Security Analysis and Implementation of Relay-Resistant Contactless Payments

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

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

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

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 ondistance 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.

CCS CONCEPTS

• **Security and privacy → Formal security models**.

KEYWORDS

security analysis, formal methods, contactless payment protocols

ACM Reference Format:
Ioana Boureanu, Tom Chothia, Alexandre Debant, and Stéphanie Delaune. 2020. Security Analysis and Implementation of Relay-Resistant Contactless Payments. In *2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3372297.3417235

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[1]". 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

[1]https://usa.visa.com/visa-everywhere/blog/bdp/2019/05/13/tap-to-pay-1557714409720.html

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. There exist a number of formal frameworks for modelling distance bounding protocols (e.g. [2, 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 physicality-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` command[2] of the TPM takes the handle for a signature scheme and some input, and it returns a signature over said input and over

$$\texttt{TPM-AttestedTime} = (\texttt{Clock}, \texttt{Time}).$$

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()` can produce a signed version of a timestamped nonce, with attested time-information.

According to the body standardising 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 considers it 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 closely approximates the roundtrip time (RTT) between the card and the reader. Moreover, PayCCR and PayBCR record this timestamping information, later to be used by the card or the issuing bank to re-verify the RTT measurements alongside other checks they normally make in PayPass-RRP. On the one hand, PayBCR does not modify the card side of PayPass-RRP and thus it is the issuing bank who does the verification of the TPM's timestamps. On the other hand, PayCCR leaves the PayPass-RRP reader-to-bank specifications unchanged, and modifies the PayPass-RRP card so that it is now the card (and not the bank – as per PayBCR) who checks the RTT time-stamping mediated by the TPM onboard the PayPass-RRP reader. Below, we will mainly recall PayBCR (see Figure 1).

The PayPass-RRP protocol, that is at the basis of the two protocols considered here, is summarised in Appendix B; this protocol is similar to the "PaySafe" protocol that was originally proposed and analysed in [9]. More details about the protocols PayBCR and PayCCR can be found in [7]. In PayBCR, firstly, the EMV reader sends its nonce N_R to the TPM to be timestamped. The TPM uses the `TPM2_GetTime` command to timestamp this nonce and it produces a randomised signature σ_1 on the timestamped nonce. The signature σ_1 from the TPM is sent to the card instead of the first nonce *UN* in PayPass-RRP. To keep the protocol compliant with PayPass-RRP, the PayBCR reader actually sends the card only a truncation of σ_1, denoted as σ_1'. The card's response (N_C as per PayPass-RRP) is sent to the TPM, which similarly yields a randomised signature σ_2. The *SDAD* (Signed Dynamic Application Data) is a digital signature by the card on the *AC*, the timing information t_d and σ_1' (in place of *UN*). Finally, the card's PayPass-RRP time-bound t_d, σ_1, σ_2, t_1 and t_2 and the *AC* are sent to the bank. With these, the bank can check the difference between the timestamps to ensure the card and EMV reader where close.

For completeness, we give the description of PayCCR too; see Figure 6, in Appendix C. Its details are very similar to those of PayBCR, only that the card does the verification of the timestamping signatures.

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 introduced 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 [1] which is the calculus used in input of the ProVerif tool [3]. 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 $n, k \in \mathcal{N}$, agent names denoted $a, b \in \mathcal{A}$, or simply public constants denoted $c, c_1, c_2 \in \Sigma_0$. We denote $\Sigma_0^+ = \Sigma_0 \uplus \mathcal{A}$. More complex messages can also be exchanged relying on cryptographic primitives modelled through a set of function symbols Σ called a *signature*. Such a signature Σ is split into *constructor* and *destructor* symbols, *i.e.,* $\Sigma = \Sigma_c \uplus \Sigma_d$. We also consider a set $\mathcal{X}$ of *message variables*, denoted $x, y, z \ldots$, as well as a set $\mathcal{W}$ of *handles*, denoted $\mathsf{w}_1, \mathsf{w}_2, \ldots$. Variables in $\mathcal{X}$ model arbitrary data expected by a protocol participant, while variables in $\mathcal{W}$ 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 D 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 *vars*(u) the variables occurring in u. A *constructor term* is a term in $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathcal{X} \cup \mathbb{R}_+)$.

EXAMPLE 1. *To model the PayBCR protocol presented in Section 2, we consider the signature* $\Sigma_{\mathrm{BCR}} = \Sigma_c \uplus \Sigma_d$*:*

- $\Sigma_c = \{\mathsf{senc}, \mathsf{shk}, \mathsf{sign}, \mathsf{pubk}, \mathsf{seck}, \mathsf{mac}, \langle\ \rangle, \mathsf{ok}\}$, *and*
- $\Sigma_d = \{\mathsf{sdec}, \mathsf{verify}, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{eq}\}$.

[2]This 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 v 1.38 is available by several manufacturers.

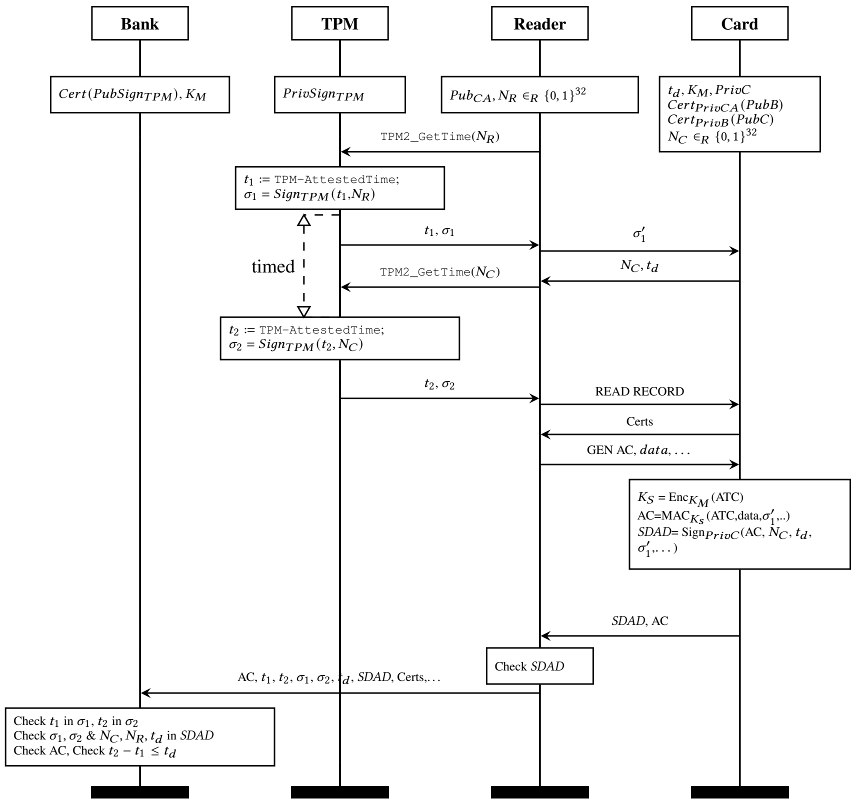

Figure 1: PayBCR [7]: Mastercard's PayPass-RRP with Collusive-Relay Protection & No Changes to the Card

The symbols in Σ_{BCR} *allows one to construct terms to represent cryptographic messages, e.g., ciphertext based on the symmetric encryption symbol* senc *(of arity* 2*), and signature using the symbol* sign *(of arity* 2*). The corresponding decryption and verification schemes are modelled by the* sdec *and* verify *symbols. The symbols* pubk *and* seck*, each of arity* 1*, are used to model public/private keys, whereas the symbol* shk *of arity* 2 *is used to model symmetric key shared between two agents. The symbol* mac *models a MAC scheme. The symbols* $\langle\ \rangle$, proj_1*, and* proj_2 *represent the pairing and projections operators. Lastly, we consider two symbols* eq *(arity* 2*) and the constant* 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 $\mathcal{T}(\Sigma_c, \mathcal{X})$ and define $=_{\mathsf{E}}$ 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 $\mathsf{g}(t_1, \ldots, t_n) \to t$ where $\mathsf{g} \in \Sigma_d$ and $t, t_1, \ldots, t_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$. A term u can be *rewritten* in v if there is a position p in u, and a rewriting rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ such that $u|_p = \mathsf{g}(t_1, \ldots, t_n)\theta$ for some substitution θ, and $v = u[t\theta]_p$, 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 $u\!\downarrow$ 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:*

$$\begin{array}{ll} \mathsf{sdec}(\mathsf{senc}(x,y),y) \to x & \\ \mathsf{verify}(\mathsf{sign}(x,y),\mathsf{pubk}(y)) \to x & \mathsf{proj}_1(\langle x_1,x_2\rangle) \to x_1 \\ \mathsf{eq}(x,x) \to \mathsf{ok} & \mathsf{proj}_2(\langle x_1,x_2\rangle) \to x_2 \end{array}$$

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* $\mathsf{sdec}(\mathsf{senc}(x,y),y) \to x$ *as we did here, or consider both symbols as constructors, together with the equation* $\mathsf{sdec}(\mathsf{senc}(x,y),y) = x$*. In the latter case,* $\mathsf{sdec}(c,k)$ *will be considered as a "valid" message. The second rule is used both to check a signature and to extract its content. The term* $\mathsf{seck}(a)$ *is used to represent the private key of the agent a, whereas* $\mathsf{pubk}(\mathsf{seck}(a))$ *models its public counterpart. The third one models an equality* *check. Note that* $\mathsf{eq}(u,v)$ *reduces to a constructor term if, and only if,* $u =_{\mathsf{E}} v$ *(i.e.,* $u = v$ *here since* $\mathsf{E} = \emptyset$*). The two rules in the second column model the projection operators.*

For modelling purposes, we split the signature Σ into two parts, Σ_{pub} and Σ_{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 $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe*, i.e., a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \Sigma_0^+ \cup \mathcal{W} \cup \mathbb{R}_+)$.

EXAMPLE 3. *Among the symbols in* Σ_{BCR}*, only the symbols* shk *and* seck *are in* Σ_{priv}*. Let* $u_0 = \mathsf{senc}(\langle n, k\rangle, \mathsf{shk}(b,c))$*, and* $u_1 = \mathsf{shk}(b,c)$*. We have that* $\mathsf{proj}_2(\mathsf{sdec}(u_0,u_1))\downarrow = k$*. The term* $\mathsf{proj}_2(\mathsf{sdec}(u_0,u_1))$ *models the application of the decryption algorithm on top of* u_0 *using the key* u_1 *followed by the application of the second projection.*

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:

$$\begin{array}{rcl} P, Q & := & 0 \\ & \mid & \mathtt{new}\ n.P \\ & \mid & \mathtt{in}(x).P \\ & \mid & \mathtt{out}(u).P \\ & \mid & \mathtt{let}\ x = v\ \mathtt{in}\ P\ \mathtt{else}\ Q \\ & \mid & \mathtt{gettime}(x).P \\ & \mid & \mathsf{check}(u_1,u_2,u_3).P \\ & \mid & \mathsf{claim}(u_1,u_2,u_3,u_4).P \end{array}$$

where $x \in \mathcal{X}$, $n \in \mathcal{N}$, $v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \Sigma_0^+ \cup \mathcal{X} \cup \mathbb{R}_+)$, and $u, u_1, \ldots, u_4 \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathcal{X} \cup \mathbb{R}_+)$.

