Modelling and Analysis of a Hierarchy of Distance Bounding Attacks

Tom Chothia
University of Birmingham
Birmingham, UK

Joeri de Ruiter
Radboud University
Nijmegen, The Netherlands

Ben Smyth
University of Luxembourg,
Luxembourg

Abstract

We present an extension of the applied pi-calculus that can be used to model distance bounding protocols. A range of different security properties have been suggested for distance bounding protocols; we show how these can be encoded in our model and prove a partial order between them. We also relate the different security properties to particular attacker models. In doing so, we identify a new property, which we call *uncompromised distance bounding*, that captures the attacker model for protecting devices such as contactless payment cards or car entry systems, which assumes that the prover being tested has not been compromised, though other provers may have been. We show how to compile our new calculus into the applied pi-calculus so that protocols can be automatically checked with the ProVerif tool and we use this to analyse distance bounding protocols from MasterCard and NXP.

1 Introduction

Contactless payment cards and "keyless" car entry systems aim to make life easier. However, they also make it possible to wirelessly-pickpocket a victim [12] or even steal their car [21]. Such exploits are not merely theoretical; criminal gangs are using such attacks to steal cars [6]. Thieves relay signals from a victim's key fob (located inside the victim's house) to the victim's car (parked outside), which enables the thieves to unlock the car, start the engine, and drive away.

Distance bounding protocols [11] use round trip times to establish an upper-bound on the distance between a "prover", e.g., a contactless payment card or key fob, and a "verifier", e.g., a payment machine or car. This can be used to enforce that a prover is co-located with a verifier. Hence, they can be used to prevent the aforementioned attacks. Round trip times are sometimes bounded by the speed of light [11] and sometimes by the lag introduced by relaying equipment [20].

A distance bounding attack occurs when a verifier is deceived into believing they are co-located with a prover, when they are not. Attackers may relay, replay and alter messages, as well as trying to predict or preempt timed challenges. Some distance bounding protocols also aim to defend against a "dishonest prover" attacker , i.e., an attacker that knows all of the secret values of a normal prover, but will misuse them to try to trick a verifier. Other attacker models consider a weaker "terrorist prover," i.e., a dishonest prover that will not reveal its long term keys. The literature on symbolic verification of distance bounding protocols includes five different types of attacks, each of which uses some combination of basic, unprivileged attackers, dishonest prover attackers, and terrorist fraud attackers. We describe these in detail in the next section.

In this paper, we extend the applied pi-calculus [2] to distinguish between co-located processes and processes at distinct locations, and we restrict communication between locations using timers. In particular, when a location's timer is active, processes at that location may only receive input from co-located processes (they cannot receive input from a remote process, i.e., a process at a different location). Our extended calculus allows us to model distance bounding protocols. Indeed, we can consider an attacker, some provers and a verifier in various locations. Moreover, timers capture bounded round trip times, in particular, a verifier cannot receive any input from a remote attacker whilst a timer is active at the verifier's location. Thus, the calculus allows us to check for and detect each of the different types of attack against distance bounding protocols. Furthermore, we define a compiler that encodes the calculus into the standard applied pi-calculus, which enables automated analysis using tools such as ProVerif [8].

Industrial distance bounding protocols such as Mastercard's RRP protocol [20] and NXP's "proximity check" [14, 25] aim to protect payments and access tokens from relay attacks. These protocols need not defend against at-

tacks requiring dishonest provers, because if an attacker gets access to the secret keys, they can clone the cards or key fobs, and make payments or gain access without a need to relay the original device, i.e., protection is only needed for an uncompromised device.

However, we expect some devices (e.g., EMV cards or car fobs) may be compromised at some point, and we would like to ensure that the compromise of a particular prover would not lead to an attacker being able to successfully attack other provers. None of the commonly considered distance bounding security properties (which are presented in the next section) match this attacker model.

Using our calculus, we are able to consider all possible combinations of verifiers, provers and dishonest provers and so enumerate all possible distance bounding attack scenarios. Defending against each of these attack scenarios gives us a security property, and under reasonable assumptions (which we detail in Section 5) we can equate many of these distance bounding attack scenarios and impose a partial order on the others so creating a hierarchy of distance bounding attacks. Different parts of this hierarchy relate to different attacker models, and each attacker model is dominated by a single security property (this ordering is presented in Figure 3 on page 11). Our ordering shows that, under reasonable assumptions, "assisted distance fraud" attacks [13] are more powerful than all other properties. Moreover, it shows that when an attacker can only act remotely, protection against "distance hijacking" attacks [13] is the most powerful property needed. Details of these attacks are given in the next section.

From our hierarchy of distance bounding protocols we identify a new distance bounding attack scenario and security property, which we call *uncompromised distance bounding* security. In an uncompromised distance bounding attack the provers being targeted are remote from the verifier and the attacker acts at both the location of the prover and the verifier. Additionally, the attacker may have compromised a number of other provers at both locations, and use these in the attack. An uncompromised distance bounding attack exists if the attacker can cause the verifier to believe that one of the uncompromised, remote targeted provers is in fact local to the verifier. Defending against this kind of attack is the strongest security property needed for protocols such as MasterCard's RRP to protect contactless payment cards or NXP's proximity check when being used to protect, e.g., access to buildings.

We demonstrate the applicability of our results by analysing MasterCard's RRP protocol for distance bounding of contactless EMV [20], and a distance bounding protocol from NXP [14, 25]. These protocols have not been studied before. In these protocols the prover will send information about how long replies are expected to take and the verifier will use this information to set the time limits used in the distance bounding protocol. If attackers can alter these time limits then they can succeed in a relay attack by telling the verifier to wait long enough to relay the messages. As our calculus is based on the applied pi-calculus we are also able to check that the protocols ensure the authenticity of the timing information to confirm that attacks on it are not possible.

Contributions: Our contributions are as follows:

- An extension of the applied pi-calculus with locations and timer actions (Section 3).
- Formalizations of security properties for distance bounding protocols (Section 4).
- A hierarchy of our security properties, relations to particular attacker models, and identification of a new security property (Section 5).
- A practical, automatic tool for the analysis of distance bounding protocols, based on compiling our calculus into the applied pi-calculus (Section 6).
- Formal analysis of distance bounding protocols, including from MasterCard and NXP (Section 7).

Our models, compiler and full paper (with proofs) are on our project website `https://cs.bham.ac.uk/~tpc/distance-bounding-protocols/`

Related work: Some prior work on the verification of distance bounding protocols has used manual reasoning, e.g., [30, 34] in the symbolic model, [4, 9, 10, 18] in the computational model and [13, 34] using theorem provers.

Some previous work on automatic analysis of distance bounding protocols has been based on the applied pi-calculus: Malladi et al. [27] analyse single runs, Chothia et al. [12] analyse an arbitrary number of runs for relay attacks, and Debant et al. [15] provide a model with a formal correctness proof, which uses a definition of relay attack that is close to our definition of uncompromised distance bounding.

Nigam et al. [31] introduce an extension to strand spaces to model security protocols that include time and Kanovich et al. [26] consider a multiset rewriting model and compare discrete and continuous time. A contribution of our paper is to show that you do not need to explicitly consider the time of actions to meaningfully analyse distance bounding protocols. Mauw et al. [28] improves on the framework of [34] looking at causality between actions to make a framework for automatically testing distance fraud and terrorist fraud.

None of the previous papers on symbolic checking of distance bounding protocols consider the full range of

distance bounding properties or makes comparisons between them.

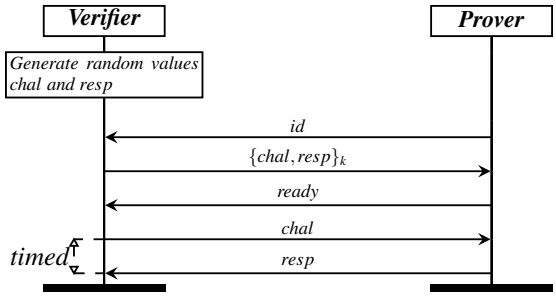
A recent survey [3] gives many examples of distance bounding protocols and attacks. Two notable protocols missing from this survey are MasterCard's RRP protocol for contactless EMV cards and NXP's "proximity check", which we both consider in this paper. MasterCard's RRP is a variant of the PaySafe protocol, which we have previously proposed for contactless EMV [12].

Past papers [15, 28, 31] have reported an attack against PayWave when the prover is dishonest. However, as we discuss in Section 5, if an EMV card has been compromised, then there is no need to relay it, hence such "distance fraud attacks" are not the correct attacker model for contactless EMV. In contrast, we relate distance bounding security properties to particular attacker models.

2 Distance bounding protocols and attacks

Distance bounding protocols aim to let a verifier place an upper-bound on the distance to a prover by timing how long it takes for certain challenges to be answered. Cryptography is used to ensure that the responder had to know the challenge before replying. Often the time taking to perform complex cryptography will vary between runs, therefore it is difficult to time cryptographic actions, and the challenge-response mechanism is typically limited to a simple exchange of nonces, with the cryptography performed before or afterwards.

Example 1. *As a running example we consider the following distance bounding protocol, in which the verifier and all provers share the same symmetric key.*



The verifier receives the identifier of the prover, generates nonces chal and resp, and sends the encrypted nonces to the prover. Once the prover indicates that it has decrypted the nonces, the verifier activates a timer, and sends nonce chal to the prover. The prover waits for nonce chal before revealing nonce resp, hence, the nonce is only revealed once the verifier's timer is running.

This protocol is not vulnerable to relay fraud because only the prover can decrypt the challenge and response, and an honest prover will not release the response until it receives the challenge, i.e., the attacker cannot learn the response until the timer has started, and then, if the prover is remote from the verifier, it will be impossible to get this response to the prover without the timer expiring.

Our example protocol does *not* defend against a dishonest prover that tries to trick the verifier, i.e., a prover can convince the verifier that it is nearer than it really is. Such a dishonest prover could be a hardware device that has been compromised by an attacker, or the owner of a device trying to mislead the verifier. Indeed, the prover can send the response early, before receiving the challenge, so the verifier receives the response just after it transmits the challenge. This will lead to a short delay between the challenge and response, making the verifier incorrectly believe that the prover is nearby.

The right security property for a distance bounding protocol, will depend on the use case. Common security properties considered in the literature on symbolical of checking distance bounding protocols include:

- Relay/Mafia Fraud [17]: The verifier and the prover are remote from each other. Attackers act at the same location as both the verifier and prover, and may relay, alter or replay messages, to trick the prover into believing that the prover is in fact local.
- Distance Fraud [16]/Lone Distance Fraud [13]: A dishonest prover, which may deviate from the protocol, is at a location remote from the verifier. This dishonest prover misleads the verifier into believing that it is local.
- Distance Hijacking [13]: A dishonest prover remotely authenticates to a verifier, as in Distance Fraud, but there are also other honest provers at the same location as the verifier, which the dishonest prover may make use of.
- Terrorist Fraud [16]: A terrorist fraud attack involves one attacker acting locally to the verifier along with a remote dishonest prover, with the goal of making the verifier believe that the remote dishonest prover is in fact local. This kind of attack always assumes that the prover has a secret key that identifies it and that the prover does not send this key to any process which is local to the verifier.
- Assisted Distance Fraud [13]: A terrorist prover remotely authenticates to a verifier, assuming the cooperation of another dishonest prover that is co-located with the verifier.

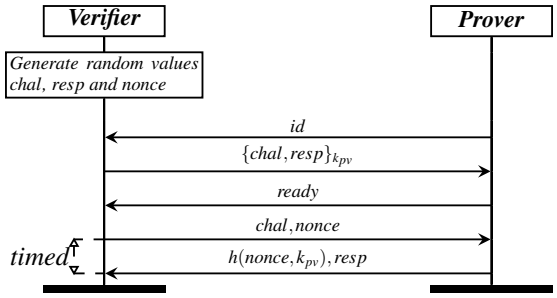
We can stop our example protocol being vulnerable to distance fraud attacks by adding a new nonce that is sent with the challenge, and also needs to be included with the

response. However, such a protocol would still be vulnerable to terrorist fraud attacks, because a remote dishonest terrorist fraud prover could decrypt the challenge and response and send them to an accomplice attacker that is local to the verifier, which can then use them to answer the verifier's challenge within the time limit.

This terrorist fraud attack can be stopped by, for instance, requiring the prover to hash the response with their secret key. Thereby providing evidence to the verifier that some local party did indeed know the secret key. However, if the same key is used by multiple provers then the protocol is vulnerable to distance hijacking and assisted distance fraud, because the dishonest prover could send the challenge and response to some honest prover that is co-located with the verifier. This honest prover would answer the verifiers challenge, which the verifier believed was the dishonest prover.

To protect against these attacks, we could require every prover to use a unique key with the verifier, thereby making it impossible for the dishonest prover to encrypt a message for some other honest prover.

Example 2. *Making the additions described above to the protocol from Example 1 we get a protocol that is secure against all the attacks listed above:*



This protocol uses a lightweight hash function, which needs to be computed before the timer expires.

3 Timer location calculus: A language for modelling distance bounding protocols

Our *timer location calculus* extends the applied pi-calculus [1, 2, 7, 33] with timers and locations. We first present the calculus syntax, illustrating this using the protocol from Example 1. We then present the semantics and explain how this captures the behaviour of timed communications.

Syntax: Each protocol role is written as a process, using the syntax of our language (Figure 1). Communication between roles is modelled by the *input* and *output* commands. The semantics, presented below, will substitute the term sent by an output command for the variable named in an enabled input. We assume that the attacker

Figure 1 The timer location calculus syntax

$M,N ::=$	terms
x,y,z	variables
a,b,c,k	names
$f(M_1,\ldots,M_n)$	constructor application
$D ::= g(M_1,\ldots,M_n)$	destructor application
$P,Q ::=$	processes
0	nil
$out(N).P$	output
$in(x).P$	input
$P \mid Q$	parallel composition
$!P$	replication
$new\ a.P$	restriction
let $x = D$ in P else Q	term evaluation
event$(M_1,\ldots,M_n)$	an event
$startTimer.P$	timer activation
$stopTimer.P$	timer termination
$S ::=$	systems
$[\{P_1,\ldots,P_n\}]_r$	a location
$new\ \tilde{a}.S$	restriction
$[\{P_1,\ldots,P_n\}]_r \mid S$	locations

controls the network, so processes are not able to ensure that a particular output goes to a particular input.

