Security Analysis and Implementation of Relay-Resistant Contactless Payments
Ioana Boureanu, Tom Chothia, Alexandre Debant, Stéphanie Delaune

To cite this version:
Ioana Boureanu, Tom Chothia, Alexandre Debant, Stéphanie Delaune. Security Analysis and Implementation of Relay-Resistant Contactless Payments. CCS, Nov 2020, Virtual Conference, United States. hal-02917076

HAL Id: hal-02917076
https://inria.hal.science/hal-02917076
Submitted on 20 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Security Analysis and Implementation of Relay-Resistant Contactless Payments

Ioana Boureanu
i.boureanu@surrey.ac.uk
University of Surrey, SCCS, UK

Alexandre Debant
alexandre.debant@irisa.fr
Univ Rennes, CNRS, IRISA, France

Tom Chothia
T.P.Chothia@cs.bham.ac.uk
University of Birmingham, UK

Stéphanie Delaune
stephanie.delaune@irisa.fr
Univ Rennes, CNRS, IRISA, France

ABSTRACT
Contactless systems, such as the EMV (Europay, Mastercard and Visa) payment protocol, are vulnerable to relay attacks. The typical countermeasure to this relies on distance bounding protocols, in which a reader estimates an upper bound on its physical distance from a card by doing round-trip time (RTT) measurements. However, these protocols are trivially broken in the presence of rogue readers. At Financial Crypto 2019, we proposed two novel EMV-based relay-resistant protocols: they integrate distance-bounding with the use of hardware roots of trust (HWRoT) in such a way that correct RTT-measurements can no longer be bypassed.

Our contributions are threefold: first, we design a calculus to model this advanced type of distance-bounding protocols integrated with HWRoT; as an additional novelty, our calculus is also the first to allow for mobility of cards and readers during a proximity-checking phase. Second, to make it possible to analyse these protocols via more standard mechanisms and tools, we consider a 2018 characterisation of distance-bounding security that does away with physical aspects and relies only on the causality of events; we cast it in our richer calculus and extend its theoretical guarantees to our more expressive models (with mobility, potentially rogue readers, and HWRoT). Due to this extension, we can carry out the security analysis in the standard protocol verification tool ProVerif. Third, we provide the first implementation of Mastercard’s relay-resistant EMV protocol PayPass-RRP as well as one of its 2019 extension with HWRoT called PayBCR. We evaluate their efficiency and their robustness to relay attacks, in presence of both honest and rogue readers. Our experiments are the first to show that Mastercard’s PayPass-RRP and its HWRoT-based extension PayBCR are both practical in preventing relay attacks of the magnitude shown thus-far in EMV.

1 INTRODUCTION
Contactless payments are now globally adopted and in 2019 alone, “50 countries have seen more than 10% increase in tap-to-pay vs. PIN-based transactions”, and “in Europe, more than two-thirds of face-to-face Visa transactions occur contactlessly”. As such, the importance of fast and secure contactless payments cannot be understated. This paper focuses on the development of a formal approach to security analysis of new contactless payments, their integration and implementation as per the EMV (Europay Mastercard Visa) standard as well as their robustness and efficiency testing.

One of the main security concerns in contactless payments is that of relay attacks. In these, a man-in-the-middle (MiM) is interposed between an EMV reader and an honest EMV card which is out of the range of the reader. The adversary captures messages honestly generated by the card and the reader, and simply forwards them back and forth between the two. In such a way, the MiM makes it look as if the card were in the range of the PoS (Point of Sale). Consequently, the MiM manages to pay fraudulently with the funds associated to the victim-card. The contactless/NFC (Near Field Communication) interface that touch-and-pay EMV builds on does not protect against these vulnerabilities. To mitigate such relay attacks, after 2016, Mastercard’s EMV specifications included the PayPass-RRP protocol. In this protocol (detailed in Appendix B and shown in Figure 5), the reader measures the round-trip times (RTTs) in certain exchanges between itself and the card. That is, the PayPass-RRP reader distance-bounds, the contactless Mastercard: if the RTTs are within a given bound, the likelihood is that the card is close to the reader and no relay attack is taking place.

In the PayPass-RRP protocol, the main assumption is that the reader/PoS behaves correctly and as such detects a relay attack and stops it. However, the reader/PoS has just one incentive: to take the payment, be it honest or relayed. What is more, in EMV (i.e., in the PayPass-RRP protocol), the card-issuing bank gets no proof that the reader performed the anti-relay checks. Rogue readers could easily conspire with the relaying MiM and allow far-away payments to go through; which we term “collusive relaying”. To this end, we introduced two contactless payment protocols that augment the PayPass-RRP reader with a TPM (Trusted Platform Module), in order to protect against collusive relaying: i.e., the RTT-measurements necessarily pass through the TPM on-board the reader, in a way that ensures that dishonest readers can no longer cheat and allow relayed-transactions through. We presented these protocols, without a proof of correctness or implementation, as a short paper at Financial Crypto 2019 [7]. Our two solutions, called PayBCR and PayCCR, build on PayPass-RRP in a way that is backwards compatible with EMV: PayBCR does not modify the PayPass-RRP card, while PayCCR does not modify the bank’s side of the PayPass-RRP protocol.

Shortcomings of State of the Art & Related Work. To the best of our knowledge, no formal analysis of the security of PayBCR or PayCCR is possible with any previously proposed framework.

1https://usa.visa.com/visa-everywhere/blog/bdp/2019/05/13/tap-to-pay-155711409720.html
There exist a number of formal frameworks for modelling distance bounding protocols (e.g. [3, 11]) that do not have automated tool support. Formal methods that lead to automated tools [8, 9, 12, 18] have been applied to analyse (authenticated) distance-bounding and proximity-checking protocols, including PayPass-RRP. However all of these works require an honest reader, and are unable to model the evidence produced by the hardware roots of trust.

Two of these methods [8,12] put forward a version of the applied-pi calculus specialised for encoding distance-bounding: i.e., with participants having locations in metric spaces, actions being timed, etc. The approach by Mauw et al. in [18] is based in turn on multiset-rewriting logics, in which the authors express two flavours of distance-bounding security: one pertaining to time and locations, and the other hinging on just a series of events on a trace. The latter is called causality-based distance bounding and [18] also proves that, under some conditions, timed distance-bounding security and causality-based distance-bounding are equivalent, thus rendering the verification problem more amenable to automation.

The reduction to causality-based verification for distance-bounding described in [18] is sound assuming a number of aspects. First, it must be one and the same agent (i.e., the reader) who timestamps the RTTs and checks these against the time-bound. This is not the case in the new and stronger versions of PayPass-RRP proposed in [7]: e.g., in PayBCR, the reader timestamps the exchange between itself and the card, then takes the RTT measurement, yet the issuing-bank also (re-)verifies these RTT timestamps. Secondly, the time-stamping party (i.e., the reader) must be honest, which is not the case in the strong, collusive-relay attacks. Third, the cards are modelled as fixed at one location throughout the protocol execution: i.e., neither does the most-related prior formalism in [18] nor others (e.g. [8,12]) capture mobility of parties, which is a realistic aspect of payment systems and could possibly lead to attacks.

The PayPass-RRP-based protocols presented in [7] were not implemented or tested (for security and/or efficiency). Furthermore, there are no public implementations of a reader for MasterCards PayPass-RRP protocol. Therefore questions remain over how well these protocols would work in practice given the physical constraints of the hardware involved. We answer these questions by fully implementing these protocols and carrying out a series of tests.

Contributions. In this work, we aim to overcome the aforementioned shortcomings of the state of the art. Specifically, our main contributions are as follows:

1) Extending the work in [12], we put forward an applied-pi calculus that can capture advanced features of distance-bounding protocols featured in new payment protocols: mobility of parties and time-stamping by one party and verification by another.

2) Following the ideas in [18], we then formulate a causality-based definition of distance-bounding (which does away with most timing aspects of these protocols). Importantly, we prove that timed-based security is equivalent to causality-based security, even in our strong model mentioned above. This theoretical result gives formal guarantees that certain physically-based protocols can be checked easily in different provers, in a language that abstracts away time and location. We encode PayBCR, PayCCR and variants, alongside a causality-based definition of secure payments in the fully automatic ProVerif prover, and we discuss the analysis results obtained on these protocols.

3) We provide concrete evidence that the currently proposed EMV distance bounding protocols can work in practice. We do this by implementing a reader for PayPass-RRP, and PayBCR including TPM calls. Using a PayPass-RRP test card from MasterCard and a tool from EMV-centred consultancy Consult Hyperion that emulates the bank backend, we fully test these implementations and provide timing information. We find that both protocols can stop relays that add a delay of 10ms or more, and therefore these protocols are effective at stopping relay attacks using e.g. mobile phones and wi-fi. However, neither protocol (when implemented on standard hardware) will be effective at stopping relays that add a delay of less than 5ms. We find that a factor here is the variance of the card processing time, and therefore no protocol implemented on such a card could stop such a relay.

This work provides the first public empirical analysis of MasterCard’s PayPass-RRP and it also shows that using TPMs can be an effective measure for stopping mobile phone based relay attacks.

Outline. We first recall in Section 2 the protocols presented in [7] which are studied in this work. We present our formal model in Section 3. Then, we revisit the characterisation proposed in [18]. Thanks to this reduction result presented in Section 4, we are able to analyse the security of the protocols presented in [7] relying on the verification tool ProVerif. These results are presented in Section 4.3. In Section 5, we report on the implementations we have performed, as well as the experiments we have done to evaluate the efficiency and the robustness to relay attacks of our own implementations of PayPass-RRP and PayBCR.

2 BACKGROUND

In this section, we recall relevant details on the protocols introduced in [7] and which are both based on Mastercard’s PayPass-RRP.

2.1 On TPMs

Our payment protocols (described below) make use of a Trusted Platform Module (TPM). A TPM is a tamper-resistant hardware chip providing various functionalities, mainly of a cryptographic nature. TPMs also facilitate robust timing functionalities, which we use in our protocols. Concretely, the TPM2_GetTime command2 of the TPM takes the handle for a signature scheme and some input, and it returns a signature over said input and over

TPM-AttestedTime = (Clock, Time).

The latter is the timing data-structure that the TPM keeps: with the first being a non-volatile representation of the real time, set when the TPM is created [16], and the second being a volatile value corresponding to the time passed since the last boot-up of the TPM (see page 205 of [16]). So, as such, TPM2_GetTime()

2This command is supported only in TPM 2.0, which is the current set of specifications of the TPM. More precisely, on TPM2.0, TPM2_GetTime() is supported from revision 1.38 (and not on earlier revisions). TPM 2.0 + 1.38 is available by several manufacturers.
can produce a signed version of a timestamped nonce, with attested
time-information.

According to the standardisation of the TPM, namely TCG, the
attacks onto TPM AttestedTime are mainly relevant w.r.t. the
TPM Clock (see 36.3 and 36.6 [16]), as this has a non-volatile
dimension, unlike Time. Notably, if the TPM is powered down, the
Clock value is correct when the TPM reboots. The threats w.r.t.
Clock documented by TCG, are arguably immaterial in practice
(see page 206 of [16]), and –as such– in this paper and as in [7],
we assume that the timestamps given by the TPM via the command
TPM2_GetTime are timing-secure (in the sense that TCG consid-
ners they overwhelmingly impossible to tamper with timestamping
in this command).

2.2 PayBCR & PayCCR: High-level Description

Both of our protocols enhance PayPass-RRP by adding a TPM
onto a PayPass-RRP reader. This TPM is called twice, each time to
timestamp an input, such that the difference of the two timestamps
is used to produce a signed version of a timestamped nonce, with attested
time-information.

In this section, we describe a formalism allowing us to faithfully
model security protocols based on time and location like those in-
troduced in Section 2. Our security model is expressive enough to
model a variety of cryptographic primitives, and is also suitable to
capture mobility, i.e., the fact that agents executing the protocol may
move during the execution, including during the timing phase. Our
formalism is close to the applied-pi calculus [2] which is the calculus
used in input of the ProVerif tool [4]. On the one hand, we extend
this calculus with several features in order to model time, location
and mobility. On the other hand, some applied-pi constructions, e.g.,
replication, parallel, are only used to define configurations (and not
the protocols themselves), i.e., these are not part of our protocol
syntax.

3 A SECURITY MODEL WITH MOBILITY

In this section, we describe a formalism allowing us to faithfully
model security protocols based on time and location like those in-
troduced in Section 2. Our security model is expressive enough to
model a variety of cryptographic primitives, and is also suitable to
capture mobility, i.e., the fact that agents executing the protocol may
move during the execution, including during the timing phase. Our
formalism is close to the applied-pi calculus [2] which is the calculus
used in input of the ProVerif tool [4]. On the one hand, we extend
this calculus with several features in order to model time, location
and mobility. On the other hand, some applied-pi constructions, e.g.,
replication, parallel, are only used to define configurations (and not
the protocols themselves), i.e., these are not part of our protocol
syntax.

3.1 Agents and Messages

Participants in a protocol are called agents, and the set of agents
is denoted \( \mathcal{A} \). We also consider a fixed and arbitrary set \( \mathcal{M} \subseteq \mathcal{A} \)
to represent malicious agents. During a protocol execution, participants exchange messages through the network. Messages can be
atomic data such as nonces, keys denoted \( n, k \in \mathcal{K} \), agent names
denoted \( a, b \in \mathcal{A} \), or simply public constants denoted \( c, c_1, c_2 \in \Sigma_0 \).
We denote \( \Sigma_0 = \Sigma_0 \cup \mathcal{A} \). More complex messages can also be ex-
changing relying on cryptographic primitives modelled through a set
of function symbols \( \Sigma \) called a signature. Such a signature \( \Sigma \) is split
into constructor and destructor symbols, i.e., \( \Sigma = \Sigma_c \uplus \Sigma_d \). We also
consider a set \( X \) of message variables, denoted \( x, y, z \ldots \), as well as a set \( W \) of handles, denoted \( w_1, w_2, \ldots \). Variables in \( X \) model
arbitrary data expected by a protocol participant, while variables in \( W \) are used to store messages learnt by the attacker. The set \( \mathbb{R}_+ \)
denotes non-negative real numbers and is used to model time. Given
a signature \( \mathcal{F} \) and a set of atomic data, we denote \( \mathcal{T} (\mathcal{F}, D) \) the set
of terms built from \( D \) using symbols in \( \mathcal{F} \). Given a term \( u \), we denote \( \text{vars}(u) \) the variables occurring in \( u \). A constructor term is a term in \( \mathcal{T} (\Sigma_c, \mathcal{N} \uplus \Sigma_0 \uplus \mathcal{X} \uplus \mathbb{R}_+) \).

Example 1. To model the PayBCR protocol presented in Section 2, we consider the signature \( \Sigma_{\text{PayBCR}} = \Sigma_c \uplus \Sigma_d \):

- \( \Sigma_c = \{ \text{enc}, \text{shk}, \text{sign}, \text{pubk}, \text{seck}, \text{mac}, \text{ok} \} \), and
- \( \Sigma_d = \{ \text{sdenc}, \text{verify}, \text{proj}_1, \text{proj}_2, \text{eq} \} \).

The symbols in \( \Sigma_{\text{PayBCR}} \) allows one to construct terms to represent
cryptographic messages, e.g., ciphertext based on the symmetric
encryption symbol \( \text{enc} \) (of arity 2), and signature using the symbol
\( \text{sign} \) (of arity 2). The corresponding decryption and verification
schemes are modelled by the \( \text{sdenc} \) and verify symbols. The symbols
\( \text{pubk} \) and \( \text{seck} \), each of arity 1, are used to model public/private
keys, whereas the symbol \( \text{shk} \) of arity 2 is used to model symmetric
key shared between two agents. The symbol \( \text{mac} \) models a MAC
scheme. The symbols \( \text{mac} \), \( \text{proj}_1 \), and \( \text{proj}_2 \) represent the pairing
and projections operators. Lastly, we consider two symbols \( \text{eq} \) (arity 2)
and the constant \( \text{ok} \) that are used to check an equality between two
terms.
In order to give a meaning to constructor symbols, we equip constructor terms with an *equational theory*. We assume a set of equations $E$ over $T(\Sigma_c, X)$ and define $\equiv_E$ as the smallest congruence containing $E$ that is closed under substitutions and under bijective renaming. Then, we give a meaning to destructors through a rewriting system, i.e., a set of rewriting rules of the form $g(t_1, \ldots, t_n) \rightarrow t$ where $g \in \Sigma_d$ and $t, t_1, \ldots, t_n \in T(\Sigma_c, X)$. A term $u$ can be *rewritten* in $v$ if there is a position $p$ in $u$, and a rewriting rule $g(t_1, \ldots, t_n) \rightarrow t$ such that $u[p] = g(t_1, \ldots, t_n)\theta$ for some substitution $\theta$, and $v = u[t\theta]$, i.e., $u$ in which the term at position $p$ is replaced by $t\theta$. Moreover, we assume that $t_1\theta, \ldots, t_n\theta$ as well as $t\theta$ are constructor terms. As usual, we consider sets of rewriting rules that yield a convergent rewriting system, and we denote $u[.]$ the normal form of a term $u$.