The first four instructions are all standard to the applied pi-calculus. As usual, the null process does nothing. The restriction $\mathtt{new}\ n.P$ generates a fresh name and then executes P. We have constructions to model input and output actions. The $\mathtt{let}$ 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 $\mathtt{gettime}(x)$ bounds the current time to the variable x. This is used to add a timestamp in some messages. The $\mathsf{check}(u_1,u_2,u_3)$ instruction indicates that a process is checking the times u_1 and u_2 against some expected time bound u_3. The $\mathsf{claim}(u_1,u_2,u_3,u_4)$ 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 $fv(P)$ for the set of *free* variables occurring in P, *i.e.,* the set of variables that are not in the scope of an $\mathtt{in}$, a $\mathtt{let}$, or a $\mathtt{gettime}$ instruction. We consider *parametrised processes*, $P(z_0, \ldots, z_n)$, where $z_0, \ldots, z_n$ are variables from a special set $\mathcal{Z}$ (disjoint from $\mathcal{X}$ and $\mathcal{W}$). 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 $fv(R) \subseteq \{z_0, \ldots, z_n\}$ 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 $\mathtt{gettime}$ 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:*

$$\begin{array}{rcl} \mathsf{TPM}(z_0) & = & \mathtt{in}(x_n). \\ & & \mathtt{gettime}(x_t). \\ & & \mathtt{out}(\mathsf{sign}(\langle x_t, x_n\rangle, \mathsf{seck}(z_0))).0 \end{array}$$

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* $\mathsf{seck}(z_0)$ *and a timestamp is added into the signature. The current time is obtained using the* $\mathtt{gettime}$ *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) $\mathsf{Loc}(a, t_a)$, i.e., the location of a at time t_a, and
(2) $\mathsf{Loc}(b, t_b)$, i.e., the location of b at time t_b

is less or equal than $(t_b - t_a) \cdot \mathsf{c}_0$. 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 $\mathsf{Dist} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}_+$ is defined as follows:

$$\mathsf{Dist}(l_1, l_2) = \frac{\|l_2 - l_1\|}{\mathsf{c}_0} \text{ for any } l_1, l_2 \in \mathbb{R}^3$$

with $\|\cdot\| : \mathbb{R}^3 \to \mathbb{R}_+$ the Euclidean norm.

A *mobility plan* Loc is a function: $\mathcal{A} \times \mathbb{R}_+ \to \mathbb{R}^3$ 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 $a \in \mathcal{A}$ and $t_1, t_2 \in \mathbb{R}_+$ such that $t_1 \leq t_2$, we have that:

$$\mathsf{Dist}(\mathsf{Loc}(a, t_1), \mathsf{Loc}(a, t_2)) \leq t_2 - t_1.$$

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* Loc_0 *such that* $\mathsf{Loc}_0(\mathsf{tpm}_0, t) = (0,0,0)$*, and* $\mathsf{Loc}_0(\mathsf{card}_0, t) = (10,0,0)$*, and* $\mathsf{Loc}_0(\mathsf{bk}_0, t) = (100,0,0)$ *for any* $t \in \mathbb{R}_+$*, and* $\mathsf{Loc}_0(a,t) = (0,0,0)$ *otherwise (i.e., for any* $a \in \mathcal{A} \smallsetminus \{\mathsf{tpm}_0, \mathsf{card}_0, \mathsf{bk}_0\}$*). This models a very simple mobility plan where all the agents are actually at a fixed position and they never move:* card_0 *is at distance* 10 *from* tpm_0 *and the agent* bk_0 *is even further. The other entities are all located at position* $(0,0,0)$ *at the same place as* tpm_0.

TIM	$(\mathcal{P};\Phi;t) \longrightarrow_{\mathsf{Loc}} (\mathcal{P};\Phi;t+\delta)$	with $\delta > 0$
NEW	$(\lfloor \mathsf{new}\ n.P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,t,\tau}_{\mathsf{Loc}} (\lfloor P\{n \mapsto n'\} \rfloor_a \uplus \mathcal{P};\Phi;t)$	with $n' \in \mathcal{N}$ fresh
OUT	$(\lfloor \mathsf{out}(u).P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,t,\mathsf{out}(u)}_{\mathsf{Loc}} (\lfloor P \rfloor_a \uplus \mathcal{P};\Phi \uplus \{\mathsf{w} \xrightarrow{a,t} u\};t)$	with $\mathsf{w} \in \mathcal{W}$ fresh
LET-THEN	$(\lfloor \mathtt{let}\ x = v\ \mathtt{in}\ P\ \mathtt{else}\ Q \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,t,\tau}_{\mathsf{Loc}} (\lfloor P\{x \mapsto v\downarrow\} \rfloor_a \uplus \mathcal{P};\Phi;t)$	when $v\downarrow \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathbb{R}_+)$
LET-ELSE	$(\lfloor \mathtt{let}\ x = v\ \mathtt{in}\ P\ \mathtt{else}\ Q \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,t,\tau}_{\mathsf{Loc}} (\lfloor Q \rfloor_a \uplus \mathcal{P};\Phi;t)$	when $v\downarrow \notin \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathbb{R}_+)$
CLAIM	$(\lfloor \mathsf{claim}(u_1,u_2,u_3,u_4).P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,t,\mathsf{claim}(u_1,u_2,u_3,u_4)}_{\mathsf{Loc}} (\lfloor P \rfloor_a \uplus \mathcal{P};\Phi;t)$	
CHECK	$(\lfloor \mathsf{check}(u_1,u_2,u_3).P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,t,\mathsf{check}(u_1,u_2,u_3)}_{\mathsf{Loc}} (\lfloor P \rfloor_a \uplus \mathcal{P};\Phi;t)$	
GTIM	$(\lfloor \mathtt{gettime}(x).P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,t,\mathtt{gettime}}_{\mathsf{Loc}} (\lfloor P\{x \mapsto t\} \rfloor_a \uplus \mathcal{P};\Phi;t')$	with $t' > t$

IN $\quad (\lfloor \mathtt{in}(x).P \rfloor_a \uplus \mathcal{P};\Phi;t) \xrightarrow{a,t,\mathtt{in}(u)}_{\mathsf{Loc}} (\lfloor P\{x \mapsto u\} \rfloor_a \uplus \mathcal{P};\Phi;t)$

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

- either $b \in \mathcal{A} \smallsetminus \mathcal{M}$ and there exists $(\mathsf{w} \xrightarrow{b,t_b} u) \in \Phi$
- or $b \in \mathcal{M}$ 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.

3.4 Semantics

Our semantics is given by a transition system over configurations that manipulates extended processes, i.e., expressions of the form $\lfloor P \rfloor_a$ with $a \in \mathcal{A}$ and P a process such that $fv(P) = \emptyset$. 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* $(\mathcal{P};\Phi;t)$ *where:*

- $\mathcal{P}$ *is a multiset of* extended processes $\lfloor P \rfloor_a$*;*
- $\Phi = \{\mathsf{w}_1 \xrightarrow{a_1,t_1} u_1, \ldots, \mathsf{w}_n \xrightarrow{a_n,t_n} u_n\}$ *is an* extended frame*, i.e., a substitution such that* $\mathsf{w}_i \in \mathcal{W}$*,* $u_i \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+ \cup \mathbb{R}_+)$*,* $a_i \in \mathcal{A} \cup \{\star\}$ *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* $\mathcal{K}_0 = (\mathcal{P}_0;\Phi_0;0)$ *with:*

$$\mathcal{P}_0 = \{\ \lfloor \mathsf{TPM}(\mathsf{tpm}_0) \rfloor_{\mathsf{tpm}_0} ;\ \lfloor \mathsf{Bank}(\mathsf{bk}_0) \rfloor_{\mathsf{bk}_0} ;\ \lfloor \mathsf{Card}(\mathsf{card}_0) \rfloor_{\mathsf{card}_0} \}.$$

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:*

$$\begin{aligned}\Phi_0 = \{\quad & \mathsf{w}_1 \xrightarrow{\star,0} \mathsf{pubk}(\mathsf{seck}(\mathsf{tpm}_0)),\\ & \mathsf{w}_2 \xrightarrow{\star,0} \mathsf{pubk}(\mathsf{seck}(\mathsf{bk}_0)),\\ & \mathsf{w}_3 \xrightarrow{\star,0} \mathsf{pubk}(\mathsf{seck}(\mathsf{card}_0))\}.\end{aligned}$$

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 add the following terms into the frame:*

- $\mathsf{sign}(\langle \mathsf{cardCert}, \langle \mathsf{card}_0, \mathsf{pubk}(\mathsf{seck}(\mathsf{card}_0))\rangle\rangle, \mathsf{seck}(\mathsf{bk}_0))$*;*
- $\mathsf{sign}(\langle \mathsf{tpmCert}, \langle \mathsf{tpm}_0, \mathsf{pubk}(\mathsf{seck}(\mathsf{tpm}_0))\rangle\rangle, \mathsf{seck}(\mathsf{bk}_0))$.

The terms cardCert *and* tpmCert *are public constants from* Σ_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 same constant* cert.

Given an extended frame Φ, and a mobility plan Loc, we say that a term u is *deducible from* Φ *by* $b \in \mathcal{A}$ *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 $R\Phi\downarrow =_{\mathsf{E}} u$, and for all $\mathsf{w} \in vars(R)$ we have that $(\mathsf{w} \xrightarrow{c,t} v) \in \Phi$ for some v, and

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

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 $\mathsf{Loc}(b,t_b)$ to where it is being received $\mathsf{Loc}(a,t)$.

