



Is Eve nearby? Analysing protocols under the distant-attacker assumption


Reynaldo Gil-Pons

Department of Computer Science
University of Luxembourg
Luxembourg
reynaldo.gilpons@uni.lu 

Ross Horne

Department of Computer Science
University of Luxembourg
Luxembourg
ross.horne@uni.lu 

Sjouke Mauw

Department of Computer Science
University of Luxembourg
Luxembourg
sjouke.mauw@uni.lu 

Alwen Tiu

School of Computing
The Australian National University
Australia
alwen.tiu@anu.edu.au 

Rolando Trujillo-Rasua

Centre for Cyber Security Research and Innovation
School of Information Technology
Deakin University
Australia
rolando.trujillo@deakin.edu.au 

Abstract—Various modern protocols tailored to emerging wireless networks, such as body area networks, rely on the proximity and honesty of devices within the network to achieve their security goals. However, there does not exist a security framework that supports the formal analysis of such protocols, leaving the door open to unexpected flaws. In this article we introduce such a security framework, show how it can be implemented in the protocol verification tool TAMARIN, and use it to find previously unknown vulnerabilities on two recent key exchange protocols.

Index Terms—security protocols, formal verification, key exchange, distance bounding, distant attacker

I. INTRODUCTION

In the past few years, we have seen the emergence of a new class of security protocols that have as a common feature that their security goals are only guaranteed under the assumption that the adversary is not in the proximity of the proper communication partners. Examples of such protocols are key exchange protocols for Body Area Networks [1], pairing protocols of smart devices [2]–[4], or protocols for memory erasure and memory attestation [5], [6].

Assuming that attackers are far or distant, called the *distant-attacker assumption*, can be motivated in various ways. A local attacker can, for instance, be excluded due to physical protection or human observation of the environment. Alternatively, attacks by local agents may be considered infeasible due to the use of out-of-band channels that open to nearby devices only, such as short-range or low-powered communication. A traceability attack, which occurs when a user can be traced based on the transcripts of the communication protocol, that requires the attacker to be close to the victim is arguably ineffective, as the victim is already being physically monitored. Lastly, memory erasure and attestation protocols have proven unable to resist a standard man-in-the-middle attacker [7], such as the Dolev-Yao attacker. The state-of-the-practice for this type of

protocols is to isolate the prover and verifier by radio jamming or hardware manipulation.

Most of the protocols that depend on the distant-attacker assumption have not been formally verified yet and may thus suffer from unexpected vulnerabilities. This lack of verification effort can mainly be explained by the use of informal, physical or out-of-band techniques that are hard to formalize in a symbolic security model. Hence there is a need for a security model that makes explicit what a distant-attacker can and cannot do, and that is amenable to formal verification.

Because the distant-attacker assumption states that the distance between the adversary and the proper communication partners is (much) larger than the mutual distance between the communication partners, this notion appears to have a strong link to the notion of distance-bounding protocols. However, while the goal of distance-bounding protocols is to ensure that the communication partners are close, protocols from the above mentioned class aim to ensure some classical security property, like secrecy or authentication, under the assumption of proximity of the communication partners. Such focus on distance bounding is reflected on the verification frameworks [8]–[13] developed for the verification of distance-bounding protocols using round-trip time, which ignore classical security properties.

The pairing protocol depicted in Figure 1, inspired by the design of Move2Auth [3], is an example of the type of protocols we are referring to: it aims at secure key exchange, relies on the distant-attacker assumption, and uses the physical properties of the communication channel to check proximity. The goal is for an agent V to create a shared key k_{vp} with another agent P . V does not have any previous cryptographic secret shared with P , but it is confident that all agents in its vicinity are honest. The prover first generates a public/private key pair, denoted $\text{pk}(k)$ and k , respectively. The public key, together with the signature of the prover's identity P with k , is sent to V . Upon reception, V generates a fresh symmetric key

This work was supported by the Luxembourg National Research Fund, Luxembourg, under the grant AFR-PhD-14565947.

k_{vp} , executes a round-trip-time (RTT) measurement with P , and sends k_{vp} encrypted with $\text{pk}(k)$. The RTT measurement, illustrated by dashed arrows, is based on the messages n_v and $n_v \oplus n_p$. Once P decrypts $\{k_{vp}\}_{\text{pk}(k)}$, it confirms reception of the key by encrypting the nonce n_p and its own identity P with k_{vp} . If the RTT is lower than a time threshold Δ , and $\langle n_p, P \rangle$ is correctly encrypted with k_{vp} , then V concludes that P is nearby and, therefore, honest (based on the distant-attacker assumption). This allows V to claim that k_{vp} is secret, i.e. unknown to attackers.

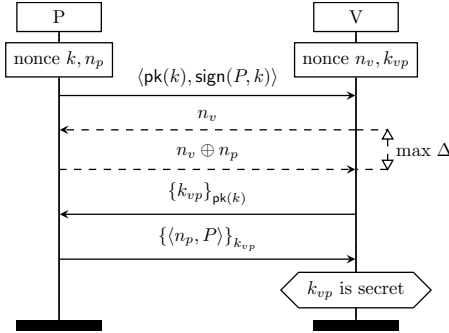


Fig. 1: An insecure pairing protocol \mathcal{P}_{ex} : a running example

Based on our study of the literature, the most common security argument used to analyse this type of protocol is to assume that some messages are unavailable to the attackers, because they are far. Case in point: if the messages used to measure RTT can only be received by honest and nearby agents, then the last message $\{\langle n_p, P \rangle\}_{k_{vp}}$ should have been generated by an honest agent, supposedly P . By using this security argument one may conclude that the protocol in Figure 1 is secure. Yet it is not. The protocol suffers from an attack known as *distance fraud* in the distance-bounding literature [14], because it relies on the ability of an attacker to inject messages from far away.

The attack works as follows. The attacker E executes the protocol with V by sending their own public key. E does not wait for the second message. Instead, E sends a random value m soon enough to be received by V right after V sends n_v . The same antenna that E uses to inject messages is used to eventually receive the verifier’s challenge n_v . This allows E to compute $n_p = n_v \oplus m$ and correctly finish the protocol with V . There is another attack on this protocol, that does not require the attacker to interfere with the RTT measurement. It consist in hijacking a session between an honest prover P and V in a similar fashion to distance-hijacking attacks on distance-bounding protocols [15]. Without providing further details on the second attack, we argue that the intuitive use of a secure or protected channel between nearby devices to formally model the physical assumptions underlying the protocol, risks missing attacks. Hence, such a modelling could lead to the conclusion that the protocol is safe, while it is not. Consequently, there is a need for a verification methodology that augments standard symbolic security protocol verification

with a distant-attacker model and incorporates techniques used for the verification of protocols with physical properties, such as distance bounding protocols [9]. This article introduces such a methodology.

Our methodology starts from a time-based security model which allows for the analysis of standard security properties (Section III). Next, we add the distant-attacker assumption and round-trip-time restrictions (Section IV). In order to prepare for efficient verification with an analysis tool like Tamarin [16], this time-based model is then reduced to a causality-based model (Section V). The input to our methodology consists of the formalized description of a security protocol, in which we modelled the distant-attacker assumption as a time-bound challenge-response loop in the protocol. Using this approach, we formally verified seven protocols for which we found a number of novel attacks (Section VI).

II. RELATED WORK

There exist various symbolic models to analyse security protocols that depend on time and location, most of them specifically targeting distance-bounding protocols. The first one [17] was proposed in 2007. That model allowed for the analysis of distance-bounding protocols while faithfully representing time and location, but it lacked support for computer-aided verification. Basin et al. [8] later addressed that limitation, introducing a security model for distance-bounding protocols with tool support. They made explicit that the arrival time of a message depends on the locations of sender and receiver, and the maximum propagation speed of the communication channel. This allows their model to formulate distance-bounding security as a statement on whether a round-trip-time measurement is lower than or equal to twice the distance to the communicating partner divided by the propagation speed. Our model is inspired by these modelling choices. To assist formal verification, Basin et al. encoded their model in the theorem proving assistant Isabelle/HOL. Their approach is not fully automated, though, requiring end-users to define several protocol-dependent lemmas.

The problem of realising a fully automated verification framework for distance-bounding protocols was solved independently in 2018 by Mauw et al. [9], Chothia et al. [10], and Debant et al. [11], [12]. In [18] a procedure to analyse these protocols is presented and integrated in Akiss¹. Our work intersects with all those seminal works in different ways. Like in [9], we frame time-based protocol requirements as causal relations of protocol events, and show they can be automatically verified in TAMARIN. To prove equivalence between a time-based model and a causality-based model, we use proving techniques similar to the one used in [10]–[12] to analyse distance-hijacking resistance. We generalise these approaches under the assumption of a distant attacker by supporting other security properties, such as secrecy, agreement and memory erasure.

¹<http://people.irisa.fr/Alexandre.Debant/akiss-db.html>

Improvements and extensions of the above verification frameworks have followed. For example, Boureau et al. [13] introduce a model that supports moving agents and use it to analyse complex payment protocols. Alturki et al. [19] propose a timed multiset rewriting model with memory bounding and timeouts features. Their analyses are of theoretical interest, but their model lacks tool support.

Outside the domain of distance-bounding protocols, there exist time-aware security models [20]–[22] that support the verification of standard authentication properties, as we do in this article. Their specification language is richer than ours, allowing for statements on timeouts and the ordering of protocol events based on timestamps. Their network model, however, neither captures the location of agents nor the propagation time of a message via the network in terms of the distance between sender and receiver. Instead, they model the network communication delay as a range within a discrete space, making these models unsuitable to formalise the notion of distant attackers.

Lastly, the notion of distant attacker was introduced in [7], from which we borrow the terminology, within the context of memory-erasure protocols. Their security model, however, is neither amenable for computer-aided verification nor extendable to other security properties.

Up to our knowledge, no symbolic model with tool support has been proposed to analyse protocols in which standard properties such as secrecy and authentication can be analysed under the distant-attacker assumption. We introduce the first such model and use it to verify key exchange and memory-erasure protocols.