**Example 2.** The properties of the cryptographic primitives introduced in Example 1 are reflected through the following rewriting system:

\[
\begin{align*}
\text{sdec}(\text{senc}(x, y), y) & \rightarrow x \\
\text{verify}(\text{sign}(x, y), \text{pubk}(y)) & \rightarrow x \\
\text{eq}(x, x) & \rightarrow \text{ok}
\end{align*}
\]

The first rule is the usual rewriting rule to model symmetric encryption. Depending on whether we want to model a decryption algorithm that may fail or not, we can either consider sdec as a destructor together with the rewrite rule sdec(senc(x, y), y) $\rightarrow x$ as we did here, or consider both symbols as constructors, together with the equation sdec(senc(x, y), y) = x.

In the latter case, sdec(c, k) will be considered as a “valid” message. The second rule is used both to check a signature and to extract its content. The term \text{seck}(a) is used to represent the private key of the agent $a$, whereas \text{pubk}(\text{seck}(a)) models its public counterpart. The third one models an equality check. Note that $\mathbf{eq}(u, v)$ reduces to a constructor term if, and only if, $u \equiv_E v$ (i.e., $u = v$ here since $E = \emptyset$). The two rules in the second column model the projection operators.

For modelling purposes, we split the signature $\Sigma$ into two parts, $\Sigma_{\text{pub}}$ and $\Sigma_{\text{priv}}$. An attacker builds his own messages by applying public function symbols to terms he already knows and which are available to him through variables in $W$. Formally, a computation done by the attacker is a *recipe*, i.e., a term in $T(\Sigma_{\text{pub}}, \Sigma_{\text{priv}} \cup W \cup \mathbb{R}_+)$.

**Example 3.** Among the symbols in $\Sigma_{\text{BCR}}$, only the symbols shk and seck are in $\Sigma_{\text{priv}}$. Let $u_0 = \text{senc}(n, k)$, shk(h, c), and $u_1 = \text{shk}(h, c)$. We have that $\text{proj}_2(\text{sdec}(u_0, u_1)) \downarrow = k$. The term
3.2 Protocols

Single protocol roles are modelled through a process algebra closed to the one used as input in the ProVerif verification tool. Processes are given by the following grammar:

\[
P, Q := \begin{align*}
&0 \\
&\text{new } n.P \\
&\text{in}(x).P \\
&\text{out}(u).P \\
&\text{let } x = v \text{ in } P \text{ else } Q \\
&\text{gettime}(x).P \\
&\text{check}(u_1, u_2, u_3).P \\
&\text{claim}(u_1, u_2, u_3, u_4).P
\end{align*}
\]

where \(x \in X, n \in N, v \in T(\Sigma, N \cup \Sigma^+ \cup X \cup \mathbb{R}_+), \) and \(u, u_1, \ldots, u_4 \in T(\Sigma_c, N \cup \Sigma^c_0 \cup X \cup \mathbb{R}_+).\)

The first four instructions are all standard to the applied pi-calculus. As usual, the null process does nothing. The restriction new \(n.P\) generates a fresh name and then executes \(P\). We have constructions to model input and output actions. The let construction tries to evaluate \(v\) to get a constructor term \(u\), then \(x\) is bound to \(u\) and \(P\) is executed; if the evaluation of \(v\) fails, then \(Q\) is executed. The last three instructions are used to model distance bounding protocols. The instruction gettime\((x)\) bounds the current time to the variable \(x\). This is used to add a timestamp in some messages. The check\((u_1, u_2, u_3)\) instruction indicates that a process is checking the times \(u_1\) and \(u_2\) against some expected time bound \(u_3\). The claim\((u_1, u_2, u_3, u_4)\) indicates that an agent executing it believes that there has been a successful run of the protocol between \(u_1\) and \(u_2\) with the challenges and responses exchanged between times \(u_3\) and \(u_4\). As we will see below, the two events, namely claim and check do not interfere with the semantics of the processes rather we rely on them to express our security properties.

We write fs\((P)\) for the set of free variables occurring in \(P\), i.e., the set of variables that are not in the scope of an \(a\), a let, or a gettime instruction. We consider parametrised processes, \(P(z_0, \ldots, z_n)\), where \(z_0, \ldots, z_n\) are variables from a special set \(Z\) (disjoint from \(X\) and \(W\)). Intuitively, these variables will be instantiated by agent names, and \(z_0\) corresponds to the name of the agent who executes the process. A role \(R = P(z_0, \ldots, z_n)\) is a parametrised process such that \(fs(R) \subseteq \{z_0, \ldots, z_n\}\) and that does not contain element in \(\mathbb{R}_+.\) This allows us to ensure that elements in \(\mathbb{R}_+\) occurring in an execution have been either introduced by our gettime instruction, or by the attacker. A protocol is a finite set of roles.

**Example 4.** The TPM check time functionality will access a nonce and sign this along with the current time. We write this functionality using our syntax as:

\[
\text{TPM}(z_0) = \langle \text{in}(x_n), \text{gettime}(x_t), \text{out}(\text{sign}(x_1, x_n), \text{seck}(z_0)), 0 \rangle
\]

The parameter \(z_0\) will be instantiated by an agent name. Such a process is waiting for a message, and outputs its signature. The signature is done using the key \(\text{seck}(z_0)\) and a timestamp is added into the signature. The current time is obtained using the gettime instruction.

3.3 Mobility Model

In order to faithfully model the fact that transmitting a message takes time, we use the notion of mobility plan whose main purpose is to indicate the location of each agent at a given time. Then, a message \(m\) sent by an agent \(a\) at time \(t_a\) and received by another agent \(b\) at time \(t_b\) must satisfy that the Euclidean distance between the two locations:

1. \(\text{Loc}(a, t_a), \) i.e., the location of \(a\) at time \(t_a,\)
2. \(\text{Loc}(b, t_b), \) i.e., the location of \(b\) at time \(t_b,\)

is less or equal than \((t_b - t_a) \cdot c_0.\) Here, \(c_0\) is the transmission speed, and is supposed to be constant in our model (e.g., the speed of the light). In this way, the physical law that messages cannot travel faster than the speed of light is made explicit. As \(c_0\) is to be constant in our model, distance between two locations will be represented by the time it takes for a message to travel from one point to the other. Hence, we have that \(\text{Dist} : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}_+\) is defined as follows:

\[
\text{Dist}(l_1, l_2) = \frac{\|l_2 - l_1\|}{c_0} \text{ for any } l_1, l_2 \in \mathbb{R}^3
\]

with \(\|\cdot\| : \mathbb{R}^3 \rightarrow \mathbb{R}_+\) the Euclidean norm.

A mobility plan \(\text{Loc}\) is a function: \(A \times \mathbb{R}_+ \rightarrow \mathbb{R}^3\) defining the position of each agent in space at any time. To avoid unrealistic behaviours where agents will travel faster than messages, we assume that for any \(a \in A\) and \(t_1, t_2 \in \mathbb{R}_+\) such that \(t_1 < t_2,\) we have that:

\[
\text{Dist}(\text{Loc}(a, t_1), \text{Loc}(a, t_2)) \leq t_2 - t_1.
\]

We note that this requires that our mobility plans are continuous, with no agents making discrete jumps in location.

**Example 5.** To illustrate the notion of mobility plan, we may consider the function \(\text{Loc}_a\) such that \(\text{Loc}_a((\text{tpm}_a, t)) = (0, 0, 0)\) and \(\text{Loc}_a((\text{card}_a, 0)) = (10, 0, 0)\) and \(\text{Loc}_a((\text{bkd}_a, t)) = (100, 0, 0)\) for any \(t \in \mathbb{R}_+,\) and \(\text{Loc}_a((a, t)) = (0, 0, 0)\) otherwise (i.e., for any \(a \in A\). This models a very simple mobility plan where all the agents are actually at a fixed position and they never move: \(\text{card}_a\) is at distance 10 from \(\text{tpm}_a\) and the agent \(\text{bkd}_a\) is even further. The other entities are all located at position \(0, 0, 0\) at the same place as \(\text{tpm}_0\).

3.4 Semantics

Our semantics is given by a transition system over configurations that manipulates extended processes, i.e., expressions of the form \(\langle P, \Phi, t \rangle\) where:

- \(P\) is a multiset of extended processes \(\langle P \rangle_a\) with \(a \in A\) and \(P\) a process such that \(fs(P) = \emptyset.\) Intuitively, \(P\) describes the actions of agent \(a.\) In order to store the messages that have been outputted so far, we extend the notion of frame to keep track of the time at which the message has been outputted and by whom. We rely on the special symbol \(\star\) to indicate that a message is known by any agent. This will be used to define the initial frame.

**Definition 1.** A configuration \(\mathcal{K}\) is a tuple \(\langle P, \Phi, t \rangle\) where:

- \(\mathcal{P}\) is a multiset of extended processes \(\langle P \rangle_a\).
TIM

$$(P; \Phi; t) \xrightarrow{\text{inLoc}} (P; \Phi; t + \delta)$$

with $\delta > 0$

NEW

$$([\text{new} \, n.P]_a \cup P; \Phi; t) \xrightarrow{a, t, r} \text{Loc} \ (\{P \mapsto n'\})_a \cup P; \Phi; t)$$

with $n' \in N$ fresh

OUT

$$([\text{out}(u).P]_a \cup P; \Phi; t) \xrightarrow{a, t, \text{out}(u)} \text{Loc} \ (\{P\}_a \cup P; \Phi; \{w \xrightarrow{a, t} u\}; t)$$

with $w \in W$ fresh

LET-THEN

$$([\text{let} \ x = v \ \text{in} \ P \ \text{else} \ Q]_a \cup P; \Phi; t) \xrightarrow{a, t, r} \text{Loc} \ (\{P[x \mapsto v]\}_a \cup P; \Phi; t)$$

when $v \downarrow \in T(\Sigma_c, N \cup \Sigma_0^+ \cup \mathbb{R}^+)$

LET-ELSE

$$([\text{let} \ x = v \ \text{in} \ P \ \text{else} \ Q]_a \cup P; \Phi; t) \xrightarrow{a, t, \text{out}(u)} \text{Loc} \ (\{P\}_a \cup P; \Phi; t)$$

when $v \uparrow \notin T(\Sigma_c, N \cup \Sigma_0^+ \cup \mathbb{R}^+)$

CLAIM

$$([\text{claim}(u_1, u_2, u_3, u_4).P]_a \cup P; \Phi; t) \xrightarrow{a, t, \text{claim}(u_1, u_2, u_3, u_4)} \text{Loc} \ (\{P\}_a \cup P; \Phi; t)$$

CHECK

$$([\text{check}(u_1, u_2, u_3).P]_a \cup P; \Phi; t) \xrightarrow{a, t, \text{check}(u_1, u_2, u_3)} \text{Loc} \ (\{P\}_a \cup P; \Phi; t)$$

GTIM

$$([\text{gettime}(x).P]_a \cup P; \Phi; t) \xrightarrow{a, t, \text{inLoc}} \text{Loc} \ (\{P[x \mapsto t']\}_a \cup P; \Phi; t')$$

with $t' > t$

IN

$$([\text{in}(x).P]_a \cup P; \Phi; t) \xrightarrow{a, t, \text{in}(x)} \text{Loc} \ (\{P[x \mapsto u]\}_a \cup P; \Phi; t)$$

when there exist $b \in A$ and $t_b \in \mathbb{R}_+$ such that $t \geq t_b + \text{Dist}(\text{Loc}(b, t_b), \text{Loc}(a, t))$ and:

- either $b \in A \setminus M$ and there exists $(w \xrightarrow{b, t_b} u) \in \Phi$
- or $b \in M$ and $\Phi \vdash u$ by $b$ at time $t_b$ w.r.t. Loc.

Figure 2: Semantics of our calculus parametrised by the mobility plan Loc.

- $\Phi = \{w_1 \xrightarrow{a_1, t_1} u_1, \ldots, w_n \xrightarrow{a_n, t_n} u_n\}$ is an extended frame, i.e., a substitution such that $w_i \in W$, $u_i \in T(\Sigma_c, N \cup \Sigma_0^+ \cup \mathbb{R})$, $a_i \in A \cup \{\ast\}$ and $t_i \in \mathbb{R}_+$ for $1 \leq i \leq n$;
- $t \in \mathbb{R}_+$ is the global time of the system.

EXAMPLE 6. A typical configuration for our running example is $K_0 = (P_0; \Phi_0; 0)$ with:

$P_0 = \{[\text{TPM}(\text{tpm}_0)]; \text{Bank}(\text{bk}_0); [\text{Card}(\text{card}_0)]\}$

This simply models a scenario where tpm$_0$, bk$_0$, and card$_0$ execute a single session of their role. Regarding the initial frame, we may consider:

$\Phi_0 = \{w_1 \xrightarrow{a_1, t_1} \text{pubk(\text{seck}(\text{card}_0))}, w_2 \xrightarrow{a_2, t_2} \text{pubk(\text{seck}(\text{bk}_0))}, w_3 \xrightarrow{a_3, t_3} \text{seck(\text{card}_0)}\}$

This initial frame reveals the public key of the 3 agents to the attacker. We may want to also reveal the certificates built by the bank for the card and the tpm. This will correspond to the following terms in the frame:

- $\text{sign}([\text{cardCert}, \text{card}_0, \text{pubk(\text{seck}(\text{card}_0))}], \text{seck}(\text{bk}_0))$;
- $\text{sign}([\text{tpmCert}, \text{tpm}_0, \text{pubk(\text{seck}(\text{tpm}_0))}], \text{seck}(\text{bk}_0))$.

The terms cardCert and tpmCert are public constants from $\Sigma_0$ used to avoid a possible confusion between the two kinds of certificates (the one issued by the bank to certify a card, and the one used to certify a TPM). In case such a confusion is possible, we may model the two types of certificates relying on the constant cert.

Given an extended frame $\Phi$, and a mobility plan Loc, we say that a term $u$ is deducible from $\Phi$ by $b \in A$ at time $t_0$, denoted $\Phi \vdash u$ by $b$ at time $t_0$ w.r.t. Loc, if there exists a recipe $R$ such that $R\Phi = u$, and for all $w \in \text{vars}(R)$ we have that $(w \xrightarrow{c, t} v) \in \Phi$ for some $v$, and:

- either $c = \ast$;
- or $t \geq t_0 + \text{Dist}(\text{Loc}(c, t), \text{Loc}(b, t_0))$.

In other words, $u$ has to be forgeable by the agent $b$ at time $t_0$ and thus messages needed to forge $u$ have to be available in due time.

Given a mobility plan Loc, the semantics of processes is formally defined by a transition system over configurations. This transition system is given in Figure 2 and is parametrised by Loc. This piece of information is omitted when clear from the context. We will only comment on some key cases.

The rule TIM allows time to elapse, and the rule GTIM allows an agent to get the global time of the system. Note that two consecutive gettime instructions will return two different values since we force time to elapse. This models the fact that such an instruction cannot be executed instantaneously, and that two remote agents cannot perfectly co-ordinate their actions. This also helps us establish the soundness of our abstraction by making it possible for us to get rid of time by replacing real numbers issuing from a gettime instruction by different public constants.

The IN rule is more complex (than in normal applied–pi) since we have to ensure that enough time has passed to make it possible for the message to travel from where it was produced Loc$(b, t_b)$ to where it is being received Loc$(a, t)$. As noted above, the claim and check instructions are simple events (as those used in ProVerif). They do not interfere with the semantics but are used to model security properties.

EXAMPLE 7. To illustrate the semantics, we consider the configuration $K'_0 = ([\text{TPM}(\text{tpm}_0)]; \Phi_0; 0)$ and the mobility plan Loc$_0$ as given in Example 5.

$K'_0 \xrightarrow{[\text{TPM}(\text{tpm}_0)]; \Phi_0; 1, 1)} \text{gettime} \text{Loc}_0 \ (\{\text{out}(m)\}_\text{tpm}_0; \Phi_0; 1, 2)$
where:
- \( m = \text{sign}(1.1, \text{ok}, \text{seck}(\text{tpm}_0)) \), and
- \( \Phi_1 = \Phi_0 \cup \{w_4 \rightarrow m\} \).

The first input is possible since \( \Phi_0 \vdash \text{ok} \) by \( c \in \mathcal{M} \) at time 1.1. Actually such a constant is even deducible at time 0.

We may note that \( m \) is deducible from \( \Phi_1 \) by \( \text{tpm}_0 \) at time 1.2. Actually any agent other than \( \text{card}_0 \) and \( \text{bb}_0 \) are able to deduce \( m \) at time 1.2. Remember that \( \text{Loc}_0(a, t) = (0, 0, 0) \) for any \( t \) and any a different from \( \text{tpm}_0 \) and \( \text{bb}_0 \).

### 3.5 Threat Model

Before we present the security property we wish to analyse, we need to define a few initial frames. Before we can give our security definition, we need to define a few initial frames. Before we can give our security definition, we need to define a few initial frames.

**Definition 2.** A configuration \( \mathcal{K} = (\mathcal{P}_0; \Phi_0; 0) \) is a valid initial configuration for a set of roles \( \mathcal{R} \), if \( \Phi_0 \) is an initial frame, and for each \( |P|_a \in \mathcal{P}_0 \), there exists \( R(z_0, z_1, \ldots, z_k) \in \mathcal{R} \) and \( a_1, \ldots, a_k \in \mathcal{A} \) such that \( P = R(a, a_1, \ldots, a_k) \).