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* $\mathcal{K}'_0 = (\lfloor \mathsf{TPM}(\mathsf{tpm}_0)\rfloor_{\mathsf{tpm}_0}; \Phi_0; 0)$ *and the mobility plan* Loc_0 *as given in Example 5.*

$$\begin{array}{l}\mathcal{K}'_0 \rightarrow (\lfloor \mathsf{TPM}(\mathsf{tpm}_0)\rfloor_{\mathsf{tpm}_0}; \Phi_0; 1.1)\\ \xrightarrow{\mathsf{tpm}_0,1.1,\mathsf{in(ok)}}\xrightarrow{\mathsf{tpm}_0,1.1,\mathsf{gettime}} (\lfloor \mathsf{out}(m)\rfloor_{\mathsf{tpm}_0}; \Phi_0; 1.2)\\ \xrightarrow{\mathsf{tpm}_0,1.2,\mathsf{out}(m)}_{\mathit{Loc}_0} (\lfloor 0\rfloor_{\mathsf{tpm}_0}; \Phi_1; 1.2)\end{array}$$

where:

- $m = \mathsf{sign}(\langle 1.1, \mathsf{ok}\rangle, \mathsf{seck}(\mathsf{tpm}_0))$*, and*
- $\Phi_1 = \Phi_0 \cup \{\mathsf{w}_4 \xrightarrow{\mathsf{tpm}_0,1.2} m\}$.

The first input is possible since $\Phi_0 \vdash \mathsf{ok}$ *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 Φ_1 *by* tpm_0 *at time* 1.2*. Actually any agent other than* card_0 *and* bk_0 *are able to deduce m at time* 1.2*. Remember that* $\mathit{Loc}_0(a,t) = (0,0,0)$ *for any t and any a different from* tpm_0 *and* bk_0*.*

3.5 Threat Model

Before we present the security property we wish to analyse, we summarise the threat model implied by Section 2 (for the TPM) and this current section (for the rest), via the language, mobility/time encodings, and all the (other) protocol semantics described above. Thus, our attacker model is as follows:

- readers, cards, TPMs[3] can become malicious/corrupted;
- with respect to cryptographic primitives, we assume a normal Dolev-Yao (DY) attacker-model for all malicious agents;
- with respect to communication channels, malicious agents acts as expected (i.e., can block, inject, modify message as a DY adversary), but they are bound by our mobility plan and our timing rules as follows:
  - in total, there can be an unbounded number of (statically[4]) corrupted/malicious agents, at any given location;
  - all corrupted/malicious agents adhere to the laws of our model/physics: i.e., corrupt agents cannot transmit messages faster than the other agents, and they cannot move faster than the other agents (who, in turn, travel overwhelmingly more slowly than messages).

3.6 Security Properties

As explained in above, we are interested in analysing a property (formalised in Def. 3) that pertains to secure/correct contactless payments made in physical proximity. Roughly, this property ensures that when a bank finishes the protocol successfully it is because a transaction took place between a card and a reader/TPM, and the card and the reader have been close during this transaction. This proximity is ensured through the use of the instruction $\mathsf{check}(t_1, t_2, \delta)$ whose intended meaning is that timing constraint $t_2 - t_1 \leq \delta$ has been verified by some honest agent.

As usual to analyse a security property, we rely on events. The event $\mathsf{claim}(\mathsf{tpm}_0, \mathsf{card}_0, t_1^0, t_2^0)$ will be launched when the bank finishes the protocol, seemingly with tpm_0 and card_0, and, this claim event occurs after a check event took place – whereby card_0 was close to tpm_0 between times t_1^0 and t_2^0.

Before we can give our security definition, we need to define a few helper-notions, like that of *configuration*. An *initial frame* Φ_0 is an extended frame such that $a = \star$ and $t = 0$ for any $(\mathsf{w} \xrightarrow{a,t} u) \in \Phi_0$.

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* Φ_0 *is an initial frame, and for each* $\lfloor P\rfloor_a \in \mathcal{P}_0$*, 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* Φ_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 payBCR. 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* Loc*, for all execution exec such that:*

$$\mathcal{K}_0 \xrightarrow{(a_1,t_1,\mathsf{act}_1)\ldots(a_n,t_n,\mathsf{act}_n)\cdot(b_0,t,\mathsf{claim}(b_1,b_2,t_1^0,t_2^0))}_{\mathit{Loc}} \mathcal{K}$$

we have that:

- *either* $b_1 \in \mathcal{M}$*, or* $b_2 \in \mathcal{M}$*;*
- *or* $\mathsf{act}_k = \mathsf{check}(t_1^0, t_2^0, t_3^0)$ *for some* $k \leq n$ *such that:*

$$\begin{array}{rcl} t_2^0 - t_1^0 & \geq & \mathit{Dist}(\mathit{Loc}(b_1, t_1^0), \mathit{Loc}(b_2, t)) \\ & & + \mathit{Dist}(\mathit{Loc}(b_2, t), \mathit{Loc}(b_1, t_2^0)) \end{array}$$

  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

[3]In practice, TPMs can only be corrupted within the TCG-driven model described in Section 2: that is, the global clocks of TPMs can essentially not be tampered with. In our formalism however, the threat model is stronger, allowing even for time-corruption on the TPM.
[4]This means that all corruption occurs before the protocol starts executing, which is usual in symbolic verification.

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.).

Observations w.r.t Definition 3. Firstly, note that the check is performed by agent a_k which is not fixed to being b_0, b_1 or b_2. Importantly, as such, in this definition, we may have the check be performed by an agent that is not b_0 who in fact makes the final claim.

Secondly, note that indeed it is in one of the possible executions that the agent b_0 is the bank, the agent b_1 is a TPM and b_2 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 b_2 is dishonest, or the TPM b_1 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 b_1 and b_2 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 strengthen 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 t_1^0 and t_2^0 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 ($t_2^0 - t_1^0$–message processing time) $\times\, c_0$. In our model (unlike in practice), this would be $(t_2^0 - t_1^0) \times c_0$, 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 ($t_2^0 - t_1^0$–message processing time) $\times\, c_0/2$. In our model (unlike in practice), this would be $(t_2^0 - t_1^0) \times c_0/2$, 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 [3], 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 [3] 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 [4], to perform a comprehensive analysis of the TLS 1.3 Draft-18 protocol [5], 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 Σ_0^{spe} of special constants in Σ_0 that will be used to abstract time. We explain how to transform a configuration $\mathcal{K} = (\mathcal{P}; \Phi; t)$ into a simple configuration that does not contain time. For sake of simplicity, we assume that variables occurring in $\mathcal{P}$ are at most bound once. The transformation $\bar{\cdot}$ applied on $\mathcal{K}$ gives us a pair $(\mathcal{P}_0; \phi_0)$ where:

- $\mathcal{P}_0$ is the untimed counterpart of $\mathcal{P}$, i.e. each $\mathtt{gettime}(x)$ instruction occurring in $\mathcal{P}$ is replaced by $\mathtt{timestamp}(c_x)$ where $c_x \in \Sigma_0^{\mathsf{spe}}$, and the occurrences of x in the remaining process are replaced by c_x;
- $\phi_0 = \{\mathsf{w} \rightarrow u \mid (\mathsf{w} \xrightarrow{c,t} u) \in \Phi\}$.

We denote $\overset{(a,\mathsf{act})}{\rightsquigarrow}$ 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

$$
\begin{array}{lll}
\text{NEW}' & (\lfloor \mathsf{new}\ n.P \rfloor_a \uplus \mathcal{P}; \Phi) \xrightarrow{a,\tau} (\lfloor P\{n \mapsto n'\} \rfloor_a \uplus \mathcal{P}; \Phi) & \text{with } n' \in \mathcal{N} \text{ fresh} \\
\vdots & \vdots & \\
\text{CHECK}' & (\lfloor \mathsf{check}(u_1, u_2, u_3).P \rfloor_a \uplus \mathcal{P}; \Phi) \xrightarrow{a,\mathsf{check}(u_1,u_2,u_3)} (\lfloor P \rfloor_a \uplus \mathcal{P}; \Phi) & \\
\text{GTIM}' & (\lfloor \mathsf{timestamp}(c_x).P \rfloor_a \uplus \mathcal{P}; \Phi) \xrightarrow{a,\mathsf{timestamp}(c_x)} (\lfloor P \rfloor_a \uplus \mathcal{P}; \Phi) & \\
\text{IN}' & (\lfloor \mathsf{in}(x).P \rfloor_a \uplus \mathcal{P}; \Phi) \xrightarrow{a,\mathsf{in}(u)} (\lfloor P\{x \mapsto u\} \rfloor_a \uplus \mathcal{P}; \Phi) & \text{with } R\Phi\downarrow =_{\mathsf{E}} u \text{ for some recipe } R
\end{array}
$$

Figure 3: Semantics a la ProVerif.

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 $\mathcal{P}$ is* causality-based secure *w.r.t. a set $\mathcal{S}$ of valid initial configurations, if for all $\mathcal{K}_0 \in \mathcal{S}$, for all execution exec such that:*

$$\overline{\mathcal{K}_0} \xrightarrow{(a_1,\mathsf{act}_1)\dots(a_n,\mathsf{act}_n)\cdot(b_0,\mathsf{claim}(b_1,b_2,c_1,c_2))} (\mathcal{P}'; \phi')$$

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 $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+)$ such that:

- $\mathsf{act}_k = \mathsf{check}(c_1, c_2, u)$*;*
- $(a_i, \mathsf{act}_i) = (b_1, \mathsf{timestamp}(c_1))$*;*
- $(a_j, \mathsf{act}_j) = (b_1, \mathsf{timestamp}(c_2))$*; and*
- $a_{k'} = b_2$.

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 $\mathcal{P}$ is* well-timed *w.r.t. a set $\mathcal{S}$ of valid initial configurations if for all $\mathcal{K}_0 \in \mathcal{S}$, for all execution exec such that:*

$$\overline{\mathcal{K}_0} \xrightarrow{(a_1,\mathsf{act}_1)\dots(a_n,\mathsf{act}_n)\cdot(b_0,\mathsf{claim}(b_1,b_2,c_1,c_2))} (\mathcal{P}'; \phi')$$

we have that there exist $i, j \leq n$ such that:

- $(a_i, \mathsf{act}_i) = (b_1, \mathsf{timestamp}(c_1))$*;*
- $(a_j, \mathsf{act}_j) = (b_1, \mathsf{timestamp}(c_2))$.

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 $\mathcal{P}$ be a protocol and $\mathcal{S}$ be a set of valid initial configurations. Assuming that $\mathcal{P}$ is well-timed w.r.t. $\mathcal{S}$, we have that:*

$\mathcal{P}$ is DB-secure w.r.t. $\mathcal{S}$
if, and only if,
$\mathcal{P}$ is causality-based secure w.r.t. $\mathcal{S}$.

A proof sketch is available in Appendix and a detailed proof can be found in [6]. First, we may note that if $\mathcal{P}$ is causality-based secure w.r.t. $\mathcal{S}$ then $\mathcal{P}$ is well-timed w.r.t. $\mathcal{S}$. 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 δ 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 δ. 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 [6].