Parallel composition ($P \mid Q$) represents two processes running concurrently, and process *replication* ($!P$) represents an arbitrary number of copies of a process running in parallel. The *new* command creates a new value that then represents, for instance, a nonce, a key or a process identity. This value will not be known to the attacker unless it is output on a public channel.

Example 3. *The following process models an arbitrary number of provers with different ids each running an arbitrary number of times*

$$ExProvers(id) \;=\; !new\ id.!PRole(id)$$

We define $PRole(id)$ in the next example to model a single run of the protocol with identity "id", so $!PRole(id)$ represents an arbitrary number of runs of the protocol with a particular id. The "$!new\ id$" term at the front of the process generates an arbitrary number of new process ids.

Cryptography is modelled using constructors and destructors, e.g., symmetric key encryption can be modelled using a binary constructor $enc(m,k)$ to represent the message m encrypted with the key k and a binary destructor function dec with the rewrite rule $dec(enc(m,k),k) = m$. Functions can be public, i.e., available for use by the attacker, or private meaning that they cay only be used

by processes specified as party of the protocol. Private functions are useful, for instance, to look up private keys which should only be known to protocol participants.

Functions are applied using the let statement, e.g., "let $pt = dec(ct,k)$ in P else Q" tries to decrypt cipher text ct with key k, and acts as P if decryption succeeds and Q otherwise. Term evaluation in the let statement can also be used to define projections on tuples, and equality checks on names. As syntactic sugar we write "$in(=a).P$", for a process that receives an input and then acts as the process P if that input value is equal to a. We refer the reader to [2] for more details on functions in the applied pi-calculus.

Example 4. *A single run of the prover role of the protocol informally described in Example 1, with identity id, can be modelled as the process:*

$$\begin{aligned} PRole(id) \;=\; & out(id)\,.\,in(x)\,. \\ & \text{let } (chal,resp) = dec(x,k) \text{ in} \\ & out(ready)\,.\,in(=chal)\,.\,out(resp) \end{aligned}$$

Events are used to annotate the protocol for automated checking. For instance, below we will add an event to the protocol to signal that the verifier believes it has correctly verified a particular prover. The syntax presented so far is from the applied pi-calculus. Next, we present our additions, namely, locations and timers.

The process *startTimer.P* represents starting a timed challenge and *stopTimer.P* represents ending a challenge. We require that every start timer action is matched by exactly one stop timer action along all possible paths, and replication and parallel composition are forbidden between start and stop timer actions.

Example 5. *The verifier role of the protocol informally described in Example 1 can be modelled as the process:*

$$\begin{aligned} ExVerifiers \;=\; & !in(id)\,.\,new\ chal\,.\,new\ resp. \\ & out(enc((chal,resp),k))\,.\,in(ready). \\ & startTimer\,.\,out(chal)\,.\,in(=resp)\,. \\ & stopTimer\,.\,\text{event}(\text{verify}(id)) \end{aligned}$$

Locations are written $[\mathscr{P}]_r$, where $\mathscr{P}$ are (co-located) processes and r denotes the number of active timers. We abbreviate $[\{P_1,\ldots,P_n\}]_r$ as $[P_1,\ldots,P_n]_r$ and $[\{P_1,\ldots,P_n\}]_0$ as $[P_1,\ldots,P_n]$. Our model assumes that processes are either co-located or at distinct locations, and we abstract away from precise distances between provers and verifiers when modelling. We assume that there is a known maximum round trip time for communication between "local" processes, i.e., co-located processes, and the timer enforces this. Hence, it will not be possible for a message to travel to processes at different locations, and back again before the timer expires.

Example 6. *The system*

$$new\ k.[\ ExProvers \mid ExVerifiers\ ]$$

represents our example provers and verifiers running at the same location, i.e., it is possible for the prover to answer the challenge within the time limit and be verified. The declaration of the key k as new means that this is a new unique value, known only in the ExProvers and ExVerifiers processes.

By comparison, the system

$$new\ k.([\ ExProvers\ ] \mid [\ ExVerifiers\ ])$$

represents the verifiers and provers at different locations. Hence, in the latter system, it should not be possible for the prover to answer the timed challenge within the time limit, therefore a correct distance bounding protocol will not allow the prover to be verified.

Semantics: Dynamic behaviour of processes (which model protocols) can be examined using the semantics of our language (Figure 2), which is defined over *system configurations*, denoted $E,\mathscr{L}$, where that E is a set of free names and $\mathscr{L}$ is a finite multiset of systems.

The set E keeps track of the names that have been assigned so far, making it possible for the new command to pick fresh previously unused names, this is done by the (NEW) rule. The (REPL) rule creates a copy of a replicated process, the (LET 1) rule can be used to apply functions, e.g., for decryption, and the (LET 2) rule selects the else branch when no function reductions are possible (this, for instance, allows us to define equality tests). These rules are a direct extension of existing applied pi-calculus rules (e.g., [1, 33]) with our syntax for locations.

The rules we have created for our modelling language define the behaviour for timers and for communication between locations. The (START) rule increments the number of timers running at a location, and the (STOP) rule reduces the number of running timers. The restriction placed upon processes ensures that the number of running timers never becomes negative. Rule (I/O LOCAL) defines local communication, which allows messages to be exchanged between co-located processes, regardless of whether timers are running.

Example 7 (Local communication)**.** *As an example we consider a verifier that sends a challenge, denoted* `a`*, to a prover, to which the prover replies with a function* `f` *applied to this and some other value* `b`*:*

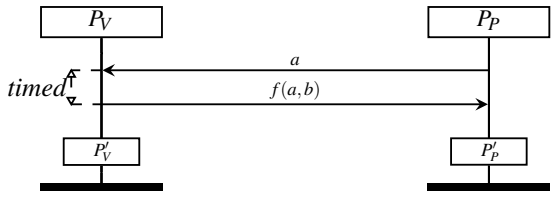


Figure 2 Operational semantics for our timer locations calculus

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{!P\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{!P, P\}]_r\ \} \quad \text{(REPL)}$$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P \mid Q\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P, Q\}]_r\ \} \quad \text{(PAR)}$$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{new\ a.P\}]_r\ \} \to E \cup \{a'\}, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P\{a'/a\}\}]_r\ \} \quad \text{(NEW)}$$
for some name $a' \notin E$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{\text{let } x = D \text{ in } P \text{ else } Q\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P\{M/x\}\}]_r\ \} \quad \text{(LET 1)}$$
if there exists M such that $D \to M$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{\text{let } x = D \text{ in } P \text{ else } Q\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{Q\}]_r\ \} \quad \text{(LET 2)}$$
if there is no M such that $D \to M'$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{out(M).P, in(x).Q\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P, Q\{M/x\}\}]_r\ \} \quad \text{(I/O LOCAL)}$$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{out(M).P\}]_r, [\mathscr{Q}]_0\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P\}]_r, [\mathscr{Q} \cup \{out(M)\}]_0\ \} \quad \text{(GLOBAL)}$$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{startTimer.P\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P\}]_{r+1}\ \} \quad \text{(START)}$$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{stopTimer.P\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P\}]_{r-1}\ \} \quad \text{(STOP)}$$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{out(M).P\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P \mid out(M)\}]_r\ \} \quad \text{(ASYNC)}$$

$$E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{\text{event}(M).P\}]_r\ \} \to E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{P\}]_r\ \} \quad \text{(EVENT)}$$

We can write these roles as processes:

$$P_V = startTimer.out(a).in(x).stopTimer.P'_V$$
$$P_P = in(x).out(f(x,b)).P'_P$$

such that processes P'_V and P'_P do not contain variable x (hence, we need not consider substitutes for x in these processes). Moreover, consider system configuration $\mathscr{C}_1 = E, \{[P_V, P_P]_0\}$ that co-locates those processes. Hence, we can observe traces in which the timed challenge succeeds. Indeed, $\mathscr{C}_1$ reduces by rule (START) *two applications of rule* (I/O LOCAL) *rule, and rule* (STOP)*:*

$$\begin{aligned}\mathscr{C}_1 &\to E, \{\,[out(a).in(x).stopTimer.P'_V, P_P]_1\,\}\\ &\to E, \{\,[in(x).stopTimer.P'_V, out(f(a,b)).P'_P]_1\,\}\\ &\to E, \{\,[stopTimer.P'_V, P'_P]_1\,\}\\ &\to E, \{\,[P'_V, P'_P]_0\,\}\end{aligned}$$

By comparison, the processes cannot complete the challenge from distinct locations. Indeed, although

$$E, \{[P_V]_0, [P_P]_0\} \to^*$$
$$E, \{\,[in(x).stopTimer.P'_V,]_1, [out(f(a,b)).P'_P\,]_0\,\},$$

the semantics do not allow any further reduction.

Rule (GLOBAL) allows an output to arrive at a new location, if no timers are active at that location. In implemented systems, it is only possible to receive outputs at particular times, yet rule (GLOBAL) allows outputs to be received at any time (in particular, after other processes have reduced). In this sense, the rule might be considered an over-approximation. However, for any communication allowed by our semantics, there exists a corresponding system execution (that takes communication and processing times into account). Thus, the rule accurately captures system behaviour, in particular, all possible interactions with an attacker are considered.

Example 8 (Preemption)**.** *A remote process may communicate with a timed process by preempting the messages needed. For instance, consider configuration $\mathscr{C}_3 = E, \{[P_V]_0, [in(x).P'_P, out(f(p,b))]_0\}$ and reduction*

$$\begin{aligned}\mathscr{C}_3 \to\ & E, \{[P_V, out(f(p,b))]_0, [in(x).P'_P]_0\}\\ \to\ & E, \{[out(a).in(x).stopTimer.P'_V, out(f(p,b))]_1,\\ & \qquad [in(x).P'_P]_0\}\\ \to^*\ & E, \{[stopTimer.P'_V]_1\ [P'_P]_0\}\\ \to\ & E, \{[P'_V]_0\ [P'_P]_0\}\end{aligned}$$

Note that the message received by P_V uses the name p rather than the challenge name a, hence, when using preemption there is no way in which the answer to the response to a timed challenge can be based on the message outputted as part of that challenge.

Rule (ASYNC) defines asynchronous communication, which prevents processes from blocking when they are ready to output. We could also avoid blocking by replacing instances of $out(M).P$ with $out(M).0 \mid P$, but introducing parallel composition reduces readability. Moreover, for purposes of compilation (Section 6), it is useful to consider only linear processes.

4 Modelling DB protocols and attacks

We define distance bounding protocols as follows:

Definition 1 (Distance bound protocol specification)**.** *A distance bounding protocol specification is a tuple* $(P(id), V, \tilde{n})$*, where*

- $P(id) = !new\ id.!Q$ *for some process* Q*;*
- $V = !V'$ *for some process* V' *that contains an event* $\text{event}(\text{verify}(id))$.
- $\tilde{n}$ *is a list of names known only to* Q *and* V.

We require that no further events are used in either process and the only free names (i.e., names not declared as new or bound by an input) used are those in $\tilde{n}$ *and the public channel* c*.*

Process Q models a single run of a prover with the identity id and $P(id)$ represents arbitrarily many distinct provers, each of which can run arbitrarily many times. Similarly, process V' models a single run of a verifier and V models arbitrarily many runs. Event "$\text{event}(\text{verify}(id))$" signifies a successful execution of the verifier with a prover that uses identity id. Anonymous protocols can use a dummy id value. It is important to note that the "verify" event does not mean that we have verified that the protocol is secure, rather it means that the verifier believes it has completed a run of the protocol. This could be because there is a prover at the same location as the verifier, or it could be because an attacker has performed a successful attack and tricked the verifier.

The names $\tilde{n}$ are secrets known to the verifier and all provers; many well designed protocols will have no such secrets, in which case $\tilde{n}$ will be the empty list, nonetheless many commercial devices continue to use global shared secrets (see e.g. [22] for one of many examples).

Example 9. *The protocol informally described in Example 1 can be modelled as specification* $(ProverE(id), VerifierE, \langle k \rangle)$*, where* $ProverE(id)$ *and* $VerifierE$ *are as described above, and* k *is the global shared key.*

Since attackers can be present at a number of different locations, we introduce *system contexts* as systems with "holes," in which a process may be placed. These holes denote the locations in a system where the attacker can act, and we write them as A. E.g., the system context $C_1 = new\ k.[VerifierE \mid A\ ] \mid [\ ProverE(id) \mid A\ ]$ represents a scenario in which the attacker can be co-located with the verifier V, and co-located with the prover Q, whereas $C_2 = new\ k.[\ VerifierE\ ] \mid [\ ProverE(id) \mid A\ ]$ represents a scenario in which the attacker is co-located with the prover and is remote from the verifier. When the context is applied to a process the A symbol is replaced with that process, to give a system. E.g., $C_2[P_A] = new\ k.[\ VerifierE\ ] \mid [\ ProverE(id) \mid P_A\ ]$.

Using our calculus, and system contexts, we can formulate the five types of attacks against distance bounding protocols described in Section 2, in which verifiers are deceived into believing they are co-located with provers. We formulate attacks as reachability requirements over traces that represent executions of distance bounding protocols. In particular, our formulations require an execution of a verifier, with a remote prover, which ends in a verify event for a particular id.

The following definition tells us if an attacker process can be found that leads to a context performing a verify event.

Definition 2. *Given a name id and a system context* C*, we write* $\text{verified}(id)$*:*C *if there exists a process* P_A *and a trace:*

$$\begin{aligned}\{c\}, C[P_A] &\rightarrow^* E, \mathscr{L} \cup \{[\mathscr{P} \cup \{new\ id.P\}]_r\} \\ &\rightarrow\ E \cup \{id'\}, \mathscr{L} \cup \{\left[\mathscr{P} \cup \{P\{^{id'}/_{id}\}\}\right]_r\} \\ &\rightarrow^* E', \mathscr{L}' \cup \{[\mathscr{P}' \cup \{\text{event}(\text{verify}(id')).P'\}]_{r'}\}\end{aligned}$$

where the only free name in P_A *is the public channel name* c *and* P_A *does not contain timers nor events.*

It follows from our definition that $\text{verified}(id)$:C denotes a successful execution of a verifier, therefore we would expect it to hold for any context that places a verifier and prover, with the identity id, at the same location. By comparison, we would not expect $\text{verified}(id)$:C_1, for the aforementioned context C_1, which places the verifier and prover at different locations, unless the protocol being modelled is insecure.