Roughly, we consider initial configurations made up of instances of the roles of the protocols, and we only consider roles executed by agents located at the right place, i.e., the agent a who executes the role must correspond to the first argument of the parametrised process. Note that, according to the definition above, a single agent can play different roles (e.g., the bank and the card).

**Example 8.** The frame \( \Phi_0 \) described in Example 6 is an initial frame, and the configurations \( \mathcal{K}_0 \), as well as the configuration \( \mathcal{K}'_0 \) given in Example 7 are valid initial configurations for the protocol payBRC. Although valid, these configurations are rather poor and additional scenarios will be (of course) considered when performing the security analysis. Typically, we will consider many agents, and we will assume that each agent can execute the protocol many times.

Our result applies on all the valid initial configurations. However, to give the possibility to discard some unrealistic configurations (that may depend on the use case) during the security analysis, our main result is parametrised by a set \( \mathcal{S} \) of valid initial configurations.

We are now able to formally state the security property we want to consider.

**Definition 3.** A protocol \( \mathcal{P} \) is DB-secure w.r.t. a set \( \mathcal{S} \) of valid initial configurations if for all \( \mathcal{K}_0 \in \mathcal{S} \), for all mobility plan \( \text{Loc} \), for all execution exec such that:

\[
\mathcal{K}_0 \xrightarrow{(a_1, t_1, \text{act}_1), \ldots, (a_n, t_n, \text{act}_n), (b_0, t, \text{claim}(b_1, b_2, t_1^{0}, t_2^{0}))} \text{Loc} \mathcal{K}
\]

we have that:
- either \( b_1 \in \mathcal{M} \), or \( b_2 \in \mathcal{M} \);
- or \( \text{act}_{k} = \text{check}(t_1^{0}, t_2^{0}, t_3^{0}) \) for some \( k \leq n \) such that:

\[
\text{Dist} \left( \text{Loc}(b_1, t_1^{0}), \text{Loc}(b_2, t) \right) \\
\geq \text{Dist} \left( \text{Loc}(b_2, t), \text{Loc}(b_1, t_2^{0}) \right)
\]

for some \( t_1^{0} \leq t \leq t_2^{0} \).

We now discuss Definition 3, from different viewpoints.

**Main Meaning of Definition 3.** First note that Definition 3 does not bind entities \( b_0, b_1, \ldots, \) to roles. So, in the next, we will try to render the meaning of the definition via one of its several possible representation. Namely, in such an instance of Definition 3, we say that a payment protocol is secure if at the end of its execution an agent \( b_0 \) (e.g., the bank) can make a claim that links an agent \( b_1 \) (e.g., a TPM) and an agent \( b_2 \) (e.g. a card) to two timestamps \( t_1^{0} \) and \( t_2^{0} \) in the following sense: in the execution, there was a check w.r.t. \( t_1^{0} \) and \( t_2^{0} \) and another time-value \( t_3^{0} \) (which is given generally as a constant time-bound in the protocol) and this check denotes that the distance between \( b_1 \) and \( b_2 \) was within the bound between the timepoints \( t_1^{0} \) and \( t_2^{0} \) (i.e., \( t_2^{0} - t_1^{0} \leq t_3^{0} \) and this gives an upper bound on the distance between \( b_1 \) and \( b_2 \)).
We are implicitly interested in the second case of the above, and (which was not the case in [18]); (c) the parties (a) the property is extended to make a statement w.r.t. three parties of Definition 3 is an upper bound on both of the processing of messages is instantaneous. We take the view that in our model (unlike in practice), this would be bound at if the agents were stationary, or their speed is negligible of light; and in fact, in our model, we do state that agents move unrealistic to assume that agents can travel at close to the speed (On the one hand, in practice, in this case, the distance between the each other occurs when they are a fixed distance from each other particular time is more subtle. The furthest the agents could be from Definition 3 is given in terms of the total travel time of the message also it strengthens that – for instance, w.r.t. to allowing for mobility of light. Finally, the property is meant to capture an extension of relay-based attack, applied to contactless payments. We consider a subset Σ of special constants in Σ₀ that will be used to abstract time. We explain how to transform a configuration K = (P; Φ; t) into a simple configuration that does not contain time. For sake of simplicity, we assume that variables occurring in P are at most bound once. The transformation τ applied on K gives us a pair (P₀; φ₀) where:

- P₀ is the untimed counterpart of P, i.e. each get time(x) instruction occurring in P is replaced by timestamp(cₓ) where cₓ ∈ Z₀, and the occurrences of x in the remaining process are replaced by cₓ;
- φ₀ = {w → u | (w(e→act)u) ∈ Φ}.

We denote the relaxed semantics that corresponds to the semantics used in a tool like ProVerif (see Figure 3). The TIM rule does not exist anymore. The rules NEW, OUT, LET-THEN and LET-ELSE are adapted in a straightforward way, i.e. by removing the timing information t. The rules CLAIM, and CHECK will correspond to “events” in ProVerif. Finally the rule GTIM is modified to become a simple event instruction, whereas the IN rule is much more simple since the travel time of messages is not taken into account anymore.

The equivalent of our DB-Secure property is then:

**Definition 4.** A protocol P is causality-based secure w.r.t. a set S of valid initial configurations, if for all K₀ ∈ S, for all execution steps in the configuration, the traces of all processes in S are valid time sequences. **Observations w.r.t. Definition 3.** Firstly, note that the check is performed by agent a₂ which is not fixed to being b₀, b₁ or b₂. Importantly, as such, in this definition, we may have the check be performed by an agent that is not b₀ who in fact makes the final claim.

Secondly, note that indeed it is in one of the possible executions that the agent b₀ is the bank, the agent b₁ is a TPM and b₂ is a card; this type of execution is of interest to us. To this end, Definition 3 stipulates two alternate security-relevant cases:

- be it: the card b₂ is dishonest, or the TPM b₁ is dishonest, or both are dishonest, in which case Definition 3 does not require any further condition and one can declare the protocol trivially DB-secure;
- alternatively, we are in the case where the TPM and the card are honest, in which case the definition contains further conditions (namely the restrictions on check have to be fulfilled for the protocol to be declared DB-secure.)

We are implicitly interested in the second case of the above, and particularly in the case where the reader is malicious.

Finally, the property is meant to capture an extension of relay-resistance or security against a man-in-the-middle attacker that performs a relay-based attack, applied to contactless payments. Our definition is in the spirit of [18]. However, note that in our semantics/model, it is strictly stronger than the definition in [18], as follows:

(a) the property is extended to make a statement w.r.t. three parties instead of two parties; (b) it contains a statement on the checking of the timestamps (as [18] did), but also a claim made on top of this timestamps’ check potentially by another party in the protocol (which was not the case in [18]); (c) the parties b₁ and b₂ are mobile (which was not the case in [18]). In this vain, not only does our definition lift [18] to cover the notion of collusive-relaying in [7], but also it strengthens that – for instance, w.r.t. to allowing for mobility of parties.

**Time-bounds vs. Distance-bounds in Definition 3.** The bound in Definition 3 is given in terms of the total travel time of the message between the agents. As the agents may move while the message is in transit, characterising the exact distance between the agents at a particular time is more subtle. The furthest the agents could be from each other occurs when they are a fixed distance from each other at all times between t₂₀ and t₁₀ and the receiver move towards the message being sent at almost the same speed as the message travels. On the one hand, in practice, in this case, the distance between the agents would be bound at (t₂₀ − t₁₀—message processing time) × c₀. In our model (unlike in practice), this would be (t₂₀ − t₁₀—message processing time) × c₀, as the processing of messages is instantaneous. However, it is highly unrealistic to assume that agents can travel at close to the speed of light; and in fact, in our model, we do state that agents move overwhelmingly more slowly than messages. On the one hand, in practice, if the agents were stationary, or their speed is negligible compared to the speed of the messages, then the distance between them would be bound at (t₂₀ − t₁₀—message processing time) × c₀/2. In our model (unlike in practice), this would be (t₂₀ − t₁₀—message processing time) × c₀/2, as the processing of messages is instantaneous. We take the view that the distance implied by Definition 3 is an upper bound on both of these thresholds.

**4 SECURITY ANALYSIS USING PROVERIF**

To provide automated tool support for checking our definition of DB-security we will encode our processes into the language used by the verification tool ProVerif [4], and automatically verify a property (namely the one in Definition 4) that we show is equivalent to Definition 3.

**4.1 ProVerif in a nutshell**

ProVerif [4] is a well-established automated protocol verifier offering very good support to detect flaws and prove security. This verifier takes as input processes written in a syntax close to one introduced in Section 3 but does not feature location and time. ProVerif can cover a wide class of cryptographic primitives and various protocols structures, yielding a very flexible tool that can be used to analyse various encoding of protocols and security properties. It handles an unbounded number of sessions and even if termination is not guaranteed, it works well in practice. For instance, this tool has been successfully used to analyse two avionic protocols that aim to secure air-ground communications [5], to perform a comprehensive analysis of the TLS 1.3 Draft-18 protocol [6], or more recently to analyse some e-voting protocols [10, 17].

Some recent work has looked at encoding and checking DB protocols in ProVerif [8, 12] however this work requires the reader to be honest, does not allow for mobility, and only allows times to be compared on completion of the DB protocol. Therefore, those methods in [8, 12] cannot be used to analyse protocols such as those described in Section 2.

**4.2 Main result**

We consider a subset Σ₀ of constant special constants in Σ₀ that will be used to abstract time. We explain how to transform a configuration K = (P; Φ; t) into a simple configuration that does not contain time. For sake of simplicity, we assume that variables occurring in P are at most bound once. The transformation τ applied on K gives us a pair (P₀; φ₀) where:

- P₀ is the untimed counterpart of P, i.e. each get time(x) instruction occurring in P is replaced by timestamp(cₓ) where cₓ ∈ Z₀, and the occurrences of x in the remaining process are replaced by cₓ;
- φ₀ = {w → u | (w(e→act)u) ∈ Φ}.

We denote the relaxed semantics that corresponds to the semantics used in a tool like ProVerif (see Figure 3). The TIM rule does not exist anymore. The rules NEW, OUT, LET-THEN and LET-ELSE are adapted in a straightforward way, i.e. by removing the timing information t. The rules CLAIM, and CHECK will correspond to “events” in ProVerif. Finally the rule GTIM is modified to become a simple event instruction, whereas the IN rule is much more simple since the travel time of messages is not taken into account anymore.

The equivalent of our DB-Secure property is then:

**Definition 4.** A protocol P is causality-based secure w.r.t. a set S of valid initial configurations, if for all K₀ ∈ S, for all execution
Security Analysis and Implementation of Relay-Resistant Contactless Payments

NEW’  \[ ([\text{new } n.P]_a \uplus P; \Phi) \xrightarrow{a,\tau} ([P[n \mapsto n’]]_a \uplus P; \Phi) \] with \( n’ \in \mathbb{N} \) fresh

\[ \vdots \] \[ \vdots \]

CHECK’  \[ ([\text{check}(u_1, u_2, u_3).P]_a \uplus P; \Phi) \xrightarrow{a,\text{check}(u_1, u_2, u_3)} ([P]_a \uplus P; \Phi) \]

GTIM’  \[ ([\text{timestamp}(x).P]_a \uplus P; \Phi) \xrightarrow{a,\text{timestamp}(c)} ([P]_a \uplus P; \Phi) \]

IN’  \[ ([\text{in}(x).P]_a \uplus P; \Phi) \xrightarrow{a,\text{in}(u)} ([P[x \mapsto u]]_a \uplus P; \Phi) \] with \( R\Phi \models \equiv u \) for some recipe \( R \)

Figure 3: Semantics a la ProVerif.

\[ \text{exec such that:} \]
\[ \frac{a_1(x_1),\ldots,a_n(x_n),b_i(\text{claim}(b_i,b_1,c_1,c_2))}{(P';\phi')}(K_0) \]

we have that either \( b_1 \in \mathcal{M} \), or \( b_2 \in \mathcal{M} \), or there exist \( i, j, k, k' \leq n \) with \( i \leq k' \leq j \), and \( u \in \mathcal{T}(\Sigma, N \cup \Sigma^*_0) \) such that:
1. \( a_{k'} = \text{check}(c_1,c_2,u) \);
2. \( (a_1, act_1) = (b_1, \text{timestamp}(c_1)) \);
3. \( (a_j, act_j) = (b_1, \text{timestamp}(c_2)) \); and
4. \( a_{k'} = b_2 \).

This definition states that for every claim that \( b_2 \) is in the vicinity of \( b_1 \) between times \( c_1 \) and \( c_2 \), a timing constraint must have been checked and \( b_2 \) must have been active in between the two events.

Before stating our main result, we have to introduce an hypothesis; since, causality-based requires the existence of timestamp events, we have to ensure that those events will be available.

**Definition 5.** A protocol \( P \) is well-timed w.r.t. a set \( S \) of valid initial configurations if for all \( K_0 \in S \), for all execution \( \text{exec such that:} \)
\[ \frac{a_1(x_1),\ldots,a_n(x_n),b_i(\text{claim}(b_i,b_2,c_1,c_2))}{(P';\phi')}(K_0) \]

we have that there exist \( i, j \leq n \) such that:
1. \( (a_i, act_i) = (b_1, \text{timestamp}(c_1)) \);
2. \( (a_j, act_j) = (b_1, \text{timestamp}(c_2)) \).

We are now able to state our main result that establishes the equivalence between the two notions DB-security and causality-based security. Causality-based security does not rely on time and location anymore and can therefore be analysed relying on ProVerif.

**Theorem 1.** Let \( P \) be a protocol and \( S \) be a set of valid initial configurations. Assuming that \( P \) is well-timed w.r.t. \( S \), we have that:
- \( P \) is DB-secure w.r.t. \( S \) if, and only if,
- \( P \) is causality-based secure w.r.t. \( S \).

A detailed proof is available in Appendix. First, we may note that if \( P \) is causality-based secure w.r.t. \( S \) then \( P \) is well-timed w.r.t. \( S \). The first implication is rather easy to establish. It mainly consists in retiming a witness of an attack against the causality-based secure property. Re-timing such an execution depends on the underlying mobility plan, and we consider the mobility plan where all the agents are located at the same place (and never move) but the two agents \( b_1 \) and \( b_2 \) are far away. Since no action will be performed by \( b_2 \) between the two timestamps events, we know that all the actions are performed by the agents located at the same place and the time elapsed between these two timestamp actions can be fixed to be less than the distance between \( b_1 \) and \( b_2 \).

The other implication is more complex to establish. We start with an attack trace w.r.t. DB-security with a value \( t_j - t_i \) smaller than some value \( \delta \) corresponding to the distance between \( b_1 \) and \( b_2 \), and then we consider the following steps:
- We first weaken this timed trace in the untimed semantics;
- Then, we clean up the trace between the two timestamps actions to ensure that all the actions between these two timestamps can not be pushed before or after. Thanks to causality-based security, we know that an action from \( b_2 \) is performed in between.
- We then lift this trace into the timed model keeping the exact same value for \( t_j - t_i \), and due to the action performed by \( b_2 \), we know that some time has elapsed between the two timestamps, and thus \( t_j - t_i \) is necessary bigger than \( \delta \). We therefore reach a contradiction.

### 4.3 Case studies

We consider the two protocols PayBCR and PayCCR respectively described in Section 2 and Appendix C, and we report the results we have obtained using ProVerif. All the ProVerif models mentioned in this section are available in [1].

**ProVerif models.** We model the protocol PayBCR following the description given in Section 2. Regarding PayCCR, it is actually not possible to state the causality-based security property (and even more the DB-security property). The problem is that, in PayCCR, the bank never receives the reader/TPM identity or the two timestamps; so, the data needed to state the final claim claim are not available to the bank. To overcome this limitation, we propose a slightly modified version of the protocol named PayCCR++ in which the TPM identity and the two timestamps are added in the \( \mathcal{A} \) message\(^5\).

More formally, we have that:
\[
\mathcal{A}_{payCCR++} = MAC(K_S, ATC, data, \sigma_1, TPM, t_1, t_2)
\]

with \( K_S = \text{sen}(ATC, \text{shk(Card, Bank)}) \) as presented in Figure 6.

**Scenarios.** For each protocol, we have considered a scenario with an arbitrary number of banks, cards, and TPMs. We do not model the reader which is assumed to be dishonest and thus fully executable by the attacker. More precisely, the role of the bank is played by many

\(^5\)In practice, this is feasible via optional fields inside the \( \mathcal{A} \), which issuing-banks already use for further data-collection.
possible entities. Each bank issues many cards, and is also used to certify many TPMs. Among these cards and these TPMs, some are honest, and some are dishonest meaning that their key material is revealed to the attacker. However, a given honest entity can not act as a card and a TPM at the same time. This is not true for a dishonest participant, and thus to avoid a dishonest participant to act as a card and a TPM, it is important to differentiate the certificates: given a bank name $bankID$, a TPM certificate of agent $a$, noted $certT(a)$, is

$$sign(TPMCert, a, pub(seck(a)), seck(bankID))$$

whereas a card certificate, $certC(a)$ will be

$$sign((cardCert, a, pub(seck(a))), seck(bankID))$$

The initial knowledge given to the attacker is:

- $pub(seck(a))$ for any agent $a$, and his associated certificate $certX(a)$;
- $seck(a)$, and $shk(a, bankID)$ when the agent $a$ is dishonest.