III. A SECURITY MODEL FOR TIMED SECURITY PROPERTIES

This section introduces a security model for protocols with round-trip-time restrictions. Like previous models for distance-bounding protocols [8], [18], our model uses a timed communication channel where protocol participants are provided with a spatial location, and the arrival time of messages is consistent with the propagation speed of the communication channel and the distance between sender and receiver. Our model, however, is used to analyse general properties of security protocols. Hence we provide the model with a simple protocol specification language (à la Cremers and Mauw [23]) and operational semantics that allows us to prove general properties of protocols with round-trip-time measurements in relation to their security goals. We note that although security models including a notion of time do exist, they are either too expressive (e.g. [8]), making it a laborious task to prove general properties of protocols, or too specific (e.g. [9]) and constrained to the analysis of distance-bounding protocols.

A. Messages

A security protocol defines the way various protocol participants, called *agents*, exchange cryptographic messages. To model them, we use an order-sorted term algebra $(\mathcal{S}, \leq, \mathcal{T}_\Sigma(\mathcal{V}, \mathcal{C}))$ where Σ is a signature, \mathcal{V} a set of variables, \mathcal{C}

a set of constants, and (\mathcal{S}, \leq) a poset of sorts [24]. We will often use $\mathcal{T}_\Sigma(\mathcal{V}, \mathcal{C})$ to refer to our sorted term algebra when the sorts are clear from context. All terms have a sort. In particular, the sorts *agent* and *nonce* are reserved for agent names and nonces. We also define the sort *msg* (short for message) to be the superset or the greatest element of the poset \mathcal{S} , i.e. $s \leq \text{msg}$ for all $s \in \mathcal{S}$. We use $t : s$ to denote that term t is of sort s .

Let $\text{Agent} = \{a \in \mathcal{C} \mid a : \text{agent}\}$ be the set of agent names and $\text{Nonce} = \{n \in \mathcal{C} \mid n : \text{nonce}\}$ be the set of nonces. Given an agent a , we use Nonce_a to denote the set of nonces that agent a can produce. We restrict agents to produce unique nonces, hence we require $\forall a, b \in \text{Agent} : a \neq b \implies \text{Nonce}_a \cap \text{Nonce}_b = \emptyset$. The set Agent is partitioned into *Honest* (honest agents) and *Dishonest* (dishonest agents). Finally, we assume that the signature Σ contains the following function symbols:

- $\text{pair}(m, m')$ denoting the pairing of two terms m and m' . We will often use $\langle m, m' \rangle$ as shorthand notation, and write $\langle m_1, \dots, m_n \rangle$ instead of $\langle m_1, \langle m_2, \dots, m_n \rangle \rangle$. The functions fst and snd allow us to recover the first and second element of a pair, respectively.
- $\text{xor}(m, m')$, often written as $m \oplus m'$, denoting the *exclusive or* of the terms m and m' . The constant zero represents the null element with respect to xor .
- $\text{k}(a, b)$ denoting the long-term symmetric key shared by agents a and b .
- $\text{sk}(a)$ denoting the long-term secret key of an agent a .
- $\text{pk}(m)$ denoting the public key associated to a term m . If it does not lead to confusion, we shall write $\text{pk}(a)$ to denote $\text{pk}(\text{sk}(a))$ when $a \in \text{Agent}$.
- $\text{aenc}(m_1, \text{pk}(k))$ and $\text{adec}(m_2, k)$ denoting, respectively, the asymmetric encryption of m_1 with the public key $\text{pk}(k)$, and the asymmetric decryption of m_2 with the secret key k .
- $\text{senc}(m_1, k_1)$ and $\text{sdec}(m_2, k_2)$ denoting, respectively, the symmetric encryption of m_1 with the key k_1 , and the symmetric decryption of m_2 with the key k_2 .
- $\text{sign}(m, k')$ denoting the signature of m with the secret key k' . The function verify and the constant true are used to verify whether a signature is correct.
- $\text{h}(m)$ denoting the hash of the term m .
- $x \cdot y$ and x^y denote, respectively, the multiplication and exponentiation of x and y , in a Diffie-Hellman group. The function inv and the constant 1 denote, respectively, the inverse function w.r.t. multiplication and the unit element.

We often write $\{X\}_Y$ to denote (a)symmetric encryption of X with (public) key Y . Further, we assume that the sort of all composed terms is the super sort *msg*.

The semantics of the function symbols above is formalised by an equational theory E that models perfect cryptography, such as the one supported by TAMARIN and ProVerif. We use the symbol $=_E$ to denote equality of two terms modulo E .

Terms with variables will be used in our model to specify the behaviour of protocol participants (roles), in such a way that their behaviour can be *instantiated* multiple times by means of variable substitution. Formally, let

$vars: \mathcal{T}_\Sigma(\mathcal{V}, \mathcal{C}) \rightarrow \mathbb{P}(\mathcal{V})$, where $\mathbb{P}(\cdot)$ denotes the power set, be an auxiliary function that, given a term t , gives all variables occurring in t . A term $t \in \mathcal{T}_\Sigma(\mathcal{V}, \mathcal{C})$ is called *ground* iff $vars(t) = \emptyset$. We use $\mathcal{T}_\Sigma(\mathcal{C})$ to denote the set of ground terms over the term algebra. A *substitution* is a function $\sigma: \mathcal{V} \rightarrow \mathcal{T}_\Sigma(\mathcal{V}, \mathcal{C})$ from variables to terms such that $\sigma(v) \neq v$ for finitely many variables. An *instantiation* of a term t via a substitution σ , denoted $t\sigma$, is inductively defined by

$$t\sigma = \begin{cases} t & \text{if } t \in \mathcal{C} \\ \sigma(t) & \text{if } t \in \mathcal{V} \\ f(\sigma(t_1), \dots, \sigma(t_n)) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

We say that a substitution σ is *type-preserving* if for every variable $v \in \mathcal{V}$ it holds that $v: s \wedge \sigma(v): s' \implies s' \leq s$. This means, for example, that a variable of type `msg` can be substituted by a term of type `nonce`, but not the other way around. In our model, we consider type-preserving substitution only, and we use Γ to denote the universe of such substitutions.

B. Protocol specification

We partition a protocol into roles. A role is composed of events it uses to communicate with other roles, security claims, time measurements, etc. An *event* is a term of the form $E_a(t)$, where E is a symbol from an unsorted signature \mathcal{E} , and t and a are terms in $\mathcal{T}_\Sigma(\mathcal{V}, \mathcal{C})$ with a : `agent`. The application of a substitution σ to an event $E_a(t)$, denoted $E_a(t)\sigma$, results in the event $E_{a\sigma}(t\sigma)$. The set of all events is denoted Ev and the set $\text{Ev}_g \subseteq \text{Ev}$ denotes the set of *ground events*, which are events with only ground terms as arguments. The function $actor(\cdot)$, defined by $actor(E_a(t)) = a$, provides the actor executing an event.

We reserve the event symbols `send`, `recv`, `claim`, `clock`, and `equal`. The events $\text{send}_a(m)$ and $\text{recv}_a(m)$ denote the sending and reception, respectively, of a message m . For the remaining reserved events we impose the following syntactical restrictions: clock events have the form $\text{clock}_a(i, j)$, where i and j are integers representing the start and end of a timer. This timer starts with the execution of the i th event of the role, and stops at the execution of its j th event. Claim events have the form $\text{claim}_a(\psi, t)$, where ψ is a constant denoting a security property name and t is an argument of the property, such as an agent's name or a nonce; equality events have the form $\text{equal}_a(\langle m_1, m_2 \rangle)$ denoting the expectation that $m_1 =_E m_2$. The impact of these event types in the behaviour of a protocol will be made precise soon.

As in [23], we consider a role specification R to be a sequence of events $r_1 \cdots r_n$ establishing a total order on the execution of the role events. We require every role specification with sequence of events $r_1 \cdots r_n$ to satisfy that $actor(r_1) = \dots = actor(r_n)$, i.e. events within a role specification are executed by the same agent. We also require $r_i = \text{clock}_a(x, y) \implies x \leq y < i$ for every $i \in \{1, \dots, n\}$; i.e. a time measurement is built upon preceding events only, and the event at which the clock stops does not precede the event at which the clock starts.

A role is a mapping from role names, such as *server* and *client*, to role specifications. We use $R = r_1 \cdots r_n$ to denote the role with name R and specification $r_1 \cdots r_n$, and we use R to denote the universe of roles.

Definition 1 (Protocol specification). *A protocol \mathcal{P} consists of a set of roles, such that no two roles share the same role name, built over an order-sorted term algebra $(\mathcal{S}, \leq, \mathcal{T}_\Sigma(\mathcal{V}, \mathcal{C}))$.*

At the semantical level, we will treat all variables within a role as local variables. This ensures that agents can only communicate by messaging each other. Given a role $R = r_1 \cdots r_n$, we use $rolevars(R)$ to obtain all role variables in R , which is defined as follows.

$$rolevars(R) = \{v \in \mathcal{V} \mid \exists i \in \{1, \dots, n\}: r_i = \text{send}_a(m) \wedge v \in vars(m) \wedge \forall j \in \{1, \dots, i-1\}: v \notin vars(r_j)\}$$

As naming convention for variables, we will use upper case letters when a variable is intended to be instantiated within a receive event, such as K and N_p in the example below, lower case letters otherwise, such as n_v and k_{vp} .

Example 1. *To illustrate our security model, we formalise the running example depicted in Figure 1. The specification within our security model of the prover and verifier roles, denoted P and V , respectively, is given below. It assumes that n_v , n_p , k_{vp} , and k are variables of type `nonce`, while the variables V and P are of type `agent`. All the other variables, namely N_v , N_p , K_{vp} , S , PK , are of type `msg`.*

$$\begin{aligned} \mathcal{P}_{ex} = \{ \\ V &= \text{recv}_v(\langle PK, S \rangle) \cdot \text{send}_v(n_v) \cdot \text{recv}_v(n_v \oplus N_p) \cdot \\ &\quad \text{clock}_v(2, 3) \cdot \text{send}_v(\{k_{vp}\}_{PK}) \cdot \text{recv}_v(\{N_p, P\}_{k_{vp}}) \cdot \\ &\quad \text{equal}_v(\langle \text{verify}(S, P, PK), \text{true} \rangle) \cdot \text{claim}_v(\text{sec}, \langle P, k_{vp} \rangle) \\ P &= \text{send}_p(\langle \text{pk}(k), \text{sign}(P, k) \rangle) \cdot \text{recv}_p(N_v) \cdot \text{send}_p(N_v \oplus n_p) \cdot \\ &\quad \text{recv}_p(\{K_{vp}\}_{\text{pk}(k)}) \cdot \text{send}_p(\{n_p, P\}_{K_{vp}}) \} \end{aligned}$$

The Δ symbol labelling a dashed arrow that connects the 2nd and 3rd protocol message in Figure 1 represents the round-trip-time measurement of the verifier, and is translated into the event $\text{clock}_v(2, 3)$ in the role specification.