Using this we can now formally define the different types of distance bounding attacks.

Definition 3. *Distance bound protocol specification* $(P(id), V, \tilde{n})$ *is vulnerable to* relay (or mafia) fraud*, if* $\text{verified}(id)$:$new\ \tilde{n}.[\ V \mid A\ ] \mid [P(id) \mid A]$.

It follows from the definition that a relay attack is possible if the prover and verifier are at different locations, and an attacker process is co-located with each of the prover and verifier. Such an attack typically involves the attacker process co-located with the verifier answering the timed challenges, using messages passed from the other location. To keep our definitions simple we require the same attacker process at both locations, though different parts of this process can act at each location. E.g.,

an attacker process $P_{PA} \mid P_{VA}$ might define process P_{PV} to interact with the verifier and P_{AV} to interact with the prover.

Example 10. *There is no process P_A such that* verified(id):$[\ VerifierE \mid P_A\ ] \mid [ProverE(id) \mid P_A]$*, i.e., no attacker can trick the verifier into believing that it has verified id when the provers are at a different location. We informally reasoned why this protocol is safe from relay attacks in Section 2 and we will verify this result automatically in Section 7.*

Relay/mafia fraud considers an attacker that does not have the secret values of a normal prover. A more powerful "dishonest prover" attacker has access to such secrets.

Definition 4. *Distance bound protocol specification $(P(id), V, \tilde{n})$ is vulnerable to:*

- distance fraud, *if* verified(id) : *new* $\tilde{n}$.$[\ V\ ] \mid [\ DP\text{-}A(id)\ ]$
- distance hijacking, *if* verified(id) : *new* $\tilde{n}$.$[\ V \mid P(id')\ ] \mid [\ DP\text{-}A(id)\ ]$

where $P(id) =\ !new\ id.!Q$ and $DP\text{-}A(id)$ denotes $!new\ id.out(id).Q' \mid A$, where Q' outputs bound and free names of Q (including names in $\tilde{n}$, which are otherwise hidden from the attacker) and the results of any private function applications in Q, and A is the context hole.

The process $DP\text{-}A(id)$ reveals all the secret values of a normal prover to the attacker, which captures a dishonest prover attacker.

Example 11. *Specification $(ProverE(id), VerifierE, \langle k \rangle)$ is vulnerable to distance fraud. The prover process does not declare new names, and there are no private functions used therefore:*

$$DP\text{-}A(id) =\ !new\ id.out(id).out(k) \mid A$$

We define P_A as the process that receives the key k from process DP-A, uses the key to decrypt the challenge and response, and sends the response, without waiting for the challenge:

$$P_A = in(k).in(x).\text{let}\ (chal, resp) = dec(x,k)\ \text{in}\ out(resp)$$

Since the response is sent before the timer starts, it has time to make it to the verifier before the timer is active. Hence, $[\ VerifierE\ ] \mid [\ !new\ id.out(id).out(k) \mid P_A\ ]$ *can reduce such that the verifier can perform the verified event, which means that* verified(id) : $[\ VerifierE\ ] \mid [\ DP\text{-}A(id)\ ]$ *holds and the attack is possible.*

The attack works because the attacker can preempt the challenge. This can be prevented if the challenge must be observed before a response can be provided, which can be achieved by including a nonce in the challenge and requiring that nonce to be included in a response. Hence, we considered the revised specification $(ProverE2(id), VerifierE2, \langle k \rangle)$, where

$$\begin{aligned} VerifierE2 \ =\ & !in(id).new\ chal.new\ resp. \\ & out(enc((chal, resp), k)). \\ & new\ c2.startTimer. \\ & out(chal, c2).in(=resp, =c2). \\ & stopTimer.\text{event}(\text{verify}(id)) \end{aligned}$$

$$\begin{aligned} ProverE2(id) \ =\ & !new\ id.!out(id)in(x). \\ & \text{let}\ (chal, resp) = dec(x,k)\ \text{in} \\ & in(=chal, x).out(resp, x) \end{aligned}$$

It can be shown that this fix suffices to defend against distance fraud attacks. Intuitively, the nonce $c2$ is only sent when the timer is running, so the attacker can never return this in time if not co-located with the verifier.

Terrorist provers are less powerful than dishonest provers, because they will not send their secret values to a third party. Nevertheless, by considering terrorist provers working with another attacker that is co-located with the verifier, we can identify further attacks.

Definition 5. *Distance bound protocol specification $(P(id), V, \tilde{n})$ is vulnerable to:*

- terrorist fraud, *if* verified(id) : *new* $\tilde{n}$.$[\ V \mid A\ ] \mid [\ TP\text{-}A(id)\ ]$
- assisted distance fraud, *if* verified(id) : *new* $\tilde{n}$.$[\ V \mid DP\text{-}A(id')\ ] \mid [\ TP\text{-}A(id)\ ]$

where $P(id) =\ !new\ id.!Q$, $DP\text{-}A(id')$ is as specified in Definition 4, and $TP\text{-}A(id)$ denotes $!new\ id.out(id).!Q' \mid A$, where Q' is the process that acts as an oracle with all relevant functions for all bound and free names and private function applications in Q, and A is the context hole.

The process $TP\text{-}A$ will perform operations on behalf of the attacker, e.g., signing, encrypting and decrypting any values the attacker wishes, but it will not reveal secret values.

Example 12. *Specification (*$ProverE2(id)$, $VerifierE2$, $\langle k \rangle$*) is vulnerable to terrorist fraud attacks. We have*

$$\begin{aligned} TP\text{-}A(id) \ =\ & !(new\ id.out(id) \\ & \mid in(x).\text{let}\ y = dec(x,k)\ \text{in}\ out(y) \\ & \mid in(x).out(enc(x,k))) \mid A \end{aligned}$$

This process can receive the encrypted challenge from the verifier, decrypt it, and send the resulting plaintext to an attacker process co-located with the verifier, all before the timer is started. At the verifier's location we consider the following attacker process $P_A = in(chal, resp).in(=$

chal,*x*).*out*(*resp*,*x*), *this process can receive the challenge information from the terrorist prover process, and then use it to complete the verifier's challenge. This suffices to show* verified(*id*):*new k*.[*V* | *A*] | [*TP-A*(*id*)], *hence, the specification is vulnerable to terrorist fraud.*

Example 13. *The second, more secure, protocol in Example 2 can be modelled in our calculus as* $(V2, P2, \langle\rangle)$ *where:*

$$\begin{aligned} P2(id) \;=\; & !new\ id\ .!out(id)\ .\ in(x)\ . \\ & \text{let } (chal, resp) = dec(x, lookup(id)) \text{ in} \\ & out(ready)\ .\ in(=chal, nonce)\ . \\ & out(xor(nonce, lookup(id)), resp) \end{aligned}$$

$$\begin{aligned} V2 \;=\; & !in(id)\ .\ new\ chal\ .\ new\ resp\ . \\ & out(enc((chal, resp), lookup(id)))\ . \\ & in(ready)\ .\ new\ nonce\ . \\ & startTimer\ .\ out(chal, nonce)\ .\ in(xb, =resp)\ . \\ & stopTimer\ .\ \text{let } xb = h(nonce, lookup(id)) \\ & \qquad \text{in event}(\text{verify}(id)) \text{ else } 0 \end{aligned}$$

and lookup is a private function used to find a unique key shared between one particular prover and the verifier, and h is a public hash function.

We show in Section 7 that there does not exist any attacker process that can make any of the system contexts that model the attacker perform a verify event for the id being tested. Therefore this protocol is secure against all of these possible, different distance bounding attacks.

We only consider two locations when capturing different types of attacks against distance bounding protocols. More attack scenarios would be possible by considering attackers at other locations, however, these scenarios are strictly weaker than those presented, so they would not lead to interesting definitions.

5 A hierarchy of attacks

We have modelled five types of attack against distance bounding protocols by considering various scenarios in which verifiers are deceived into believing they are co-located with provers. These scenarios consider the following terms:

- $V \mid A$, a verifier co-located with a basic attacker (relay fraud and terrorist fraud);
- V, a verifier in isolation (distance fraud);
- $V \mid P(id')$, a verifier co-located with honest provers (distance hijacking);
- $V \mid DP\text{-}A(id')$, a verifier co-located with dishonest provers (assisted distance fraud);
- $P(id) \mid A$, remote provers co-located with an attacker (relay fraud);
- $DP\text{-}A(id)$, remote dishonest provers in isolation (distance fraud and distance hijacking); and
- $TP\text{-}A(id)$, remote terrorist provers in isolation (terrorist fraud and assisted distance fraud).

Yet, numerous combinations of these terms were not considered by the definitions in the previous section, e.g., we have not considered a verifier co-located with a basic attacker and some other prover, along with a remote prover and a basic attacker: $[V \mid A \mid P(id')] \mid [P(id) \mid P(id)]$. We also have not considered co-location of remote dishonest provers, e.g., $DP\text{-}A(id') \mid TP\text{-}A(id)$.

We now consider a more general setting whereby a verifier is co-located with zero or more of a basic attacker A, honest provers $P(id')$, terrorist provers $TP\text{-}A(id')$, and dishonest provers $DP\text{-}A(id')$. These provers all use identifiers that are distinct from the identifier id, which is being used in an attempt to deceive the verifier. Moreover, at a distinct, remote location, we consider one or more of honest provers $P(id)$, terrorist provers $TP\text{-}A(id)$, and dishonest provers $DP\text{-}A(id)$. Furthermore, the remote location may additionally include one or more of a basic attacker A, honest provers $P(id')$, terrorist provers $TP\text{-}A(id')$, and dishonest provers $DP\text{-}A(id')$. This gives way to $2^4 \cdot 2^3 \cdot 2^4 = 2048$ scenarios. Albeit, we can disregard scenarios in which identifier id is absent (since without this any attack will be an attack on authentication, rather than a distance bounding attack, and authentication attacks can be found using a range of other well established methods, e.g. [1]). This gives us $2^4 \cdot (2^3 - 1) \cdot 2^4 = 1792$ scenarios to consider, significantly more than the five scenarios that have been identified in the literature.

We can reduce the number of scenarios we need to consider by observing that there is a strict order on the capabilities of the different attacker processes:

Lemma 1. *For any distance bounding protocol specification* $(P(id), V, \tilde{n})$, *from which we derive DP-A and TP-A, and for all system contexts C, sets of names E and names* $x \in \{id, id'\}$, *we have*

$$\begin{aligned} & \text{verified}(id){:}C[A \mid P(x)] \\ & \quad \Rightarrow \text{verified}(id){:}C[TP\text{-}A(x)] \\ & \quad \Rightarrow \text{verified}(id){:}C[DP\text{-}A(x)] \end{aligned}$$

Moreover, no reverse implication holds.

By filling a context's hole with a process containing a hole (as above), we derive a context (which is required by the verified predicate).

It follows from Lemma 1 that we need not consider more than one of the terms $P(x)$, $DP\text{-}A(x)$, or

$TP\text{-}A(x)$ at a particular location. For instance, the verifier can perform the verify event in the context $[V] \mid [P(id) \mid A \mid DP\text{-}A(id)]$ if and only if it can perform the event in the context $[V] \mid [DP\text{-}A(id)]$. Hence, we need not consider both these contexts; we need only consider the latter, simpler context.

Honest and dishonest provers represent an arbitrary number of provers. (The bound name used as the id of these provers will be substituted for another value by the (NEW) rule.) Hence, we have:

Lemma 2. *For any distance bounding protocol specification $(P(id), V, \tilde{n})$, from which we derive DP-A and TP-A, and for any system contexts $C[\_]$, sets of names E, names id and id', and $X \in \{P, DP\text{-}A, TP\text{-}A\}$, we have:*

$$\text{verified}(id){:}C[X(id') \mid X(id)] \Leftrightarrow \text{verified}(id){:}C[X(id)]$$

It follows from Lemma 2 that if process $X(id)$ is present, then it is not necessary to consider the corresponding $X(id')$ process as well.

When there is a dishonest prover at a different location to a basic attacker process, the dishonest prover could send all of its secrets to the basic attacker process enabling it to also act as a dishonest prover:

Lemma 3. *For any distance bounding protocol specification $(P(id), V, \tilde{n})$, from which we derive DP-A, and for all processes P and Q, names id, tuple of names $\tilde{n}$, and sets of names E, and all names x (including $x = id$), we have that*

$$\begin{aligned} &\text{verified}(id){:}\mathit{new}\ \tilde{n}.[P \mid A] \mid [DP\text{-}A(x) \mid Q] \\ \Leftrightarrow\ &\text{verified}(id){:}\mathit{new}\ \tilde{n}.[P \mid DP\text{-}A(x)] \mid [DP\text{-}A(x) \mid Q] \\ \Leftrightarrow\ &\text{verified}(id){:}\mathit{new}\ \tilde{n}.[P \mid DP\text{-}A(x)] \mid [A \mid Q] \end{aligned}$$

Our observations reduce the number of interesting, distinct, system contexts to 27, each of which models a different distance bounding attack scenario, and protection against which offers a distinct security property. These 27 contexts are given in the figure in the Appendix.

Lemma 1 lets us order contexts in terms of the strength of the security properties they represent. For instance, if we replace $TP\text{-}A(id)$ with $DP\text{-}A(id)$, then the attacker is strictly more powerful, and the security properties they represent are stronger. Additionally, we note that adding processes to a context will not affect the verified, predicate, e.g., $\text{verified}(id){:}C[A] \Rightarrow \text{verified}(id){:}C[A \mid P(x)]$. The partial order this leads to is shown in the figure in the Appendix.

For any protocol, if it is secure against an attack scenario in this ordering then it is also secure against the attack scenarios directly below it. Additionally, we can find examples to show that all the attack scenarios are different, and that attack scenario that are not directly above or below each other are unrelated.