**Security Properties.** Firstly, we consider the *causality-based property* as stated in Definition 4.

In addition, we consider the following extra *authentication property*:

query $TPMID:bitstring$, $cardID:bitstring$,

$$t1:bitstring, t2:bitstring,$$

$$event(claim(TPMID, cardID, t1, t2)) \implies$$

$$\forall event(TPM(TPMID)) \&\& event(card(cardID)).$$

This property expresses the fact that when the bank ends a session apparently with $TPMID$ and $cardID$, then $TPMID$ is a TPM identity whereas $cardID$ is a card identity.

In these protocols the times are checked against a time-bound $td$ that is specific to each card and sent by the card in the SDAD message. In this context, an attacker may try to replace it by an excessively large value to make the bank accept the transaction. To avoid this undesired behaviour, we add at the end of the bank process a new event $receivedBound(card_0, td)$ and check the following extra property:

query $card_0:bitstring$, $timeboundinfo:bitstring$,

$$event(receivedBound(card_0, timeboundinfo)) \implies$$

$$timeboundinfo = timebound(card_0).$$

where card_0 is an honest agent.

This property means that for each accepted transaction with an honest card, the time-bound received by the bank is correct, i.e. it is the correct time-bound for the card.

**Verification Results.** The protocols have been analysed w.r.t. the causality-based security property and the authentication property mentioned above. To make ProVerif conclude, we added additional data in the check and timestamp events (e.g., the fresh nonce $nC$ generate by the card during a session). In addition to highlight the actions performed by the card (i.e. $by_2$ in Definition 4) we had a new event $proverAction$ in the role of the card. One may note that if a protocol satisfies the resulting query then it is causality-based secure. Indeed, the more precise the events are the stronger the security property is.

ProVerif always returns in less than 1s. All the results are presented in the following table:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Role authentication</th>
<th>Time-bound authentication</th>
<th>Causality-based security</th>
</tr>
</thead>
<tbody>
<tr>
<td>PayCCR++</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PayBCR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Results Significance.** As expected the two protocols are causality-based secure, i.e., the physical proximity of the two agents involved in a transaction is ensured, as soon as they are honest.

Moreover, note that, for such transactions, if time-bound authentication holds, then this inherently implies that the check of timestamps occurring in the causality-based property had been correctly performed.

However, if the PayBCR protocol satisfies role authentication, PayCCR++ does not. This means that, in PayCCR++, a bank may accept a rogue transaction, for instance one involving a card acting as a TPM (or inversely).

This weakness exhibited is however not surprising. Indeed, checking the TPM certificates is part of the role of the card in PayCCR++, while it is performed by the bank in PayBCR. This means that certificates may not have been checked if the card’s role is executed by a malicious agent in PayCCR++, whereas in PayBCR they will be always correctly checked since the bank is assumed honest.

**Properties’ Significance.** Lastly, to go further, one can ask themselves if the DB-security/causality-based definition is too strong (since it cannot even be cast to PayCCR), or it is the correct property to require. Note that the property’s last claim does link the card’s ID and the TPM’s ID together with specific timestamps, which can arguably be seen a strong demand for a payment protocol. Instead, one could –for instance– view that the final claim be made just on a session ID of the whole protocol execution for which a check took place. However, we cannot prove the DB-security to causality-based security reduction for such a weaker claim. In other words, the causality-based definition we verify (which can be cast on PayCCR++ but not on PayCCR) is also imposed by our theoretical results. That said, we are pleased to say that not only can we cast this property and the authentication property on PayBCR, but we see that they both hold for this protocol. This is a measure of showing that DB-security (together with the accompanying authentication property) may indeed be the right security definitions for strong-relay resistant payments.

## 5 IMPLEMENTATION

Our formal work above lets us verify the protocol design, however certain practical questions remain, e.g., is it possible to integrate a TPM and an EMV reader? Is the clock on the TPM accurate enough to enforce a reasonable distance bound on the card and reader? To answer these questions, in this section we describe our implementation of PayBCR, and discuss its experimentally-ascertained functional correctness, efficiency and security. We also include a discussion of our own implementation of a PayPass-RRP reader. All of the code for our implementations is available in [1].
5.1 Implementing a PayPass-RRP and PayBCR Reader

Why Implement a PayPass-RRP Reader. Despite the fact that the PayPass-RRP specification has existed since 2016, the PayPass-RRP protocol is not deployed “in the wild”. However, we were able to obtain PayPass-RRP test cards, one implemented by ICC Solutions and one directly given to us by MasterCard. These cards run JAVA card OS and a proprietary Mastercard applet, designed fully compliant with EMV v4. On the reader side, no public implementation of the PayPass-RRP reader is available, and there are no public experiments to say if the protocol will work in practice on standard hardware. So, we have created our own implementation of the PayPass-RRP reader.

Why Implement PayBCR. Given the above, we chose to implement PayBCR rather than PayCCR as the former is arguably of more interest for real-life deployment, in that –unlike PayCCR—it does not require change to the widest-spread elements of EMV, that is the cards. Also, as Section 4.3 concludes, PayBCR has the advantage of authenticating the TPM to the bank, which PayCCR does not.

Using TPMs & NFC Readers for PayBCR. Due to the lack of TPMs on current EMV readers, we use a Vostro Notebook 5471 Base\(^6\) with a TPM2.0 v1.38 on board, running Windows 10 and a standard NFC reader\(^7\). To emulate the banks side of the transaction, we use a proprietary EMV emulation and test suite, known as “CardCracker”, from EMV manufacturer and consultancy called Consult Hyperion (https://chyp.com/). We note that this proprietary software is not necessary to replicate our data, it does however ensure that our reader implementation can correctly complete a transaction from the point of view of the back end banking network.

The PayBCR Terminal. The modification that we make to a PayPass-RRP terminal to lift it to a PayBCR terminal is the fact that –as part of the ERRD command\(^8\)– we include TPM2_GetTime calls to the on-board TPM. In terms of interactions with the TPM, the two TPM calls are done over one connection to the TPM, unless the connection has been (incidentally) cut by the latter. As part of this, we also had to declare new “EMV” fields for the terminal to store the two signatures generated by the TPM commands, as well as implement the logic of these be sent to the (CardCracker-emulated) bank and be verified by the later. We implement two versions of the terminal an honest version that behaves as expected, and a “timing-rogue” version that does not perform any checks and forwards messages to the TPM for time-stamping directly. For these implementation details, please refer to Appendix D.

We implemented our own EMV relay-attack. We tested it using an iZettle EMV reader. We ascertained that our implementations of both PayPass-RRP and PayBCR do stop the relay when the implementations are those with an honest reader; and, indeed, the bank would accept a relayed transaction form a “timing-rogue” PayPass-RRP terminal, but would reject a delayed transaction form a “timing-rogue” PayBCR terminal. In Appendix D, we give details on this implementation, and on our testing of the correct as well as “under-relay” functionalities.

5.2 Performance Testing: Honest and Relayed Cases

To show that the protocols discussed here are useful in practice, we will need to show that time bounds exist that will protect the card against relaying. I.e., we need to show that the time variance of the transaction must be small when compared to the time added by a practical relay.

To get the needed time data from our implementation, we used CardCracker to time software steps and an external “APDU-spying” tool – National Instruments (Micropross) ACL1 – to measure the times of messages in transit. This allows us to time each step of the transactions. The timing of the distance bounding ERRD command for both PayBCR and PayPass-RRP is shown in Table 1. The card processing times and transmission times are the same for PayBCR and PayPass-RRP, the difference comes in the reader processing step which takes longer, and has a higher standard deviation in the PayBCR protocol due to the calls to the TPM.

We note that for these tests, we only considered single attempt of the protocol, aborting failed runs. Therefore, care would have to be taken when generalising our results to implementations that would automatically try to resend messages following a failed exchange\(^9\).

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card Processing Time</td>
<td>13,399</td>
<td>1,850</td>
</tr>
<tr>
<td>Card-to-reader Transmitting Time</td>
<td>1,548</td>
<td>418</td>
</tr>
<tr>
<td>Reader Processing Time PayPass-RRP</td>
<td>8,238</td>
<td>938</td>
</tr>
<tr>
<td>Reader Processing Time PayBCR</td>
<td>24,615</td>
<td>3,815</td>
</tr>
<tr>
<td>Reader-to-card Transmitting Time</td>
<td>1,217</td>
<td>205</td>
</tr>
<tr>
<td>Total PayPass-RRP</td>
<td>24,402</td>
<td>2,121</td>
</tr>
<tr>
<td>Total PayBCR</td>
<td>40,779</td>
<td>4,265</td>
</tr>
</tbody>
</table>

Table 1: Durations of ERRD Commands/Responses for our PayPass-RRP and PayBCR Implementations (in microseconds)

Discussion. We consider these protocols to be practical if it is possible to find a time bound that will allow honest transactions to go through while still stopping relayed transactions. We make the assumption that a time bound will be set to make false positives (i.e., rejected but honest transactions) and false negatives (i.e, accepted yet relayed transactions) equally likely, i.e., the time bound is half the relay speed plus the mean transaction time. In Figure 4 we use the standard deviation from Table 1 to plot the false positive and negative rates for our implementation of PayPass-RRP and PayBCR for a range of possible relay speeds.

\(^6\)CPU: Intel i5-8250U (6MB Cache, up to 3.4GHz), memory: 8GB, DDR4, 2400MHz.

\(^7\)We used a SCM Microsystems INC SDI011G.

\(^8\)This is the EMV command that views the round-trip measurements and it is explained in Appendix D.1.
What is more, whilst the EMV standard has added relay-protection to well to stop relays that added a delay of more than 10ms. The robustness of EMV payments is formulated under the assumption and, as such, has not been probed in practice. Also, most security and contactless payments in 2016, this protection has not been deployed less investigated and, certainly, less so from a formal perspective. However, their newer and rising contactless dimension has been time. Nonetheless, the EMV standard has added relay-protection to contactless EMV transaction is capped. Additionally, our experiments for protecting EMV transactions, especially since the value of a contactless equipment that can relay in microseconds. It is clear that neither protocol can defend against such a relay. Indeed, based on the card processing time variance alone, a relay that adds a 3ms delay (or less) will lead to a false positive and false negative rate of more than 2%. Therefore, we can entail that it is not possible for any design of reader to time bound a MasterCard PayPass-RRP, built using the same methods as our test card, to protect against specialist equipment that can relay in microseconds.

Overall, the take-home message of our experiments is that both PayPass-RRP and PayBCR are capable of stopping relay attacks based on COTS hardware, such as mobile phones, so are practical for protecting EMV transactions, especially since the value of a contactless EMV transaction is capped. Additionally, our experiments add to the work in [7] and show that enhancing an EMV-reader specification with a TPM as a hardware root of trust (a la PayBCR) can be a practical option for protecting relays in the presence of rogue readers.

6 CONCLUSION

Payments and their security have been formally looked at for some time. However, their newer and rising contactless dimension has been less investigated and, certainly, less so from a formal perspective. What is more, whilst the EMV standard has added relay-protection to contactless payments in 2016, this protection has not been deployed and, as such, has not been probed in practice. Also, most security and robustness of EMV payments is formulated under the assumption that the EMV readers/terminals are honest; in this day and age, where many of these run on open and updatable firmware, this crux assumption is at least challengeable and was indeed effortlessly refuted in 2019, in [7]. The latter work proposed that EMV terminals be retrofitted with hardware roots of trust (such as TPMs) to aid enhance aspects of their contactless security (namely, measurements of time and distance that are used in relay-counteractions). Last but not least, the said terminals are no longer fixed points-of-sale, they are mobile iZettle, with the merchants moving them around even as payments take place. To this end, their contactless range and measurements fluctuate with such motions.

To sum up, in this work, we addressed the aforementioned gaps and new developments in contactless-payments security, both from a formal analysis and from a practical/implementation perspective. Concretely, we presented a calculus to model relay-resistant protocols, included the most recent ones that offer strong relay-protection via their integration with hardware roots of trust (HWRoT). In the same modern and novel vein, our calculus is also the first to allow for mobility of cards and readers within the proximity-checking primitive used for relay-protection. To be able to formally analyse these protocols, the physical aspects of time and distance are cumbersome to deal with in practical, security analysis tools. As such, we also extended a causality-based characterisation of proximity-checking security which was initially developed by S. Mauw et al. in 2018, such that we eliminate the need to account for physicalities in our (even stronger) protocol models and security analyses. In turn, this allowed us to verify the proximity-related security of a series of contactless payments, included the new ones in [7], in the popular verification tool ProVerif. Last but not least, we provided the first implementation of Mastercard’s relay-resistant EMV protocol called PayPass-RRP, as well as of its 2019 extension with HWRoT – called PayBCR; we evaluated their efficiency and robustness of these payment protocols to relays attacks, in presence of both honest and rogue readers. Importantly, our experiments are the first experiments to show that both PayPass-RRP and PayBCR are actually practical protocols in offering this type of relay protection for EMV (for delays in the 5ms to 10ms range, which have deemed meaningful [9] in this application domain).

In future work, we would like to pursue a series of avenues, part of which are linked to more faithful modelling of the mobility of agents. For instance, we are keen to analyse more distance-bounding protocols where the mobility of parties can lead to (in)security concerns, in ways akin to the area of authenticated ranging.

ACKNOWLEDGMENTS

This work has been partially supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR) and by the NCSC-funded “TimeTrust” project. Also, we especially thank Consultant Hyperion (https://chyp.com/), and particularly Katie Facey and Haritha Gunniguntala from Consult Hyperion, for working with us to integrate PayPass-RRP in their CardCracker tool and giving us an 1-year licence for the tool.

REFERENCES

[1] [n.d.]. See supplementary files attached to this report.
To ease the proofs, we now extend the semantics with annotations. The annotations will feature labels identifying which process in the multiset has performed the action (session identifier). This will allow us to identify which specific agent performed some action. We also put in the annotation the global time at which the action has been done. In case of an output, the annotation will indicate the name of the handle w that has been used to store the output in the frame. In case of an input, the annotation will indicate a tuple (b, t, R) the name b of the agent responsible of the corresponding output, the time at which this output has been performed, as well as the recipe R used to build this output.

Formally, a label is either empty (for the TIM rule) or of the form (a, t, α) with α of the form: τ, gettime, claim(a₁, u₂, u₃, u₄), check(a₁, u₂, u₃), out(u₄), or in(u₄)). Thus, an annotated action is:

- empty for the TIM rule;
- (a, α, s, t, w) when the underlying label (a, t, α) is of the form (a, t, out(u₄)). In such a case, s is the session identifier of the agent responsible of this action, and w is the handle added in the frame.
- (a, α, s, t, (b, tₙ, R)) when the underlying label (a, t, α) is of the form (a, t, in(u₄)). In such a case, s is the session identifier of the agent responsible of this action, b is the agent responsible of the corresponding output, tₙ the time at which this output has been done (tₙ ≤ t), and R the recipe that has been used to forge this output.
- (a, α, s, t, Ø) otherwise.

In the untimed semantics, similar annotations can be added. Of course, in such a case, timing information are not relevant. Annotations are thus of the form (a, α, s, r) with either r = w (case of the output), r = (b, R) (case of the input), or r = Ø otherwise.

In order to establish Theorem 1, we prove two propositions: one for each direction of our main result.

A.1 From DB-security to causality

This section is devoted to the proof of the following proposition.

Proposition 1. Let P be a protocol and S be a set of valid initial configurations. Assuming that P is well-timed w.r.t. S, and P is DB-secure, we have that P is causality-based secure.

This proof is quite straightforward and consists in re-timing a witness of attack against the causally-based secure property. Re-timing an execution depends on the underlying mobility plan. We therefore introduce the notion of timed formula \( C^{\text{Loc}}_{\text{exec}} \) associated to an annotated relaxed execution exec, and a mobility plan Loc.

Given a trace \( t₁ \ldots t_n \) we note IN\( t₁ \ldots t_n \) the set of all the indices corresponding to input actions. Similarly we note TS\( t₁ \ldots t_n \) the set of all the indices corresponding to timestamp events. Given a set S we note \#S its size. Finally, we note \( \text{orig}(i) \) the index of the i\(^{th}\) output in the trace.