C. Role instantiation.

Protocols are executed by instantiating their roles. Syntactically, the instantiation of a role results in a sequence of ground events that respect the order of the events established by the role specification. Given a role $R = r_1 \cdots r_n$, we define all its instantiations, denoted $insts(R)$, as follows.

$$\begin{aligned} insts(R) = \{e_1 \cdots e_i \in \text{Ev}_g^* \mid i \leq n \wedge \\ \exists \sigma \in \Gamma: e_1 = r_1\sigma, \dots, e_i = r_i\sigma\} \end{aligned}$$

Note that the empty sequence of events ϵ is a valid instantiation of all role specifications.

In the operational semantics provided further below, we restrict role variables of type `nonce` to be assigned a *fresh* value, i.e. a term that has not been used in other role instantiations during the protocol execution. Therefore, we introduce

$$\begin{array}{c}
\frac{a \vdash_s m \quad m =_E m'}{a \vdash_s m'} \text{I0} \qquad \frac{m \in \text{Agent} \cup \text{Nonce}_a}{a \vdash_s m} \text{I1} \\
\frac{b \in \text{Agent}}{a \vdash_s \langle \mathbf{k}(a, b), \mathbf{k}(b, a), \mathbf{sk}(a), \mathbf{pk}(\mathbf{sk}(b)) \rangle} \text{I2} \\
\frac{a \vdash_s m_1 \quad \dots \quad a \vdash_s m_n \quad f \in \Sigma \setminus \{\mathbf{sk}, \mathbf{k}\}}{a \vdash_s f(m_1, \dots, m_n)} \text{I3} \\
\frac{(t, \text{recv}_a(m)) \in s}{a \vdash_s m} \text{I4}
\end{array}$$

Fig. 2: Inference rules

an auxiliary function to extract the set of nonces used in an instantiation, for every $e_1 \dots e_i \in \text{insts}(R)$,

$$\begin{aligned}
\text{nonces}(e_1 \dots e_i, R) &= \{t \in \text{Nonce} \mid \exists v \in \text{rolevars}(R), \sigma \in \Gamma: \\
&\quad \sigma(v) = t \wedge v : \text{nonce} \wedge e_1 = r_1 \sigma, \dots, e_i = r_i \sigma\}
\end{aligned}$$

D. Inference

Before establishing how agents exchange messages, we need to define how agents obtain and create knowledge. We model this by means of an inference relation $\vdash \subseteq \text{Agent} \times \mathbb{P}(\text{Ev}) \times \text{Msg}$. we use the shorthand notation $a \vdash_s m$ to denote $(a, s, m) \in \vdash$, indicating that agent a can infer message m from a set of events s . The relation \vdash is defined as the least set that is closed under the inference rules in Figure 2. These rules express the following: (I0) If an agent a can infer a term m , then a can infer all terms within the equivalence class of m defined by $=_E$. (I1) an agent a can infer agent names and its own set of nonces; (I2) agents can infer their shared secret keys with other agents, long term public keys of any agent and its own long term secret key; (I3) all function symbols in Σ , except the reserved symbols for secret keys \mathbf{k} and \mathbf{sk} , can be used to infer arbitrary terms constructed over already inferable terms; (I4) a receive event $\text{recv}_a(m)$ allows agent a to infer m .

E. Protocol semantics

We model the execution of a protocol \mathcal{P} as a Labelled Transition System (LTS) $l = (Q, \Lambda, \rightarrow, s_0)$, where Q is a set of states, Λ is a set of labels used to annotate the transition of the system from one state to another, $\rightarrow: Q \times \Lambda \times Q$ is a transition relation, and $s_0 \in Q$ is the initial state. We will often use $s \xrightarrow{\ell} s'$ as a shorthand notation for $(s, \ell, s') \in \rightarrow$.

States. A state in the system is composed of a set of *runs*, where a run is an instantiation of a role by an agent. A run contains a possibly partial execution of a role and a run identifier. The latter allows agents to play multiple roles in parallel. A run also includes the time-stamps at which the events are executed. This allows us to reason about time properties, such as consistency between the time-of-flight of a transmitted message with respect to the speed of

the communication channel and the location of sender and receiver.

Definition 2 (Run). Let Id be a countably infinite set of run identifiers. A run is any tuple $(\text{id}, (t_1, e_1) \dots (t_n, e_n), R, a) \in \text{Id} \times (\mathbb{R}^+ \times \text{Ev}_{\mathbf{g}})^* \times \text{R} \times \text{Agent}$ satisfying that $e_1 \dots e_n \in \text{insts}(R)$ is an instantiation of the role R , a is the actor executing the events e_1, \dots, e_n , i.e. $a = \text{actor}(e_1) = \dots = \text{actor}(e_n)$, and $t_1 \leq t_2 \leq \dots \leq t_n$.

A run is an execution of a role by an actor, containing instantiations of events in the order imposed by the role specification, and timestamps indicating the time at which each event was instantiated. The order of the timestamps \leq is consistent with the order of the events in the instantiated role. An *empty run* contains no events.

A *state* in our LTS is a set of runs. The *initial state* s_0 is the empty set. Given a state s , the functions $\text{labels}(s)$ and $\text{nonces}(s)$ return, respectively, all timed events and fresh values occurring in s .

$$\begin{aligned}
\text{labels}(s) &= \{(t, e) \mid \exists (\text{id}, (t_1, e_1) \dots (t_n, e_n), R, a) \in s: \\
&\quad (t, e) = (t_i, e_i) \text{ for some } i \in \{1, \dots, n\}\} \\
\text{nonces}(s) &= \bigcup_{(\text{id}, (t_1, e_1) \dots (t_n, e_n), R, a) \in s} \text{nonces}(e_1 \dots e_n, R)
\end{aligned}$$

Given a state s , a role R , and a substitution σ mapping the role R to its instantiation $e_1 \dots e_n$, we use $e_1 \dots e_n \in_s \text{insts}(R)$ to denote that the instantiation $e_1 \dots e_n$ of R satisfies: (1) $\text{nonces}(e_1 \dots e_i, R) \cap \text{nonces}(s) = \emptyset$ and (2) for every $x, y \in \text{rolevars}(R)$ of type *nonce*, $x \neq y$ implies $\sigma(x) \neq \sigma(y)$. That is, no instantiation in s exists sharing fresh values with the instantiation $e_1 \dots e_n$ of R , and the role variables are instantiated with pairwise distinct nonces.

Labels and execution traces. A transition in our LTS will either add a new empty run to the current state or add a timed event to an existing run. Each transition is labelled with a timestamp, a description of the state update and the id of the run modified in that transition (this will become clear later). We denote the creation of a new run by $\text{create}_a(R)$, where $a \in \text{Agent}$ is an agent and $R \in \text{R}$ is a role. The addition of a protocol event will be labelled by using the events themselves as labels. Therefore, an *execution* of the protocol is an interleaved sequence of states and labels of the type $s_0 \xrightarrow{(t_1, \ell_1)^{\text{id}_1}} s_1 \dots s_{n-1} \xrightarrow{(t_n, \ell_n)^{\text{id}_n}} s_n$, where s_0, \dots, s_n are states, t_1, \dots, t_n timestamps, and ℓ_i is either a protocol event or a label of the type $\text{create}_a(R)$, for $i \in \{1, \dots, n\}$. A *trace* is the resulting sequence of LTS labels $\tau = (t_1, \ell_1)^{\text{id}_1} \dots (t_n, \ell_n)^{\text{id}_n}$. In this case, we say that τ has cardinality n , denoted $|\tau|$, and we use τ_i to denote the i th element of τ , i.e. $\tau_i = (t_i, \ell_i)^{\text{id}_i}$. When $n = 0$, we use ϵ to denote the empty trace, while the initial state s_0 alone represents the empty execution. Lastly, we will omit the run ids from traces when they are not necessary.

We write $a \vdash_\tau m$ to denote $a \vdash_s m$ where s is the set of events occurring in τ .

Transition relation. The transition relation of the protocol LTS is defined by a set of derivation rules, similar to the inference

rules above. The premise of a rule is a formula with no free variables. Its conclusion is a transition of the form $s \xrightarrow{(t,\ell)^{td}} s'$. Unless otherwise specified, variables in our derivation rules are universally quantified. For run identifiers, we use $id \in_s \text{Id}$ to denote that id is chosen from the set Id in such a way that id has not been used in the state s . The function $\text{max_time}: Q \rightarrow \mathbb{R}^+$ gives the maximum timestamp used by an event in a given state. If $\text{labels}(s) = \emptyset$, then $\text{max_time}(s) = 0$, otherwise $\text{max_time}(s) = \max\{t \mid (t, e) \in \text{labels}(s)\}$.

Our first rule is one that adds an empty run to a state by instantiating either a protocol role or an adversarial role.

$$\frac{s \in Q, t \geq \text{max_time}(s), R \in \mathbf{R}, a \in \text{Agent}, \quad id \in_s \text{Id}, a \in \text{Honest} \implies R \in \text{roles}(\mathcal{P})}{s \xrightarrow{(t, \text{create}_a(\mathcal{P}(R)))^{id}} s \cup \{(id, \epsilon, R, a)\}} \text{Create}_{\mathcal{P}}$$

Here \mathcal{P} denotes the protocol specification whose behaviour is being modelled. This rule ensures that honest agents instantiate a protocol role, i.e. that honest agents do not deviate from the protocol specification. Dishonest agents, on the other hand, can influence the protocol execution by instantiating arbitrary role specifications, which we refer to as an adversarial role.