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 AC message[5]. More formally, we have that:

$$AC_{\mathsf{PayCCR++}} = MAC(K_S, ATC, data, \sigma_1, TPM, t_1, t_2)$$

with $K_S = \mathsf{senc}(ATC, \mathsf{shk}(Card, Bank))$ 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 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 as the same time. This is not true for a dishonest participants, 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

$$\mathsf{sign}(\langle \mathsf{TPMCert}, a, \mathsf{pubk}(\mathsf{seck}(a))\rangle, \mathsf{seck}(bankID))$$

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

$$\mathsf{sign}(\langle \mathsf{cardCert}, a, \mathsf{pubk}(\mathsf{seck}(a))\rangle, \mathsf{seck}(bankID)).$$

The initial knowledge given to the attacker is:

- $pub(seck(a))$ for any agent a, and his associated certificate $certX(a)$;
- $\mathsf{seck}(a)$, and $\mathsf{shk}(a, bankID)$ 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)) ==>
    (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 t_d 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 $\mathsf{receivedBound}(card_0, t_d)$ and check the following extra property:

```
query card_0:bitstring, timeboundinfo:bitstring;
  event(receivedBound(card_0, timeboundinfo)) ==>
    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.

[5]In practice, this is feasible via optional fields inside the AC, which issuing-banks already use for further data-collection.

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. b_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:

Protocol	Role authentication	Time-bound authentication	Causality-based security
PayCCR++	✗	✓	✓
PayBCR	✓	✓	✓

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 [6].

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 from a "timing-rogue" PayPass-RRP terminal, but would reject a relayed transaction from 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].

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

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

[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.

[9]As Appendix D explains, we implemented PayPass-RRP and PayBCR as per the EMV standard for PayPass-RRPand the *ERRD* will be repeated twice if it fails to get the correct time-bound, under the first attempt.

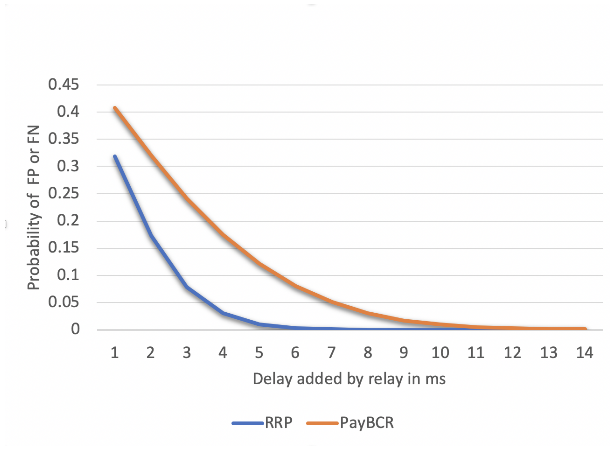

Figure 4: The possible false negative and false positive rates for our implementation of RRP and PayBCR for a range of possible relay speeds

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.

Both implementations would be easily vulnerable to a relay that added a total additional delay of 1ms or less, and both would work well to stop relays that added a delay of more than 10ms. PayPass-RRP can enforce a much tighter bound than PayBCR, namely in the 5ms to 10ms range. Past work on EMV relay that used standard, off-the-shelf equipment (smart phones) (e.g. [9, 15]) has reported a delay of anywhere between 36ms to 100ms. To this end, we can conclude that both of our PayPass-RRP and PayBCR implementations will stop relay attacks with such COTS hardware.

Relay attacks that target cars and use specialist equipment can achieve relay speeds of a few microseconds [14]. 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 vain, 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] M. Abadi and C. Fournet. 2001. Mobile Values, New Names, and Secure Communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, 104–115.

[2] David Basin, Srdjan Capkun, Patrick Schaller, and Benedikt Schmidt. 2009. Let's Get Physical: Models and Methods for Real-World Security Protocols. In *Theorem Proving in Higher Order Logics*. Springer Berlin Heidelberg, 1–22.

[3] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1, 1-2 (2016), 1–135. https://doi.org/10.1561/3300000004

[4] Bruno Blanchet. 2017. Symbolic and Computational Mechanized Verification of the ARINC823 Avionic Protocols. In *30th IEEE Computer Security Foundations Symposium (CSF'17)*. IEEE, Santa Barbara, CA, USA, 68–82.

[5] Bruno Blanchet and Ben Smyth. 2016. *Automated reasoning for equivalences in the applied pi calculus with barriers*. Research report RR-8906. Inria. Available at https://hal.inria.fr/hal-01306440.

[6] Ioana Boureanu, Tom Chothia, Alexandre Debant, and Stéphanie Delaune. 2020. *Security Analysis and Implementation of Relay-Resistant Contactless Payments*. Research Report. Univ Rennes, CNRS, IRISA, France ; University of Surrey ; University of Birmingham. https://hal.inria.fr/hal-02917076

[7] Tom Chothia, Ioana Boureanu, and Liqun Chen. 2019. Making Contactless EMV Robust Against Rogue Readers Colluding With Relay Attackers, In 23rd International Conference on?Financial Cryptography and Data Security (FC 19). *Proceedings of the 23rd International Conference on Financial Cryptography and Data Security (FC 19)*. http://epubs.surrey.ac.uk/850544/

[8] Tom Chothia, Joeri de Ruiter, and Ben Smyth. 2018. Modelling and Analysis of a Hierarchy of Distance Bounding Attacks. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 1563–1580. https://www.usenix.org/conference/usenixsecurity18/presentation/chothia

[9] Tom Chothia, Flavio D. Garcia, Joeri de Ruiter, Jordi van den Breekel, and Matthew Thompson. 2015. Relay Cost Bounding for Contactless EMV Payments. In *Proc. 19th International Conference on Financial Cryptography and Data Security (FC'15) (Lecture Notes in Computer Science)*, Vol. 8975.

[10] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. 2019. BeleniosVS: Secrecy and Verifiability Against a Corrupted Voting Device. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. 367–381.

[11] Cas Cremers, Kasper B Rasmussen, Benedikt Schmidt, and Srdjan Capkun. 2012. Distance hijacking attacks on distance bounding protocols. In *Proc. IEEE Symposium on Security and Privacy (S&P'12)*. IEEE, 113–127.

[12] Alexandre Debant, Stéphanie Delaune, and Cyrille Wiedling. 2018. A Symbolic Framework to Analyse Physical Proximity in Security Protocols. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*. 29:1–29:20. https://doi.org/10.4230/LIPIcs.FSTTCS.2018.29

[13] EMVCo. 2018. Book C-2 Kernel 2 Specification v2.7. EMV Contactless Specifications for Payment System.

[14] Aurélien Francillon, Boris Danev, and Srdjan Capkun. 2010. Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars. *IACR Cryptology ePrint Archive* 2010 (2010), 332. http://dblp.uni-trier.de/db/journals/iacr/iacr2010.html#FrancillonDC10

[15] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. 2011. Practical Relay Attack on Contactless Transactions by Using NFC Mobile Phones. *IACR Cryptology ePrint Archive* 2011 (01 2011), 618. https://doi.org/10.3233/978-1-61499-143-4-21

[16] Trusted Computing Group. 2016. *Trusted Platform Module Library Family 2.0, Specification - Part 1: Architecture, Revision 1.38 and Part 3: Commands, Revision 1.38*. Technical Report.

[17] Lucca Hirschi and Cas Cremers. 2019. Improving Automated Symbolic Analysis of Ballot Secrecy for E-Voting Protocols: A Method Based on Sufficient Conditions. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*. 635–650.

[18] S. Mauw, Z. Smith, J. Toro-Pozo, and R. Trujillo-Rasua. [n.d.]. Distance-Bounding Protocols: Verification without Time and Location. In *2018 IEEE Symposium on Security and Privacy (S&P)*, Vol. 00. 152–169. https://doi.org/10.1109/SP.2018.00001

A PROOFS OF THEOREM 1

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 by a tuple (b, t_b, 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, $\mathtt{gettime}$, $\mathsf{claim}(u_1, u_2, u_3, u_4)$, $\mathsf{check}(u_1, u_2, u_3)$, $\mathtt{out}(u)$, or $\mathtt{in}(u)\}$. Thus, an annotated action is:

- empty for the TIM rule;
- $(a, \alpha, s, t, \mathsf{w})$ when the underlying label (a, t, α) is of the form $(a, t, \mathtt{out}(u))$. 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.
- $(a, \alpha, s, t, (b, t_b, R))$ when the underlying label (a, t, α) is of the form $(a, t, \mathtt{in}(u))$. 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, t_b the time at which this output has been done ($t_b \leq t$), and R the recipe that has been used to forge this output.
- $(a, \alpha, s, t, \emptyset)$ 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 $r = \mathsf{w}$ (case of the output), $r = (b, R)$ (case of the input), or $r = \emptyset$ 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 $\mathcal{P}$ be a protocol and $\mathcal{S}$ be a set of valid initial configurations. Assuming that $\mathcal{P}$ is well-timed w.r.t. $\mathcal{S}$, and $\mathcal{P}$ is DB-secure, we have that $\mathcal{P}$ 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^{Loc}_{exec} associated to an annotated relaxed execution $exec$, and a mobility plan Loc.

Given a trace $tr_1 \ldots tr_n$ we note $\mathsf{IN}(tr_1 \ldots tr_n)$ the set of all the indices corresponding to input actions. Similarly we note $\mathsf{TS}(tr_1 \ldots tr_n)$ the set of all the indices corresponding to timestamp events. Given a set S we note $\#S$ its size. Finally, we note $\mathtt{orig}(i)$ the index of the $\mathrm{i^{th}}$ output in the trace.

Timed formula associated to an execution. Given an annotated execution $exec = \mathcal{K}_0 = (\mathcal{P}_0; \Phi_0) \xrightarrow{tr_1.....tr_n} \mathcal{K}_n$ (with $tr_i = (a_i, \alpha_i, s_i, r_i)$ for $1 \leq i \leq n$) of size n and a mobility plan Loc, the timed formula C^{Loc}_{exec}, built upon the set of variables $\mathcal{Z} = \{z_1, \dots, z_n\} \cup \{z^b_i \mid i \in \mathsf{IN}(tr_1, \dots, tr_n)\}$, is the conjunction of the following formulas:

- $0 \leq z_1 \leq z_2 \leq \ldots \leq z_n$;
- $z_i < z_{i+1}$ for all $i \in \mathsf{TS}(tr_1, \dots, tr_{n-1})$;
- for all $i \in \mathsf{IN}(tr_1, \dots, tr_n)$,
$$z_i \geq z^b_i + \mathsf{Dist}(\mathsf{Loc}(a_i, z_i), \mathsf{Loc}(b_i, z^b_i));$$
- for all $i \in \mathsf{IN}(tr_1, \dots, tr_n)$, for all j such that $\mathsf{w}_j \in vars(R_i) \smallsetminus \mathsf{dom}(\Phi_0)$,
$$z^b_i \geq z_{\mathtt{orig}(j)} + \mathsf{Dist}(\mathsf{Loc}(b_i, z^b_i), \mathsf{Loc}(a_{\mathtt{orig}(j)}, z_{\mathtt{orig}(j)})).$$

We note that given a configuration $\mathcal{K}$, the transformation $\overline{\cdot}$ 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 Loc be a mobility plan.*

For any execution $exec = \overline{\mathcal{K}_0} \xrightarrow{tr_1...tr_n} \mathcal{K}_n$ with $tr_i = (a_i, \alpha_i, s_i, r_i)$ and function φ satisfying C^{Loc}_{exec} we have:

$$\mathcal{K}_0 \xrightarrow{tr'_1...tr'_n}_{Loc} \mathcal{K}'_n$$

with:

$$tr'_i = \begin{cases} (a_i, \mathtt{gettime}, s_i, \varphi(z_i), \emptyset) \text{ if } \alpha_i = \mathsf{timestamp}(c_i) \\ (a_i, \alpha_i\varphi_c, s_i, \varphi(z_i), r_i\varphi_c) \text{ otherwise} \end{cases}$$

and $\varphi_c(c_i) = \varphi(z_i)$ for all $i \in \mathsf{TS}(tr_1 \dots tr_n)$. Moreover, $\overline{\mathcal{K}'_n}\varphi_c = \mathcal{K}_n$.

Proof. The proof follows the definition of C^{Loc}_{exec}. Indeed this formula contains all the timing constraints to trigger each action. Moreover, by definition of C^{Loc}_{exec} we obtain that φ_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.

Proof. We assume that $\mathcal{P}$ is not causality-based 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 $exec$ such that:

$$exec = \overline{\mathcal{K}_0} \xrightarrow{tr_1...tr_n.(b_0, \mathsf{claim}(b_1, b_2, c^0_1, c^0_2), s, \emptyset)} (\mathcal{P}'; \phi')$$

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

(1) there is no $k \leq n$ such that $\alpha_k = \mathsf{check}(c^0_1, c^0_2, u)$ for some $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma^+_0)$; or
(2) there is no $i \leq n$ (resp. $j \leq n$) such that $(a_i, \alpha_i) = (b_1, \mathsf{timestamp}(c^0_1))$ (resp. $(a_j, \alpha_j) = (b_1, \mathsf{timestamp}(c^0_2))$); or
(3) there exist $i_0, j_0, k_0 \leq n$ and $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma^+_0)$ such that $\alpha_{k_0} = \mathsf{check}(c^0_1, c^0_2, u)$, $(a_{i_0}, \alpha_{i_0}) = (b_1, \mathsf{timestamp}(c^0_1))$, and $(a_{j_0}, \alpha_{j_0}) = (b_1, \mathsf{timestamp}(c^0_2))$ but there is no $i_0 \leq k' \leq j_0$ such that $a_{k'} = b_2$.

Let $tr_{n+1} = (b_0, \mathsf{claim}(b_1, b_2, c^0_1, c^0_2), s, t, r))$. We consider each case separately.

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

Let φ be the function such that:

- $\varphi(z_i) = \#\mathsf{TS}(tr_1 \dots tr_{i-1})$ for $i \in \{1, \dots, n+1\}$,
- for all $i \in \mathsf{IN}(tr_1 \dots tr_n)$, we have that:
  - $\varphi(z^b_i) = \varphi(z_{\mathtt{orig}}(j))$ if $b_i \notin \mathcal{M}$ and $R_i = \mathsf{w}_j$,
  - $\varphi(z^b_i) = \varphi(z_i)$ otherwise.

We can show that this function φ satisfies C^{Loc}_{exec}. Indeed, we have that:

- $0 \leq \varphi(z_1) \leq \varphi(z_2) \leq \ldots \leq \varphi(z_n)$;
- $\varphi(z_i) < \varphi(z_{i+1})$ for $i \in \mathsf{TS}(\mathrm{tr}_1, \dots, tr_{n-1})$;
- for all $i \in \mathsf{IN}(tr_1, \dots, tr_n)$, we have that either $\varphi(z^b_i) = \varphi(z_{\mathtt{orig}}(j)) \leq \varphi(z_i)$ for some j such that $\mathtt{orig}(j) \leq i$, or $\varphi(z^b_i) = \varphi(z_i)$. In both cases, we have that:
$$\varphi(z_i) \geq \varphi(z^b_i) + \mathsf{Dist}(\mathsf{Loc}(a_i), \mathsf{Loc}(b_i))$$
since $\mathsf{Dist}(\mathsf{Loc}(a_i), \mathsf{Loc}(b_i)) = 0$. Remember that all the agents are at the same location.
- for all $i \in \mathsf{IN}(tr_1, \dots, tr_n)$, for all j such that $\mathsf{w}_j \in vars(R_i) \smallsetminus \mathsf{dom}(\Phi_0)$, we have that $\mathtt{orig}(j) < i$, and thus $\varphi(z_{\mathtt{orig}(j)}) \leq \varphi(z_i)$. Therefore, we have that:
$$\varphi(z^b_i) \geq \varphi(z_{\mathtt{orig}(j)})$$
and we conclude since $\mathsf{Dist}(\mathsf{Loc}(b_i), \mathsf{Loc}(a_{\mathtt{orig}(j)})) = 0$. 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 φ_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 Loc be the mobility plan such that:

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

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

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

- $2 \cdot \#\mathsf{IN}(tr_1 \dots tr_i) + \#\mathsf{TS}(tr_1 \dots tr_{i-1})$ for $i < i_0$;
- $\varphi(z_{i_0-1}) + 1 + \frac{1}{\mathsf{c}_0 \cdot n}\#\mathsf{TS}(tr_{i_0} \dots tr_{i-1})$ for $i_0 \leq i \leq j_0$;
- $\varphi(z_{j_0}) + 2 \cdot \#\mathsf{IN}(tr_{j_0+1} \dots tr_i) + \#\mathsf{TS}(tr_{j_0+1} \dots tr_{i-1})$ for $1 > j_0$.

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 $1/(\mathsf{c}_0 \cdot n)$

after each timestamp to ensure that time increases between two timestamps as required by the semantics.

In addition, for all $i \in \mathsf{IN}(tr_1 \dots tr_n)$, if $b_i \notin \mathcal{M}$ then

$$\varphi(z_i^b) = \varphi(z_{\mathtt{orig}(j)}) \text{ where } R_i = \mathtt{w}_j \text{, otherwise:}$$

$$\varphi(z_i^b) = \begin{cases} \varphi(z_i) - 1 & \text{if } i < i_0 \text{ or } i > j_0 \\ \varphi(z_i) & \text{if } i_0 \le i \le j_0. \end{cases}$$

We can show that this function φ satisfies C_{exec}^{Loc}. Indeed, we have that:

- $0 \le \varphi(z_1) \le \varphi(z_2) \le \dots \le \varphi(z_n)$;
- $\varphi(z_i) < \varphi(z_{i+1})$ for $i \in \mathsf{TS}(\mathsf{tr}_1, \dots, tr_{n-1})$;
- Regarding the remaining constraints in C_{exec}^{Loc}, we consider $i \in \mathsf{IN}(tr_1, \dots, tr_n)$, and we show the result by distinguishing two sub-cases: Case $i < i_0$ or $j_0 < i$, and Case $i_0 \le i \le j_0$.

Once this is done, we can apply Lemma 1 to re-time the execution. We obtain that:

$$\mathcal{K}_0 \xrightarrow{tr'_1 \dots tr'_n . (b_0, \mathsf{claim}(b_1, b_2, c_1^0\varphi_c, c_2^0, \varphi_c), s, \emptyset)}_{\mathsf{Loc}} \mathcal{K}'_{n+1}$$

with:

$$tr'_i = \begin{cases} (a_i, \mathtt{gettime}, s_i, \varphi(z_i), \emptyset) \text{ if } \alpha_i = \mathsf{timestamp}(c_i) \\ (a_i, \alpha_i\varphi_c, s_i, \varphi(z_i), r_i\varphi_c) \text{ otherwise} \end{cases}$$

and $\varphi_c(c_i) = \varphi(z_i)$ for all $i \in \mathsf{TS}(tr_1 \dots tr_n)$.

By construction, we have that $\varphi_c(c_1^0) = \varphi(z_{i_0})$, $\varphi_c(c_2^0) = \varphi(z_{j_0})$, and, by definition of φ, we have that $\varphi(z_{i+1}) - \varphi(z_i) \le 1/(\mathsf{c}_0 \cdot n)$ when $i \in \{i_0, \dots, j_0 - 1\}$. Therefore, we have that:

$$\begin{array}{rcl} \varphi(z_{j_0}) - \varphi(z_{i_0}) & \le & (j_0 - i_0 + 1)/(\mathsf{c}_0 \cdot n) \le 1/\mathsf{c}_0 \\ & < & 2/\mathsf{c}_0 \\ & \le & 2 \times \mathsf{Dist}(\mathsf{Loc}(b_1), \mathsf{Loc}(b_2)) \end{array}$$

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