**Timed formula associated to an execution.**

Given an annotated execution \( \text{exec} = \mathcal{K}₀ = (P₀; ñ₀) \stackrel{t₁ \ldots t_n}{\longrightarrow} \mathcal{K}_n \) (with \( t₁ = (aᵢ, sᵢ, tᵢ, rᵢ) \) for \( 1 \leq i \leq n \)) of size \( n \) and a mobility plan Loc, the timed formula \( C^{\text{Loc}}_{\text{exec}} \), built upon the set of variables \( Z = \{z₁, \ldots, zₙ\} \cup \{zᵢ^b | i \in \text{IN}(t₁, \ldots, tₙ)\} \), is the conjunction of the following formulas:

- \( 0 \leq i ≤ z ≤ i ≤ zₙ \);
- \( z₁ < zᵢ \) for all \( i \in \text{TS}(t₁, \ldots, tₙ₋₁) \);
- for all \( i \in \text{IN}(t₁, \ldots, tₙ) \),
  \[ zᵢ ≥ zᵢ^b + \text{Dist}(\text{Loc}(aᵢ, zᵢ), \text{Loc}(bᵢ, zᵢ^b)) ; \]
- for all \( i \in \text{IN}(t₁, \ldots, tₙ) \), for all \( j \) such that \( wᵢ \in \text{vars}(Rᵢ) \) \( \text{dom}(ðᵢ₀) \),
  \[ zᵢ^b ≥ z_{\text{orig}(j)} + \text{Dist}(\text{Loc}(bᵢ, zᵢ^b), \text{Loc}(a_{\text{orig}(j)}, z_{\text{orig}(j)})) . \]
We note that given a configuration $\mathcal{K}$, the transformation $\tau$ defined in Section 4.2 is uniquely defined as soon as each variable is bound at most once in $\mathcal{K}$. In such a case, $\rho : x \mapsto c_x$ is a bijective renaming.

**Lemma 1.** Let $\mathcal{P}$ be a protocol and $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0; 0)$ be a valid initial configuration. Let $\operatorname{Loc}$ be a mobility plan.

For any execution $\operatorname{exec} = \overrightarrow{\mathcal{K}_0, t_1 \ldots t_n} \mathcal{K}_n$ with $t_1 = (a_i, a_i, s_i, r_i)$ and function $\phi$ satisfying $C_{\operatorname{Loc}}^{\operatorname{exec}}$ we have:

$$\mathcal{K}_0 \xrightarrow{t_1 \ldots t_n} \mathcal{K}_n$$

with:

$$t_i' = \begin{cases} (a_i, \text{gettime}, s_i, \phi(z_i), 0) & \text{if } a_i = \text{timestamp}(c_i) \\ (a_i, \alpha_{\neq c_i}, s_i, \phi(z_i), r_i) & \text{otherwise} \end{cases}$$

and $\varphi_c(z_i) = \phi(z_i)$ for all $i \in \text{TS}(t_1 \ldots t_n)$. Moreover, $\mathcal{K}_n \varphi_c = \mathcal{K}_n$.

**Proof.** The proof follows the definition of $C_{\operatorname{Loc}}^{\operatorname{exec}}$. Indeed this formula contains all the timing constraints to trigger each action. Moreover, by definition of $C_{\operatorname{Loc}}^{\operatorname{exec}}$ we obtain that $\varphi_c$ is a bijective function. This preserves equalities and inequalities between the untimed and the timed execution. 

We are now able to prove Proposition 1.

**Proposition 1.** We assume that $\mathcal{P}$ is well-timed and $\mathcal{P} \neq \text{DB-secure}$. We establish that $\mathcal{P}$ is not DB-secure. Since $\mathcal{P}$ is not causality-based secure, we know that there exist a valid initial configuration $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0; 0)$ and an annotated execution such that:

$$\operatorname{exec} = \overrightarrow{\mathcal{K}_0, t_1 \ldots t_n} (b_0, \text{claim}(b_1, b_2, c_1, c_2, s, 0)) = (\mathcal{P}', \phi')$$

with $t_i = (a_i, a_i, s_i, r_i)$ ($1 \leq i \leq n$) and $b_1 \notin \mathcal{M}$, $b_2 \notin \mathcal{M}$ and either:

1. there is no $k \leq n$ such that $a_k = \text{check}(c_1^0, c_2^0)$ for some $u \in T(\mathcal{E}, N \cup \Sigma_0)$; or
2. there is no $i \leq n$ (resp. $j \leq n$) such that $(a_i, a_i) = (b_1, \text{timestamp}(c_1^0))$ (resp. $(a_j, a_j) = (b_1, \text{timestamp}(c_2^0))$); or
3. there exists $b_0, b_0, b_0 \leq n$ and $u \in T(\mathcal{E}, N \cup \Sigma_0)$ such that $a_k = \text{check}(c_1^0, c_2^0, u)$, $a_k = (b_1, \text{timestamp}(c_1^0))$, and $(a_k, a_k) = (b_1, \text{timestamp}(c_2^0))$ but there is no $i \leq k \leq j_0$ such that $a_{k'} = b_2$.

Let $t_{n+1} = (b_0, \text{claim}(b_1, b_2, c_1, c_2, s, t, r))$. We consider each case separately.

**Case 1:** Let $\mathcal{Loc}$ be the mobility plan such that $\mathcal{Loc}(a, t) = (0, 0, 0)$ for any $a \in \mathcal{M}$ and $t \in \mathbb{R}_+$. Since the location of each agent does not depend on time, for sake of readability, we write $\mathcal{Loc}(a)$ instead of $\mathcal{Loc}(a, t)$.

Let $\phi$ be the function such that:

- $\phi(z_i) = \#\text{TS}(t_1 \ldots t_{i-1})$ for $i \in \{1, \ldots, n + 1\}$;
- for all $i \in \text{IN}(t_1 \ldots t_n)$ we have that:
  - $\phi(z_i^b) = \phi(\text{or}(j))$ if $b_i \notin \mathcal{M}$ and $R_i = w_j$;
  - $\phi(z_i^b) = \phi(z_i)$ otherwise.

We can show that this function $\phi$ satisfies $C_{\mathcal{Loc}}^{\operatorname{exec}}$. Indeed, we have that:

- $0 \leq \phi(z_1) \leq \phi(z_2) \leq \cdots \leq \phi(z_n)$;
- $\phi(z_i) < \phi(z_{i+1})$ for $i \in \text{TS}(t_1 \ldots t_{n-1})$;
- for all $i \in \text{IN}(t_1, \ldots, t_n)$, we have that either $\phi(z_i^b) = \phi(\text{or}(j)) \leq \phi(z_i)$ for some $j$ such that $\text{or}(j) \leq i$, or $\phi(z_i^b) = \phi(z_i)$. In both cases, we have that:
  $$\phi(z_i) \geq \phi(z_i^b) + \operatorname{Dist}((\mathcal{Loc}(a_i), \mathcal{Loc}(b_i)))$$

since $\operatorname{Dist}((\mathcal{Loc}(a_i), \mathcal{Loc}(b_i)) = 0$. Remember that all the agents are at the same location.

- for all $i \in \text{IN}(t_1, \ldots, t_n)$, for all $j$ such that $w_j \in \text{vars}(R_i) \setminus \text{dom}(\Phi_0)$, we have that $\text{or}(j) < i$, and thus $\phi(\text{or}(j)) \leq \phi(z_i)$. Therefore, we have that:
  $$\phi(z_i^b) \geq \phi(\text{or}(j))$$

and we conclude since $\operatorname{Dist}((\mathcal{Loc}(b_i), \mathcal{Loc}(a_i))) = 0$. Remember that all the agents are at the same location.

Finally, applying Lemma 1 we obtain that the execution can be lifted into the timed semantics. This timed execution immediately falsifies the DB-secure property because there is no check event in the timed execution. Indeed the timed execution is equal to the untimed one up to the times and the bijective function $\varphi_c$ defined in Lemma 1.

**Case 2:** Since $\mathcal{P}$ is well-timed w.r.t. $\mathcal{S}$, this case is not possible.

**Case 3:** Let $\mathcal{Loc}$ be the mobility plan such that:

- $\mathcal{Loc}(a, t) = (0, 0, 0)$ for any $a \notin \mathcal{M}$, and $t \in \mathbb{R}_+$;
- $\mathcal{Loc}(b_2, t) = (1, 0, 0)$ for any $t \in \mathbb{R}_+$.

Since the location of each agent does not depend on time, for sake of readability, we write $\mathcal{Loc}(a)$ instead of $\mathcal{Loc}(a, t)$.

Let $\phi$ be the function such that $\phi(z_i)$ is as follows:

- $2 \cdot \#\text{IN}(t_1 \ldots t_n) + \#\text{TS}(t_1 \ldots t_{n-1})$ for $i < i_0$;
- $\phi(z_{i_0-1}) + 1 + i_0 \#\text{IN}(t_1 \ldots t_{i-1})$ for $i_0 \leq i \leq j_0$;
- $\phi(z_{j_0} + 2 + \#\text{IN}(t_{j_0+1} \ldots t_{j-1}) + \#\text{TS}(t_{j_0+1} \ldots t_{j-1}))$ for $j > j_0$.

Informally, for all action outside the critical phase delimited by indices $i_0$ and $j_0$, we apply a delay of 2 before each input, and a delay of 1 after each timestamp. During the critical phase we do not apply delay before inputs and only apply a short delay of $1/(c_0 - n)$ after each timestamp to ensure that time increases between two timestamps as required by the semantics.

In addition, for all $i \in \text{IN}(t_1 \ldots t_n)$, if $b_i \notin \mathcal{M}$ then

$$\phi(z_i^b) = \phi(\text{or}(j))$$

where $R_i = w_j$, otherwise:

$$\phi(z_i^b) = \begin{cases} \phi(z_i) & \text{if } i < i_0 \text{ or } i > j_0 \\ \phi(z_i) & \text{if } i_0 \leq i \leq j_0. \end{cases}$$

We can show that this function $\phi$ satisfies $C_{\mathcal{Loc}}^{\operatorname{exec}}$. Indeed, we have that:

- $0 \leq \phi(z_1) \leq \phi(z_2) \leq \cdots \leq \phi(z_n)$;
- $\phi(z_i) < \phi(z_{i+1})$ for $i \in \text{TS}(t_1 \ldots t_{n-1})$;
- Regarding the remaining constraints in $C_{\mathcal{Loc}}^{\operatorname{exec}}$, we consider $i \in \text{IN}(t_1, \ldots, t_n)$, and we distinguish two sub-cases:
Case $i < i_0$ or $j_0 < j$: if the agent $b_1$ forging the received message is honest then we have that $r_i = w_j$ for some $j$ such that $or_{i}(j) < i$ and $a_{or_{i}(j)} = b_1$. Therefore, we have that:

$$\varphi(z_{i}) \geq \varphi(z_{i-1}) + 2$$
$$\geq \varphi(z_{or_{i}(j)}) + 2$$
$$\geq \varphi(z_{b_1}) + \text{Dist}(\text{Loc}(a_{or_{i}(j)}), \text{Loc}(b_1))$$

Moreover, $\varphi(z_{b_1}) = \varphi(z_{or_{i}(j)}) + \text{Dist}(\text{Loc}(a_j), \text{Loc}(b_1))$ since $a_{or_{i}(j)} = b_1$.

Otherwise, we have that $b_1 \in M$, and we have that $\varphi(z_{i}) = \varphi(z_{b_1}) + 1$ by definition of $\varphi$, and since $\text{Dist}(\text{Loc}(b_1), \text{Loc}(a_j)) \leq 1$, this allows us to conclude that:

$$\varphi(z_{i}) \geq \varphi(z_{b_1}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_j))$$

Now, let $w_j \in \text{vars}(R_i)$. We have that:

$$\varphi(z_{b_1}) \geq \varphi(z_{i}) - 1$$
$$\geq \varphi(z_{i-1}) + 2 - 1$$
$$\geq \varphi(z_{or_{i}(j)}) + 1$$
$$\geq \varphi(z_{or_{i}(j)}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)}))$$

Case $i_0 \leq i \leq j_0$: if the agent $b_1$ forging the received message is $b_2$ ($b_2$ is honest) then we have that $r_i = w_j$ for some $j$ such that $or_{i}(j) < i_0$ (remember that no action of $b_2$ occurs between $i_0$ and $j_0$). Therefore, we have that:

$$\varphi(z_{i}) \geq \varphi(z_{i_0-1}) + 1$$
$$\geq \varphi(z_{or_{i}(j)}) + 1$$
$$\geq \varphi(z_{b_2}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)}))$$

By definition of $\varphi$, we have that: $\varphi(z_{b_2}) \geq \varphi(z_{or_{i}(j)})$, and thus

$$\varphi(z_{b_2}) \geq \varphi(z_{or_{i}(j)}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)}))$$

since Dist(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)})) = 0 (we have that $b_1 = a_{or_{i}(j)}$).

Otherwise, if $b_1$ is honest but different from $b_2$, then we have:

$$\varphi(z_{i}) \geq \varphi(z_{i-1})$$
$$\geq \varphi(z_{or_{i}(j)})$$
$$\geq \varphi(z_{b_2}) + 0$$
$$\geq \varphi(z_{b_2}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)}))$$

By definition of $\varphi$, we have that $\varphi(z_{b_2}) = \varphi(z_{or_{i}(j)})$, and thus

$$\varphi(z_{b_2}) \geq \varphi(z_{or_{i}(j)}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)}))$$

since Dist(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)})) = 0 (we have that $b_1 = a_{or_{i}(j)}$).

Now, if $b_1 \in M$, we have that Loc($b_1$) = Loc($a_j$) (since $b_2$ is the only agent not located at the same place of the others, $b_2 \notin M$ and we know that there is no occurrence of an action performed by $b_2$ between $i_0$ and $j_0$, thus $a_i \neq b_2$). We have that $\varphi(z_{i}) = \varphi(z_{b_2})$ by definition of $\varphi$, and since Dist(\text{Loc}(b_1), \text{Loc}(a_j)) = 0, this allows us to conclude that:

$$\varphi(z_{i}) \geq \varphi(z_{b_2}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_j))$$

Now, let $w_j \in \text{vars}(R_i)$. In case $a_{or_{i}(j)} \neq b_2$, we have that Dist(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)})) = 0, and thus

$$\varphi(z_{b_2}) \geq \varphi(z_{or_{i}(j)}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)}))$$

In case $a_{or_{i}(j)} = b_2$, we know that $or_{i}(j) < i_0$, and thus

$$\varphi(z_{or_{i}(j)}) \leq \varphi(z_{b_2}) - 1 \leq \varphi(z_{i}) - 1$$

Thus, we have that:

$$\varphi(z_{i}) = \varphi(z_{i})$$
$$\geq \varphi(z_{or_{i}(j)}) + 1$$
$$\geq \varphi(z_{or_{i}(j)}) + \text{Dist}(\text{Loc}(b_1), \text{Loc}(a_{or_{i}(j)}))$$

Now, we have shown that $\varphi$ satisfies $C_{\text{Loc,exec}}$, we can apply Lemma 1 to re-time the execution. We obtain that:

$$K_0 \xrightarrow{tr'_1 \ldots tr'_n(b_1,\text{claim}(b_1, b_2, c_i^1, c_i^2, q_j, \varnothing))} \text{Loc} K'_{n+1}$$

with:

$$tr_i = \begin{cases} (a_i, \text{get time}, s_i, \varphi(z_{i})), \text{if } a_i = \text{timestamp}(c_i) \\
(a_i, c_i, s_i, \varphi(z_{i}), r_i) \text{ otherwise} \end{cases}$$

and $\varphi(c_i) = \varphi(z_{i})$ for all $i \in T(r_1 \ldots r_n)$.

By construction, we have that $\varphi(c_i^1) = \varphi(z_{b_1}), \varphi(c_i^2) = \varphi(z_{b_2})$, and, by definition of $\varphi$, we have that $\varphi(z_{i+1}) - \varphi(z_{i}) \leq 1/(c_0 \cdot n)$ when $i \in \{i_0, i_0 + 1, \ldots, j_0 - 1 \}$. Therefore, we have that:

$$\varphi(z_{j_0}) - \varphi(z_{i_0}) \leq (j_0 - i_0 + 1)/(c_0 \cdot n) \leq 1/c_0 \leq 2/\alpha \leq 2 \times \text{Dist}(\text{Loc}(b_1), \text{Loc}(b_2))$$

Hence, we have that $\mathcal{P}$ is not DB-secure.

A.2 From causality to DB-security

This section is devoted to the proof of the following proposition.

**Proposition 2.** Let $\mathcal{P}$ be protocol and $S$ a set of valid initial configurations. If $\mathcal{P}$ is causality-based secure w.r.t. $S$ then $\mathcal{P}$ is DB-secure w.r.t. $S$.

This implication is more complex than the previous one, and we start by establishing some lemmas allowing one to reorder some actions in a trace, and to derive timing constraints between dependent actions.

A.2.1 Preliminaries. First, we introduce the notion of dependence between actions.

**Definition 6.** Given an execution $K_0 \xrightarrow{L_1} \text{Loc} \cdots \xrightarrow{L_n} \text{Loc} K_n$ with $L_i = \{ a_1, a_2, s_i, t_i, r_i \}$. Let $J = \{ a_1, a_2, s_j, t_j, r_j \}$, we say that $L_j$ is dependent of $L_i$, denoted $L_j \leadsto L_i$, if $i < j$, and:

- either $s_j = s_j$ (and thus $a_1 = a_2$), and in that case $L_j$ is sequentially-dependent of $L_i$, denoted $L_j \leadsto_s L_i$;
- or $a_1 = \text{out}(v), a_j = \text{in}(u)$, and $r_i \in \text{vars}(R_i)$ with $r_j = \{b_j, t_j, R_j\}$, and in that case $L_j$ is data-dependent of $L_i$, denoted $L_j \leadsto_d L_i$.