This partial ordering of attack scenarios the Appendix tells us that protection against distance hijacking attacks is strictly stronger than security against distance fraud attacks, and that security against assisted distance fraud is stronger than security against terrorist fraud attacks, which in turn is a stronger property than security against relay attacks. However, distance hijacking and assisted distance fraud are not directly comparable properties. To illustrate this we could consider a verifier with an override mode: if a process sent it the secret key of a prover then it would accept it as local. Such a protocol could be secure against assisted distance fraud but would not be secure against distance hijacking.

On the other hand we could consider a verifier that would correctly distance bound a process and would then accept any identity from that local process. Such a protocol could be secure against distance hijacking but not against assisted distance fraud. Therefore, the strongest property that a distance bounding protocol can have is protection from both distance hijacking and assisted distance fraud.

To separate many of the distance bounding properties we need to consider a verifier that will verify any process that sends it a secret key. This is the difference between what a dishonest prover and a terrorist prover can do, however there are currently no proposals for distance bounding protocols with this behaviour. Therefore, it is a safe assumption that for any proposed distance bounding protocol, if there is no local attacker process, then the ability to send a secret key does not add any additional power. This means that:

Assumption 1. *Distance bounding protocols will not be designed so that a correct prover could send their secret key to the verifier. I.e.,*

$$\begin{aligned} &\text{verified}(id){:}\mathit{new}\ \tilde{n}.[V] \mid [DP\text{-}A(id)] \\ &\qquad \Leftrightarrow \text{verified}(id){:}\mathit{new}\ \tilde{n}.[V] \mid [TP\text{-}A(id)] \end{aligned}$$

All examples of distance bounding protocols we have seen in the literature do not distance bound the verifier to the prover. This would mean that the attacker does not gain any additional power by being local to the prover, rather than local to the verifier. This further reduces the number of interesting cases we need to consider.

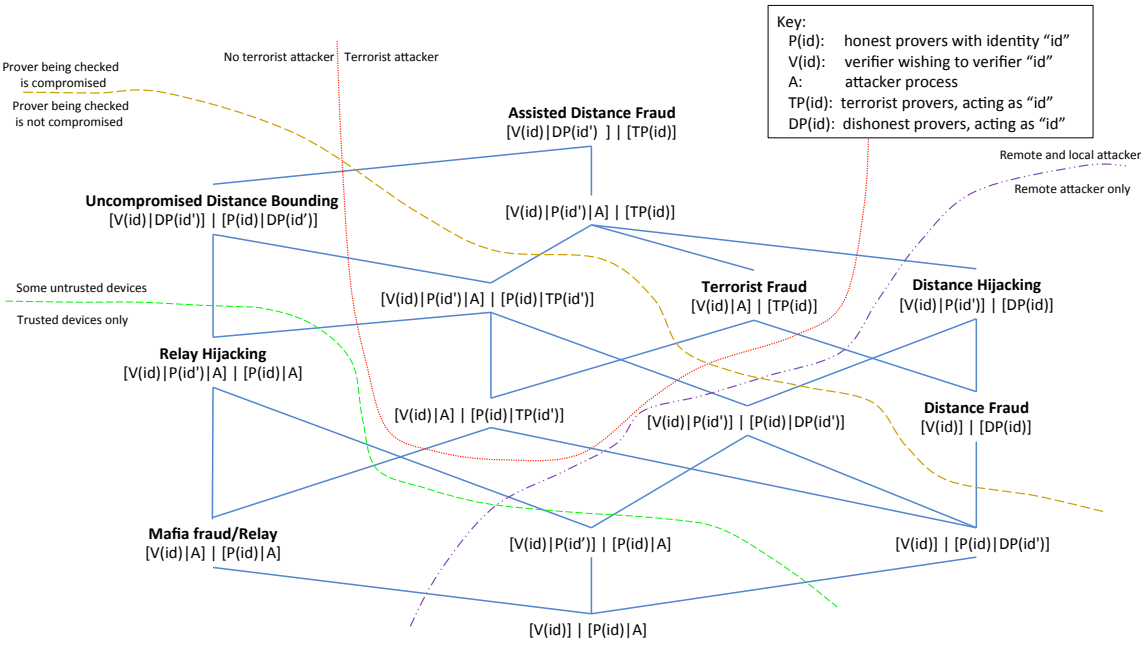
Assumption 2. *In the protocols we consider the prover does not also distance bound the verifier. I.e.,*

$$\begin{aligned} &\text{verified}(id){:}\mathit{new}\ \tilde{n}.[V \mid A] \mid [P(id)] \\ &\qquad \Leftrightarrow \text{verified}(id){:}\mathit{new}\ \tilde{n}.[V \mid A] \mid [P(id) \mid A] \end{aligned}$$

These assumption, along with the lemmas above, leave us with 14 distance bounding attack scenarios, which can be ordered using the lemmas above. This ordering is shown in Figure 3.

Discussion: With the assumption that transmitting the secret key does not matter, assisted distance fraud becomes the most powerful distance bounding property. If

Figure 3 Ordering of distance bounding attack scenarios that follows from lemmas 1, 2 and 3 and assumptions 1 and 2. Higher properties imply those below them. We write $[V(id) \mid P] \mid [Q]$ for verified(id):$[V \mid P] \mid [Q]$



a distance bounding protocol is secure against this attack scenario, then none of the other attacks are possible. However, this property is very strong; industrial distance bounding protocols such as MasterCard's RRP or NXP's proximity check do not have this property nor do they need it: If a bank card or key fob has been fully compromised, then an attacker may send all key information from this device to the same location as the verifier and so pass the verification.

The lines which dissect Figure 3 each represent different possible attacker models, and each area is dominated by a single property, which, if checked, will prove security for that particular attacker model. Assisted distance fraud, and all of the other attack scenarios that require a terrorist fraud attacker process (as indicated by the red dotted line in Figure 3), rely on the terrorist fraud attacker simply deciding not to send their key. While such an attacker could exist, there is nothing to stop an attacker, that has compromised a device, from sharing the secret key. Therefore, the additional protection provided by protecting against a terrorist attacker is questionable in some attacker models.

The brown, large dashed lines separates the properties in which the verifier is checking a compromised prover from an uncompromised prover. Many of the use cases for distance bounding protocols aim to protect a device against relay attack, thereby preventing criminals from taking a victim's car or making a payment with the victim's EMV card, for instance. In this attacker model, if the attackers have compromised the device, then they can simply clone it, making the distance bounding attack unnecessary. In this model, checking verified(id): $[V \mid DP\text{-}A(id') \mid A] \mid [P(id) \mid DP\text{-}A(id')]$ ensures that all of the possible relevant security properties hold. For this security property to hold, the attacker should not be able to pretend to be an uncompromised device, regardless of how many other devices are compromised. We define this property as uncompromised distance bounding:

Definition 6 (Uncompromised Distance Bounding attack)**.** *Given a name id' and a distance bounding protocol $(V, P(id), \tilde{n})$, from which we derive a dishonest prover DP-A(id'), we say that the protocol is vulnerable to an uncompromised distance bounding attack if:* verified(id):*new* $\tilde{n}.[V \mid DP\text{-}A(id')] \mid [P(id) \mid DP\text{-}A(id')]$ *otherwise we say that it is safe from this attack.*

As we are dealing with dishonest provers, by Lemma 3:

$$\begin{aligned}&\text{verified}(id)\text{:}new\ \tilde{n}.[V \mid DP\text{-}A(id')] \mid [P(id) \mid DP\text{-}A(id')]\\ &\Leftrightarrow \text{verified}(id)\text{:}new\ \tilde{n}.[V \mid A] \mid [P(id) \mid DP\text{-}A(id')]\\ &\Leftrightarrow \text{verified}(id)\text{:}new\ \tilde{n}.[V \mid DP\text{-}A(id')] \mid [P(id) \mid A]\end{aligned}$$

therefore any of these system contexts could be used to represent uncompromised distance bounding attacks.

We choose the one that makes it clear that the dishonest prover can act at both locations.

The purple dot-dashed line separates the attack scenarios that have only a remote attacker from those that let the attacker act both locally to the verifier and remotely. In the case where transmissions from the verifier can be picked up remotely and the attacker can only act remotely, the strongest possible property is distance hijacking. However, in many applications the messages from the verifier are limited to the local area (e.g. due to the RFID technology as used by contactless EMV cards), therefore the attacker must be able to act locally to the verifier and these attack scenarios do not apply.

The green small dashed line marks out the attack scenarios that assume trusted hardware from those that allow some provers to be compromised. Our ordering shows that verified(id):*new* $\tilde{n}$.$[V \mid P(id) \mid A] \mid [P(id) \mid A]$ is the most powerful property that can be tested in this category. This attacker corresponds to, for instance, a relay attack against an EMV card, which uses another, different EMV card at the verifier's location. The use of this other EMV card that is co-located with the verifier makes it a more powerful attacker than a basic relay attack, but it is still less powerful than an uncompromised distance bounding attack, because it does not require any cards to be compromised. We do not believe this particular attack scenario has been identified before, as a distinction from relay attacks, so we call this "relay hijacking".

In summary, our ordering tells us that:

- If the protocol is aiming to defend against terrorist fraud attackers, then it should be checked against *assisted distance fraud*.
- If the attacker model does not include terrorist fraud attackers, then the strongest protection a protocol can have is against both *distance hijacking* and *uncompromised distance bounding* attacks.
- If the attacker model does not require protection for a compromised prover, then the strongest attack that needs to be defended against are *uncompromised distance bounding* attacks.
- If a distance bounding protocol assumes trusted hardware devices, then the strongest attack that needs to be defended against is *relay hijacking*: verified(id):$[V \mid P(id') \mid A] \mid [P(id) \mid A]$.
- If the attacker model only considers attackers that are remote from the verifier, then the strongest attack that needs to be defended against is *distance hijacking*.

6 Automated reasoning

To enable automated reasoning, we define a compiler from our timer location calculus to a dialect of the applied pi calculus with phases [7], which can be automatically reasoned with using the ProVerif tool [8]. Phases are used to define an ordering on reduction, e.g., processes in phase 1 can only be executed before the processes in phase 2, which come before the processes in phase 3, etc. Beyond phases, the applied pi-calculus adds named communication channels, e.g., $out(c,m)$ outputs message m on the channel c. Channels can be public or private, and the attacker can only send and receive messages on public channels. The applied pi-calculus does not have timers or locations, and our compiler encodes the start timer, stop timer and locations using other primitives. Thus, compilation enables distance bounding protocols to be verified automatically using ProVerif.

We restrict compilation to extended linear processes that contain at most one timer:

Definition 7. *A* linear process *is a process without parallel composition or replication. Moreover, an* extended linear process *is a process new* $\tilde{n}.L_1 \mid \cdots \mid L_i \mid !L_{i+1} \mid \cdots \mid !L_n$*, where* $L_1, \ldots, L_n$ *are linear processes.*

Linear processes allow us to express all distance bounding protocols from the literature, so they do not reduce the usefulness of our method.

Using linear processes, we introduce a technique to simplify the detection of vulnerabilities and define a compiler that allows us to take advantage of that technique.

Proof technique: It follows from Definition 2 that: if verified(id):*new* $\tilde{n}$.$[!L_1 \mid L_2 \mid A] \mid [L_3 \mid A]$ such that only L_1 contains a timer, then there exists a successful execution of L_1. Moreover, the following lemma shows that it is sufficient to consider $L_1 \mid !\text{blind}(L_1)$ in place of $!L_1$, where $\text{blind}(L_1)$ is L_1 after removing timer actions ($startTimer$ and $stopTimer$) and events, hence, it suffices to isolate timers and events to a single instance of L_1.

Lemma 4. *For all system contexts new* $\tilde{n}$.$[!V_L \mid L_v \mid A] \mid [L_p \mid A]$*, sets of names E and name id, such that* V_L*,* L_v *and* L_p *are linear processes and only* V_L *contains a timer, we have:* verified(id):*new* $\tilde{n}$.$[!V_L \mid L_v \mid A] \mid [L_p \mid A] \Rightarrow$ verified(id):*new* $\tilde{n}$.$[V_L \mid !\text{blind}(V_L) \mid L_v \mid A] \mid [L_p \mid A]$.

It follows from Lemma 4 that distance bounding attacks can be detected by checking whether a single instance of the verifier is deceived. Moreover, we need only consider a single unreplicated timer.

Our compiler: Intuitively, the goal of our compiler is to encode a single timer using phases. In particular, all processes should initially be in phase 0, hence, all processes are initially active. Once the timer is activated, we advance all processes at the same location as the timer to phase 1, hence, only processes at the timer's location are active. Finally, once the timer is deactivated, we advance

all processes to phase 2, hence, all processes are active. Thus, compilation encodes timers as phases.

Encoding the activation and deactivation of timers as phases is straightforward, indeed, we merely replace *startTimer.P* with $1:P$ and *stopTimer.Q* with $2:Q$. But, encoding the advancement of other processes at the same location as the timer from phase 0 to phase 1 is problematic, as is advancing processes at different locations from phase 0 to phase 2, because we cannot know when processes should advance. We overcome this problem by over-approximating advancement.

We over-approximate by ensuring processes can advance between phases at any time. It suffices to consider advancements just before input operations, because processes ready to output can be reduced by an attacker that receives those outputs before an advancement and replays the messages received afterwards, and other processes do not produce communications, so it does not matter whether they happen before or after an advancement. We define the following function to produce all ways in which advancements can be inserted into a process before inputs.

Definition 8. *Given a timer location calculus process P, and a non-empty list of integers ds, we define the function phases, to applied pi-calculus processes, as follows*

$$phases(P,ds) = !P_1 \mid !P_2 \mid \cdots \mid !P_n$$

where $\{P_1,\ldots,P_n\} = phasesSet(P',ds)$, P' *equals* P *with every in(x) replaced with in(c,x) and every out(M) replaced with out(c,M) and function phasesSet is defined as follows:*

$$\begin{aligned}
&phasesSet(P,[d]) \\
&\quad = \{C[d:in(M,x).P'] \;:\; P = C[in(M,x).P']\} \cup \{P\} \\
&phasesSet(P,d_1::d_2::ds) \\
&\quad = \{C[d_1:in(M,x).P''] \;:\; P = C[in(M,x).P'] \wedge P'' \in \\
&\qquad phasesSet(P',d_2::ds)\} \cup phasesSet(P,d_2::ds)
\end{aligned}$$

Using function *phases*, we define our compiler, first for systems with verifiers co-located with attackers and then for systems with remote attackers.

