the challenges c issued by $d\mathbb{V}$;
6. $\mathcal{A}$ calls `replace`($\mathsf{P}, sid, *, \{d\mathbb{V}\}, \{sid\}, tag_{\mathcal{A}}$), where $tag_{\mathcal{A}} = f_{x_{d\mathbb{P}}}(x_{d\mathbb{P}}, NV, c)$, to replace the final message of P by his own.
7. $\mathcal{A}$ returns sid.

The session sid authenticates $d\mathbb{P}$: all authenticating messages in the session are computed with the authentication material of $d\mathbb{P}$. Therefore, the prover $d\mathbb{P}$ is accepted by $d\mathbb{V}$, even though $d(\mathsf{pos}_{d\mathbb{P}}, \mathsf{pos}_{d\mathbb{V}}) > \mathbb{B}$ and $d(\mathsf{pos}_{\mathcal{A}}, \mathsf{pos}_{d\mathbb{V}}) > \mathbb{B}$.

A.2 (2-*Weak-Insider*, *Full*)-Attacks against Protocols Other than DB3

The same attack that we showed in Subsection A on top of DB3 [12] actually applies to other distance-bounding protocols. A non-exhaustive list of which is given in Table 4.

In the DB protocols in Table 4, the timed-phase response function always uses a bitstring a which, in turn, is the output a PRF on two nonces used in the initialisation phase. However, compared with DB3, some other details may differ. Columns 2 and 3 of Table 4 capture such differences: column 2 indicates whether NP is sent before NV; column 3 indicates whether messages are sent after the timed phase.

For instance, for the protocols where V sends his nonce before the prover's, steps 2 and 3 of the attack against DB3 would be inverted. For the protocols where no messages are sent during after the end of the timed phase, step 5 is not executed. Finally, in the protocol in [5], an additional value v_0, derived from a, is sent by the prover before the timed phase: $\mathcal{A}$ can either send it, or let the close-by prover send it.

B New (1-*Strong-Insider*,Full)-Attack on DB Protocols

We consider adversaries that can chose their secret keys. Our attack targets protocols designed to be terrorist-fraud resistant, in which the two responses $r0_j, r1_j$ at round j are such that $r0_j \oplus r1_j = x_j$, where x is the secret key of the prover. These protocols often have a structure similar to the Swiss-Knife (SK) protocol [29]: we present our attack on SK, noting that it applies to other protocols of the same family.

Protocol	NP first	Final message
Kim and Avoine [28]	✗	✗
Benfarah *et al.* [8] (both versions)	✗	✗
TMA [39]	✗	✗
Hancke and Kuhn [26]	✓	✗
Munilla et Peinado [34]	✓	✓
Avoine et Tchamkerten [5]	✓	✗
Poulidor [38]	✓	✗
NUS [25]	✓	✓
Lee *et al.* [30]	✓	✗
LPDB [33]	✓	✗
EBT [23]	✓	✗
Baghernejad *et al.* [24]	✓	✗

Table 4. Certain DB Protocols Vulnerable to Generalised Distance Fraud via Programmable PRF, in $FlexiDB$.

The Swiss-Knife Protocol [29] & Its Security In the SK protocol, the verifier sends a nonce NA, and receive a nonce NB in return. Both P and V compute $a = f_x(CB, NB)$ (where CB is a constant, and f is a PRF. The response at round j is computed as a_j if $c_j = 0$, and $a_j \oplus x_j$ if $c_j = 1$. In the end, P sends $T_B = f_x(c, ID, NA, NB)$, where c is the concatenation of all the challenges and ID is the identifier of P. The verifier replies with $f_x(NB)$.

Our Attack on the Swiss-Knife Protocol In our attack, $\mathcal{A}$ picks a key $x_{d\mathbb{P}} = 0$. Therefore, for all rounds, it holds that $r0_j = r1_j = a_j$. Since the response is independent of the challenge, $\mathcal{A}$ can send it before V issues the challenge, so that it arrives in time.

This attack is executed in our GDF setting: a $\mathcal{A}_{1\text{-}SI,Full}$ adversary $\mathcal{A}$ and the designated prover $d\mathbb{P}$ are both at a distance greater than $\mathbb{B}$ of $d\mathbb{V}$. As long as these conditions are satisfied, their exact position does not change the validity of our attack. We therefore write $\mathsf{pos}_{\mathcal{A}}, \mathsf{pos}_{d\mathbb{P}}$ and $\mathsf{pos}_{d\mathbb{V}}$ to denote their respective positions, without loss of generality. Note that $d\mathbb{P}$ is not actually used in this attack, as the insider adversary, knowing his key, authenticates from a distance on his behalf.