We note $\leadsto^*$ the transitive closure of $\leadsto$. Finally we note $L_j \nleq^* L_i$ when $L_j$ is not dependent of $L_i$.

As established in [12] in a slightly different setting, we have the following result.
Lemma 2. Given an execution $\mathcal{K}_0 \xrightarrow{\ell_1} \mathcal{K} \xrightarrow{\ell_2} \mathcal{K}_2$ such that $L_2 \nrightarrow L_1$. We have that $\mathcal{K}_0 \xrightarrow{\ell_1} \mathcal{K}' \xrightarrow{\ell_1} \mathcal{K}_2$ for some $\mathcal{K}'$.

Proof. Let $\mathcal{K}_0 = (P_0, \Phi_0)$, $\mathcal{K} = (P, \Phi)$, and $\mathcal{K}_2 = (P_2, \Phi_2)$ be such that $\mathcal{K}_0 \xrightarrow{\ell_1} \mathcal{K} \xrightarrow{\ell_2} \mathcal{K}_2$ with $L_2 \nrightarrow L_1$. Let $L_1 = (a_1, a_2, s_1, r_1)$ and $L_2 = (a_2, s_2, r_2)$. Since $L_2 \nrightarrow L_1$ we have that $s_1 \neq s_2$. Therefore, we have that:

- $P_0 = \{\text{act}_1, P_{11} a_1 \cup \text{act}_2, P_{22} a_2 \} \cup Q$, where $\text{act}_1$ and $\text{act}_2$ are actions of the form $\text{in}(x)$, new $n$, out$(u)$, gettime$(x)$, let $x = v$ in $P$ else $Q$, claim$(u_1, u_2, u_3, u_4)$, or check$(u_1, u_2, u_3)$.

In case $a_1 = \text{out}(v)$ and $a_2 = \text{in}(u)$, we have that $P_2 = \Phi = \Phi_0 \cup \{w \overset{\overset{a_1}{\text{out}}}{\rightarrow} v\}$ and since $L_2 \nrightarrow L_1$, we know that $w \notin \text{vars}(r_2)$. Thus, we have that $\text{vars}(r_2) \subseteq \text{dom}(\Phi_0)$. Now, let $\mathcal{K}' = (\{\text{act}_1, P_{11} a_1 \cup P_{12} a_2 \} \cup Q, \Phi_0)$. Relying on the fact that $w \notin \text{vars}(r_2)$ in case $a_1 = \text{out}(v)$ and $a_2 = \text{in}(u)$, it is easy to see that $\mathcal{K}_0 \xrightarrow{\ell_1} \mathcal{K}' \xrightarrow{\ell_1} \mathcal{K}_2$. The other cases can be treated in a rather similar way.

Corollary 1. Given a trace $\mathcal{K}_0 \xrightarrow{t_{r_1} \ldots t_{r_n}} \mathcal{K}$ with $n \geq 2$ there exists a bijection $\varphi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ such that:

- $t_{r_1} = \varphi(i)$ for all $i \in \{1, \ldots, n\}$;
- for all $j$ such that $\varphi(1) < j < \varphi(n)$, we have that $t_{r_1} \varphi(j) \xrightarrow{\ast} t_{r_2} \varphi(j')$;
- for all $j_1, j_2$ such that $\varphi(1) \leq j_1 < j_2 \leq \varphi(n)$, we have that $\varphi^{-1}(j_1) < \varphi^{-1}(j_2)$; and

$\mathcal{K}_0 \xrightarrow{t_{r_1} \ldots t_{r_n}} \mathcal{K}$.

Proof. We split the proof in two parts: first we prove that there exists a bijection $\varphi_1$ cleaning the trace between $t_{r_1}$ and $t_{r_2}$ moving actions independent from $t_{r_1}$ before it. Then we prove that there exists a bijection $\varphi_2$ cleaning the trace moving actions from which $t_{r_n}$ does not depend on after it. Considering $\varphi = \varphi_2 \circ \varphi_1$ we will be able to conclude.

Claim 1. Given a trace $\mathcal{K}_0 \xrightarrow{t_{r_1} \ldots t_{r_n}} \mathcal{K}$ with $n \geq 1$ there exists a bijection $\varphi_1 : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ such that:

- $t_{r_1} = \varphi_1(i)$ for all $i \in \{1, \ldots, n\}$;
- for all $j > \varphi_1(1)$, $t_{r_1} \varphi_1(j) \xrightarrow{\ast} t_{r_1} \varphi_1(j')$;
- for all $j_1, j_2$ such that $\varphi_1(1) \leq j_1 < j_2 \leq \varphi_1(n)$, we have that $\varphi_1^{-1}(j_1) < \varphi_1^{-1}(j_2)$; and

$\mathcal{K}_0 \xrightarrow{t_{r_1} \ldots t_{r_n}} \mathcal{K}$.

Proof. We make the proof by induction on the length of the execution. If $n = 1$ then the results holds considering $\varphi_1 = \text{id}$ and $t_{r_1} = t_{r_1}$.

Otherwise, let $\mathcal{K}_0 \xrightarrow{t_{r_1} \ldots t_{r_n}} \mathcal{K}_n \xrightarrow{t_{r_{n+1}}} \mathcal{K}_{n+1}$ be an execution. By induction hypothesis we have that there exists a bijection $\varphi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ such that:

- $t_{r_1} = \varphi(i)$ for all $i \in \{1, \ldots, n\}$;
- for all $j$ such that $\varphi(1) < j \leq n$, we have $t_{r_1} \varphi(j) \xrightarrow{\ast} t_{r_1} \varphi(j)$;

- for all $j_1, j_2$ such that $\varphi(1) \leq j_1 < j_2 \leq \varphi(n)$, we have that $\varphi^{-1}(j_1) < \varphi^{-1}(j_2)$; and

- for all $j_1, j_2$ such that $\varphi(1) \leq j_1 < j_2 \leq n$, we have that $\varphi^{-1}(j_1) < \varphi^{-1}(j_2)$.

For sake of uniformity, let $t_{r_1} = t_{r_1}$, and we extend $\varphi_1$ on $\{1, \ldots, n\}$ as follows: $\varphi_1(n) = n$.

Then we apply Claim 2 on the resulting execution starting at $t_{r_1} \varphi_1(n) + 1$ to obtain that there exists $\varphi_2 : \{\varphi_1(n) + 1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ such that:

- $t_{r_1} = \varphi_2(i)$ for all $i \in \{1, \ldots, n\}$;
- for all $j$ such that $\varphi_1(n) + 1 < j \leq n$, we have $t_{r_1} \varphi_2(j) \xrightarrow{\ast} t_{r_1} \varphi_2(j)$;

- for all $j_1, j_2$ such that $\varphi_1(n) + 1 \leq j_1 < j_2 \leq n$, we have that $\varphi_2^{-1}(j_1) < \varphi_2^{-1}(j_2)$; and

- for all $j_1, j_2$ such that $\varphi_1(n) + 1 \leq j_1 < j_2 \leq n$, we have that $\varphi_2^{-1}(j_1) < \varphi_2^{-1}(j_2)$.

Claim 2. Given a trace $\mathcal{K}_0 \xrightarrow{t_{r_1} \ldots t_{r_n}} \mathcal{K}_n$ with $n \geq 1$ there exists a bijection $\varphi_2 : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ such that:

- $t_{r_1} = \varphi_2(i)$ for all $i \in \{1, \ldots, n\}$;
- for all $j < \varphi_2(n)$, we have that $t_{r_1} \varphi_2(j) \xrightarrow{\ast} t_{r_1} \varphi_1(j)$;
- for all $j_1, j_2$ such that $1 \leq j_1 < j_2 \leq \varphi_2(n)$, we have that $\varphi_2^{-1}(j_1) < \varphi_2^{-1}(j_2)$;

$\mathcal{K}_0 \xrightarrow{t_{r_1} \ldots t_{r_n}} \mathcal{K}_n$.

Proof. We follow a very similar proof to the previous claim. We first apply the induction hypothesis to $\mathcal{K}_1 \xrightarrow{t_{r_2} \ldots t_{r_n}} \mathcal{K}_{n-1}$ and we obtain $\varphi : \{2, \ldots, n\} \rightarrow \{2, \ldots, n\}$ (a shift of 1 has been applied to ease the reasoning). If $t_{r_1} \varphi_2(n) = t_{r_2}(n)$ then we conclude considering $\varphi_2 = \varphi \cup \{1 \mapsto 1\}$. Otherwise, by repeatedly applying Lemma 2 we can move $t_{r_1}$ at the right place in the trace, i.e. just after $t_{r_1} \varphi_2(n)$.
The next lemma consists in transforming a timed execution into an untimed one. Even if this transformation can be done in a rather straightforward way, we state it with some details in order to maintain a strong relationship between the two executions.

To do so, we first precise the transformation presented in Section 4.2. When we apply this transformation, we rely on the function $\sigma_{\text{spe}} : X \rightarrow \Sigma_{\text{spe}}$ which is used to replace an action of the form $\text{gettime}(x)$ by the action timestamp($\sigma_{\text{spe}}(x)$).

Similarly, given an execution $\text{exec} = \mathcal{K}_0 \xrightarrow{\text{tr}_1 \ldots \text{tr}_n} \mathcal{K}_n$, we denote $\sigma_{\text{time}} : X \rightarrow \mathbb{R}_+$ the function that associates to each variable occurring in a gettime instruction, the current time at which this instruction has been executed.

**Lemma 3.** Let $\mathcal{P}$ be a protocol and $\mathcal{K}_0$ be a valid initial configuration for $\mathcal{P}$. For any execution

$$\text{exec} = \mathcal{K}_0 \xrightarrow{\text{tr}_1 \ldots \text{tr}_n} \mathcal{K}_n$$

such that $\text{tr}_i = (a_i, a_i, s_i, t_i, r_i)$ for $i \in \{1, \ldots, n\}$, we have that $\mathcal{K}_0 \xrightarrow{\text{tr}_1 \ldots \text{tr}_n} \mathcal{K}_n$ where $\mathcal{K}_n = \mathcal{K}_0 \sigma$ and for any $i \in \{1, \ldots, n\}$, we have:

$$\text{tr}_i = \begin{cases} (a_i, \text{timestamp}(t_1, \sigma), s_i, 0) & \text{if } a_i = \text{gettime} \\ (a_i, a_i, a_i, s_i, (b_i, R_i)) & \text{if } r_i = (b_i, L^2_i, \sigma) \\ (a_i, a_i, a_i, s_i, (b_i, r_i, \sigma)) & \text{otherwise} \end{cases}$$

where $\sigma = \sigma_{\text{spe}} \circ \sigma_{\text{time}}^{-1}$ assuming that $\sigma_{\text{spe}}$ is the function used to transform $\mathcal{K}_0$ into $\mathcal{K}_n$ and $\sigma_{\text{time}}$ is the one associated to the execution exec.

**Proof.** This proof is immediate because the configurations only differ from the bijective function $\sigma$ (the equalities are thus preserved) and the rules in the untimed semantics are less restrictive than the rules in the timed semantics.

The following lemma gives us some constraints about dependent actions. Indeed given two actions $\text{tr}_1$ and $\text{tr}_2$ such that $\text{tr}_2 \xrightarrow{\ast} \text{tr}_1$ we know that enough time must has elapsed after the execution of $\text{tr}_1$ to be able to trigger $\text{tr}_2$.

**Lemma 4.** Let $\mathcal{L}$ be a mobility plan, and $\text{exec} = \mathcal{K}_0 \xrightarrow{\text{tr}_1 \ldots \text{tr}_n} \mathcal{K}_n$ be an execution with $\text{tr}_i = (a_i, a_i, s_i, t_i, r_i)$ for $i \in \{1, \ldots, n\}$. Let $i, j \in \{1, \ldots, n\}$ such that $\text{tr}_j \xrightarrow{\ast} \text{tr}_i$. We have that:

$$t_j \geq t_i + \text{Dist}(%(\mathcal{L}(a_i, t_i), \mathcal{L}(a_j, t_j))).$$
In such a case, we have that

\[ tr_j = tr_{t_1} \iff tr_{t_2} \iff \ldots \iff tr_{t_k} = tr_t. \]

We do the proof by induction on the length of this sequence. If \( k = 1 \) then \( tr_j = tr_t \) and thus \( t_j = t_t \) and \( a_i = a_j \). The result trivially holds. Otherwise, we have that:

\[ tr_j = tr_{t_1} \iff tr_{t_2} \iff \ldots \iff tr_{t_k} = tr_t. \]

By induction hypothesis, we have that:

\[ t_i \geq t_t + \text{Dist}(\text{Loc}(a_i, t_1), \text{Loc}(a_i, t_2)). \]

We distinguish two cases depending on the nature of the dependency \( tr_j \iff tr_{t_i} \).

Case \( tr_j \iff t_{t_i} \). In such a case, we have that \( a_i = a_{t_i} \) and \( j \geq t_i \). Moreover, we have that \( t_j \geq t_{t_i} \), and by definition of a mobility plan we know that

\[ t_j - t_{t_i} \geq \text{Dist}(\text{Loc}(a_j, t_j), \text{Loc}(a_j, t_{t_i})). \]

Relying on our induction hypothesis, we have that:

\[ t_j \geq t_{t_i} + \text{Dist}(\text{Loc}(a_j, t_j), \text{Loc}(a_j, t_{t_i})) \]

\[ \geq t_{t_i} + \text{Dist}(\text{Loc}(a_j, t_{t_i}), \text{Loc}(a_j, t_{t_i})) \]

\[ \geq t_{t_i} + \text{Dist}(\text{Loc}(a_j, t_{t_i}), \text{Loc}(a_j, t_{t_i})). \]

Note that the last inequality comes from the fact Dist is a distance, and thus satisfies the triangle inequality.

Case \( tr_j \iff t_{t_i} \). In such a case, we have that \( r_j = (b, t_b, R), r_{t_i} = w \) and \( w \in \text{vars}(R) \). By definition of the IN rule we have that:

- \( t_j \geq t_b + \text{Dist}(\text{Loc}(a_j, t_j), \text{Loc}(b, t_b)), \) and
- \( t_b \geq t_{t_i} + \text{Dist}(\text{Loc}(b, t_b), \text{Loc}(a_j, t_{t_i})). \)

Combining these two inequalities together with the triangle inequality we obtain:

\[ t_j \geq t_{t_i} + \text{Dist}(\text{Loc}(a_j, t_j), \text{Loc}(a_j, t_{t_i})). \]

Finally, relying on our induction hypothesis, we have that:

\[ t_j \geq t_{t_i} + \text{Dist}(\text{Loc}(a_j, t_j), \text{Loc}(a_j, t_{t_i})). \]

This concludes the proof. □

A.2.2 Proposition 2. We are now able to prove Proposition 2. The proof starts with an attack trace w.r.t. DB-security such that \( t^0_2 - t^0_1 < \delta \) (for some threshold \( \delta \)) and then follows the following steps:

1. We first apply Lemma 3 to weaken the trace in the untimed semantics.
2. Then, we apply Corollary 1 to clean up the trace between the two timestamp actions.
3. We applyLemma 1 to lift this execution in the timed model keeping the value \( t^0_2 - t^0_1 \) unchanged.
4. Assuming that the protocol is causality-based secure, there is still an action executed by the prover between the two timestamps. By construction this action depends on the two timestamps. Applying Lemma 4, we will therefore obtain that \( t^1_2 - t^1_1 \geq \delta \) leading to contradiction.

Proposition 2. Let \( P \) be protocol and \( S \) a set of valid initial configurations. If \( P \) is causality-based secure w.r.t. \( S \) then \( P \) is DB-secure w.r.t. \( S \).

Proof. We assume that \( P \) is not DB-secure, and thus there exist a valid initial configuration \( K_0 \in S \) and an execution exec such that:

\[ \text{exec} = \text{exec}' = K_0 \rightarrow K_{n+1} \]

with \( b_1 \notin M \) and \( b_2 \notin M \) and either:

1. there is no index \( k \leq n \) such that \( tr_k = (a_k, \text{check}(t^0_k, t^0_k, t^0_k, s_k, t_k, 0)) \); or
2. for any \( i \) with \( t^0_i < t^0_k \), we have that:

\[ t^0_i - t^0_k < \text{Dist}(\text{Loc}(b_i, t^0_i), \text{Loc}(b_i, t^0_k)) \]

Below, we note \( tr_i = (a_i, a_i, s_i, t_i, r_i) \) for \( i \in \{1, \ldots, n\} \). First, we apply Lemma 3. Therefore, we have that:

\[ \text{exec}' = \text{exec}'' = K_0 \rightarrow K_{n+1} \]

where \( K'_{n+1} = K_{n+1} \) and for any \( i \in \{1, \ldots, n+1\} \), we have that:

\[ tr_i = \begin{cases} (a_i, \text{timestamp}(t_i, \sigma), s_i, 0) & \text{ if } a_i = \text{get time} \\ (a_i, a_i, s_i, (b_i, R_i)) & \text{ if } r_i = (b_i, t^0_i, R_i) \\ (a_i, a_i, s_i, r_i, \sigma) & \text{ otherwise} \end{cases} \]

where \( \sigma = \sigma_{\text{spe}} \circ \sigma^{-1}_{\text{time}} \) assuming that \( \sigma_{\text{spe}} \) is the function used to transform \( K_0 \) into \( K_0 \) and \( \sigma_{\text{time}} \) is the one associated to the execution exec.