Definition 9. *Given a system context* $S = new\ \tilde{n}.([!V_L \mid L_v \mid A] \mid [!new\ id.!P_L \mid L_p \mid A])$ *and name id, we define the* $compile(id,S)$ *as*

$$\begin{aligned}
&new\ \tilde{n}.(tToPh(V_L) \mid \\
&\quad phases(\mathrm{blind}(V_L),[1,2]) \mid phases(L_v,[1,2]) \mid \\
&\qquad !new\ id.phases(P_L,[2]) \mid phases(L_p,[2]))
\end{aligned}$$

where $tToPh(L)$ *is L after replacing startTimer.P with* $1:P$ *and stopTimer.Q with* $2:Q$ *and every in(x) replaced with in(c,x) and every out(M) replaced with out(c,M)*

Timers limit communication between locations. Hence, once timers have been encoded as phases, we no longer require locations. Thus, our compiler also removes locations. (Once locations are removed, we can consider a single hole, rather than multiple holes. Such a hole can be left implicit, because it will be introduced by Definition 11, below.) It follows that our compiler outputs processes in the applied pi calculus with phases, which can be automatically reasoned with using ProVerif.

When the verifier and attacker are not co-located, we must prevent the attacker communicating with the verifier's location whilst the timer is running. To do this, we replace the public channel "*c*" with a private channel "*priv*" between phase 1 and 2 (i.e., whilst the timer is active), thereby denying the attacker access to the communication channel. To maintain equivalence between compiled processes in the applied pi-calculus with phases and the original process in the timer location calculus, compilation introduces the following processes:

- $!in(c,x).1:out(priv,x)$, respectively $!1:in(priv,x).2:out(c,x)$, which allows messages sent on public channel c in phase 0 (before the timer starts), respectively private channel *priv* in phase 1 (whilst the timer is running), to be received on private channel *priv* in phase 1, respectively public channel c in phase 2 (after the timer stops).

The first process permits preemption, whereby a message is sent before a timer starts and received when the timer is running, and the second permits a message sent whilst a timer is running to be received after the timer stops.

- $!1:in(priv,x).out(priv,x)$, which allows messages sent on private channel *priv* to be buffered, i.e., received and relayed.

This final process ensures that any reduction by the (ASYNC) rule on private channel *priv* in our timer location calculus can be mapped to a reduction in the applied pi-calculus, which has no such rule (a similar processes isn't required for reductions by the (ASYNC) rule on public channel c, because the attacker process can simulate such reductions).

Definition 10. *Given a system context* $S = new\ \tilde{n}.[!V_L \mid L_v] \mid [!new\ id.!P_L \mid L_p \mid A]$ *and a name id, we define* $compile(id,S)$ *as*

$$\begin{aligned}
&new\ priv.new\ \tilde{n}.(renameC(priv,tToPh(V_L) \\
&\mid\ renameC(priv,phases(\mathrm{blind}(V_L),[1,2])) \\
&\mid\ renameC(priv,phases(L_v,[1,2])) \\
&\mid\ !new\ id.(phases(P_L,[2])) \mid phases(L_p,[2]) \\
&\mid\ !in(c,x).1:out(priv,x) \mid !1:in(priv,x)2:out(c,x) \\
&\mid\ !1:in(priv,x).out(priv,x)
\end{aligned}$$

where $tToPh(L)$ is defined above and $renameC(a,P)$ is process P with every occurrence of the channel c used for input and output between all $1:$ *and* $2:$ *actions replaced with the channel priv.*

The ProVerif tool [8] can test to see if there exists an attacker process that can make an event reachable.In this paper we only require events that are a function application to new names, which can be defined as follows. Although ProVerif can test such properties, the corresponding definition has not previously been formally defined, we do so here:

Definition 11. *We write* $\text{ev}(f(a_1,\ldots,a_i)), Init:P$ *if there exists a process Q such that the free names of Q are a subset of the names Init and Q does not contain any events, and a trace:*

$\mathscr{T} = Init, \{P|Q\} \rightarrow^* E, \{\text{event}(f(b_1,\ldots,b_i)).P'\} \cup \mathscr{P}$ *and for* $1 \leq j \leq i$ *the trace* $\mathscr{T}$ *contains the reductions:* $E_j, \mathscr{P}_j \cup \{new\ a_j.P_j\} \rightarrow E_j \cup \{b_j\}, \mathscr{P}_i \cup \{P_j\{b_j/a_j\}\}$

The following theorem tells us that we can check the compiled system in the applied pi-calculus and concluded security results about the system with locations:

Theorem 1. *Given a system context* $S = new\ \tilde{n}.[!V_L \mid L_v \mid A] \mid [!new\ id.!P_L \mid L_p \mid A]$ *or* $S = new\ \tilde{n}.[!V_L \mid L_v] \mid [!new\ id.!P_L \mid L_p \mid A]$*, and a name id, we have*

not $ev(\text{verify}(id)), \{c\}:compile(id,S) \Rightarrow \neg\ \text{verified}(\{c\}, id):S$

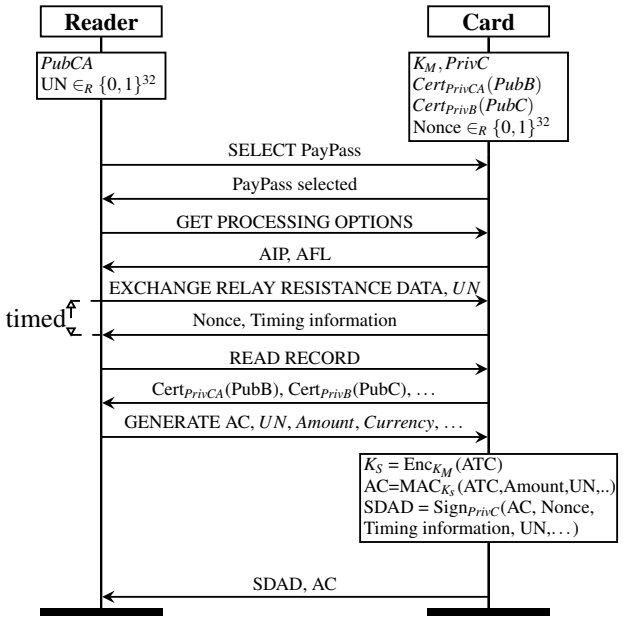
7 Case studies

We have implemented the compiler introduced in the previous section. Using this tool and ProVerif we analysed various distance bounding protocols. The tool and all of the model files mentioned in this section are available on the website given in the introduction.

Contactless payment protocols: Smart cards use the EMV protocol to perform contact-based and contactless payments via payment terminals [19, 20]. EMV Contactless cards make use of ISO/IEC 14443 for the communication between the card and terminal. ISO/IEC 14443 is a standard that specifies near-field communication at 13.56 MHz. This standard is widely used for bank cards and cards for access control (e.g. for buildings) and public transport. Due to its physical characteristics it is not possible to communicate over a long distance using ISO/IEC 14443. Even with a very powerful antenna active communication is only possible up to around a meter [23].

The EMV protocol comprises of an exchange of transaction data and then the card generates a MAC (called the Application Cryptogram or AC) using a session key based on a key shared between the smart card and the

Figure 4 MasterCard's Relay Resistance Protocol



card issuer and the Application Transaction Counter (ATC), which equals the number of times the card has been used and will provide freshness to the transaction. The AC is used for verification of the transaction by the card issuer. As the payment terminal cannot read the AC, the card also signs the transaction data, known as the Signed Dynamic Application Data (SDAD) and the payment terminal uses this to verify the transaction.

MasterCard's Relay Resistance Protocol (RRP) [20], as part of an EMV transaction, is presented in Figure 4. RRP is an extension of the EMV protocol, for which a new command is added, namely the *EXCHANGE RELAY RESISTANCE DATA* command. In a regular EMV session, a transaction is initiated by executing the *SELECT* command, to select the EMV applet on the smart card, and then the *GET PROCESSING OPTIONS* command to provide information about the capabilities of the terminal to the card.

The card will typically respond to the *GET PROCESSING OPTIONS* message with the Application Interchange Profile (AIP) and Application File Locator (AFL), used to indicate the capabilities of the card and the location of data files respectively. To finalise a transaction the *GENERATE AC* command is used. This command includes a nonce, known as the Unpredictable Number (UN), to provide freshness to the transaction, and an AC, and if the card supports it the SDAD, are them returned.

The new command added in RRP is the *EXCHANGE RELAY RESISTANCE DATA* command, which will be timed and is typically executed after the *GET PROCESS-*

ING OPTIONS command. The terminal will send a nonce (*Terminal Relay Resistance Entropy*), which will also be used as the Unpredictable Number for the rest of the transaction. The card will respond with another nonce (*Device Relay Resistance Entropy*) and three timing estimates (minimum time for processing, maximum time for processing and estimated transmission time). The maximum time serves as an upper bound for the terminal's timer. Both random numbers and the timing information are included in the SDAD. If the card does not respond in time, it is assumed that it is not actually present at the current location and the data may be relayed.

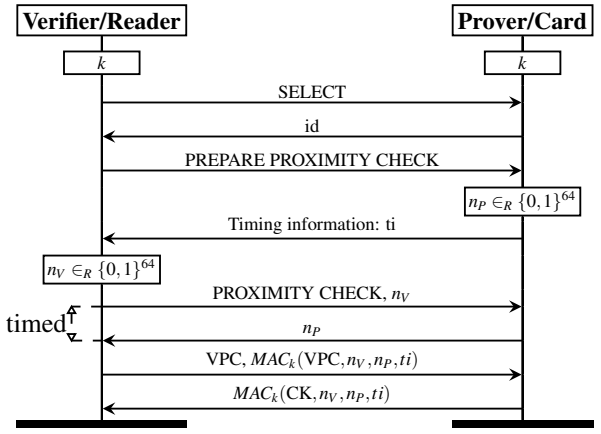
MasterCard's RPP is similar to PaySafe [12], though PaySafe makes fewer changes to the previous EMV specification. No new commands are introduced; rather than sending the nonce using the *EXCHANGE RELAY RESISTANCE DATA* as in RRP, it is included in the *GET PROCESSING OPTIONS* command and a nonce is added in the corresponding response. This exchange is timed to detect possible relay attacks.

Mauw et al. [28] looked at PaySafe and observed that it is vulnerable to distance fraud attacks and suggested adding the UN nonce to the timed response to protect against this. We note that the same weakness to distance fraud applies to MasterCard's protocol. Due to the physical characteristics of ISO/IEC 14443, we consider distance fraud attacks not to be applicable to protocols using this standard, as it will always be necessary to have a local adversary in order to be able to communicate with the local reader. Furthermore, once a card is compromised, it should not lead to a compromise of other cards but the compromised card should be considered lost as the information on it can be used to clone the card, as discussed in Section 5. This means that we do not consider attacks such as terrorist fraud or distance hijacking applicable to these protocols.

NXP's distance bounding protocols: NXP's Mifare Plus cards are used in, for example, public transport and for building access control and also make use of the ISO/IEC 14443 specification for contactless communication. The cards use a proprietary distance bounding protocol. It is not publicly known what protocol is used. Nevertheless, NXP have been granted a patent [25] and have filed a further patent application [14] for distance bounding technology.

We present the protocol from the granted patent [25] in Figure 5. As with any protocol on top of ISO/IEC 14443, the session starts with the reader sending a *SELECT* command to the card and the card responding with its ID. The distance bounding check will be initialised by sending a *PREPARE PROXIMITY CHECK* command. The card generates a random 8-byte number n_P and sends timing information to the reader indicating how long a

Figure 5 NXP's patented distance bounding protocol. The timed step can be repeated up to 8 times



reply to the distance bounding check should take. After receiving the timing information the reader generates its own random 8-byte number (n_V), sends this to the card using a *PROXIMITY CHECK* command and starts its timer.

In reply to the *PROXIMITY CHECK* command the card sends its own random number and on receiving this the reader stops its timer and checks the time against the timing information previously sent by the card. These steps can send the whole 8-byte nonces in one message, or the nonces can be split into up to eight exchanges of 1 byte each, so giving multiple time measurements.

Finally, the reader sends a *VERIFY PROXIMITY CHECK* with a MAC of the nonces and the timing information. The card checks whether the nonces and timing information are correct, and if so the card replies with a MAC of its own, again including the nonces and timing information. The card and readers MAC are distinguished by the inclusion of a different constant in each. The reader checks the card's MAC, and if it is correct it verifies the card as being at the same location.

NXP's other patent application [14] presents the same protocol but without the timing information (we refer to this as NXP's variant 1 below). It also presents a variant of the protocol in which the reader does not include a MAC with the *PROXIMITY CHECK* command (we refer to this as NXP's variant 2 below). Similar protocols are claimed which use encryption rather than MACs. It is not specified whether there is a unique key per card, or a global key that is shared between many cards.

Checking prover provided timing information: In the protocols above the prover sends the verifier information about how long responses should take. When testing security properties for these protocols we also need to ensure that the timing information is correctly authenticated.

The authentication for the timing information should be independent of how the information is used, or the location of the processes, therefore we may reasonably over-approximate the correctness of the timing information by removing the timer actions and running all processes in parallel in the applied pi-calculus, along with any required dishonest provers. The ProVerif tool lets us check the authenticity of information by checking correspondences between events. For protocols that strongly authenticate the prover's identity we check the authenticity of the timing information by adding an event($start(ti, id)$) to the start of the prover being tested, where *it* is a name representing the timing information, and *id* is the identity of the prover. We add an event($end(ti, id)$) to the verifier at the point it accepts the timing information as valid for prover *id*. For protocols that are anonymous, or do not authenticate the prover's identity, we replace the *id* in the event with the session nonces. We check that every end event has a corresponding start event, i.e., the verifier only accepts timing information as valid for a prover if the prover also performed a session with that timing information.