Concretely, the attack goes as follows:

1. During the learning phase, $\mathcal{A}$ registers $d\mathbb{V}$ by calling `join`(verifier,$\mathsf{pos}_{d\mathbb{V}}$), and $d\mathbb{P}$ by calling $\texttt{join}^{SI}(d\mathbb{P}, \mathsf{pos}_{d\mathbb{P}})$. He also calls `enable-broadcast`() oracle to enable full broadcast mode, and returns the setting ($\mathsf{pos}_{\mathcal{A}}$, $d\mathbb{P}, d\mathbb{V}$);
2. During the attack phase, $\mathcal{A}$ calls $\texttt{init}(d\mathbb{V})$ to start a session sid with $d\mathbb{V}$, receives NA, sends a random nonce NB with $\texttt{send}^{Dum}(d\mathbb{V}, sid, NB)$ and computes $a = f_{x_{d\mathbb{P}}}(CB, NB)$;
3. At each round j, $\mathcal{A}$ uses $\texttt{send}^{Dum}(d\mathbb{V}, sid, a_j)$ in advance;
4. $\mathcal{A}$ uses $\texttt{send}(d\mathbb{V}, sid, T_B)$, where $T_B = f_{x_{d\mathbb{P}}}(c, d\mathbb{P}, NA, NB)$.
5. $\mathcal{A}$ returns sid.

The session *sid* authenticates $d\mathbb{P}$: all authenticating messages in the session are computed with the authentication material of $d\mathbb{P}$. Therefore, the prover $d\mathbb{P}$ is accepted by $d\mathbb{V}$, even though $d(\mathsf{pos}_{d\mathbb{P}}, \mathsf{pos}_{d\mathbb{V}}) > \mathbb{B}$ and $d(\mathsf{pos}_{\mathcal{A}}, \mathsf{pos}_{d\mathbb{V}}) > \mathbb{B}$.

Counteraction & Applicability of Our Attack on SK While the Swiss-Knife protocol, and other similar ones, are vulnerable to this attack due to the mechanism introduced to counter terrorist-fraud (the presence of the key in the response function), it is noteworthy that a fix can be applied. For instance, in [3], a random bitmask m, chosen by the verifier at each session, is applied. The response to challenge $c_i = 1$ becomes $a_i \oplus x_i \oplus m_i$, while the response to the challenge zero remains a_i. Therefore, fixing the secret key to only permits to send responses in advance for half the rounds on average (the ones where $m_i = 0$).

The attack presented above can be seen as a *destructive* attack, as the adversary implicitly leaks his secret key to potential eavesdroppers by executing it. Nonetheless, it remains relevant in settings where the adversary has no rationale interest in protecting his credentials, *i.e.*, the gain from executing the attack is greater than the loss incurred by leaking his secret key.

C Proof Sketch for Theorem 4

Proof. The proof is formalised in EasyCrypt. At its core, the proof relies on refactoring the prover and verifier as adversaries against the signature scheme, and folding them into the adversary and oracle code. The resulting construction forms the core of our reduction $\mathcal{B}$. It is then easy to show that any response accepted by the verifier that did not come from the prover can be used to produce a valid forgery, while also proving that any response that did involve the prover must have been received by the prover after time $tc + 2 \cdot d(\mathsf{pos}_{\mathsf{V}}, \mathsf{pos}_{\mathsf{P}})$, or reused a challenge nonce, which occurs with probability at most $q/2^{\ell}$. □

D Discussion on Easycrypt-Mechanised Proofs for EMV-RRP

This Appendix includes some more tangential discussions that may be of interest to the reader.

D.1 Enforcement vs Assumption of Physical Constraints

In our model, we choose to let the Environment *enforce* physical constraints on the propagation of messages and information. Another popular alternative when discussing the violation of trust assumptions or other constraints is to explicitly include the advantage of an adversary in violating these constraints whilst still allowing them.

In this small proof of concept, enforcement makes the most sense, for two reasons: i. It allowed us to convince ourselves early on in the formalisation that all

physical properties we wished to rely on in proofs were accurately captured; ii. It allowed us to directly use the constraints as invariants on the state in proofs. For example, we knew that, at any point in any execution, it was always the case that a message read had already been sent. Further, one could argue that physical constraints are in fact being enforced by the real-world, and violating them is not simply a "cheating" behaviour.

However, it is worth considering that some attacks on DB protocols rely on the adversary's ability to break abstractions, inferring information from partial signals, and reacting before an honest party would have fully "received" the information [18]. Capturing this information as advantage terms would make security claims safer by keeping them explicit, and would also support a compositional analysis of lower-level mechanisms aimed at reducing the probability of such attacks succeeding.

We suspect that any reduction in a model with physical constraints as an explicit assumption would start by a transition to an enforcement model with the probability of the physical assumption being broken appearing as a simple term in the advantage. As such, the "enforcement-style" proof would in fact be a part of the "assumption-style" proof itself.

D.2 Modelling Multi-Location Adversaries

We do not capture, in our model, adversaries found at multiple physical positions. EasyCrypt supports proving properties with multiple abstract modules—in fact, our proof keeps the signature scheme $\mathcal{S}$ entirely abstract throughout. However, EasyCrypt currently only handles multiple abstract modules by assuming and enforcing that they do not share state. Properly treating colluding adversaries in different physical positions will require some care to ensure that the various adversarial components do not share *state* but can still share *information*—within the constraints placed on information propagation by the Environment.

D.3 Corrupted Participants and Control Messages

Although we do not model adversaries that can corrupt otherwise honest participants, future developments in such models will need to take care of the fact that *control* messages used to control corruption or release a corrupted party's state to the adversary must be passed through the environment in order to avoid any problems with the teleportation of information. Before tackling models that require more extensive use of control messages, it may be worth extending the environment-based framework to capture control messages in a separate queue: as noted in the context of UC, the ability to easily distinguish between control and protocol messages, and to apply different processing to them, is often a key ingredient in complex proofs.