We assume by contradiction that \( P \) is causality-based secure, thus we know that there exist \( i_0, j_0, k, k' \leq n \) with \( i_0 \leq k' \leq j_0 \), and \( u \in T(\Sigma, \mathcal{N} \cup \Sigma_t^\tau) \) such that:

- \( a_k \sigma = \text{check}(t^0_1, t^0_2, u, s_k, t_k, 0) \);
- \( (a_k, t_k \sigma) = (b_i, \text{timestamp}(t_i \sigma)) \);
- \( (a_{k'}, t_k \sigma) = (b_i, \text{timestamp}(t_i \sigma)) \);
- \( a_k = b_2 \).

By definition of \( \text{exec}' \), we have that \( tr_k = (a_k, \text{check}(t^0_1, t^0_2, u), s_k, t_k, 0) \) for some time \( t_k \in \mathbb{R}_+, \) which leads to a contradiction with item 1). Thus, we can assume from now that the condition stated in item 2) holds.

Now, we apply Corollary 1 to the sub-execution \( K_{n-1} \rightarrow K_{n-1} \rightarrow K_{n} \) of \( \text{exec}' \), and we obtain that there exists a bijection \( \varphi_{\text{act}} : \{i_0, \ldots, j_0\} \rightarrow \{i_0, \ldots, j_0\} \) such that:

\[ tr_i' = tr''_{\varphi_{\text{act}}} (i) \quad \text{for all } i \in \{i_0, \ldots, j_0\} \]

\[ \text{for all } j \text{ such that } \varphi_{\text{act}}(i) < j < \varphi_{\text{act}}(j) \text{, we have that} 
\]

\[ tr''_{\varphi_{\text{act}}}(i) = tr''_{\varphi_{\text{act}}}(j) \]

\[ \text{for all } j, j' \text{ such that } \varphi_{\text{act}}(i) \leq j' < j \leq \varphi_{\text{act}}(j) \text{, we have that} \]

\[ \varphi_{\text{act}}(j) < \varphi_{\text{act}}(j') \text{ and} \]

\[ K_{n-1} \rightarrow K_{n} \rightarrow K_{n+1} \]
We have thus an execution $exec''$ such that:

$$exec'' = \frac{tr_i^1 \ldots tr_i^n}{K_0} \frac{tr_j^1 \ldots tr_j^n}{K_j} \frac{tr_j^1 \ldots tr_j^n}{b_i, claim(b_i, b_j, t_j^1, \sigma, s, j)} \frac{\sigma}{K_j} \frac{\sigma}{K_j} = K_{n+1}.$$

For sake of simplicity, for $i < i_0$ and $i > j_0$ we define $tr_i^j = tr_i$. For all $i \in \{1, \ldots, n\}$ we define $a_{\phi}(i)$ as the name of the agent executing $tr_i^j$. If $i \in \text{In}(tr_j^1 \ldots tr_j^n)$, we also define $R''_i$ (resp. $b''_i$) the recipe (resp. agent name) occurring in $tr_i^j$. We have that $a''_{\phi}(i) = a_i$, $R''_i = R_i \sigma$ and $b''_i = b_i$.

In the following we aim at re-timing the trace $exec''$ without changing the amount of time that elapses between the two time-stamps instructions. Once this is done, the dependencies

$$tr''_j \leftarrow \text{tr}_j^i \leftarrow \text{tr}_j'^{\phi_{ac}(i)}$$

for any $j$ such that $\phi_{ac}(i) < j < \phi_{ac}(j)$ together with the fact that there exists such a $j$ such that $a_j = b_j$ (since $P$ is assumed to be causality-based secure) will lead us to a contradiction thanks to Lemma 4. The remaining of the proof is there to formalise this idea.

We start by defining a function $\phi$ and a mobility plan $\text{Loc}'$ such that $\phi$ satisfies $C_{exec'}$. Let $A_\phi$ be the set of all the agents involved in the execution $exec'$ (i.e. executing an action or forging a message used to fill an input). We denote $\delta(t)$ the maximal distance w.r.t. Loc among two agents in $A_\phi$ at time $t$. i.e.

$$\delta(t) = \max\{\text{Dist}(\text{Loc}(a, t), \text{Loc}(b, t)) \mid a, b \in A_\phi\}.$$

We define $\phi$ as follows for $i \in \{1, \ldots, n\}$:

$$\phi(z_i) = \begin{cases} 2 \times \delta(t_{i_0}) \times \#\text{In}(tr_i'^{\phi} \ldots tr_i'^{\phi}) + \#\text{TS}(tr_i'^{\phi} \ldots tr_i'^{\phi}) & \text{if } i < \phi_{ac}(i) \\ t_{\phi_{ac}(i)} + \Delta & \text{if } \phi_{ac}(i) \leq i \leq \phi_{ac}(j) \\ \phi(z_{\phi_{ac}(i)} + 1 + 2 \times \delta(t_{j_0}) \times \#\text{In}(tr_i'^{\phi} \phi_{ac}(i)+1 \ldots tr_i'^{\phi}) & \text{if } i > \phi_{ac}(j) \end{cases}$$

with $\Delta = \phi(z_{\phi_{ac}(i)-1}) + 2 \times \delta(t_{i_0}) + 1$.

In addition, for all $i \in \text{In}(tr_i'^{\phi} \ldots tr_i'^{\phi})$, if $b_i' \in A \setminus M$, we define $\phi_{\phi_{ac}(i)-1}(i)$ the current frame when executing the action $tr_i'^{\phi_{ac}(i)}$ in $exec'$ and we define $\phi(z_i')$ as follows:

$$\phi(z_i') = \begin{cases} \phi(z_i) - \delta(t_{i_0}) & \text{if } i < \phi_{ac}(i) \\ \phi(z_{\phi_{ac}(i)}) - \delta(t_{i_0}) & \text{if } \phi_{ac}(i) \leq i \leq \phi_{ac}(j) \\ \phi_{\phi_{ac}(i)} + \Delta & \text{if } \phi_{ac}(i) \leq i \leq \phi_{ac}(j) \end{cases}$$

We consider the mobility plan $\text{Loc}'$ where $\text{Loc}'(a, t)$ is equal to:

$$\begin{align*}
\text{Loc}(a, t_{i_0}) & \quad \text{if } t \leq \phi(z_{\phi_{ac}(i)})(= t_{i_0} + \Delta) \\
\text{Loc}(a, t - \Delta) & \quad \text{if } \phi(z_{\phi_{ac}(i)})(< t < \phi(z_{\phi_{ac}(i)})) \\
\text{Loc}(a, t_{j_0}) & \quad \text{if } t \geq \phi(z_{\phi_{ac}(j)})(= t_{j_0} + \Delta).
\end{align*}$$

We may note that $\text{Loc}'$ is indeed a mobility plan. Now, we show that $\phi$ satisfies $C_{exec'}$.

We first establish that the two first items are indeed satisfied, i.e.

- $0 \leq \phi(z_1) \leq \phi(z_2) \leq \ldots \leq \phi(z_n)$; and
- $\phi(z_i) < \phi(z_{i+1})$ for all $i \in \text{TS}(tr_i'^{\phi} \ldots tr_{i-1}').$

First, we note that $0 \leq \phi(z_1)$, and $\phi(z_i) \leq \phi(z_{i+1})$ on the three intervals $1 \leq \phi(z_{\phi_{ac}(i)}), \phi(z_{\phi_{ac}(i)}) \leq i \leq \phi_{ac}(j_0)$ and $i > \phi_{ac}(j_0)$. In addition, by definition of $\phi$, we have that:

- $\phi(z_{\phi_{ac}(i)}) = t_{i_0} + \Delta > \phi(z_{\phi_{ac}(i)-1})$; and
- $\phi(z_{\phi_{ac}(j)+1}) > \phi(z_{\phi_{ac}(j)}).$

This concludes the proof regarding the first item. Now, we consider item 2. Let $i \in \text{TS}(tr_i'^{\phi} \ldots tr_{i-1}').$ We distinguish several cases:

- Case $i < \phi_{ac}(i) - 1$ or $i > \phi_{ac}(i)$. In such a case, we have that
  $$\phi(z_{i+1}) - \phi(z_i) = \#\text{TS}(tr_i'^{\phi} \ldots tr_{i-1}') > 0.$$

- Case $i = \phi_{ac}(i) - 1$ or $i = \phi_{ac}(i)$. In such a case, we have seen just before that:
  $$\phi(z_{\phi_{ac}(i)}) - \phi(z_i) = \phi(z_{\phi_{ac}(i)-1}) > 0.$$

- Case $\phi_{ac}(i) \leq i < \phi_{ac}(j)$. In such a case, Corollary 1 (item 3) tells us that $\phi_{\phi_{ac}(i)}(i) < \phi_{\phi_{ac}(i)+1}(i)$, and thus $\phi_{\phi_{ac}(i)}(i) + 1 \leq \phi_{\phi_{ac}(i)+1}(i)$. From these two inequalities, and since time is non decreasing during an execution, we deduce that:
  $$\phi(z_i) < \phi(z_{i+1}).$$

Note that the strict inequality comes from the fact that $\phi_{\phi_{ac}(i)}(i)$ corresponds to an application of GTIM. Hence, we have that $\phi(z_i) < \phi(z_{i+1})$. Therefore, we deduce that:
  $$\phi(z_{i+1}) - \phi(z_i) = (t_{\phi_{ac}(i)}(i+1) + \Delta) - (t_{\phi_{ac}(i)}(i) + \Delta) > 0.$$

This allows us to conclude regarding item 2.

We now establish that the two last items are also satisfied, i.e. for any $i \in \text{In}(tr_i'^{\phi} \ldots tr_{i-1}'')$,

- $\phi(z_i) \geq \phi(z_i') + \text{Dist}(\text{Loc}'(a, \phi(z_i)), \text{Loc}'(b_i', \phi(z_i')))$; and
- for all $j$ such that $w_j \in \text{vars}(R''_i) \setminus \text{dom}(\phi_i)$, $\phi(z_i') \geq \phi(z_{\phi_{ac}(j)}) + \text{Dist}(\text{Loc}'(b_i', \phi(z_i'))), \text{Loc}'(a, \phi(z_{\phi_{ac}(j)})))$.

We distinguish several cases.

Case $i < \phi_{ac}(i)$. In such a case, by definition of Loc', for any $a \in A$ and $t \leq \phi(z_{\phi_{ac}(i)})$, we have that $\text{Loc}'(a, t) = \text{Loc}(a, t_{i_0})$. Moreover, a delay of at least $2 \times \delta(t_{i_0})$ is applied between an input and any action occurring before it. By consequence, for all
This case is similar to the previous one. Note that we have seen that:
\[
\phi(z_i) \geq \phi(z_{\text{orig}(j)}) + 2 \times \delta(t_i).
\]
In addition, since \( \text{orig}(j) < \phi_{\text{act}(i)} \), thanks to item 1, we know that:
\[
\phi(z_{\text{orig}(j)}) \leq \phi(z_i) \leq \phi(z_{\text{act}(i)}).
\]
Therefore, we have that:
\[
\phi(z_i) \geq \phi(z_{\text{orig}(j)}) + 2 \times \delta(t_i).
\]
\[
\phi(z_{\text{orig}(j)}) + \text{Dist}(\text{Loc}(a''_{\text{orig}(j)}, t_i), \text{Loc}(a'', t_i)) \geq \phi(z_i).
\]
Hence we have:
\[
\phi(z_i) \geq \phi(z_{\text{orig}(j)}) + 2 \times \delta(t_i).
\]
\[
\phi(z_{\text{orig}(j)}) + \text{Dist}(\text{Loc}(a''_{\text{orig}(j)}, t_i), \text{Loc}(a'', t_i)) \geq \phi(z_i).
\]
\[
\phi(z_i) \geq \phi(z_{\text{orig}(j)}) + 2 \times \delta(t_i).
\]
\[
\phi(z_{\text{orig}(j)}) + \text{Dist}(\text{Loc}(a''_{\text{orig}(j)}, t_i), \text{Loc}(a'', t_i)) \geq \phi(z_i).
\]
Finally, since \( \text{Loc}^* \) is a mobility plan we have that:
\[
\phi(z_i) - \phi(z_{\text{orig}(j)}) \geq \text{Dist}(\text{Loc}(a''_{\text{orig}(j)}, \phi(z_i)), \text{Loc}(a'', \phi(z_{\text{orig}(j)}))).
\]
Combining these two inequalities, and using the triangle inequality, we deduce that:
\[
\phi(z_i) \geq \phi(z_{\text{orig}(j)}) + \text{Dist}(\text{Loc}(b''_{\phi}(t_i), \phi(z_i)), \text{Loc}(a'', \phi(z_{\text{orig}(j)}))).
\]
Now, if \( b'' \in M \), then we have that \( \phi(z_i) \geq \phi(z_{\text{orig}(j)}) \) and \( a''_{\text{orig}(j)} = b'' \). Therefore, from the previous inequality, we deduce the result:
\[
\phi(z_i) \geq \phi(z_{\text{orig}(j)}) + \text{Dist}(\text{Loc}(b''_{\phi}(t_i), \phi(z_i)), \text{Loc}(a'', \phi(z_{\text{orig}(j)}))).
\]
If \( b'' \in M \), then we have that \( \phi(z_i) = \phi(z_{\text{orig}(j)}) + \delta(t_i) \). For the same reasons as before we have that:
\[
\phi(z_i) \geq \phi(z_{\text{orig}(j)}) + \delta(t_i).
\]
\[
\phi(z_{\text{orig}(j)}) + \text{Dist}(\text{Loc}(a''_{\text{orig}(j)}, t_i), \text{Loc}(a'', t_i)) \geq \phi(z_i) + \delta(t_i).
\]
Therefore, we have:
\[
\phi(z_i) \geq \phi(z_{\text{orig}(j)}) + \delta(t_i).
\]
\[
\phi(z_{\text{orig}(j)}) + \text{Dist}(\text{Loc}(a''_{\text{orig}(j)}, t_i), \text{Loc}(a'', t_i)) \geq \phi(z_i) + \delta(t_i).
\]
Regarding item 3, we have that:
\[
\phi(z_i) = \phi(z_{\text{act}(i)}) + \delta(t_i).
\]
\[
\phi(z_{\text{act}(i)}) + \text{Dist}(\text{Loc}(a''_{\phi}(t_i), \phi(z_i)), \text{Loc}(a'', \phi(z_{\text{act}(i)}))).
\]
Combining these two inequalities, we obtain:
\[
\phi(z_i) \geq \phi(z_{\text{act}(i)}) + \text{Dist}(\text{Loc}(a''_{\phi}(t_i), \phi(z_i)), \text{Loc}(a'', \phi(z_{\text{act}(i)}))).
\]
This concludes this case.
We know also that there exists $k$ such that $w_k \in \text{vars}(R'') \setminus \text{dom}(\varphi''_{\text{act}(i_k)})$.

Actually, we have that
\[
\varphi(z'_b) = t^b_{\varphi''_{\text{act}(i)}} + \Delta.
\]

This equality is trivial when $b'' \in \mathcal{M}$ and also satisfied when $b'' \in \mathcal{N} \setminus \mathcal{M}$. Indeed, in this last case, we have that $\varphi(z'_b) = \varphi(z_{\text{orig}(k)})$.

Since $\text{orig}(k) > \varphi_{\text{act}(i_k)}$ we know that $\varphi(z_{\text{orig}(k)}) = t^b_{\varphi''_{\text{act}(i)}} + \Delta$ and since we have that $t^b_{\varphi''_{\text{act}(i)}} = t^b_{\varphi''_{\text{act}(i)}}$ we conclude that
\[
\varphi(z'_b) = t^b_{\varphi''_{\text{act}(i)}} + \Delta.
\]

Moreover, we know that $t^b_{\varphi''_{\text{act}(i)}} \geq t_b$ since the input has been executed in the initial execution $\text{exec}$ and the output corresponding to $w_k$ in $\text{exec}$ must have been executed after $t_b$ (Remember that the trace has been clean up between $i_0$ and $j_0$ only). Thus, we have that
\[
\varphi(z'_b) = t^b_{\varphi''_{\text{act}(i)}} + \Delta \geq t + \Delta = \varphi(z_{\text{act}(i_k)}).
\]

Therefore, we deduce that:
\[
\text{Loc}'(b''_1, \varphi(z'_b)) = \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}} + \Delta) = \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}}).
\]

Since the IN rule has been triggered in $\text{exec}$ we have that:
\[
t^b_{\varphi''_{\text{act}(i)}} \geq t^b_{\varphi''_{\text{act}(i)}} + \text{Dist} = \text{Loc}(a''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}}))
\]

Therefore, regarding item 3, we have that:
\[
\varphi(z') = t^b_{\varphi''_{\text{act}(i)}} + \Delta \\
\geq t^b_{\varphi''_{\text{act}(i)}} + \text{Dist} = \text{Loc}(a''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}})) + \Delta \\
= \varphi(z'_b) + \text{Dist} = \text{Loc}(a''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}})) \\
= \varphi(z'_b) + \text{Dist} = \text{Loc}(a''_1, \varphi(z'_b)), \text{Loc}(b''_1, \varphi(z'_b))
\]

This concludes the proof regarding item 3.