The remaining transition rules express how a run of the system state can make progress by executing a single event. There are three of these rules, which can all be defined as an extension of the following rule template:

$$\frac{s \in Q, t \geq \text{max_time}(s), (id, \tau, R, a) \in s, \quad \tau = (t_1, e_1) \cdots (t_i, e_i), \quad e_1 \cdots e_i e_{i+1} \in_s \setminus (id, \tau, R, a) \text{ insts}(R), \quad \boxed{}}{s \xrightarrow{(t, e_{i+1})^{id}} (s \setminus \{(id, \tau, R, a)\}) \cup \{(id, \tau \cdot (t, e_{i+1}), R, a)\}}$$

where the bracketed part $\boxed{}$ is a placeholder to add premises. This rule template triggers the execution of an event e_{i+1} at time t if i) t is greater than or equal to the largest timestamp in s , and ii) there exists a run $(id, (t_1, e_1) \cdots (t_i, e_i), R, a)$ in the state satisfying that $e_1 \cdots e_{i+1}$ is an instantiation of R . The condition $e_1 \cdots e_i e_{i+1} \in_s \setminus (id, \tau, R, a) \text{ insts}(R)$ ensures that nonces assigned to role variables in other runs, i.e. in s minus (id, τ, R, a) , are not used in the current run.

We use $\rightarrow [p_1, \dots, p_n]$ to denote the rule obtained from the above rule template by replacing the placeholder $\boxed{}$ by the set of premises $\{p_1, \dots, p_n\}$. Using this notation, we define the remaining rules for our LTS in the following.

The Send rule:

$$\rightarrow [e_{i+1} = \text{send}_a(m), a \vdash_{\{e_1, \dots, e_i\}} m] \quad \text{Send}$$

allows agents to send messages to the network. The Send rule restricts both honest and dishonest agents to send messages whose content is inferable from their initial knowledge, constants, and the sequence of events already executed in the run. This is expressed by the premise $a \vdash_{\{e_1, \dots, e_i\}} m$, and means that our model does not consider an omnipresent adversary overseeing all events sent to the network. Instead, our model forces dishonest agents to collaborate by messaging each other.

The Recv_d rule:

$$\rightarrow \left[\begin{array}{l} e_{i+1} = \text{recv}_a(m), (t', \text{send}_b(m')) \in \text{labels}(s), \\ d(a, b) \leq c(t - t'), m =_E m' \end{array} \right] \quad \text{Recv}_d$$

models how agents receive messages from other agents, enforcing that the time of flight of a message exchange is consistent with the distance between sender and receiver. The rule is parameterised on a distance metric $d: \text{Agent} \times \text{Agent} \rightarrow \mathbb{R}^+$, and assumes a constant propagation speed c of the communication channel.

The rule Recv_d triggers the execution of a receive event $\text{recv}_a(m)$ at time t if $e_{i+1} = \text{recv}_a(m)$ and there exists a timed send event $(t', \text{send}_b(m'))$, where $m =_E m'$, in some run in the state such that the distance between the sender and receiver is smaller than or equal to $c(t - t')$.

It is worth pointing out that the Recv_d rule does not consider that messages may fade away as they travel, implying that secrets revealed to nearby agents leak to the entire network. This is a deliberate choice made with the goal of making no assumptions about signal strength, nor about the distance at which a message can be eavesdropped. For example, RFID eavesdropping on messages at a range of 20m or more has proven feasible, depending on the power of the devices [25].

The Equal rule

$$\rightarrow \left[\begin{array}{l} e_{i+1} = \text{equal}_a(\langle m_1, m_2 \rangle), m_1 =_E m_2, \\ a \vdash_{\{e_1, \dots, e_i\}} m_1, a \vdash_{\{e_1, \dots, e_i\}} m_2 \end{array} \right] \quad \text{Equal}$$

states that an event $\text{equal}_a(\langle m_1, m_2 \rangle)$ only executes when m_1 and m_2 are equal modulo the equational theory E . This type of event is used, e.g., to model the verification of signatures.

The Signal rule

$$\rightarrow [e_{i+1} \in \text{SignalEvent}] \quad \text{Signal}$$

models the execution of *signal* events. Signal events are useful for instrumenting security properties, which often rely on expectations announced by agents by means of signal events, such as claim events. Formally, the set of signal events is defined by $\text{SignalEvent} = \{e \in \text{Ev}_g \mid \nexists a, m: e \in \{\text{send}_a(m), \text{recv}_a(m), \text{equal}_a(m)\}\}$.

Definition 3 (Protocol semantics). *The semantics of a protocol \mathcal{P} w.r.t. a distance function d is the LTS $(Q, \Lambda, \rightarrow, s_0)$ where,*

- $Q = \mathbb{P}(\text{Id} \times (\mathbb{R}^+ \times \text{Ev}_g)^* \times \mathbf{R} \times \text{Agent})$
- $\Lambda = \mathbb{R}^+ \times (\text{Ev}_g \cup \{\text{create}_a(R) \mid a \in \text{Agent}, R \in \mathbf{R}\}) \times \text{Id}$
- $\rightarrow = \text{Create}_{\mathcal{P}} \cup \text{Send} \cup \text{Recv}_d \cup \text{Equal} \cup \text{Signal}$.
- $s_0 = \emptyset$

We use $\llbracket \mathcal{P} \rrbracket_d$ to denote the set of traces obtained from \mathcal{P} 's semantics with respect to the distance function d .

Example 2. *Let E an adversarial role specification with all role variables of type nonce:*

$$E = \text{send}_e(\langle \text{pk}(k_e), \text{sign}(P, k_e) \rangle) \cdot \text{recv}_e(N_v) \cdot \text{send}_e(N_v \oplus n_e) \cdot \text{recv}_e(\{K_{vp}\}_{\text{pk}(k_e)}) \cdot \text{send}_e(\{(n_e, P)\}_{K_{vp}})$$

Assuming that the pairwise distance between the agents a , b and c is $100c$, and that $s = \text{sign}(b, k')$, an execution trace of our running example protocol \mathcal{P}_{ex} is the following:

$$\begin{aligned}
s_0 &\xrightarrow{(0, \text{create}_a(id1, \mathcal{P}_{ex}(V)))} s_1 \xrightarrow{(0, \text{create}_b(id2, \mathcal{P}_{ex}(P)))} s_2 \\
&\xrightarrow{(0, \text{create}_e(id3, E))} s_3 \xrightarrow{(0, \text{send}_e(\langle \text{pk}(k'), s \rangle))} s_4 \\
&\xrightarrow{(100, \text{recv}_a(\langle \text{pk}(k'), s \rangle))} s_5 \xrightarrow{(100, \text{send}_a(n'))} s_6 \xrightarrow{(200, \text{recv}_e(n'))} s_7 \\
&\xrightarrow{(200, \text{send}_e(n' \oplus n))} s_8 \xrightarrow{(300, \text{recv}_a(n' \oplus n))} s_9 \xrightarrow{(300, \text{clock}_a(2, 3))} s_{10} \\
&\xrightarrow{(300, \text{send}_a(\{k''\}_{\text{pk}(k')})} s_{11} \xrightarrow{(400, \text{recv}_e(\{k''\}_{\text{pk}(k')})} s_{12} \\
&\xrightarrow{(400, \text{send}_e(\{n, b\}_{k''}))} s_{13} \xrightarrow{(500, \text{recv}_a(\{n, b\}_{k''}))} s_{14} \\
&\xrightarrow{(500, \text{equal}_a(\langle \text{verify}(s, b, \text{pk}(k')), \text{true} \rangle))} s_{15} \xrightarrow{(500, \text{claim}_a(\text{sec}, \langle b, k'' \rangle))} s_{16}
\end{aligned}$$

F. Security properties, claims, and protocol correctness

We define a security property ψ as a predicate on traces and integers such that $\psi(\tau, i)$ means that ψ is satisfied at step i of the trace τ . For illustration purposes, we define next the secrecy property used in our running example, whereby an agent expects the adversary to not know a given term.

$$\begin{aligned}
\text{sec}(\langle t_1, e_1 \rangle, \dots, \langle t_n, e_n \rangle, i) &\iff \\
e_i = \text{claim}_a(\text{sec}, \langle b, m \rangle) \wedge b \in \text{Honest} &\implies \\
\exists c \in \text{Dishonest}: c \vdash_{\{e_1, \dots, e_n\}} m &
\end{aligned}$$

As in this example, we consider properties instrumented by claim events. A security claim denotes a belief about the protocol execution that led to the claim, e.g. $\text{claim}_a(\text{sec}, \langle b, m \rangle)$ denotes a 's belief that as long as its communicating partner is honest, no adversary knows the secret term m . Note the slight abuse of notation in the definition of secrecy: sec is used as a claim identifier and a predicate symbol. We shall keep this convention from now on to make the connection between claims and their intended meaning explicit. That is, for every claim event $\text{claim}_a(\psi, m)$, we consider ψ to be the predicate giving ψ its meaning. Hence the following definition of protocol correctness with respect to a security claim follows.

Definition 4 (Claim correctness). *A claim event $\text{claim}_a(\psi, m)$ is said to be correct in a protocol \mathcal{P} , denoted $\mathcal{P} \triangleright \psi$, if for every distance function d , trace $\tau \in \llbracket \mathcal{P} \rrbracket_d$, and index $i \in \{1, \dots, |\tau|\}$, $\psi(\tau, i)$ holds.*

The next section is dedicated to extending the definition of claim correctness with time restrictions and assumptions about the honesty of agents in relation to their distance to another agent, making it possible for \mathcal{P}_{ex} , and several other modern protocols, to formalise their security goals.

IV. MODELLING SECURITY REQUIREMENTS BASED ON THE DISTANT-ATTACKER ASSUMPTION

The formal model introduced in the previous section can be used to specify the structure and behaviour of a security protocol with time measurements. This section introduces a new class of security requirements that captures the intended security goals of a relatively recent wave of protocols based

on proximity [1], [3], [4], [6], [7], [26], [27]. Such protocols aim at classical security properties, such as secrecy and authentication, but rely on a particular trust assumption that has not been captured within a formal security model. We are referring to trust assumptions that are based on the (dis)honesty of agents in the neighbourhood of another agent, usually a verifier. We call this assumption the *distant-attacker assumption*, as it considers the neighbourhood of a verifier to be free of attackers.