Analysis and results: We modelled MasterCard's RRP, PaySafe, NXP's protocols and several protocols from the literature as well as our example protocols in our calculus. Using our tool we compiled these to the applied pi-calculus with phases, and analyzed the resulting models with ProVerif. Table 1 summarizes the results of our analysis for the different protocols and attack scenarios. The compiled models can be significantly larger, as they scale linearly with the number of input operations. For example, the PayWave model becomes about 4 times longer than the original model when checking it for mafia fraud. For the results in Table 1, the verification with ProVerif finishes within a second on a system with an Intel Core i7-4550U and 8GB of RAM. For the protocols from the literature we used similar abstractions to model these as used in [28] and [15]. All models are available online. For the protocols from the literature [5, 24, 29, 30, 32, 35, 36] our analysis did not find any new results, so we focus on the industrial protocols.

We found that all the payment protocols protect against relay attacks and are safe in the uncompromised distance bounding scenario. It follows that your bank card is safe from relay attacks, even if someone else's card is compromised. PaySafe and MasterCard's RRP protocol do not defend against distance fraud, but Mauw et al.'s extension does. However, as noted above, distance fraud attacks are not applicable to protocols using ISO/IEC 14443, as it is always required to have a local adversary in order to communicate with the payment terminal. All of the protocols fail to protect against terrorist fraud attacks but, as discussed, we do not consider these applicable to the EMV attacker model. Therefore, we can conclude that all of the payment protocols meet their security goals with regard to relay attacks.

	Mafia Fraud / Relay	Uncompromised Distance Bounding	Distance Fraud	Terrorist Fraud	Timing information authenticity
Example 1 (Section 2)	OK	Attack	Attack	Attack	N/A
Example 2 (Section 2)	OK	OK	OK	OK	N/A
PaySafe	OK	OK	Attack	Attack	N/A
PaySafe with changes [28]	OK	OK	OK	Attack	N/A
MasterCard's RRP	OK	OK	Attack	Attack	OK
NXP's protocol (unique keys)	OK	OK	Attack	Attack	OK
NXP's protocol (global key)	OK	Attack	Attack	Attack	OK
NXP's variant 1 (unique keys)	OK	OK	Attack	Attack	N/A
NXP's variant 2 (unique keys)	OK	OK	Attack	Attack	N/A
Meadows et al. [30]	OK	OK	OK	Attack	N/A
MAD (One-Way) [36]	OK	OK	OK	Attack	N/A
CRCS [32]	OK	OK	OK	Attack	N/A
Hancke and Kuhn [24] Poulidor [35] Tree-based [5] Uniform [29]	OK	OK	OK	OK	N/A

Table 1: Results of our verification. The last four protocols use the same underlying distance bounding method.

NXP's protocols with a unique key for every device provide the same security against relay attacks as MasterCard's RRP and PaySafe. Here we again consider both distance and terrorist fraud attacks not applicable due to the underlying ISO/IEC 14443 protocol. However, if we assume that a global key is shared across a range of devices then security against relay attacks holds, but uncompromised distance bounding security does not. This is due to the fact that the compromise of one device is equal to the compromise of the complete system. This would represent a major security risk with, for example, a single compromised key fob putting all cars at risk.

The only property that can distinguish the case where one compromised device leads either to an attack only on this one device or to the compromise of the complete system, is our proposed uncompromised distance bounding property. None of the properties suggested in previous papers can detect the difference between a global and unique key used in the NXP protocol, so highlighting the need for our work.

Regarding the authentication of timing information, our analysis shows that MasterCard's RRP, PaySafe and NXP's protocols with unique keys correctly bind the identity to the timing information. As NXP's protocol with a global key does not authenticate the identity, we check the timing information against the session nonces, and find that it correctly binds these. Therefore, for these protocols, attacks aimed at the timing information will not work.

8 Conclusion

We have presented an applied pi-calculus based modelling framework for distance bounding protocols and attacks. We built a hierarchy of distance bounding attack

scenarios, and we have identified a new scenario for protocols that do not aim to protect against a compromised prover. We have defined a compiler from our calculus to the applied pi-calculus and use this compiler to analyse several distance bounding protocols, including protocols by MasterCard and NXP. We have also shown how the timing profiles used in these protocols can be verified.

Acknowledgements This work has been supported by the Netherlands Organisation for Scientific Research (NWO) through Veni project 639.021.750. We would like to thank Ioana Boureanu and Sjouke Mauw for useful comments on a draft of this paper.

References

[1] ABADI, M., BLANCHET, B., AND FOURNET, C. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. *JACM 65*, 1 (2017).

[2] ABADI, M., AND FOURNET, C. Mobile values, new names, and secure communication. In *POPL'01* (2001).

[3] AVOINE, G., BINGOL, M., BOUREANU, I., CAPKUN, S., HANCKE, G., KARDAS, S., KIM, C., LAURADOUX, C., MARTIN, B., MUNILLA, J., AND ET AL. Security of distance-bounding: A survey. *CSUR 4* (2017).

[4] AVOINE, G., BINGÖL, M. A., KARDAŞ, S., LAURADOUX, C., AND MARTIN, B. A framework for analyzing RFID distance bounding protocols. *JCS 19*, 2 (2011).

[5] AVOINE, G., AND TCHAMKERTEN, A. An efficient distance bounding RFID authentication protocol: Balancing false-acceptance rate and memory requirement. In *ISC'09* (2009).

[6] BBC. Car theft 'relay' devices seized in Birmingham. `http://www.bbc.com/news/uk-england-birmingham-42370086`.

[7] BLANCHET, B., ABADI, M., AND FOURNET, C. Automated verification of selected equivalences for security protocols. *JLAP* (2008).

[8] BLANCHET, B., SMYTH, B., CHEVAL, V., AND SYLVESTRE, M. ProVerif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial, 2018.

[9] BOUREANU, I., MITROKOTSA, A., AND VAUDENAY, S. Practical and provably secure distance-bounding. *JCS 23*, 2 (2015).

[10] BOUREANU, I., AND VAUDENAY, S. Optimal proximity proofs. vol. 8957 of *LNCS*.

[11] BRANDS, S., AND CHAUM, D. Distance-bounding protocols. In *EUROCRYPT'93* (1994).

[12] CHOTHIA, T., GARCIA, F. D., DE RUITER, J., VAN DEN BREEKEL, J., AND THOMPSON, M. Relay cost bounding for contactless EMV payments. In *FC'15*, vol. 8975 of *LNCS*. 2015.

[13] CREMERS, C. J. F., RASMUSSEN, K. B., SCHMIDT, B., AND CAPKUN, S. Distance hijacking attacks on distance bounding protocols. In *S&P'12* (2012), IEEE.

[14] DE, J., HUBMER, P., MURRAY, B., NEUMANN, H., STERN, S., AND THUERINGER, P. Decoupling of measuring the response time of a transponder and its authentication, 2011. EP Patent App. EP20,080,874,469.

[15] DEBANT, A., DELAUNE, S., AND WIEDLING, C. Proving physical proximity using symbolic models. Research report, Univ Rennes, CNRS, IRISA, France, 2018.

[16] DESMEDT, Y. Major security problems with the 'unforgeable' (Feige)-Fiat-Shamir proofs of identity and how to overcome them. In *SECURICOM* (1988).

[17] DESMEDT, Y., GOUTIER, C., AND BENGIO, S. Special uses and abuses of the Fiat-Shamir passport protocol. In *CRYPTO* (1987), vol. 293 of *LNCS*.

[18] DÜRHOLZ, U., FISCHLIN, M., KASPER, M., AND ONETE, C. A formal approach to distance-bounding RFID protocols. In *ISC'11* (2011).

[19] EMVCO. EMV – Integrated Circuit Card Specifications for Payment Systems, version 4.3, 2011.

[20] EMVCO. EMV Contactless Specifications for Payment Systems, version 2.6, 2016.

[21] FRANCILLON, A., DANEV, B., AND CAPKUN, S. Relay attacks on passive keyless entry and start systems in modern cars. In *NDSS'11* (2011).

[22] GARCIA, F. D., OSWALD, D., KASPER, T., AND PAVLIDÈS, P. Lock it and still lose it —on the (in)security of automotive remote keyless entry systems. In *USENIX Security'16* (2016), USENIX.

[23] HABRAKEN, R., DOLRON, P., POLL, E., AND DE RUITER, J. An RFID skimming gate using higher harmonics. In *RFIDSec'15*, vol. 9440 of *LNCS*. 2015.

[24] HANCKE, G., AND KUHN, M. An RFID distance bounding protocol. In *SecureComm'05* (2005), IEEE, pp. 67–73.

[25] JANSSENS, P. Proximity check for communication devices, 2017. US Patent 9,805,228.

[26] KANOVICH, M., KIRIGIN, T. B., NIGAM, V., SCEDROV, A., AND TALCOTT, C. Towards timed models for cyber-physical security protocols. In *FCS-FCC'14* (2014).

[27] MALLADI, S., BRUHADESHWAR, B., AND KOTHAPALLI, K. Automatic analysis of distance bounding protocols. *CoRR abs/1003.5383* (2010).

[28] MAUW, S., SMITH, Z., TORO-POZO, J., AND TRUJILLO-RASUA, R. Distance-bounding protocols: Verification without time and location. In *S&P'14* (2018).

[29] MAUW, S., TORO-POZO, J., AND TRUJILLO-RASUA, R. A class of precomputation-based distance-bounding protocols. In *EuroS&P'16* (2016).

[30] MEADOWS, C. A., POOVENDRAN, R., PAVLOVIC, D., CHANG, L., AND SYVERSON, P. F. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks* (2007).

[31] NIGAM, V., TALCOTT, C., AND URQUIZA, A. A. Towards the automated verification of cyber-physical security protocols: Bounding the number of timed intruders. In *ESORICS'16* (2016).

[32] RASMUSSEN, K. B., AND ČAPKUN, S. Realization of RF distance bounding. In *USENIX Security'10* (2010).

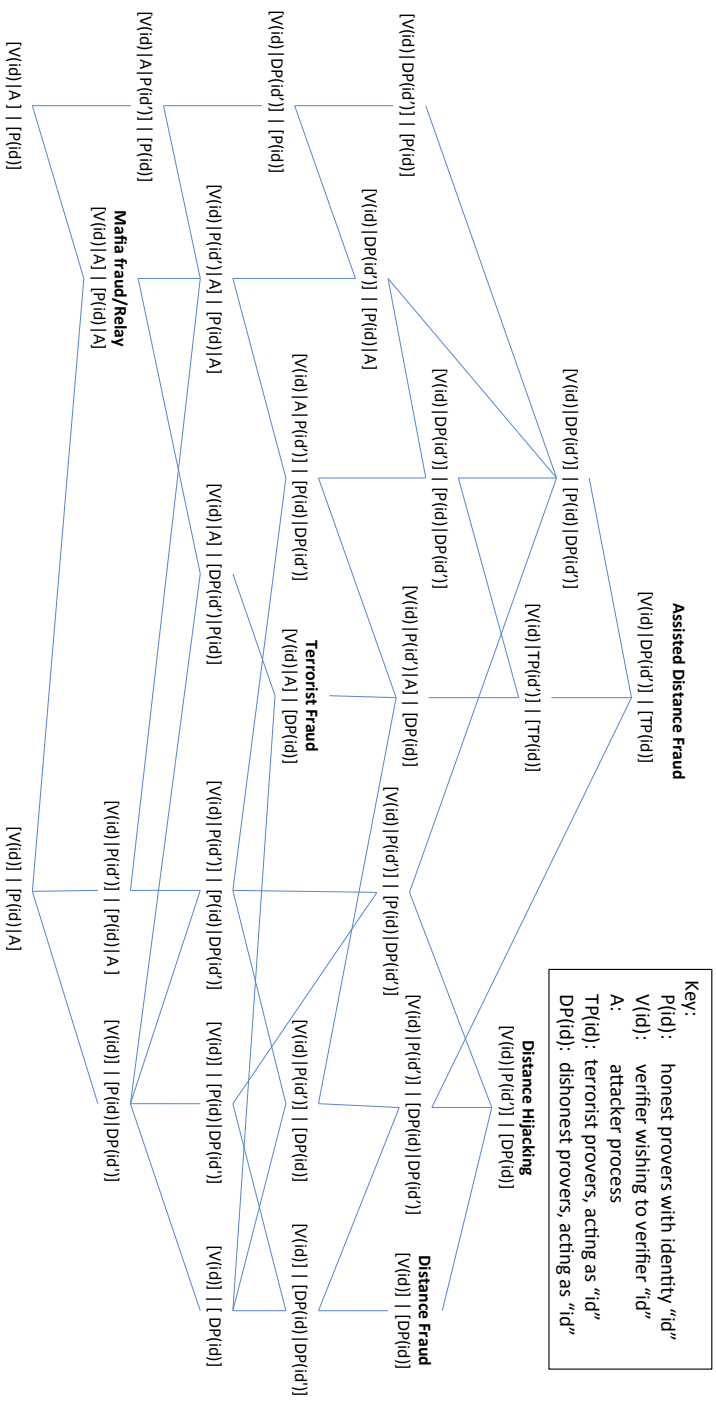
[33] RYAN, M. D., AND SMYTH, B. Applied pi calculus. In *Formal Models and Techniques for Analyzing Security Protocols*. IOS, 2011, ch. 6.

[34] SCHALLER, P., SCHMIDT, B., BASIN, D., AND CAPKUN, S. Modeling and verifying physical properties of security protocols for wireless networks. In *CSF'09* (2009).

[35] TRUJILLO-RASUA, R., MARTIN, B., AND AVOINE, G. The poulidor distance-bounding protocol. In *RFIDSec'10* (2010).

[36] ČAPKUN, S., BUTTYÁN, L., AND HUBAUX, J.-P. Sector: Secure tracking of node encounters in multi-hop wireless networks. In *SASN'03* (2003).

Figure 6 Ordering of all distance bounding attack scenarios that follows from lemmas 1, 2 and 3



A Proofs

Lemma 1 For all possible system contexts $C[\_]$ and sets of names *Init* for both $x = id$ and $x = id'$

$$\text{verified}(id){:}C[0] \Rightarrow \text{verified}(id){:}C[A] \Rightarrow \text{verified}(id){:}C[P(x)|A] \Rightarrow \text{verified}(id){:}C[TP\text{-}A(x)] \Rightarrow \text{verified}(id){:}C[DP\text{-}A(x)]$$

and

$$\text{verified}(id){:}C[0] \Rightarrow \text{verified}(id){:}C[P(x)] \Rightarrow \text{verified}(id){:}C[P(x)|A]$$

Furthermore, none of the implications hold in the reverse direction.