Regarding item 4, let $w \in \text{vars}(R'')$. We know that $w \in \text{vars}(R''_{\varphi_{\text{act}(i_k)}})$, and $(w \xrightarrow{c \xrightarrow{t_c}} u) \in \Phi$ (with $\Phi$ the current frame when executing the input in $\text{exec}$). If $c \neq *$ then
\[
t^b_{\varphi''_{\text{act}(i)}} \geq t + \text{Dist} = \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(c, t_b)).
\]

By construction, we have that $(w \xrightarrow{c \xrightarrow{t_c}} u') \in \Phi'$ (with $\Phi'$ the current frame when executing the input in $\text{exec}'$) with either $t'_c \leq \varphi(z_\text{act(ik)})$ or $t'_c = t_c + \Delta$.

In the first case we have that $\text{Loc}'(c, t'_c) = \text{Loc}(c, t_b)$. By definition of a mobility plan we know that
\[
t_c - t_b \geq \text{Dist} = \text{Loc}(c, t_b, \text{Loc}(c, t_b)).
\]

Combining this inequality with the previous one and using the triangle inequality, we obtain:
\[
t^b_{\varphi''_{\text{act}(i)}} \geq t_b + \text{Dist} = \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(c, t_b)).
\]

Thus, we have that:
\[
\varphi(z'_b) = t^b_{\varphi''_{\text{act}(i)}} + \Delta \\
\geq t_b + \text{Dist} = \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(c, t_b)) + \varphi(z_{\text{act}(i_k)}) + 2 \times (t_b + 1) \\
= t'_c + \text{Dist} = \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(c, t_b)) \\
= t'_c + \text{Dist} = \text{Loc}(b''_1, \varphi(z'_b), \text{Loc}'(c, t'_c)) \\
= t'_c + \text{Dist} = \text{Loc}(b''_1, \varphi(z'_b), \text{Loc}(c, t'_c))
\]

In the second case, we have that $t'_c = t_c + \Delta$ and thus $\text{Loc}'(c, t'_c) = \text{Loc}(c, t_c)$. Therefore, we have that:
\[
\varphi(z'_b) = t^b_{\varphi''_{\text{act}(i)}} + \Delta \\
\geq t_c + \text{Dist} = \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(c, t_c)) + \Delta \\
= t'_c + \text{Dist} = \text{Loc}(b''_1, \varphi(z'_b), \text{Loc}(c, t'_c)) \\
= t'_c + \text{Dist} = \text{Loc}(b''_1, \varphi(z'_b), \text{Loc}(c, t'_c))
\]

This allows us to conclude regarding item 4.

In conclusion we have that Lemma 1 applies with $\text{exec}'$ and $\varphi$. We have thus a timed execution $\text{exec}_{\text{time}}$ in the mobility plan $\text{Loc}'$ such that
\[
\text{exec}_{\text{time}} = \text{Loc}(a''_1, t^b_{\varphi''_{\text{act}(i)}}, \text{Loc}(b''_1, t^b_{\varphi''_{\text{act}(i)}}))
\]

with $\varphi(z_c) = \varphi(z_i)$ for all $i \in \text{TS}(t''_1 \ldots t''_n)$.

Since the causality-based secure property holds, we know by unicity of the timestamps, that there exists an index $k'$ such that $\varphi(z_i) \leq k' \leq \varphi(z_j)$ and $a_{k'} = b_2$. Moreover, from the cleaning performed relying on Corollary 1 we have that $\text{exec}_{\text{act}(j_0)} \leftrightarrow * \text{exec}_{\text{act}(j_0)} \leftrightarrow * \text{exec}_{\text{act}(k_1)}$ which is equivalent to $t''_{\varphi_{\text{act}(i_k)}} \rightarrow * \text{exec}_{\text{act}(j_0)} \rightarrow * \text{exec}_{\text{act}(k_1)}$.

Applying Lemma 4 we obtain that:
\[
\varphi(z_{\varphi_{\text{act}(i_k)}}) = \varphi(z_{\varphi_{\text{act}(i_k)}}) \\
\geq \text{Dist} = \text{Loc}(a''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}), \text{Loc}(b''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}))) \\
\geq \text{Dist} = \text{Loc}(a''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}), \text{Loc}(b''_1, \varphi(z_{\varphi_{\text{act}(i_k)}})) + \Delta) \\
\geq \text{Dist} = \text{Loc}(a''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}), \text{Loc}(b''_1, \varphi(z_{\varphi_{\text{act}(i_k)}})))
\]

Similarly we have that:
\[
\varphi(z_{\varphi_{\text{act}(i_k)}}) \geq \text{Dist} = \text{Loc}(a''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}), \text{Loc}(b''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}))
\]

Combining these two inequalities we obtain:
\[
\varphi(z_{\varphi_{\text{act}(i_k)}}) = \varphi(z_{\varphi_{\text{act}(i_k)}}) \\
\geq \text{Dist} = \text{Loc}(a''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}), \text{Loc}(b''_1, \varphi(z_{\varphi_{\text{act}(i_k)}})) + \Delta) \\
\geq \text{Dist} = \text{Loc}(a''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}), \text{Loc}(b''_1, \varphi(z_{\varphi_{\text{act}(i_k)}})) + \Delta)
\]

and thus
\[
t_{\varphi_{\text{act}(i_k)}} - t_{\varphi_{\text{act}(i_k)}} \geq \text{Dist} = \text{Loc}(a''_1, \varphi(z_{\varphi_{\text{act}(i_k)}}), \text{Loc}(b''_1, \varphi(z_{\varphi_{\text{act}(i_k)}})) + \Delta)
\]

This leads to a contradiction. Hence the results.
Figure 5: The PayPass-RRP Protocol

B DETAILS ON EMV PAYPASS-RRP

High-level Description of PayPass-RRP. We include here some essentials on PayPass-RRP, to help the reader grasp immediately the main idea of the protocol. In essence, in 2016, when Mastercard introduced PayPass-RRP (shown in Figure 5 below), they added a subprotocol/process called the Relay Resistance Protocol (RRP) to their de-facto contactless-payment protocol called PayPass. A PayPass transaction is lifted to a PayPass-RRP transaction in the following way: a new command, called ERRD (“Exchange Relay Resistance Data”) is introduced, and it is sent by a PayPass-RRP-capable reader to a PayPass-RRP-capable card right after the GPO (GET PROCESSING OPTIONS) command.

As with all EMV protocols, the PayPass-RRP card includes:

1. a private key PrivC;
2. a symmetric key KM that it shares with the bank;
3. a certificate chain CertPrivCA(PubC) for the card’s public key PubC.

Figure 6: PayCCR [7]: Mastercard’s PayPass-RRP with Collusive-Relay Protection & No Changes to the Issuing Bank

The reader has the public key PubCA of the Certificate Authority, and so can extract and verify the card’s public key. PayPass-RRP starts with a setup phase (not shown in Figure 5), in which the reader asks the card what protocols it supports and selects one to run. The card and reader then generate single-use random numbers NC and UN, respectively.

The reader then sends an ERRD command to the card, which contains the nonce UN. The card immediately replies with its own nonce NC, and the reader times this round trip time. The card also provides timing information, which tells the reader how long this exchange should take. The reader compares the time taken with the timing information on the card. If the time taken was too long, the reader stops the transaction as a suspected relay attack. Otherwise, the reader requests that the card generates a “cryptogram” (a.k.a. AC). The card uses the unique key KM, which it shares with the bank, to encrypt its application transaction counter ATC (which equals the number of times the card has been used). This encryption equates to a session-key denoted KS. The cryptogram AC is a MAC keyed with KS of data including the ATC, the nonce UN, and the transaction information. As the reader cannot check the AC, the card generates the “Signed Dynamic Application Data (SDAD)”: the card’s signature on a message including UN, amount, currency, ATC, NC. The reader checks the SDAD before accepting the payment.
C DETAILS ON PAYCCR

PayCCR from [7] is shown in Figure 6. It modifies the EMV protocol on the card and the EMV reader’s side, yet the bank system backend remains unchanged from the current standard. As with MasterCard’s PayPass-RRP protocol, the time bound \( t_{\text{CARD}}(\text{cardID}) \) to be enforced for the proximity-checking phase is embedded in each card; we denote it \( t_{\text{R}} \). Before the RRP process, the reader sends the card a certificate chain for the TPM’s public part of the signing key.

The EMV reader will then send a nonce \( N_{R} \) to the TPM to be timestamped. The TPM receives this bitstring \( N_{R} \) passed to the \( \text{TPM2\_GetTime} \) command, the TPM timestamps it with \( \text{TPM\_AttestedTime} \), and using a randomised signing algorithm to produce the signature \( \sigma_{1} \). Then, the EMV reader forwards \( \sigma_{1} \) to the card. The rest follows as per PayBCR only that it is the card which does the timestamps’ checks and the bank does not receive the data for this.

D IMPLEMENTATION DETAILS

We implemented the PayPass-RRP and PayBCR terminals/readers’ logic. The implementation is in C\# in order to be (easily) integrated with the CardCracker tool. We recall that CardCracker is a tool by the EMV-orientated company called ConsultHyperion. It is developed in C\# and it offers rich smart-card testing suite. It provides in-built functions, e.g., crypto libraries, EMV functionality, etc., but also a full scripting language that supports EMV commands. This is useful in both development and testing of any EMV prototype.

We developed/extended CardCracker, in C\#, so that we can expand it primarily as follows: (a) with the \textit{ERRD} command; (b) with a new \textit{ERRD} command that included communicating with a TPM 2.0. We also developed the scripting language of CardCracker so that then we can run easily the tests we describe below. We used this implementation for all our security/efficiency measurements described below.

On the one hand, for all PayPass-RRP-inherent matters, our reader-side implementation are based on the “EMV Contactless Book C-2” of the EMV specifications [13], which we will henceforth refer to as \textit{EMV-C2}. On the other hand, any TPM-related aspects (in PayBCR), which are added on top of PayPass-RRP, are as per [7].

We implement two types of terminals: (a) the type of readers that behave as per the specifications; (b) the type of readers that attempt to cheat on the RTT measurements. We call the former variety “\textit{honest}” and the latter – “\textit{timing rogue}”. We detail this below.

D.1 The PayPass-RRP Terminal

More details on EMV commands and responses we will describe below can be found in the EMV-C2 specs [13]. Any PayPass-RRP-compliant terminal should implement and exhibit in testing the following flow w.r.t. EMV commands and responses:

1. Their transactions should exhibit EMV messages in a flow/order as per the EMV-C2 specs, that is: “\textit{Select PSE}”, “\textit{Select AID}”, “\textit{GPO}”, “\textit{ERRD}”, “\textit{Read Records}”, “\textit{GEN AC}”.

2. In the above, the \textit{ERRD} command should contain the “Terminal Relay Resistance Entropy” (i.e., in our Figure 5 which describes PayPass-RRP, this is denoted as the reader-generated \( UN \) nonce; in our Figure 1 which describes PayBCR, this is denoted \( \sigma_{1}^{1} \)).

3. The \textit{ERRD} response should contain the following:
   - Device Relay Resistance Entropy, i.e., the nonce returned by the card and denoted on Figure 5 as \( N_{C} \), or to be precise – “\textit{ERRD}-r(N_{C})”
   - three timing estimates from the card:
     - Min Time for Processing Relay Resistance APDU, i.e., the minimal time the card takes to process this \textit{ERRD} command;
     - Max Time for Processing Relay Resistance APDU, i.e., the maximal time the card takes to process this \textit{ERRD} command;
     - Device Estimated Transmission Time For Relay Resistance R-APDU, i.e., the RTT-estimate as specified on the card.

4. The RRP process\textsuperscript{10} should be performed as per EMV-C2, Sections 3.10, 5.3 and 6.6. I.e., this means that every check passed or failed should be reflected in the bytes of the APDU as per these specifications.

D.1.1 The Honest PayPass-RRP Terminal. To determine if the PayPass-RRP terminal is exhibiting the (in)correct functional behaviour, one should inspect the following APDU responses, inside gathered EMV logs:

   - \textit{AIP} (Application Interchange Profile) response
   - \textit{TVR} (Terminal Verification Result) response.

Any honest PayPass-RRP terminal must:

(i) Approve a successful transaction having performed PayPass-RRP when no relay attack occurred. Concretely, as per EMV-C2, the following should occur:
   - \text{AIP} = 1981 (meaning the card supports PayPass-RRP);
   - \text{TVR} = 0000000000 (meaning PayPass-RRP was performed and passed).

(ii) Detect and abort a transaction having performed PayPass-RRP if a relay attack occurs. Concretely, as per EMV-C2, the following should occur:
   - \text{AIP} = 1981 (meaning the card supports PayPass-RRP);
   - \text{TVR} = 0000000000 (meaning PayPass-RRP was performed but the relay-resistance threshold was exceeded and the relay-resistance time limits were exceeded).

D.1.2 The Timing-Rogue PayPass-RRP Terminal. The timing-rogue PayPass-RRP terminal will always report the bit of TVR linked to PayPass-RRP as per case (i) above. Moreover, in the error sections/APDUs of EMV, it will report no errors. This conveys that a rogue PayPass-RRP terminal would detect relay attacks by performing PayPass-RRP, but approve said transactions anyway.

D.2 The PayBCR Terminal

First, recall that PayBCR is designed such that its terminal looks identical to the PayPass-RRP terminal as far as the EMV input/outputs are concerned (i.e., the communication with TPM is internal to the reader and the visible outputs of the reader are as per

\textsuperscript{10}This is the process of checking the above parameters against the reader’s clock. We stress that we use “PayPass-RRP” for a protocol and “RRP” for the process standardised by Mastercard inside this protocol.
PayPass-RRP). So, in the honest case, there should be absolutely no difference between the two terminals: concretely, they both behave like in case (i) above. Second, recall that the difference between PayBCR and PayPass-RRP terminal appears in the “timing-rogue” case: the terminal’s implementation should be such that it detects and reports relay attacks. In other words, the “timing-rogue” PayBCR terminal is in fact not corruptible, and, in case of relays, it behaves as an honest PayPass-RRP terminal – as per case (ii) above.

The actual modification that we make to a PayPass-RRP terminal to lift it to a PayBCR terminal is the fact that –as part of the ERRD command– we include TPM2_GetTime calls to the onboard TPM. In terms of interactions with the TPM, the two TPM calls are done over one connection to the TPM, unless the connection has been (incidentally) cut by the latter. As part of this, we also had to declare new “EMV” fields for the terminal to store the two signatures generated by the TPM commands, as well as implement the logic of these be sent to the (CardCracker-emulated) bank and be verified by the later. This will become clearer when we detail further on the functional correctness and security of the implementation in Section D.4.

D.3 A PayPass relay application

We implemented a software-based EMV-relay mechanism tailored for Mastercard’s PayPass protocol. Our relay “box” is formed of two Android apps running on two phones: one as a PayPass-card emulator and one as a PayPass-reader emulator. These are connected to the same wireless network to communicate wirelessly with a “server” which technically implements all the relay logic; this server is written in Python 3.7.

There are no special optimisations in the relay logic or in the applications, neither at the NFC-stack level nor at the EMV stack level: i.e., we just send back and forth plain EMV APDU commands. This is because relay “box” is simply a supporting apparatus for testing the behaviour of our PayBCR and PayPass-RRP implementations. That said, when tested using working, bank-issued PayPass cards and a working iZettle reader, this relay “box” performs robustly and reasonably fast. Concretely, a relay payment from a card found at 20m from the reader takes on average 3s (over 100 iterations). And, the iZettle reader accepts these payments without any intervention/modification onto it. About 2% of relay transactions fail due to synchronisation-issues between the card and the reader emulator.

D.4 Functional & Security Testing

In the functional-testing part, we look to see that the terminals implemented as per above behave as expected: i.e., when no relay “box” is present and the terminals are honest, the execution logs should correspond to the descriptions in EMV-C2 and [7]. To do this, we ran experiments with CardCracker (over 30 PayBCR and 30 PayPass-RRP payments), and the results are evidenced via the logs\(^\text{11}\) gathered and discussed below. Due space constraints, we mainly detail here in the case of PayBCR (and not the case of PayPass-RRP).

\(^{11}\)These logs contain some code in CardCracker-proprietary syntax, however in their overwhelming majority they contain standard-EMV keywords and, as such, they are largely self-explained to a reader familiar with EMV language.
The last crucial part of the functional testing is to attest that the bank checks the timestamps and the signatures by the TPM. In Figure 9, we show a short snippet of logs: this exhibits that an honest PayBCR reader—alongside the AC—does send the TPM signatures (see lines 259–262 in Figure 9). The last lines of logs in Figure 9 indicate that the bank does the checks of timings and outputs the result back.

In the security-testing cases, we used our relay “box”. These ascertain that the implemented behaviour complies to the expectations: (a) the “timing-rogue” terminals implemented would elude the RTT checks; (b) that in PayPass-RRP this is not caught by the bank and in PayBCR this is caught by the bank.
Figure 9: Snippet of CardCracker Logs – Part of the Bank Rechecking RTTs