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 $\mathcal{K}_0 \xrightarrow{L_1}_{Loc} \cdots \xrightarrow{L_n}_{Loc} \mathcal{K}_n$ with $L_i = (a_i, \alpha_i, s_i, t_i, r_i)$, $L_j = (a_j, \alpha_j, s_j, t_i, r_j)$, we say that L_j is* dependent *of L_i, denoted $L_j \hookrightarrow L_i$, if $i < j$, and:*

- *either $s_i = s_j$ (and thus $a_i = a_j$), and in that case L_j is* sequentially-dependent *of L_i, denoted $L_j \hookrightarrow_s L_i$;*
- *or $\alpha_i = \mathtt{out}(v)$, $\alpha_j = \mathtt{in}(u)$, and $r_i \in vars(R_j)$ with $r_j = (b_j, t_j^b, R_j)$, and in that case L_j is* data-dependent *of L_i, denoted $L_j \hookrightarrow_d L_i$.*

We note $\hookrightarrow^*$ the transitive closure of $\hookrightarrow$. Finally we note $L_j \not\hookrightarrow^* L_i$ 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 $\mathcal{K}_0 \overset{L_1}{\rightsquigarrow} \mathcal{K} \overset{L_2}{\rightsquigarrow} \mathcal{K}_2$ such that $L_2 \not\hookrightarrow L_1$. We have that $\mathcal{K}_0 \overset{L_2}{\rightsquigarrow} \mathcal{K}' \overset{L_1}{\rightsquigarrow} K_2$ for some $\mathcal{K}'$.*

Proof. Let $\mathcal{K}_0 = (\mathcal{P}_0; \Phi_0)$, $\mathcal{K} = (\mathcal{P}; \Phi)$, and $\mathcal{K}_2 = (\mathcal{P}_2; \Phi_2)$ be such that $\mathcal{K}_0 \overset{L_1}{\rightsquigarrow} \mathcal{K} \overset{L_2}{\rightsquigarrow} \mathcal{K}_2$ with $L_2 \not\hookrightarrow L_1$. Let $L_1 = (a_1, \alpha_1, s_1, r_1)$ and $L_2 = (a_2, \alpha_2, s_2, r_2)$. Since $L_2 \not\hookrightarrow L_1$ we have that $s_1 \ne s_2$. Therefore, we have that:

- $\mathcal{P}_0 = \lfloor \mathsf{act}_1.P_1 \rfloor_{a_1} \cup \lfloor \mathsf{act}_2.P_2 \rfloor_{a_2} \uplus Q$,
- $\mathcal{P} = \lfloor P'_1 \rfloor_{a_1} \cup \lfloor \mathsf{act}_2.P_2 \rfloor_{\mathsf{act}_2} \uplus Q$,
- $\mathcal{P}_2 = \lfloor P'_1 \rfloor_{a_1} \cup \lfloor P'_2 \rfloor_{a_2} \uplus Q$

where act_1 and act_2 are actions of the form $\mathtt{in}(x)$, $\mathtt{gettime}(x)$, $\mathtt{let}\ x = v\ \mathtt{in}\ P\ \mathtt{else}\ Q$, $\mathsf{claim}(u_1, u_2, u_3, u_4)$, $\mathtt{new}\ n$, $\mathtt{out}(u)$, or $\mathsf{check}(u_1, u_2, u_3)$.

In case $\alpha_1 = \mathtt{out}(v)$ and $\alpha_2 = \mathtt{in}(u)$, we have that $\Phi_2 = \Phi = \Phi_0 \uplus \{\mathtt{w} \xrightarrow{a_1} v\}$ and since $L_2 \not\hookrightarrow_d L_1$, we know that $\mathtt{w} \notin vars(r_2)$. Thus, we have that $vars(r_2) \subseteq \mathsf{dom}(\Phi_0)$. Now, let $\mathcal{K}' = (\lfloor \mathsf{act}_1.P_1 \rfloor_{a_1} \uplus \lfloor P'_2 \rfloor_{a_2} \uplus Q; \Phi_0)$. Relying on the fact that $\mathtt{w} \notin vars(r_2)$ in case $\alpha_1 = \mathtt{out}(v)$ and $\alpha_2 = \mathtt{in}(u)$, it is easy to see that $\mathcal{K}_0 \overset{L_2}{\rightsquigarrow} \mathcal{K}' \overset{L_1}{\rightsquigarrow} \mathcal{K}_2$. The other cases can be treated in a rather similar way. □

Corollary 1. *Given a trace $\mathcal{K}_0 \overset{tr_1 \dots tr_n}{\rightsquigarrow} \mathcal{K}_n$ with $n \ge 2$ there exists a bijection $\varphi : \{1, \dots, n\} \to \{1, \dots, n\}$ such that:*

- *$tr_i = tr'_{\varphi(i)}$ for all $i \in \{1, \dots, n\}$;*
- *for all j such that $\varphi(1) < j < \varphi(n)$, we have that $tr'_{\varphi(n)} \hookrightarrow^* tr'_j \hookrightarrow^* tr'_{\varphi(1)}$;*
- *for all j_1, j_2 such that $\varphi(1) \le j_1 < j_2 \le \varphi(n)$, we have that $\varphi^{-1}(j_1) < \varphi^{-1}(j_2)$; and*
- *$\mathcal{K}_0 \overset{tr'_1 \dots tr'_n}{\rightsquigarrow} \mathcal{K}_n$.*

Proof. (sketch) We split the proof in two parts: first we prove that there exists a bijection φ_1 cleaning the trace between tr_1 and tr_n moving actions independent from tr_1 before it. Then we prove that there exists a bijection φ_2 cleaning the trace moving actions from which tr_n does not depend on after it. Considering $\varphi = \varphi_2 \circ \varphi_1$ we will be able to conclude. □

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 s strong relationship between the two executions.

To do so, we first precise the transformation $\bar{\cdot}$ presented in Section 4.2. When we apply this transformation, we rely on the function $\sigma_{\mathsf{spe}} : \mathcal{X} \to \Sigma_0^{\mathsf{spe}}$ which is used to replace an action of the form $\mathtt{gettime}(x)$ by the action $\mathsf{timestamp}(\sigma_{\mathsf{spe}}(x))$.

Similarly, given an execution $exec = \mathcal{K}_0 \xrightarrow{tr}_{\mathsf{Loc}} \mathcal{K}_0$, we denote $\sigma_{\mathsf{time}} : \mathcal{X} \to \mathbb{R}_+$ the function that associates to each variable occurring in a $\mathtt{gettime}$ 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*

$$exec = \mathcal{K}_0 \xrightarrow{tr_1 \ldots tr_n}_{Loc} \mathcal{K}_n$$

such that $tr_i = (a_i, \alpha_i, s_i, t_i.r_i)$ for $i \in \{1, \ldots, n\}$, we have that $\overline{\mathcal{K}_0} \overset{tr'_1 \ldots tr'_n}{\rightsquigarrow} \mathcal{K}'_n$ where $\mathcal{K}'_n = \overline{\mathcal{K}_n}\sigma$ and for any $i \in \{1, \ldots, n\}$, we have that:

$$tr'_i = \begin{cases} (a_i, \mathsf{timestamp}(t_i\sigma), s_i, \emptyset) & \text{if } \alpha_i = \mathtt{gettime} \\ (a_i, \alpha_i\sigma, s_i, (b_i, R_i\sigma)) & \text{if } r_i = (b_i, t_i^b, R_i) \\ (a_i, \alpha_i\sigma, s_i, r_i\sigma) & \text{otherwise} \end{cases}$$

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

Proof. This proof is immediate because the configurations only differ from the bijective function σ (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 tr_1 and tr_2 such that $tr_2 \hookrightarrow^* tr_1$ we know that enough time must has elapsed after the execution of tr_1 to be able to trigger tr_2.

Lemma 4. *Let Loc be a mobility plan, and $exec = \mathcal{K}_0 \xrightarrow{tr_1 \ldots\ldots tr_n}_{Loc} \mathcal{K}_1$ be an execution with $tr_i = (a_i, \alpha_i, s_i, t_i, r_i)$ for $i \in \{1, \ldots, n\}$. Let $i, j \in \{1, \ldots, n\}$ such that $tr_j \hookrightarrow^* tr_i$. We have that:*

$$t_j \geq t_i + \mathit{Dist}(\mathit{Loc}(a_i, t_i), \mathit{Loc}(a_j, t_j)).$$

Proof. By definition of $\hookrightarrow^*$, we know that there exists a sequence $n \geq i_1 > i_2 > \ldots > i_k \geq 1$ such that:

$$tr_j = tr_{i_1} \hookrightarrow tr_{i_2} \hookrightarrow \ldots \hookrightarrow tr_{i_k} = tr_i.$$

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

$$tr_j = tr_{i_1} \hookrightarrow tr_{i_2} \hookrightarrow \ldots \hookrightarrow tr_{i_k} = tr_i.$$

By induction hypothesis, we have that:

$$t_{i_2} \geq t_i + \mathsf{Dist}(\mathsf{Loc}(a_i, t_i), \mathsf{Loc}(a_{i_2}, t_{i_2})).$$

We distinguish two cases depending on the nature of the dependency $tr_j \hookrightarrow tr_{i_2}$.

<u>Case $tr_j \hookrightarrow_s tr_{i_2}$.</u> In such a case, we have that $a_j = a_{i_2}$ and $j \geq i_2$. Moreover, we have that $t_j \geq t_{i_2}$, and by definition of a mobility plan we know that

$$t_j - t_{i_2} \geq \mathsf{Dist}(\mathsf{Loc}(a_j, t_j), \mathsf{Loc}(a_j, t_{i_2})).$$

Relying on our induction hypothesis, we have that:

$$\begin{aligned} t_j \geq\ & t_{i_2} + \mathsf{Dist}(\mathsf{Loc}(a_j, t_j), \mathsf{Loc}(a_j, t_{i_2})) \\ \geq\ & t_i + \mathsf{Dist}(\mathsf{Loc}(a_i, t_i), \mathsf{Loc}(a_{i_2}, t_{i_2})) \\ & + \mathsf{Dist}(\mathsf{Loc}(a_j, t_j), \mathsf{Loc}(a_j, t_{i_2})) \\ \geq\ & t_i + \mathsf{Dist}(\mathsf{Loc}(a_i, t_i), \mathsf{Loc}(a_j, t_j)). \end{aligned}$$

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

<u>Case $tr_j \hookrightarrow_d tr_{i_2}$.</u> In such a case, we have that $r_j = (b, t_b, R)$, $r_{i_2} = \mathsf{w}$ and $\mathsf{w} \in \mathit{vars}(R)$. By definition of the IN rule we have that:

- $t_j \geq t_b + \mathsf{Dist}(\mathsf{Loc}(a_j, t_j), \mathsf{Loc}(b, t_b))$, and
- $t_b \geq t_{i_2} + \mathsf{Dist}(\mathsf{Loc}(b, t_b), \mathsf{Loc}(a_{i_2}, t_{i_2}))$.

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

$$t_j \geq t_{i_2} + \mathsf{Dist}(\mathsf{Loc}(a_j, t_j), \mathsf{Loc}(a_{i_2}, t_{i_2})).$$

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

$$\begin{aligned} t_j \geq\ & t_i + \mathsf{Dist}(\mathsf{Loc}(a_i, t_i), \mathsf{Loc}(a_{i_2}, t_{i_2})) \\ & + \mathsf{Dist}(\mathsf{Loc}(a_j, t_j), \mathsf{Loc}(a_{i_2}, t_{i_2})) \\ \geq\ & t_i + \mathsf{Dist}(\mathsf{Loc}(a_j, t_j), \mathsf{Loc}(a_i, t_i)). \end{aligned}$$

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 $t_2^0 - t_1^0 < \delta$ (for some threshold δ) 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 apply Lemma 1 to lift this execution in the timed model keeping the value $t_2^0 - t_1^0$ 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 $t_2^0 - t_1^0 \geq \delta$ leading to contradiction.

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

Proof. (sketch) We assume that $\mathcal{P}$ is not DB-secure, and thus there exist a valid initial configuration $\mathcal{K}_0 \in \mathcal{S}$ and an execution *exec* such that:

$$exec = \mathcal{K}_0 \xrightarrow{tr_1 \ldots tr_n.(b_0, \mathsf{claim}(b_1, b_2, t_1^0, t_2^0), s, t, \emptyset)}_{\mathsf{Loc}} \mathcal{K}_{n+1}$$

with $b_1 \notin \mathcal{M}$ and $b_2 \notin \mathcal{M}$ and either:

(1) there is no index $k \leq n$ such that $tr_k = (a_k, \mathsf{check}(t_1^0, t_2^0, t_3^0), s_k, t_k, \emptyset)$; or
(2) for any t with $t_1^0 \leq t \leq t_2^0$, we have that:

$$t_2^0 - t_1^0 < \begin{aligned}[t] & \mathsf{Dist}(\mathsf{Loc}(b_1, t_1^0), \mathsf{Loc}(b_2, t)) \\ & + \mathsf{Dist}(\mathsf{Loc}(b_2, t), \mathsf{Loc}(b_1, t_2^0)) \end{aligned}$$

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

$$exec' = \overline{\mathcal{K}_0} \overset{tr'_1 \ldots tr'_n.(b_0, \mathsf{claim}(b_1, b_2, t_1^0\sigma, t_2^0\sigma), s, \emptyset)}{\rightsquigarrow} \mathcal{K}'_{n+1}$$

where $\mathcal{K}'_{n+1} = \overline{\mathcal{K}_{n+1}}\sigma$ and for any $i \in \{1, \dots, n+1\}$, we have that:

$$tr'_i = \begin{cases} (a_i, \mathsf{timestamp}(t_i\sigma), s_i, \emptyset) & \text{if } \alpha_i = \mathtt{gettime} \\ (a_i, \alpha_i\sigma, s_i, (b_i, R_i\sigma)) & \text{if } r_i = (b_i, t_i^b, R_i) \\ (a_i, \alpha_i\sigma, s_i, r_i\sigma) & \text{otherwise} \end{cases}$$

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

We assume by contradiction that $\mathcal{P}$ 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 $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \Sigma_0^+)$ such that:

- $\alpha_k\sigma = \mathsf{check}(t_1^0\sigma, t_2^0\sigma, u\sigma)$;
- $(a_{i_0}, \alpha_i\sigma) = (b_1, \mathsf{timestamp}(t_1^0\sigma))$;
- $(a_{j_0}, \alpha_j\sigma) = (b_1, \mathsf{timestamp}(t_2^0\sigma))$; and
- $a_{k'} = b_2$.

By definition of *exec'*, we have that $tr_k = (a_k, \mathsf{check}(t_1^0, t_2^0, u), s_k, t_k, \emptyset)$ for some time $t_k \in \mathbb{R}_+$, and this 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 $\mathcal{K}'_{i_0-1} \xrightarrow{tr'_{i_0} \dots tr'_{j_0}} \mathcal{K}'_{j_0}$ of *exec'*, and we obtain that there exists a bijection $\varphi_{\mathrm{act}} : \{i_0, \dots, j_0\} \to \{i_0, \dots, j_0\}$ such that:

- $tr'_i = tr''_{\varphi_{\mathrm{act}}(i)}$ for all $i \in \{i_0, \dots, j_0\}$;
- for all j such that $\varphi_{\mathrm{act}}(i_0) < j < \varphi_{\mathrm{act}}(j_0)$, we have that $tr''_{\varphi_{\mathrm{act}}(j_0)} \hookrightarrow^* tr''_j \hookrightarrow^* tr''_{\varphi_{\mathrm{act}}(i_0)}$;
- for all j_1, j_2 such that $\varphi_{\mathrm{act}}(i_0) \leq j_1 < j_2 \leq \varphi_{\mathrm{act}}(j_0)$, we have that $\varphi_{\mathrm{act}}^{-1}(j_1) < \varphi_{\mathrm{act}}^{-1}(j_2)$; and
- $\mathcal{K}'_{i_0-1} \xrightarrow{tr''_{i_0} \dots tr''_{j_0}} \mathcal{K}'_{j_0}$.

We have thus an execution *exec''* such that:

$$\begin{aligned} exec'' = \quad & \overline{\mathcal{K}_0} \xrightarrow{tr'_1 \dots tr'_{i_0-1}} \mathcal{K}'_{i_0-1} \xrightarrow{tr''_{i_0} \dots tr''_{j_0}} \mathcal{K}'_{j_0} \\ & \xrightarrow{tr'_{j_0+1} \dots tr'_n . (b_0, \mathsf{claim}(b_1, b_2, t_1^0\sigma, t_2^0\sigma), s, \emptyset)} \mathcal{K}'_{n+1}. \end{aligned}$$

For sake of simplicity, for $i < i_0$ and $i > j_0$ we define $tr''_i = tr'_i$. For all $i \in \{1, \dots, n\}$ we define a''_i the name of the agent executing tr''_i. If $i \in \mathsf{IN}(tr''_1 \dots tr''_n)$, we also define R''_i (resp. b''_i) the recipe (resp. agent name) occurring in tr''_i. We have that $a''_{\varphi_{\mathrm{act}}(i)} = a_i$, $R''_{\varphi_{\mathrm{act}}(i)} = R_i\sigma$ and $b''_{\varphi_{\mathrm{act}}(i)} = b_i$.

Then, we aim at re-timing the trace *exec''* without changing the amount of time that elapses between the two timestamps instructions. Once this is done, the dependencies

$$tr''_{\varphi_{\mathrm{act}}(j_0)} \hookrightarrow^* tr''_j \hookrightarrow^* tr''_{\varphi_{\mathrm{act}}(i_0)}$$

for any j such that $\varphi_{\mathrm{act}}(i_0) < j < \varphi_{\mathrm{act}}(j_0)$ together with the fact that there exists such a j such that $a_j = b_2$ (since $\mathcal{P}$ is assumed to be causality-based secure) will lead us to a contradiction thanks to Lemma 4. □

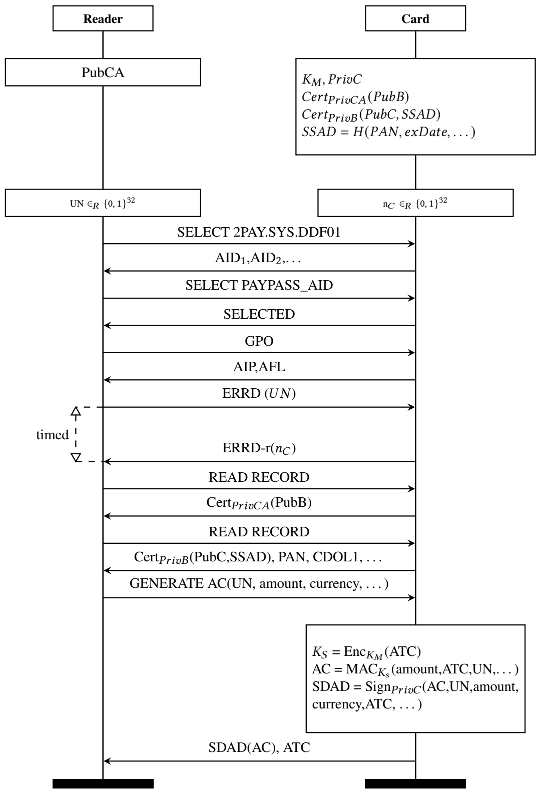

Figure 5: The PayPass-RRP Protocol

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 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 $Priv_C$;
(2) a symmetric key K_M that it shares with the bank;
(3) a certificate chain $Cert_{Priv_{CA}}(Pub_C)$ for the card's public key Pub_C.

The reader has the public key Pub_{CA} 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

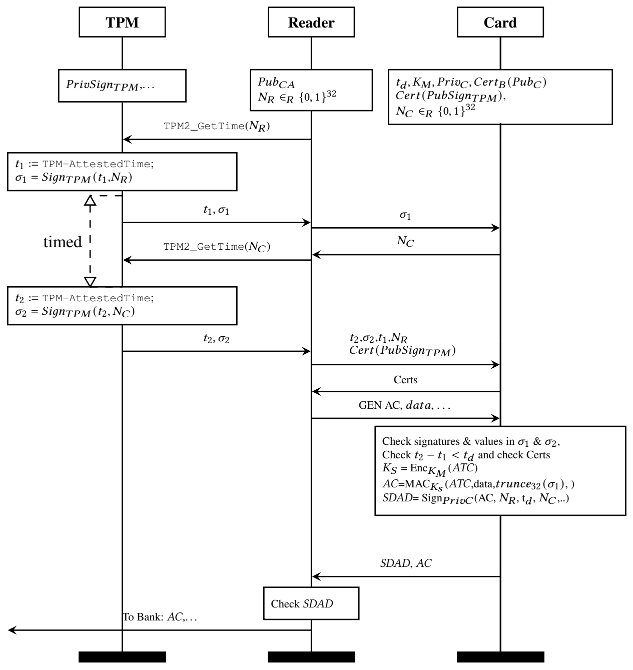
TPM
Reader
Card
$PrivSign_{TPM}, \ldots$
Pub_{CA} $N_R \in_R \{0,1\}^{32}$
$t_d, K_M, Priv_C, Cert_B(Pub_C)$ $Cert(PubSign_{TPM})$, $N_C \in_R \{0,1\}^{32}$
TPM2_GetTime(N_R)
t_1 := TPM-AttestedTime; $\sigma_1 = Sign_{TPM}(t_1, N_R)$
t_1, σ_1
σ_1
timed
TPM2_GetTime(N_C)
N_C
t_2 := TPM-AttestedTime; $\sigma_2 = Sign_{TPM}(t_2, N_C)$
t_2, σ_2
t_2, σ_2, t_1, N_R $Cert(PubSign_{TPM})$
Certs
GEN AC, $data, \ldots$
Check signatures & values in σ_1 & σ_2, Check $t_2 - t_1 < t_d$ and check Certs $K_S = \text{Enc}_{K_M}(ATC)$ $AC = \text{MAC}_{K_S}(ATC, \text{data}, trunce_{32}(\sigma_1),)$ $SDAD = \text{Sign}_{Priv_C}(\text{AC}, N_R, t_d, N_C \ldots)$
SDAD, AC
Check SDAD
To Bank: $AC, \ldots$

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

and reader then generate single-use random numbers N_C 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 N_C, 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 K_M, 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 K_S. The cryptogram *AC* is a MAC keyed with K_S 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*, N_C. 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_d(cardID) to be enforced for the proximity-checking phase is embedded in each card; we denote it t_d. 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 *TPM2_GetTime* command, the TPM timestamps it with `TPM-AttestedTime`, and using a randomised signing algorithm to produce the signature σ_1. Then, the EMV reader forwards σ_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 *ERRD* command ; (b) with a new *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 *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 ***"honest''*** and the latter – ***"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: "`Select` *PPSE*", "`Select` *AID*", "*GPO*", "*ERRD*", "`Read Records`", "*GEN AC*".
(2) In the above, the *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 $\sigma_1^{'}$);
(3) The *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 – "*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 *ERRD* command;
  - Max Time for Processing Relay Resistance APDU, i.e., the maximal time the card takes to process this *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[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:

- `AIP` (Application Interchange Profile) response
- `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: (a) `AIP` = 1981 (meaning the card supports PayPass-RRP); (b) `TVR` = 0000000002 (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: (a) `AIP` = 1981 (meaning the card supports PayPass-RRP); (b) `TVR` = 000000000E (meaning that 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 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 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. 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 a 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 this 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[11] gathered and discussed below. Due space constraints, we mainly detail herein for the case of PayBCR (and not the case of PayPass-RRP).

Figure 7 shows part of the *ERRD* command as demonstrably executed by our honest PayBCR terminal. Lines 89-91 show that we store if RRP is performed and, for this honest terminal, the logs

[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.

[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.