Proof: $\text{verified}(id) : C[0] \Rightarrow \text{verified}(id) : C[A] \Rightarrow \text{verified}(id) : C[P(x)|A]$ and $\text{verified}(id) : C[0] \Rightarrow \text{verified}(id) : C[P(x)] \Rightarrow \text{verified}(id){:}C[P(x)|A]$ follow directly from inclusion.

For $\text{verified}(id){:}C[P(x)|A] \Rightarrow \text{verified}(id){:}C[TP\text{-}A(x)]$, by definition $TP\text{-}A(x) = A|P$ for some process P that acts as an oracle for all new and free names and function applications in $P(x)$, therefore the attacker process used to show $\text{verified}(id){:}C[P(x)|A]$ can be extended to use the oracle process to act as $P(x)$, it can then be used to show $\text{verified}(id){:}C[TP\text{-}A(x)]$.

For $\text{verified}(id) : C[TP\text{-}A(x)] \Rightarrow \text{verified}(id) : C[DP\text{-}A(x)]$, the dishonest prover $DP\text{-}A(x)$ is defined as an attack process A along with a process that broadcasts all of the secret names and function applications in $P(x)$. The $TP\text{-}A(x)$ is defined as an attacker process and a process that acts as an oracle for all applications of secret names in P, therefore if there exists a process P_A which acts as an attacker to show that $\text{verified}(id){:}C[TP\text{-}A(x)]$ then we can extend the process P_A with another process that receives the secret names and uses the function applications from $DP\text{-}A(x)$ and then acts as an oracle, mimicking $TP\text{-}A(x)$. Therefore if $\text{verified}(id){:}C[TP\text{-}A(x)]$ then $\text{verified}(id){:}C[DP\text{-}A(x)]$.

To show that none of the reverse implications hold, we assume the existence of a distance bounding protocol $(P(id), !V, \tilde{n})$, where $P(id) = !\mathit{new}\ id.!P$ and that this protocol is secure against all of the attacks defined in this paper. We now extend this protocol and define system contexts that separate each case:

- $\text{verified}(id){:}C[0] \not\Leftarrow \text{verified}(id){:}C[A]$. We define a verifier $\hat{V} = !(V|\mathit{new}\ a.\mathit{out}(a).\mathit{in}(= a, id).\text{event}(\mathit{verify}(id)))$ i.e., this new verifier will automatically verify any id sent to it with a new value. We further assume that the prover does not perform this action. We consider $C[\_] = [\hat{V}|\_] \mid [P(id)]$. In which case we have $\text{verified}(id){:}C[A]$ but not $\text{verified}(id){:}C[0]$.

- $\text{verified}(id){:}C[A] \not\Leftarrow \text{verified}(id){:}C[P(x)|A]$: We consider a version of the verifier $\hat{V}$ which is the same as $!V$ but $\mathit{event}(\mathit{verify}(id))$ is replaced with $\mathit{in}(x).\mathit{event}(\mathit{verify}(x))$. We then define $C[\_] = [\hat{V}|\_] \mid [P(id)]$ and observe that $\text{verified}(id){:}C[P(x)|A]$ but not $\text{verified}(id){:}C[A]$.

- $\text{verified}(id){:}C[P(x)|A] \not\Leftarrow \text{verified}(id){:}C[TP\text{-}A(x)]$: We add a public/private key pair to the verifier and prover, and extend the verifier so that it verifies any process that sends it the constant *accept* signed by this key. For C with a TP-Athe process $C[P(x)|A]$ cannot produced this signed name, because the attacker will not get access to the key. However, TP-A(x) will include an oracle of the use of this key, and so we will get $\text{verified}(id){:}C[TP\text{-}A(x)]$, whenever $TP\text{-}A(x)$ can be reached in C.

- $\text{verified}(id){:}C[TP\text{-}A(x)] \not\Leftarrow \text{verified}(id){:}C[DP\text{-}A(x)]$: We consider our secure verifier, and extend it so that all provers and the verifier share a secret key. We also extend the verifier so that if it is ever sent the secret key in the clear it automatically verifiers any id that is sent with it. By definition TP-A(x) cannot sent the private key, but $DP\text{-}A(x)$ therefore we have $\text{verified}(id){:}C[DP\text{-}A(x)]$ but not $\text{verified}(id){:}C[TP\text{-}A(x)]$

- $\text{verified}(id){:}C[0] \not\Leftarrow \text{verified}(id){:}C[P(x)]$. We consider a verifier that, if a single prover ever correctly verifies itself then the timer checks are removed for all future verification attempts. We define $C[\_] = [\hat{V}|\_] \mid [P(id)]$ therefore we will have $\text{verified}(id){:}C[P(x)]$ but not $\text{verified}(id){:}C[0]$

- $\text{verified}(id){:}C[P(x)] \not\Leftarrow \text{verified}(id){:}C[P(x)|A]$. This follows from the same reasoning use to show that $\text{verified}(id){:}C[0] \not\Leftarrow \text{verified}(id){:}C[A]$.

Lemma 2 For all possible system contexts $C[\_]$, names id' and sets of names $Init$:

$$\begin{array}{rll} & \text{verified}(id){:}C[P(id')|P(id)] & \Leftrightarrow \text{verified}(id){:}C[P(id)] \\ \text{and} & \text{verified}(id){:}C[TP\text{-}A(id')|TP\text{-}A(id)] & \Leftrightarrow \text{verified}(id){:}C[TP\text{-}A(id)] \\ \text{and} & \text{verified}(id){:}C[DP\text{-}A(id')|DP\text{-}A(id)] & \Leftrightarrow \text{verified}(id){:}C[DP\text{-}A(id)] \end{array}$$

Proof In each case the $\Leftarrow$ direction is trivial. We observe that, by definition, $P(x) = !new\ x.!P$, for some P, therefore if $P(id')$ is used in the reduced then the (NEW) rule: $E, \mathscr{L} \cup \{[\mathscr{P} \cup \{new\ id'.!P\}]_r\} \rightarrow E \cup \{a\}, \mathscr{L} \cup \{[\mathscr{P} \cup \{!P\{a/id'\}\}]_r\}$ must be used. However, $P(id)$ can reduce to the same process: $E, \mathscr{L} \cup \{[\mathscr{P} \cup \{new\ id.!P\}]_r\} \rightarrow E \cup \{a\}, \mathscr{L} \cup \{[\mathscr{P} \cup \{!P\{a/id\}\}]_r\}$. Therefore, $\text{verified}(id){:}C[P(id')|P(id)] \Rightarrow \text{verified}(id){:}C[P(id)]$, as any process that may appear on the left can also appear on the right. In a similar way, we observe that $TP\text{-}A(x) = !new\ x.!P'$ and $DP\text{-}A(x) = !new\ x.!P'$ for some P' and P'' so here too, the *newid* and *newid'* terms can reduce to use the same names and similar reasoning as the P case establishes the other $\Rightarrow$ cases.

Lemma 3 For any P, Q, id, $\tilde{n}$, E, and for both $x = id$ and $x \neq id$ we have that

$$\text{verified}(id){:}new\ \tilde{n}.[P|A]||[DP\text{-}A(x)|Q] \Leftrightarrow \text{verified}(id){:}new\ \tilde{n}.[P|DP\text{-}A(x)]||[DP\text{-}A(x)|Q] \Leftrightarrow \text{verified}(id){:}new\ \tilde{n}.[P|DP\text{-}A(x)]||[A|Q]$$

Proof: The implications $\text{verified}(id){:}new\ \tilde{n}.[P|A]||[DP\text{-}A(x)|Q]] \Rightarrow \text{verified}(id){:}new\ \tilde{n}.[P|DP\text{-}A(x)]||[DP\text{-}A(x)|Q]]$ and $\text{verified}(id){:}new\ \tilde{n}.[P|DP\text{-}A(x)]||[A|Q]] \Rightarrow \text{verified}(id){:}new\ \tilde{n}.[P|DP\text{-}A(x)]||[DP\text{-}A(x)|Q]]$ follow from the definition of $DP\text{-}A(x)$ as equal to $R|A$ for some process R. In the opposite directions, we note that the R process just outputs all names from a prover. The attacker process at the location with the $DP\text{-}A$ can be extended to receive these names and output them twice. There by replicating the R process (and so $DP\text{-}A$) at both locations.

Lemma 4 For linear processes L_v and L_p with no timer actions or events, any linear verifier V_L, and any $id, \tilde{n}, Init$: $\text{verified}(id){:}new\ \tilde{n}.[!V_L|L_v|A] \mid [L_p|A] \Rightarrow \text{verified}(id){:}new\ \tilde{n}.[V_L|!\text{blind}(V_L)|L_v|A] \mid [L_p|A]$

Proof We assume that $\text{verified}(id){:}new\ \tilde{n}.[!V_L|L_v|A] \mid [L_p|A]$, i.e., there exists a process P_A such that

$$\begin{array}{rll} Init, \{\ new\ \tilde{n}.\,[\{!V_L|L_v|P_A\}]_0 \mid [\{L_p|P_A\}]_0\ \} & \rightarrow^* & E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{new\ id.P\}]_r\ \} \\ & \rightarrow & E \cup \{id'\}, \mathscr{L} \cup \{\ \left[\mathscr{P} \cup \{P\{^{id'}/_{id}\}\}\right]_r\ \} \\ & \rightarrow^* & E', \mathscr{L}' \cup \{\ [\mathscr{P}' \cup \{\text{event}(\text{verify}(id')).P'\}]_{r'}\ \} \end{array}$$

Without loss of generality we can assume that we unwind all necessary instances of the V_L process first, and that the $\text{event}(\text{verify}(id'))$ term is the first unguarded verify event to appear in the trace, i.e.,

$$\begin{array}{rll} Init, \{\ new\ \tilde{n}.\,[\{!V_L|L_v|P_A\}]_0 \mid [\{L_p|P_A\}]_0\ \} & \rightarrow^* & Init \cup \{\tilde{n}'\}, \{\ ([\{V_L, V_L, \ldots, V_L, !V_L, L_v|P_A\}]_0, [\{L_p|P_A\}]_0)\{\tilde{n}'/\tilde{n}\}\ \} \\ & \rightarrow^* & E, \mathscr{L} \cup \{\ [\mathscr{P} \cup \{new\ id.P\}]_r\ \} \\ & \rightarrow & E \cup \{id'\}, \mathscr{L} \cup \{\ \left[\mathscr{P} \cup \{P\{^{id'}/_{id}\}\}\right]_r\ \} \\ & \rightarrow^* & E', \mathscr{L}' \cup \{\ [\mathscr{P}' \cup \{\text{event}(\text{verify}(id')).P'\}]_{r'}\ \} \end{array}$$

As we are considering a trace in which $\text{event}(\text{verify}(id'))$ appears unguarded only in the final step, and the replication rule is only used at the start, we may remove all other event actions and there will still exist a trace:

$$\begin{array}{rll} Init \cup \{\tilde{n}'\}, \{\ ([\{V_L, V_L', \ldots, V_L', L_v|P_A\}]_0 | [\{L_p|P_A\}]_0)\{\tilde{n}'/\tilde{n}\}\ \} & \rightarrow^* & E, \mathscr{L}_2 \cup \{\ [\mathscr{P}_2 \cup \{new\ id.P\}]_r\ \} \\ & \rightarrow & E \cup \{id'\}, \mathscr{L}_2 \cup \{\ \left[\mathscr{P}_2 \cup \{P\{^{id'}/_{id}\}\}\right]_r\ \} \\ & \rightarrow^* & E', \mathscr{L}_2' \cup \{\ [\mathscr{P}_2' \cup \{\text{event}(\text{verify}(id')).P'\}]_{r'}\ \} \end{array}$$

where V_L' equals V_L with all events removed.

We note that V_L' with all instances of *startTimer* and *stopTimer* removed equals $\text{blind}(V_L)$ and that removing the timer actions does not stop any of the none timer actions in the trace above. Therefore there exists a trace:

$$\begin{array}{rll} Init \cup \{\tilde{n}'\}, \{\ ([\{V_L, \text{blind}(V_L), \ldots, \text{blind}(V_L), L_v|P_A\}]_0 | [\{L_p|P_A\}]_0)\{\tilde{n}'/\tilde{n}\}\ \} & \rightarrow^* & E, \mathscr{L}_3 \cup \{\ [\mathscr{P}_3 \cup \{new\ id.P\}]_r\ \} \\ & \rightarrow & E \cup \{id'\}, \mathscr{L}_3 \cup \{\ \left[\mathscr{P}_3 \cup \{P\{^{id'}/_{id}\}\}\right]_r\ \} \\ & \rightarrow^* & E', \mathscr{L}_3' \cup \{\ [\mathscr{P}_3' \cup \{\text{event}(\text{verify}(id')).P'\}]_{r'}\ \} \end{array}$$

And as

$Init, \{new\ \tilde{n}.\,[V_L|!\text{blind}(V_L)|L_v|P_A)]_0 \mid [\{L_p|P_A\}]_0\} \rightarrow^* Init\{\tilde{n}'\}, \{\,([\{V_L, \text{blind}(V_L), \ldots, \text{blind}(V_L), !\text{blind}(V_L), L_v|P_A\}]_0, [\{L_p|P_A\}]_0)\{\tilde{n}'/\tilde{n}\}$

we can conclude verified(id):$new\ \tilde{n}.[V_L|!\text{blind}(V_L)|L_v|A] \mid [L_p|A]$

Theorem 1 Given a name id and a system $S = new\ \tilde{n}.[!V_L \mid L_v \mid A] \mid [!new\ id.!P_L \mid L_p \mid A]$ or $S = new\ \tilde{n}.[!V_L|L_v]|[!new\ id.!P_L|L_p|A]$ then:

$$not\ ev(\text{verify}(id)), \{c\} : compile(id, S) \Rightarrow \neg\ \text{verified}(\{c\}, id):S$$