The goal of this section is to formalise the distant-attacker assumption and instrument it within classical security properties. The result is a class of security requirements expressed as a premise-conclusion formula, where the premise is a proximity check in conjunction with a distant-attacker claim, and the conclusion is a standard trace-based property. In the subsequent sections, we show such a class of security requirements to be sufficient for the verification of a large number of security protocols based on proximity which currently lack formal correctness proofs.

A. The distant-attacker assumption

The security goals of many modern protocols are contingent upon the assumption that no attacker is in the vicinity of the verifier. When formalising this assumption, we shall allow the verifier's communicating partner (the prover) to be both malicious and close, as it allows us to model memory attestation and erasure properties [7] for free. Hence, our task next is to formalise the following statement - *agents making a security claim are aware of what attackers (if any) are in their vicinity.*

We define the vicinity of an agent as the locus of a circle with radius δ centred in the agent's location. We also use the auxiliary function $\text{dishonest_agents}(\text{claim}_a(\psi, m))$ which gives the set of dishonest agents that a allows to be close when claiming the property ψ . For the case of secrecy, for example, it follows that $\text{dishonest_agents}(\text{claim}_a(\text{sec}, \langle b, m \rangle)) = \emptyset$ for every b and m . This is, indeed, the case for authentication properties, as the verifier does not expect to authenticate a corrupt prover. In remote memory erasure and attestation protocols, however, the prover is considered dishonest, implying that the function $\text{dishonest_agents}(\cdot)$ should return the prover agent for erasure and attestation claims. This leads to the following formalisation of the distant-attacker assumption as a predicate with domain $\mathcal{T} \times \mathbb{Z}^+$, where \mathcal{T} is the universe of traces. For every $\tau = (t_1, e_1) \cdots (t_n, e_n)$, distance threshold δ and index i ,

$$\begin{aligned}
\text{dist_attacker}_\delta(\tau, i) &\iff e_i = \text{claim}_a(\psi, m) \wedge \\
\forall c \in (\text{actors}(\tau) \cap \text{Dishonest}): d(a, c) > \delta \vee \\
c \in \text{dishonest_agents}(e_i) &
\end{aligned}$$

The distant-attacker assumption holds for a claim $e_i = \text{claim}_a(\psi, m)$ in a trace $\tau = (t_1, e_1) \cdots (t_n, e_n)$, if all dishonest agents in the trace are either far from a , i.e. at a distance larger than δ , or are part of a list of expected dishonest agents $\text{dishonest_agents}(e_i)$.

B. Round-trip-time restrictions

Intuitively, for a protocol to use the distant-attacker assumption effectively, it needs to provide agents with the ability to measure distance to other agents. Our security model allows protocols to accomplish this by means of clock events, syntactically establishing the calculation of the time difference between two events. Semantically, time measurements are local to protocol runs. That is, any two events involved in the calculation of a time measurement should be part of the same protocol session or run. We thus need a mechanism to extract runs from traces, which we obtain by exploiting the run identifiers present in protocol traces. Given a trace $\tau = (t_1, e_1)^{id_1} \dots (t_n, e_n)^{id_n}$ and run identifier id , we use $run(\tau, id)$ to denote the subtrace $(t_{i_1}, e_{i_1})^{id} \dots (t_{i_k}, e_{i_k})^{id}$ of maximum cardinality, i.e. $run(\tau, id)$ denotes the full run in τ with run identifier id .

Here we formalise an interpretation of time measurements in relation to a security claim and the distant-attacker assumption, leading to a definition of correctness that we show is applicable to a large class of protocols. Clock events are used to measure the round-trip-time of a message exchange with a communicating partner, with the expectation that the communicating partner is within a δ radius, for some distance parameter δ . If a time measurement is below $2\delta/c$ where c is the speed of the communication medium, then we say that such measurement is correct with respect to the distance bound δ . If all clock events in a given protocol run are correct, then we say that such a run has correct time measurements. Formally, the correctness of the time measurements of the run $run(\tau, id_i) = (t_{i_1}, e_{i_1}) \dots (t_{i_k}, e_{i_k})$ in the trace $\tau = (t_1, e_1)^{id_1} \dots (t_n, e_n)^{id_n}$, where e_i is a claim event by agent a is defined by:

$$\begin{aligned} \text{correct_time}_\delta(\tau, i) &\iff \forall j \in \{i_1, \dots, i_k\}: \\ (e_j = \text{clock}_a(x, y) \wedge j < i) &\implies t_{i_y} - t_{i_x} < 2 \cdot \delta / c \end{aligned}$$

Note that in the definition of $\text{correct_time}_\delta(\tau, i)$ we only consider clock events that precede the event e_i . The predicate ensures that for every possible run of the protocol resulting in the trace τ , the time measurements performed by the agent that produced the event e_i are all below the threshold $2 \cdot \delta / c$.

Now we are ready to define a class of security requirements that extend classical security properties by making them conditional to round-trip-time restrictions and the distant-attacker assumption.

Definition 5 (Claim correctness under the distant-attacker assumption). *Let \mathcal{P} be a protocol and δ a distance value. A claim event $\text{claim}_a(\psi, m)$ is said to be correct in \mathcal{P} under the distant-attacker assumption, denoted $\mathcal{P} \triangleright_\delta \psi$, if for every distance function d , trace $\tau \in \llbracket \mathcal{P} \rrbracket_d$, and index $i \in \{1, \dots, |\tau|\}$,*

$$\text{correct_time}_\delta(\tau, i) \wedge \text{dist_attacker}_\delta(\tau, i) \implies \psi(\tau, i)$$

As a security requirement, correctness with respect to \triangleright_δ is weaker than correctness with respect to \triangleright (see Definition 4), i.e. for every protocol \mathcal{P} , every claim $\text{claim}_a(\psi, m)$, and every

$\delta > 0$, it follows that $\mathcal{P} \triangleright \psi \implies \mathcal{P} \triangleright_\delta \psi$. That said, we argue that the relation \triangleright_δ better captures the intended goal of many other modern protocols, including the running example used in this article. Therefore, we dedicate the remainder of this article to introducing a verification framework for (dis)proving the correctness of protocols with respect to \triangleright_δ .

V. A CAUSALITY-BASED INTERPRETATION OF LOCALITY

A security model that explicitly carries the notion of time and location, as introduced in the previous section, is useful to reach consensus on the formal definition of the distant-attacker assumption and how to instrument it as a security requirement. However, it defies computer-aided reasoning since modern protocol verification tools, such as Tamarin and ProVerif, do not cope well with temporal and spatial information.

Our goal in this section is to provide a timeless protocol semantics, denoted $\llbracket \cdot \rrbracket_\pi$, and a correctness relation of protocols with respect to a security property ψ , denoted $\llbracket \mathcal{P} \rrbracket_\pi \triangleright_\sim \psi$ where \sim is an equivalence relation on the set of agents that encodes proximity, such that $\forall d: \llbracket \mathcal{P} \rrbracket_d \triangleright_\delta \psi \iff \forall \sim: \llbracket \mathcal{P} \rrbracket_\pi \triangleright_\sim \psi$ is valid. This will allow us to analyse the property ψ on the timeless semantics by considering \triangleright_\sim to be the security goal, rather than \triangleright_δ .

Due to space restrictions, we have moved most proofs to the Appendix B.

A. Properties of the timed semantics

We start by proving properties of the time-based semantics introduced in Section III.

The relation defined in the following definition is essential for determining when a trace can be generated by our timed semantics, as it represents a precedence relation between events in the trace.

Definition 6. *Given a trace $\tau = (t_1, e_1) \dots (t_n, e_n) \in \llbracket \mathcal{P} \rrbracket_d$, let \rightsquigarrow_τ be the relation defined as follows:*

$(t_i, e_i) \rightsquigarrow_\tau (t_j, e_j)$ iff $i < j$ and either:

- *actor(e_i) = actor(e_j) or*
- *$e_i = \text{send}_a(m_i) \wedge e_j = \text{recv}_b(m_j) \wedge a \neq b \wedge m_i =_E m_j \wedge d(a, b) \leq (t_j - t_i) \cdot c \wedge$
 $\neg(\exists k \in \{1, \dots, n\}, \exists m_k: k < i \wedge e_k = \text{send}_c(m_k) \wedge m_i =_E m_k \wedge d(c, b) \leq (t_j - t_k) \cdot c)$.*

We denote the transitive closure of \rightsquigarrow_τ by \rightsquigarrow_τ^ .*

When timed events are related by \rightsquigarrow_τ^* , there are constraints on the times at which they can occur, as shown by the next two lemmas.

Lemma 1. *Let $\tau \in \llbracket \mathcal{P} \rrbracket_d$ be a trace. Then for each pair of timed events such that $(t_i, e_i) \rightsquigarrow_\tau^* (t_j, e_j)$ we have:*

$$t_j - t_i \geq \frac{d(\text{actor}(e_i), \text{actor}(e_j))}{c}$$

Lemma 2. *Let $\tau = (t_1, e_1) \dots (t_n, e_n) \in \llbracket \mathcal{P} \rrbracket_d$ be a trace, $i, j \in \{1, \dots, n\}$ such that $\text{actor}(e_i) = \text{actor}(e_j) = a$, and $\delta \in \mathbb{R}_+$ a constant. If $t_j - t_i \leq \frac{2\delta}{c}$, then for all $k \in \{i, \dots, j\}$ such that $(t_i, e_i) \rightsquigarrow_\tau^* (t_k, e_k) \rightsquigarrow_\tau^* (t_j, e_j)$ we have $d(a, \text{actor}(e_k)) \leq \delta$.*

The following definition states the minimal necessary condition that needs to be satisfied by a sequence of timed events in order for it to be a valid trace of some protocol: for each recv event there is a corresponding send event.