```
 85/          /! --------------------------------
 86/          /!    ERRD
 87/          /! -------------------------------
 88/          /
 89/          /[DECLARE RRP_PERFORMED = Y]
 90/          /[PRINT RRP_PERFORMED]
 91/          /:RRP_PERFORMED = Y
      ...
 95/          /[IN_CLA = 80]
 96/          /[IN_INS = EA]
 97/          /[IN_P1  = 00]
 98/          /[IN_P2  = 00]
 99/          /[IN_LC  = 04]
100/          /[IN_CDATA = [IN_RRP_TRRE]]
101/          /[IN_CDATA = 11121314]
102/          /[IN_LE  = 00]
103/          /! ------------
104/          /![START_TIMER = MICROTICKS()]
105/          /[TMP = START]
106/          /[TPM_CONN = TPM2CONNECT()]
107/          /[TESTEQ [TPM_CONN], TRUE]
107/          /[TESTEQ TRUE, TRUE]
108/              /[DOIFEQ [TMP], START]
 108/   1/          /[DOIFEQ START, START]
 108/   2/          /!CALL TPM GET2TIME([IN_RRP_TRRE]{UN})
 108/   3  /[GETTIME_RESULT = TPM2GETATTESTEDTIME([IN_RRP_TRRE])]
 108/   4  /[GETTIME_RESULT = TPM2GETATTESTEDTIME(11121314)]
 108/   5/          /[TPM_TIMESTAMP_1 = TPM2GET("TIMEINFO","TIME")]
 108/   6/          /[TPM_SIGNATURE_1 = TPM2GET("SIGNATURE","SIG")]
 108/   7/          /[IN_RRP_TRRE = LEFT([TPM_SIGNATURE_1], 04)]
 108/   8/          /[IN_RRP_TRRE = LEFT(405C6808E4153207973FF84A133
430EFA07B298433D2F2A1E9E60F76E8433D7EAC058C4F78C04D967F3D29A19044C59
26093564847EE991364B4D65AA63A489CF9245A48378B138B74483C925A5E48866F8
6D40049DD5C03845F83DD73028FFD2BC0140B53936AE07A5257A97F123EA661781EF
14A9352319C1064A5C052B9A83A00DAA70CD7C6BCC07022B6E39788F337C9A97F4DF
CF40E574F6CA389BB0FDF054BE6DA155A13430C1436FD597AF47BE3CAB280DDB796C
C168EFA5CCC3130F5AF517115522A6A271BAB8CE96CAED270D66A8EE8EAADD
108/     9/ /[IN_CDATA = 405C6808]
108/   10/ /[DOIFEND] ...
108/  11/ /[DOIFEQ [TMP], STOP]
108/  12/ /[DOIFEQ START, STOP]
108/  13/ /! DOIF !!CALL TPM GET2TIME(Card Nonce)
108/  14/ /! DOIF ![IN_RRP_DRRE = MID([OUT], 03, 04)]
108/  15/ /! DOIF ![GETTIME_RESULT = TPM2GETATTESTEDTIME([IN_RRP_DRRE])]
108/  16/ /! DOIF ![TPM_TIMESTAMP_2 = TPM2GET("TIMEINFO","TIME")]
108/  17/ /! DOIF ![TPM_SIGNATURE_2 = TPM2GET("SIGNATURE","SIG")]
108/  18/ /! DOIF ![TPM_CONN = TPM2CLOSE()]
108/  19/ /! DOIF ![TESTEQ [TPM_CONN], ]
```

Figure 7: Snippet of CardCracker Logs – Part1 of *ERRD* Command in a Honest PayBCR Terminal

show that it is indeed carried out. Lines 95–102 show standard EMV APDU headers. In our implementation, `IN_RRP_TRRE` stands for a list that stores data linked to the RRP process on the terminal side. In a similar list, denoted `IN_RRP_DRRE`, we do this for the card side as well. Line 106 shows that a connection to the TPM is opened, and in lines 108/2 and 108/3 – the UN nonce from the reader is gathered inside the `IN_RRP_TRRE` and passed to the TPM as an argument to the `TPM2_GetTime` command. Lines 108/13 show that, similarly, the nonce from the card is gather inside `IN_RRP_DRRE` and passed to the TPM. In variables denoted `TPM_TIMESTAMP_1` and `TPM_SIGNATURE_1` (respectively ..._2), we stored the timestamps and signatures returned from the `TPM2_GetTime` commands called to the TPM.

In Figure 8, we partly show that our (honest) PayBCR reader does also then follow the RRP process as per EMV-C2's Sections 3.10, 5.3 and 6.6 and partly recalled by us in Subsection D.1.

Line 104 in Figure 7 shows that we are taking timings for performance analysis purposes.

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