Proof

We show that $\text{verified}(id) : S \Rightarrow ev(\text{verify}(id)), \{c\} : compile(id, S)$ and we first consider $S = new\ \tilde{n}.[!V_L \mid L_v \mid A] \mid [!new\ id.!P_L \mid L_p \mid A]$. By Lemma 4:

$$\text{verified}(id):new\ \tilde{n}.[!V_L \mid L_v \mid A] \mid [!new\ id.!P_L \mid L_p \mid A] \Rightarrow \text{verified}(id): new\ \tilde{n}.[V_L \mid !\text{blind}(V_L) \mid L_v \mid A] \mid [!new\ id.!P_L \mid L_p \mid A]$$

I.e., there exists a process an attacker P_A such that, without loss of generality we may assume that $P_A = P_{A1}|P_{A2}$ where P_1 and P_2 are linear and only act at a single location.:

$$\begin{array}{lll}
 & \{c\}, & new\ \tilde{n}.\,[\{V_L, !\text{blind}(V_L), L_v, P_{A1}\}]_0 \mid [\{!new\ id.!P_L, L_p, P_{A2}\}]_0 \\
\rightarrow^* & \{c, \tilde{n}'\}, & ([\{V_L, !\text{blind}(V_L), L_v, P_{A1}\}]_0 \mid [\{!new\ id.!P_L, L_p, P_{A2}\}]_0)\{\tilde{n}'/\tilde{n}\} \\
\rightarrow & \{c, \tilde{n}'\}, & ([\{V_L, !\text{blind}(V_L), L_v, P_{A1}\}]_0 \mid [\{new\ id.!P_L, !new\ id.!P_L, L_p, P_{A2}\}]_0)\{\tilde{n}'/\tilde{n}\} \\
\rightarrow & \{c, \tilde{n}', id'\}, & ([\{V_L, !\text{blind}(V_L), L_v, P_{A1}\}]_0 \mid [\{!P_L\{id'/id\}, !new\ id.!P_L, L_p, P_{A2}\}]_0)\{\tilde{n}'/\tilde{n}\} \\
\rightarrow^* & E''', & \mathscr{L}' \cup \{\ [\mathscr{P}' \cup \{\text{event}(\text{verify}(id')).P_e\}]_0\ \}
\end{array}$$

Because we are using linear processes we can apply all necessary uses of the replication rule at the start. Therefore, we may have:

$$\begin{array}{lll}
\rightarrow & \{c, \tilde{n}', id'\}, & ([\{V_L, !\text{blind}(V_L), L_v, P_{A1}\}]_0 \mid [\{!P_L\{id'/id\}, !new\ id.!P_L, L_p, P_{A2}\}]_0)\{\tilde{n}'/\tilde{n}\} \\
\rightarrow^* & \{c, \tilde{n}', id'\}, & ([\{V_L, \text{blind}(V_L)_1, \ldots \text{blind}(V_L)_i, L_{v1}, \ldots L_{vj}, P_{A1}\} \cup P_{D1}]_0 \\
 & & \mid \left[\{P_{L1} \ldots P_{Lk}, L_{p1}, \ldots L_{pl}, P_{A2}\} \cup P_{D2}\right]_0)\{\tilde{n}'/\tilde{n}\} \\
\rightarrow^* & E''', & \left[\{\text{event}(\text{verify}(id')).P_e, V'''_{L1}, \ldots V'''_{Li}, L'''_{v1}, \ldots L'''_{vj}, 0\} \cup Outputs'''_V \cup P_{D1}\right]_0 \\
 & & \mid \left[\{P'''_{L1} \ldots P'''_{Lk}, L'''_{p1}, \ldots L'''_{pl}, 0\} \cup Outputs'''_P \cup P_{D2}\right]_0
\end{array}$$

where $Outputs'''_V$ and $Outputs'''_P$ contain only output actions.

The last reduction stage in the above trace does not use parallel or replication rules, just linear reductions. This means that (apart from the outputs) the number of processes we have remains static. We have made the assumption that the verifier uses a single timer, therefore we can further refine the trace to:

$$\begin{array}{lll}
\rightarrow^* & \{c, \tilde{n}', id'\}, & ([\{V_L, \text{blind}(V_L)_1, \ldots \text{blind}(V_L)_i, L_{v1}, \ldots L_{vj}, P_{A1}\} \cup P_{D1}]_0 \\
 & & \mid \left[\{P_{L1} \ldots P_{Lk}, L_{p1}, \ldots L_{pl}, P_{A2}\} \cup P_{D2}\right]_0)\{\tilde{n}'/\tilde{n}\} \\
\rightarrow^* & E', & \left[\{startTimer.V'_L, V'_{L1}, \ldots V'_{Li}, L'_{v1}, \ldots L'_{vj}, P'_{A1}\} \cup Outputs'_v \cup P_{D1}\right]_0 \\
 & & \mid \left[\{P'_{L1} \ldots P'_{Lk}, L'_{p1}, \ldots L'_{pl}, P'_{A2}\} \cup Outputs'_P \cup P_{D2}\right]_0 \\
\rightarrow^* & E'', & \left[\{stopTimer.V''_L, V''_{L1}, \ldots V''_{Li}, L''_{v1}, \ldots L''_{vj}, P''_{A1}\} \cup Outputs''_V \cup P_{D1}\right]_1 \\
 & & \mid \left[\{P'_{L1} \ldots P'_{Lk}, L'_{p1}, \ldots L'_{pl}, P'_{A2}\} \cup Outputs'_P \cup P_{D2}\right]_0 \\
\rightarrow^* & E''', & \left[\{\text{event}(\text{verify}(id')).P_e, V'''_{L1}, \ldots V'''_{Li}, L'''_{v1}, \ldots L'''_{vj}, 0\} \cup Outputs'''_P \cup P_{D1}\right]_0 \\
 & & \mid \left[\{P'''_{L1} \ldots P'''_{Lk}, L'''_{p1}, \ldots L'''_{pl}, 0\} \cup Outputs'''_V \cup P_{D2}\right]_0
\end{array}$$

We choose the linear processes in the second and third term so that each starts with an input or is a null process, and we do not make any reductions at the non-verifier location between these terms. We further require this trace to avoid any repeated application of the (GLOBAL) rule to move a single outputs repeatedly between locations.

We can now characterise each of the linear processes for each of the reduction stages, i.e, for processes at the same verifier location:

$$\begin{array}{ccccccccc}
V_L = C_{vn}[startTimer.V'_L] & \text{and} & V'_L = C'_{vn}[stopTimer.V''_L] & \text{and} & V''_L = C''_{vn}[\text{event}(\text{verify}(id')).P_e] & & \\
\text{blind}(V_L)_n = C_{vln}[V'_{Ln}] & \text{and} & V'_{Ln} = C'_{vln}[V''_{Ln}] & \text{and} & V''_{Ln} = C''_{vln}[V'''_{Ln}] & & \text{for } 1 \leq n \leq i \\
L_{vn} = C_{lvn}[L'_{vn}] & \text{and} & L'_{vn} = C'_{lvn}[L''_{vn}] & \text{and} & L''_{vn} = C''_{lvn}[L'''_{vn}] & & \text{for } 1 \leq n \leq j \\
 & & P_{A1} = C_{A1}[P'_{A1}] & \text{and} & P'_{A1} = C'_{A1}[P''_{A1}] & &
\end{array}$$

and for processes remote from the verifier we have:

$$\begin{array}{llll} P_{Ln} = C_{Ln}[P'_{Ln}] & \text{and} & P'_{Ln} = C'_{Ln}[P'''_{Ln}] & \text{for } 1 \leq n \leq j \\ L_{pn} = C_{pn}[L'_{pn}] & \text{and} & L'_{pn} = C'_{pn}[L'''_{pn}] & \text{for } 1 \leq n \leq j \\ & & P_{A2} = C_{A2}[P'_{A2}] & \end{array}$$

Our encoding will produce corresponding processes such that in front of each of these processes, which is not null, we have a phase jump: phase 1 between the first and second processes at the same location as the verifier, phase 2 between the second and third processes at the same location as the verifier, and phase 2 between the first and second processes remote from the verifier. Additionally the compilation replaces the *startTimer* in V_L with a jump to phase 1, and the *stopTimer* with a jump to phase 2. Explicitly we define:

$$\begin{array}{ll} \lfloor V_L \rfloor & = C_{vn}[1 : C'_{vn}[2 : C''_{vn}[\text{event}(\text{verify}(id')).P_e]]] \\ \lfloor \text{blind}(V_L)_n \rfloor & = C_{vln}[1 : C'_{vln}[2 : C''_{vln}[V'''_{Ln}]]] \text{ for } 1 \leq n \leq i \\ \lfloor L_{vn} \rfloor & = C_{lvn}[1 : C'_{lvn}[2 : C''_{lvn}[L'''_{vn}]]] \text{ for } 1 \leq n \leq j \\ \lfloor P_{Ln} \rfloor & = C_{Ln}[2 : C'_{Ln}[P'''_{Ln}] \text{ for } 1 \leq n \leq j \\ \lfloor L_{pn} \rfloor & = C_{pn}[2 : C'_{pn}[L'''_{pn}]] \text{ for } 1 \leq n \leq j \\ \lfloor P_{A1} \rfloor & = C_{A1}[1 : C'_{A1}[2 : P''_{A1}] \\ \lfloor P_{A2} \rfloor & = C_{A2}[2 : P'_{A2}] \end{array}$$

and we consider the applied pi-calculus process:

$$P_{pi} = \lfloor V_L \rfloor || \lfloor \text{blind}(V_L)_1 \rfloor | \ldots | \lfloor \text{blind}(V_L)_i \rfloor || \lfloor L_{v1} \rfloor | \ldots \lfloor L_{vj} \rfloor || \lfloor P_{L1} \rfloor | \ldots | \lfloor P_{Lk} \rfloor || \lfloor L_{p1} \rfloor | \ldots | \lfloor L_{pl} \rfloor || \lfloor P_{A1} \rfloor || \lfloor P_{A2} \rfloor | A_{pi}$$

where the process A_{pi} is defined below. We note that $\lfloor V_L \rfloor = tToPh(V_L)$ and $\lfloor \text{blind}(V_L)_n \rfloor$ will be in $phases(\text{blind}(V_L, [1,2])$ and $\lfloor L_{vn} \rfloor$ will be in $phases(L_v), [1,2])$ and $\lfloor P_{Lk} \rfloor$ will be in $phases(L_p, [2])$ and that there exists a member of $!new\ id.(phases(P_L, [2]))$ that can reduce to $\lfloor P_{Ln} \rfloor$. Therefore $compile(id, S) || \lfloor P_{A1} \rfloor || \lfloor P_{A2} \rfloor | A \rightarrow^* P_{pi} | Q$ for some Q.

We will next show that P_{pi} can follow the same reduction path as S, so performing the event, which we will do by induction on the length of the reduction. The base case, of 0 steps, is trivial. For the step case we consider each rule in turn.

If the (REPL), (PAR), (NEW), (LET 1) or (LET 2) rules are possible in the location calculus then, by definition, the matching step will be possible in the applied pi-calculus. No events are executed in the trace from S, therefore the (EVENT) rule does not apply.

The trace in the location calculus only contains a single application of the (START) rule. We observe that at this point all processes in the applied pi-calculus process, at the verifier's location, are prefixed with a $1:$ action. The application of the (START) rule is match by an application of the (ORDER) rule for all of these processes. Likewise, there is a single application of the (STOP) rule in the location trace, for which all applied pi-calculus processes are prefixed with a $2:$ action, and the (START) reduction is match by a (ORDER) for all of these traces. For both steps, we extend the attacker process A_{pi} to move any necessary outputs forward a phase.

If the reduction in the location calculus takes place with the (ASYNC) rule then it can be matched by the attacker process A_{pi} receiving the message and outputting it again.

If the reduction in the location calculus takes place with the (I/O LOCAL) rule then we note all of the processes at both the verifier location and the prover location always remain in the same phase, and so this rule can be matched by a reduction in the applied pi-calculus using the (I/O) rule.

For the (GLOBAL) rule, if the timer r on the outputting process is 0 (i.e., before the *startTimer* action when both locations are at phase 0, or after the *stopTimer* action, when both locations are at phase 2), then this reduction can be matched by a reduction of the (I/O) rule. If the timer r equals 1 (i.e., between the *startTimer* and *stopTimer* actions, when the verifier location processes are in phase 1), we note that the prover location processes do not make any reductions until after the *stopTimer* action (phase 2) and our requirements on the trace mean that each output will only be moved between locations once, therefore we may match the (GLOBAL) reduction by extending the attacker process to receive the output in phase 1 and forward it at phase 2.

Hence, the applied pi-calculus process will match the reduction of the location calculus system and reduce to perform the event, i.e., $ev(\text{verify}(id)), \{c\} : compile(id, S)$.

We now consider the case in which the attacker does not act at the verifier's location, i.e., $S = new\ \tilde{n}.[!V_L|L_v]|[!new\ id.!P_L|L_p|A]$. We observe that Lemma 4 does not require the attacker process at the verifier location, therefore the equivalent lemma holds for this S.

This case follows the same reasoning as above, but without the P_{A1} process in the location calculus model and the $\lfloor P_{A1} \rfloor$ process in the applied pi-calculus model, and with the addition of the helper processes: $!in(x).1:out(x)|!1:in(x)2:out(x)|!1:in(x).out(x)$ to the applied pi-calculus model.

The proof follows in the same way as above, except for the cases of the (GLOBAL) (I/O LOCAL) and (ASYNC) rules that are applied while the timer is running. Unlike above, in these cases, the applied pi-calculus model will be using a private channel that is not available to the attacker.

In the case of the (GLOBAL) rule used to move an output from the verifier location, while the timer is running, to the prover location, this reduction can be match by the helper process $1:in(x)2:out(x)$.

In the case of the (I/O LOCAL) rule being used to receive an output made available before the timer started (i.e., phase 0, on public channel c) at a input while the timer is running (i.e., phase 1, on the private channel), we use the helper process $in(x).1:out(x)$ to make the output available on the private channel, and then the (I/O) rule to perform the communication.

In the case of the application of the (ASYNC) rule while the timer is running we use the helper process $1:in(x).out(x)$ to make a matching reduction.

Therefore, when $S = new\ \tilde{n}.[!V_L|L_v]|[!new\ id.!P_L|L_p|A]$ we also get $ev(\text{verify}(id)), \{c\}:compile(id,S)$.