Definition 7. A sequence of timed events $(t_1, e_1) \cdots (t_n, e_n)$ is time valid if $t_1 \leq \dots \leq t_n$ and:

$$\begin{aligned} \forall i \in \{1, \dots, n\}, \forall m: e_i = \text{recv}_a(m) \\ \implies \exists j \in \{1, \dots, n\}, \exists m': (t_j, e_j) \rightsquigarrow_\tau (t_i, e_i) \wedge \\ e_j = \text{send}_b(m') \wedge m =_E m'. \end{aligned}$$

The following definition introduces a notation for the local view of a trace by an agent.

Definition 8. Given a trace $\tau = (t_1, e_1) \cdots (t_n, e_n) \in \llbracket \mathcal{P} \rrbracket_d$, and an agent $a \in \text{Agent}$, we define τ_a to be the sequence of timed events in τ executed by a . The timeless projection of τ , denoted by $\pi(\tau)$ is the list $e_1 \cdots e_n$.

At this point, we state a general result that relates two traces given by the semantics of a protocol.

Lemma 3. Let $\tau = (t_1, e_1) \cdots (t_n, e_n)$ and $\tau' = (t'_1, e'_1) \cdots (t'_n, e'_n)$ be two time valid sequences of events, and \mathcal{P} a protocol. If $\forall a \in \text{Agent}: \pi(\tau_a) = \pi(\tau'_a)$ then $\tau \in \llbracket \mathcal{P} \rrbracket_d \iff \tau' \in \llbracket \mathcal{P} \rrbracket_d$.

The next lemma proves the existence of a trace in a protocol which will be useful for the main result in this section. Given a trace, for each pair of events by the same actor, it is possible to construct another trace for which all the events in between the pair are executed by close agents. In the new trace some events are postponed and others are anticipated with respect to the pair. This result relies heavily on Lemmas 2 and 3.

Lemma 4. Let \mathcal{P} be a protocol and $a \in \text{Agent}$. Let $\tau = (t_1, e_1) \cdots (t_n, e_n) \in \llbracket \mathcal{P} \rrbracket_d$ be a trace such that there exist two timed events $(t_u, e_u), (t_v, e_v) \in \tau$ with $u < v$, $t_v - t_u \leq \frac{2 \cdot \delta}{c}$ and $\text{actor}(e_u) = \text{actor}(e_v) = a$. Then there exists a trace $\tau' = (t'_1, e'_1) \cdots (t'_n, e'_n) \in \llbracket \mathcal{P} \rrbracket_d$ and a bijection f from $\{1, \dots, n\}$ to $\{1, \dots, n\}$ such that:

- $\forall i \in \{1, \dots, n\}: e'_{f(i)} = e_i$; the events of τ' are a permutation of the events in τ
- $\forall a \in \text{Agent}: \pi(\tau_a) = \pi(\tau'_a)$; the permutation preserves the local order of events for each agent
- $t'_{f(v)} - t'_{f(u)} \leq \frac{2 \cdot \delta}{c}$ and $f(u) < f(v)$; the time restriction in τ translates to a corresponding time restriction in τ'
- $\forall k \in \{f(u), \dots, f(v)\}: d(a, \text{actor}(e'_k)) \leq \delta$; agents executing events between $f(u)$ and $f(v)$ have a bounded distance to a

The following example shows how the transformation of the previous lemma works for a simple trace.

Example 3. Let τ be the following execution of \mathcal{P}_{ex} :

$$\begin{aligned} s_0 &\xrightarrow{(0, \text{create}_a(id1, \mathcal{P}_{ex}(V)))} s_1 \xrightarrow{(0, \text{create}_b(id2, \mathcal{P}_{ex}(P)))} s_2 \\ &\xrightarrow{(0, \text{create}_e(id3, E))} s_3 \xrightarrow{(0, \text{send}_b(\langle \text{pk}(k), \text{sign}(n, k) \rangle))} s_4 \\ &\xrightarrow{(100, \text{recv}_a(\langle \text{pk}(k), \text{sign}(n, k) \rangle))} s_5 \xrightarrow{(100, \text{send}_a(n'))} s_6 \xrightarrow{(200, \text{recv}_b(n'))} s_7 \\ &\xrightarrow{(200, \text{send}_b(n' \oplus n))} s_8 \xrightarrow{(250, \text{recv}_e(n'))} s_9 \xrightarrow{(250, \text{send}_e(n' \oplus m'))} s_{10} \\ &\xrightarrow{(300, \text{recv}_a(n' \oplus n))} s_{11} \xrightarrow{(300, \text{clock}_a(2, 3))} s_{12} \xrightarrow{(300, \text{send}_a(\{k'\}_{\text{pk}(k)})} s_{13} \\ &\xrightarrow{(400, \text{recv}_b(\{k'\}_{\text{pk}(k)})} s_{14} \xrightarrow{(400, \text{send}_b(\langle (n, b) \rangle_{k'})} s_{15} \\ &\xrightarrow{(500, \text{recv}_a(\langle (n, b) \rangle_{k'})} s_{16} \xrightarrow{(500, \text{equal}_a(\langle \text{verify}(\text{sign}(n, k), b, \text{pk}(k)), \text{true} \rangle))} s_{17} \\ &\xrightarrow{(500, \text{claim}_a(\text{sec}, \langle b, k' \rangle))} s_{18} \end{aligned}$$

Then according to Lemma 4 with $\delta = \frac{c \cdot 200}{2}$, $d(a, b) = 0$, $d(a, e) > \delta$, $u = 6$, $v = 11$, one possible τ' is:

$$\begin{aligned} s_0 &\xrightarrow{(0, \text{create}_a(id1, \mathcal{P}_{ex}(V)))} s_1 \xrightarrow{(0, \text{create}_b(id2, \mathcal{P}_{ex}(P)))} s_2 \\ &\xrightarrow{(0, \text{create}_e(id3, E))} s_3 \xrightarrow{(0, \text{send}_b(\langle \text{pk}(k), \text{sign}(n, k) \rangle))} s_4 \\ &\xrightarrow{(100, \text{recv}_a(\langle \text{pk}(k), \text{sign}(n, k) \rangle))} s_5 \xrightarrow{(100, \text{send}_a(n'))} s_6 \xrightarrow{(200, \text{recv}_b(n'))} s_7 \\ &\xrightarrow{(200, \text{send}_b(n' \oplus n))} s_8 \xrightarrow{(300, \text{recv}_a(n' \oplus n))} s_9 \xrightarrow{(300, \text{clock}_a(2, 3))} s_{10} \\ &\xrightarrow{(300, \text{send}_a(\{k'\}_{\text{pk}(k)})} s_{11} \xrightarrow{(301, \text{recv}_e(n'))} s_{12} \xrightarrow{(301, \text{send}_e(n' \oplus m))} s_{13} \\ &\xrightarrow{(400, \text{recv}_b(\{k'\}_{\text{pk}(k)})} s_{14} \xrightarrow{(400, \text{send}_b(\langle (n, b) \rangle_{k'})} s_{15} \\ &\xrightarrow{(500, \text{recv}_a(\langle (n, b) \rangle_{k'})} s_{16} \xrightarrow{(500, \text{equal}_a(\langle \text{verify}(\text{sign}(n, k), b, \text{pk}(k)), \text{true} \rangle))} s_{17} \\ &\xrightarrow{(500, \text{claim}_a(\text{sec}, \langle b, k' \rangle))} s_{18} \end{aligned}$$

B. Security properties

The following corollary applies Lemma 4 to a protocol trace and to the events defined inside a clock event of interest.

Corollary 5. Let $\tau \in \llbracket \mathcal{P} \rrbracket_d$ be an execution trace. If $e_i = \text{claim}_a(\psi, m)$ is a claim in τ and $\text{correct_time}_\delta(\tau, i)$ is true, then there exists a trace $\tau' \in \llbracket \mathcal{P} \rrbracket_d$ such that the following conditions hold:

- 1) $\forall c \in \text{Agent}: \pi(\tau_c) = \pi(\tau'_c)$.
- 2) $e'_i = e_i$
- 3) $\{e_1, e_2, \dots, e_i\} = \{e'_1, e'_2, \dots, e'_i\}$
- 4) If $\text{run}(\tau', id_i) = (t'_{i_1}, e'_{i_1})^{id_i} \dots (t'_{i_k}, e'_{i_k})^{id_i}$ then:
 $\forall j \in \{i_1, \dots, i_k\}: e'_j = \text{clock}_a(x, y) \implies$
 $(\forall z: (i_x < z < i_y \implies d(\text{actor}(e'_z), a) \leq \delta))$

Before introducing the main result in this section, we need to restrict the security properties in our model, specified in the next definition, so that the previous lemma is applicable.

Definition 9. Let \mathcal{P} a protocol and ψ a security property. We say ψ is a robust security property iff for all $\tau, \tau' \in \llbracket \mathcal{P} \rrbracket_d$ such that $e_i = \text{claim}_a(\psi, m)$, and the first three conditions in Corollary 5 hold, we also have

$$\text{dist_attacker}_\delta(\tau, i) \wedge \text{correct_time}_\delta(\tau, i) \implies \psi(\tau, i)$$

$$\iff$$

$$\text{dist_attacker}_\delta(\tau', i) \wedge \text{correct_time}_\delta(\tau', i) \implies \psi(\tau', i)$$

The next lemma enables the use of any *robust* security property in our main equivalence result.

Lemma 6. *Let ψ be a robust security property. Then for all $\tau \in \llbracket \mathcal{P} \rrbracket_d$, such that $e_i = \text{claim}_a(\psi, m)$, τ' as defined in Corollary 5 with respect to τ , then:*

$$\begin{aligned} \text{dist_attacker}_\delta(\tau, i) \wedge \text{correct_time}_\delta(\tau, i) &\implies \psi(\tau, i) \\ \iff \\ \text{dist_attacker}_\delta(\tau', i) \wedge \text{correct_time}_\delta(\tau', i) &\implies \psi(\tau', i) \end{aligned}$$

Next, we define formally the main security properties our model supports, which we employ in the case studies later.

Definition 10 (Secrecy).

$$\begin{aligned} \forall \tau \in \llbracket \mathcal{P} \rrbracket_d, \tau_i = (t_i, \text{claim}_a(\text{sec}, (b, k))): \\ \text{dist_attacker}_\delta(\tau, i) \wedge \text{correct_time}_\delta(\tau, i) &\implies \\ (\nexists c \in \text{Dishonest}: c \vdash_\tau k) \vee b \in \text{Dishonest} \end{aligned}$$

For some protocols the communication partner is not known. The next definition covers this case, which we call anonymous secrecy.