```
  9/ /[RRP_TIME_TAKEN = SUB([RRP_TIME_TAKEN],
       [TERM_EXPECT_TRANSMIT_TIME_RRP_CAPDU])]
 10/ /[RRP_TIME_TAKEN = SUB(14A0, 0012)]
 11/ /[RRP_TIME_TAKEN = SUB([RRP_TIME_TAKEN], [RRP_MIN_CALC])]
 11/ /[RRP_TIME_TAKEN = SUB(148E, 0018)]
 12/ /
 13/ /[DOIFGE 0, [RRP_TIME_TAKEN]]
 13/ /[DOIFGE 0, 1476]
 14/ /! DOIF ![RRP_MAX_CALC = 0]
 15/ /[DOIFELSE]
 16/ /[RRP_MAX_CALC = [RRP_TIME_TAKEN]]
 16/ /[RRP_MAX_CALC = 1476]
 17/ /[DOIFEND]
 18/ /
 19/ /[MEASURED_RRP_PROCESSING_TIME = [RRP_MAX_CALC]]
 19/ /[MEASURED_RRP_PROCESSING_TIME = 1476]
 20/ /[PRINT MEASURED_RRP_PROCESSING_TIME]
 20/ /:MEASURED_RRP_PROCESSING_TIME = 1476
 21/ /[AS_BCD = HEXTOBCD([MEASURED_RRP_PROCESSING_TIME])]
 21/ /[AS_BCD = HEXTOBCD(1476)]
 22/ /[PRINT AS_BCD]
 22/ /:AS_BCD = 5238
 23/ /
 24/ /! C-2 SR1.19
 25/ /[RRP_MAX_CALC = SUB([RRP_MIN_GRACE_PERIOD],
                           [IN_RRP_MIN_TFPRRAPDU])]
 25/ /[RRP_MAX_CALC = SUB(0014, 0000)]
```

Figure 8: Snippet of CardCracker Logs – Part2 of *ERRD* Command in a Honest PayBCR Terminal

indicate that the bank does the checks of timings and outputs the result back.