Definition 11 (Anonymous Secrecy).

$$\begin{aligned} \forall \tau \in \llbracket \mathcal{P} \rrbracket_d, \tau_i = (t_i, \text{claim}_a(a_sec, k)): \\ \text{dist_attacker}_\delta(\tau, i) \wedge \text{correct_time}_\delta(\tau, i) &\implies \\ (\nexists c \in \text{Dishonest}: c \vdash_\tau k) \end{aligned}$$

In our case studies we use the non-injective agreement property [28], its anonymous variant and the secure remote erasure [7], as defined in appendix C. We prove all these properties are *robust* in Proposition 9.

C. A timeless protocol semantics

Now we simplify the security model by eliminating the notion of time from the original model, and provide a causality-based characterisation of the `dist_attacker` predicate. We do so by removing all occurrences of the time variables in the rules `CreateP`, `Send`, `Recvd`, `Equal` and `Signal`, producing their timeless equivalent `CreateP π` , `Send π` , `Recv π` , `Equal π` and `Signal π` . For reference, the modified rules can be found in Appendix A. The timeless protocol semantics associated with these rules produces a sequence of events, rather than a sequence of timed events.

Definition 12 (Timeless protocol semantics). *The timeless semantics of a protocol \mathcal{P} is the LTS $(Q, \Lambda, \rightarrow, s_0)$ where,*

- $Q = \mathbb{P}(\text{Id} \times \text{Ev}_g^* \times \text{R} \times \text{Agent})$
- $\Lambda = \text{Ev}_g \times \text{Id} \cup \{\text{create}_a(id, R)^{id} \mid a \in \text{Agent}, id \in \text{Id}, R \in \text{R}\}$
- $\rightarrow = \text{Create}_{P_\pi} \cup \text{Send}_\pi \cup \text{Recv}_\pi \cup \text{Equal}_\pi \cup \text{Signal}_\pi$.
- $s_0 = \emptyset$

We use $\llbracket \mathcal{P} \rrbracket_\pi$ to denote the set of traces obtained from \mathcal{P} 's timeless semantics.

D. A causality-based interpretation of the distant-attacker assumption

In the previous section, we introduced the notion of a distant attacker. This notion depended on a distance parameter δ that defines the vicinity of the actor in question. Here we provide a similar definition adapted to the timeless case. Let \sim be an equivalence relation on the set of actors with two equivalence classes. Intuitively, actors in the same class ($a \sim b$) are near, and those in different classes are far. The `dist_attacker π` predicate is defined by:

$$\begin{aligned} \text{dist_attacker}_\pi(\tau, i) &\iff \forall c \in (\text{actors}(\tau) \cap \text{Dishonest}): \\ &\quad \neg(a \sim c) \vee c \in \text{dishonest_agents}(e_i) \end{aligned}$$

Our definition of `correct_time` for the timeless case is inspired by the causal characterisation of distance bounding given in [9]. Our main result later shows how this definition exactly fits our objective. For every trace $\tau = e_1^{id_1} \dots e_n^{id_n}$ and every index i such that e_i is a claim event by agent a and $\text{run}(\tau, id_i) = e_{i_1} \dots e_{i_k}$ we have:

$$\begin{aligned} \text{correct_time}_\pi(\tau, i) &\iff \forall j \in \{i_1, \dots, i_k\} \forall b \in \text{Agent}: \\ &\quad (e_j = \text{clock}_a(x, y) \wedge j < i \wedge \neg(a \sim b)) \\ &\implies \nexists k: (i_x \leq k \leq i_y \wedge \text{actor}(e_k) = b) \end{aligned}$$

A protocol \mathcal{P} satisfies ψ conditional to the trust assumption `dist_attacker π` and the proximity check `correct_time π` within the timeless semantics if, for every trace $\tau \in \llbracket \mathcal{P} \rrbracket_\pi$ and every $i \in \{1, \dots, |\tau|\}$, where e_i is a claim event with security property ψ , it holds that `dist_attacker π (τ, i) \wedge correct_time π (τ, i) \implies $\psi(\tau, i)$. We denote this property by $\llbracket \mathcal{P} \rrbracket_\pi \triangleright_\sim \psi$.`

E. Equivalence between the timed and the timeless semantics

Finally we prove that, for every robust security property ψ defined as a first-order statement on sequence of events, ψ is correct in the timeless protocol semantics if and only if it is secure in the timed protocol semantics.

Lemma 7. *Let $\tau \in \llbracket \mathcal{P} \rrbracket_d$ and $\pi(\tau)$ be the timeless projection of τ . Then $\pi(\tau) \in \llbracket \mathcal{P} \rrbracket_\pi$.*

Proof sketch. Notice that if τ is generated according to the rules in the timed semantics of \mathcal{P} , then $\pi(\tau)$ can also be generated by the corresponding rules in the timeless semantics, as the latter is less restrictive than the former. \square

Now we are ready to formulate the main result.

Theorem 8. *Given a protocol \mathcal{P} and a robust security property ψ . Then $\forall d: \llbracket \mathcal{P} \rrbracket_d \triangleright_\delta \psi \iff \forall \sim: \llbracket \mathcal{P} \rrbracket_\pi \triangleright_\sim \psi$.*

VI. AUTOMATED VERIFICATION: CASE STUDIES

This section demonstrates how the theoretical development of previous sections can be used to analyse the (in)security of protocols that rely on the distant-attacker assumption. The section starts by introducing a verification methodology, which we showcase using three recent protocols. Then we report on the security analysis of a number of protocols which, to

the best of our knowledge, were lacking a formal analysis. This analysis was aided by the Tamarin prover. The coding of each protocol in Tamarin, together with their security lemmas, can be accessed online at <https://gitlab.uni.lu/regil/distant-attacker-tamarin>.

A. Methodology

Our methodology consists of six steps, whose implementation are showcased by carrying on the analysis of three recent protocols: Move2Auth [3], Amigo [4] and SPEED [6].

- 1) Description of the original protocol, security requirements and assumptions.
- 2) Abstraction of the original protocol into a symbolic model.
- 3) Definition of the security property to be verified as a member of the property class given in Definition 5, i.e. as a timeless security property conditional to the distant-attacker assumption.
- 4) Prove that the underlying timeless property satisfies Definition 9.
- 5) Replacement of proximity checks, such as signal strength and visual inspection, by distance bounding based on round-trip-time measurements. This is a workaround intended to fit into our framework proximity-checking protocols that are not based on round-trip-time.
- 6) Use of a protocol verification tool, such as Tamarin, to implement and verify the resulting protocol within the symbolic model introduced in Section V.
- 7) If attacks are found, map them back to the original setting of the protocol to ensure they are not a result of the abstraction step.

Steps 2 and 5 above may impose a risk of losing accuracy with respect to the original protocol. The former is standard in symbolic verification, the latter a choice made to analyse several similar protocols within our framework. We acknowledge Step 5 to be a workaround and not a proper solution to the problem of analysing protocols based on signal-strength. The workaround is useful, though, since some attacks found in a protocol modelled with round-trip-time are explainable in a context where signal strength is used (see attacks on Amigo [4] and Move2Auth [3] below). Because radio waves travel very close to the speed of light, it is harder for an attacker to manipulate their distance to honest participants by increasing the propagation speed of the communication channel, compared to manipulating signal strength (which can be amplified). The last step of the methodology is used to tackle the risk of over-abstracting or miss-representing a protocol, by ensuring that attacks found in the abstracted model apply to the original protocol.

B. Analysing Move2Auth

The protocol Move2Auth [3] aims to provide a secure communication channel between an IoT device and a smartphone. This protocol uses variations in the Received Signal Strength (RSS) perceived by a single antenna to detect proximity. The protocol starts when the smartphone connects to the WiFi

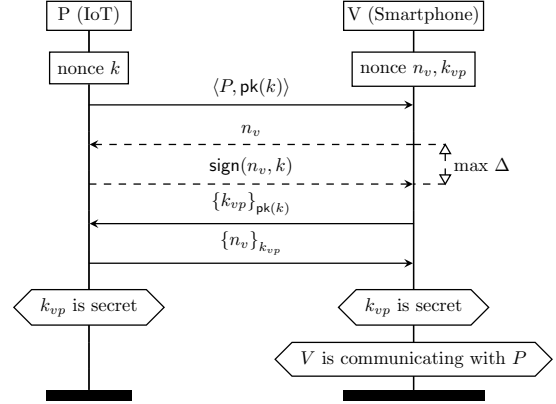


Fig. 3: A symbolic specification of Move2Auth-RTT

network of the device, identified by its SSID. Then the IoT device sends a public key to the smartphone. Immediately after that, the IoT device uses the corresponding private key to encrypt information about its MAC address and its identity. This encrypted data is sent several times to the smartphone, which uses variations on the signal strength of the received packets to determine proximity to the IoT device. The smartphone also verifies that the received packets are correct with respect to the public key received at the start of the protocol. If both signatures and signal strength measurements are correct, the smartphone sends the freshly generated key k_{vp} encrypted with the public key, and concludes that it has correctly exchanged the secret key k_{vp} with the IoT device.

To analyse Move2Auth, we substitute RSS measurements by round-trip-time measurements using a technique established in the literature [14]. This technique consists of a verifier sending a nonce to a prover, expecting to receive as a response a message containing the nonce and a secret key identifying the prover. We added an extra message at the end to let the prover use the exchanged key, as it would in a real scenario. The resulting protocol is displayed in Figure 3.

In our abstraction of Move2Auth (Move2Auth-RTT), the device (P) sends its identity (which could represent the SSID of the Wi-Fi network) and a fresh public key $pk(k)$. Then the smartphone (V) sends a nonce n_v to be used during the fast phase, and expects a message consisting of the signature of n_v with key k , which could only be constructed by the device. Then V generates a fresh key k_{vp} and sends it encrypted to P . Finally, P replies with the nonce n_v encrypted by k_{vp} , showing V that it received the shared key.