```
259/ [PRINT OUT_AC]
259/ :OUT_AC =
260/ [PRINT TPM_TIMESTAMP_1]
260/ :TPM_TIMESTAMP_1 = F56959
261/ [PRINT TPM_TIMESTAMP_2]
261/ :TPM_TIMESTAMP_2 = F57348
262/ [PRINT TPM_SIGNATURE_1]
262/ :TPM_SIGNATURE_1 =
94EF2052B4753010B28419A88E0CF3B3CF2682468D6708EBD7DCC9959F41916
A5ECAD3786557C556FA0918ADDA1012E98F44F4146E96669024064E478E37BF
49F147114D3B9913024B100E2EEF22B059E48812B273608B39C16F0F77D042C
057D7FF68A4C07DE3EE861B71CBAB37FF89483A07C9F789647F230DCD360487
A04FD707C7E3D53459F48C488D8B1E3B5EA0A1F0B3E7FD393BEA989EC01933B
DE44BDF3A36288C6B2F19F3E1CF4D3E1BC3F8FE79AE1C8C6D681ECCE900F7AD
40659F3FEEDE75C4804319B0BF3D5CAC91DA51AFDD3ABCE3A949128EC4789F5
FE8301DE755405C320C7B9663776253ABEBF22D915F8C03F9534991AF2BF9F9
C6212B0D
263/ [PRINT TPM_SIGNATURE_2]
263/ :TPM_SIGNATURE_2 =
A9DF2C92173E30D3EB9562FA05A7CF9EB9DAED9367C7D9719673FD6BCF3CB64
4460ABD6E8A90E7802481A041808DA55475F1E7D2066FF28B632105441B98B2
28B0A377519B026538A3E52C50E54882A0ABFC49D6E13460CABC1F0A07BDC3D
973467F1F6AEFB139E144EB09B5631A5B500210F39B1F1FC1A7CAABC566AEF6
E95EA288A418D517BA8F948A372FF24148E8B138D23828BB92C629EB71D8238
7DC73784161527BEFEF5EF96E95CD0DE1C1A02BD3140D34A1BAA8285D4FCF1A
4EEECF66D76CF421B6FC6AACFE78A4C38CC2D19E812C8F268AF2C6D2A6F50FC
477EB997D855667381263EE10BF1A76EE5C431FC9EC7E996E151F1232B45734
1AAAB2F3
...
25/ [RECALC_RRP_MAX_CALC = [RECALC_TIMETAKEN]]
26/ [RECALC_RRP_MAX_CALC = 632C]
27/ [DOIFEND]
28/
29/ [RECALC_MEASURED_RRP_PROCESSING_TIME = [RECALC_RRP_MAX_CALC]
...
30/ [PRINT RECALC_MEASURED_RRP_PROCESSING_TIME]
...
31/ [AS_BCD = HEXTOBCD([RECALC_MEASURED_RRP_PROCESSING_TIME])]
31/ [AS_BCD = HEXTOBCD(632C)]
32/ [PRINT AS_BCD] 32/ :AS_BCD = 25388
```

Figure 9: Snippet of CardCracker Logs – Part of the Bank Rechecking RTTs

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.