We modelled the authentication property from the smartphone’s point of view as non-injective agreement on the key k_{vp} . We found this property to be false. The attack consists of an attacker that modifies the identity in the first message. We modelled the secrecy claim by the device as (anonymous) secrecy and found this property to be false, as expected, given that the P does not execute a proximity check on the verifier. The secrecy claim made by the smartphone, on the other hand,

is proven correct by Tamarin.

Attacks: The attack on secrecy by P is realized in a trace in which a distant attacker impersonates an honest agent in the verifier role. In the original setting the same attacks applies, as the last message can be sent by the attacker itself. This vulnerability can be potentially exploited because it allows the attacker to control the IoT device by sending encrypted commands, which the IoT device will accept as correct. Regarding the authentication claim by V , the vulnerability derives from the fact that the identity in the first message is not tied to the rest of the messages of the protocol, and it is not protected cryptographically, so the attacker can easily modify it. In the original protocol, this attack would correspond to a situation in which the attacker sets up a fake Wi-Fi network.

C. Analysing Amigo

Amigo [4] is another protocol for mobile device authentication that depends on proximity. Similar to Move2Auth, each device computes a signature based on the radio environment, and uses it to detect if the other device is near. This is coupled with a key exchange based on Diffie-Hellman. As in Move2Auth-RTT, we modified the protocol to rely on round trip time rather than signal strength.

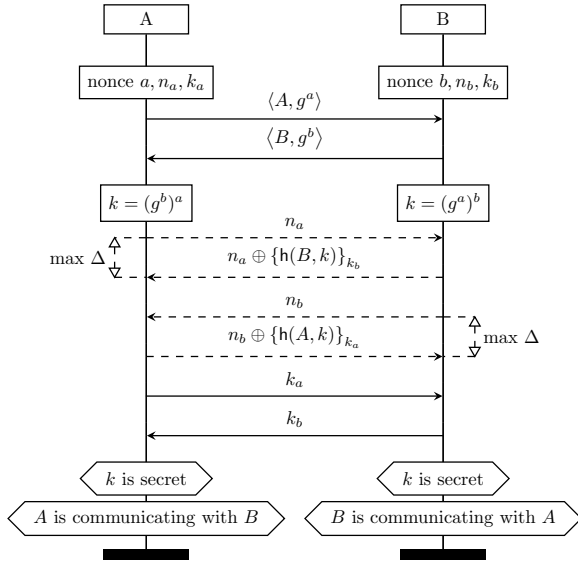


Fig. 4: A representation of the Amigo-RTT protocol

The modified protocol Amigo-RTT starts by a traditional Diffie-Hellman exchange, after which a fast phase is executed by each agent. The rapid phase uses a commitment scheme by encrypting the hash of the agent’s identity (A or B) and the common key (k). The encryption key (k_a or k_b) is revealed at the end of the protocol. The abstracted protocol is shown in Figure 4.

Attacks: We modelled non-injective agreement and secrecy claims with respect to key k for each role. All claims are invalid. The attack on secrecy for role A works as follows.

Two nearby honest agents A and B execute the protocol. The attacker E captures the first two messages, and modifies them so that the new messages are $\langle A, 1 \rangle$ and $\langle A, 1 \rangle$ where 1 is the unit value in the multiplication group. The protocol continues without any intervention from E , resulting in an exchanged key equal to 1, which is known to E . We note that this attack can be prevented by following a Diffie-Hellman public key validation². Such validation is optional in the specification, but our analysis shows it to be necessary in Amigo-RTT. We found this attack by using the TAMARIN extension developed in [29].

The attack on the role A ’s authentication claim is as follows. Two nearby honest agents X , Y execute the protocol. The attacker E manipulates the messages so that X finishes the execution in role A with Y , while Y was also executing the role A rather than B . Both X and Y send messages $\langle X, g^x \rangle$ and $\langle Y, g^y \rangle$, which the attacker delays such that both are received by the other agent as the second message. When Y starts the proximity check by sending n_y , the attacker responds with a random message m . Notice Y cannot check whether this message is correct until later, so it continues executing the protocol. Then X starts the proximity check by sending n_x , and Y sends the correct response $n_x \oplus \{h(Y, k)\}_{k_y}$, even though for Y this is the second proximity check, while for X it is still the first. Y continues the protocol by sending k_y . At this point, E has enough information to complete the protocol with X , by executing the final proximity check in role B .

We finish the analysis of Amigo-RTT by noting that attacks on the claims made by role B are similar to those shown above for role A .

D. Analysing SPEED

SPEED [6] aims to guarantee that an IoT device has erased its memory. By using a combination of software memory isolation techniques and distance bounding based on round trip time measurements, the protocol enables a device acting as verifier to erase the memory of another device, of low computational power, acting as prover, only when the verifier is nearby the prover. The protocol starts with a distance-bounding phase based on [30], which allows the prover to check the verifier is nearby. The information exchange in this phase is then used by the prover to compute a fresh key k and a MAC on its memory. The abstraction of the protocol is shown in Figure 5. For our security analysis we used the memory erasure property from [7]. This property is formally stated and proved robust in Appendix B. The analysis revealed that the security claim made by the verifier is invalid. This is not a surprise given that the verifier does not check its proximity to the prover. Given that prover and verifier do not share cryptographic information before the protocol execution, a trivial impersonation attack can be executed by a distant attacker.

²2.1.5. Public Key Validation <https://tools.ietf.org/html/rfc2631>

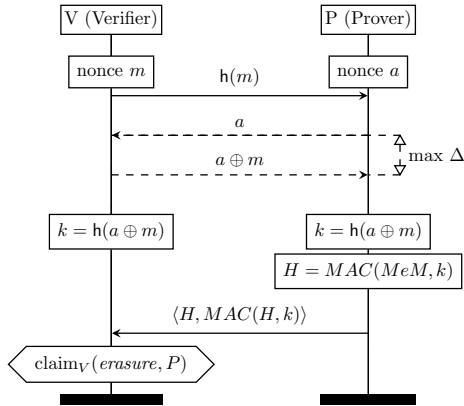


Fig. 5: A representation of the SPEED protocol

E. Summary of analysis results

We extended the analysis methodology described above to four more protocols [2], [4], [6], [27] (see Table I). The analysed protocols include (anonymous) secrecy, (anonymous) authentication and memory erasure properties. All of these properties can be defined without the notion of time, and, as proved in the previous section, they are compatible with our equivalence results. All of them have the common characteristic of using proximity checks to (hopefully) assure that the communication partner is in the vicinity.

Most protocols that consider the distant-attacker assumption are key-exchange protocols. Two close agents without any previously shared data nor public key infrastructure need to communicate in a secure way using a network that may be controlled by distant attackers. These protocols necessarily use some kind of asymmetric cryptography, given that the adversary will receive (with some delay) all the transmitted messages, and from that it should be impossible to deduce the secret shared keys.

Protocol	Secrecy		Auth.		MemE
	P	V	P	V	
DB-Based-Diffie-Hellman [2]	✓	✓	✓	✓	-
MedicalDB [1]	✓	✓	✓	✓	-
BluetoothJW-RTT [27]	✓	✓	-	✓	-
Move2Auth-RTT [3]	✗	✓	-	✗	-
Amigo-RTT [4]	✗	✗	✗	✗	-
SPEED [6]	-	-	-	-	✗
DB-Based-Erasure-Protocol [7]	-	-	-	-	✓

TABLE I: Analysis results

Table I shows that three out of five of the key exchange protocols analysed are correct. The other two, namely Move2Auth and Amigo, fail to ensure secrecy of the key exchanged. Our analysis results on memory erasure protocols coincide with those provided in [7], including the proof of correctness for the memory erasure protocol introduced there. We note,

however, that [7] provides manual proofs, while our proofs are computer-generated.

F. Comments on the Tamarin encoding

Our Tamarin code needs to mark adversary actions in the trace, as they are part of our security properties. We do so by creating rules representing a channel that can be used by honest agents and adversaries alike. When modelling all the network interactions using this channel, we faced non-termination issues. For this reason, we decided to restrict this channel usage to the messages directly related to the time measurement. This resulted in an over-approximation, in the sense that with this encoding some false attacks could appear, given that not all actions by adversaries are marked. That said, we manually checked this was not the case for any of the protocols analysed.

VII. CONCLUSION

In this article we identified and formalised the distant-attacker assumption, which has so far been used informally to establish the security requirements of various communication protocols for, e.g., IoT devices. We did so by introducing a time-based security model where round-trip-time measurements and the location of agents is used to determine whether the neighbourhood of an agent is free of attackers. To enable computer-aided verification of protocols written in our specification language, we provided a reduction of the time-based model by eliminating the notions of time and location, and defining proximity checks and the distant-attacker assumption as causal relations on the protocol events. We also introduced a class of security requirements that we proved hold in both the time-based and the causality-based model. Because the causality-based model is translatable to TAMARIN, we were able to formally verify, for the first time, the (in)security of five key exchange protocols and two memory erasure protocols, finding unreported vulnerabilities on three of them.

The results presented in this article can be extended in various ways. For example, the protocol specification language we use does not allow for conditionals and non-determinism, and only supports basic round-trip-time calculations. Hence extending our results to richer specification languages would contribute to capturing a larger class of protocols. It is also worth generalising our proofs to a causality-based specification model that is a subset of the model supported by a state-of-the-art protocol verification tool, such as TAMARIN, since it would reduce the gap between theory and practice. Lastly, like previous verification frameworks for distance-bounding protocols, our methodology assumes that agents do not move. Dropping that assumption is of interest for both classes of protocols.

REFERENCES

- [1] K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Capkun, "Proximity-based access control for implantable medical devices," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009, pp. 410–419.

- [2] D. Singelee and B. Preneel, “Key establishment using secure distance bounding protocols,” in *2007 Fourth Annual International Conference on Mobile and Ubiquitous Systems: Networking Services (MobiQuitous)*, Aug 2007, pp. 1–6.
- [3] J. Zhang, Z. Wang, Z. Yang, and Q. Zhang, “Proximity based IoT device authentication,” in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.
- [4] A. Varshavsky, A. Scannell, A. LaMarca, and E. De Lara, “Amigo: Proximity-based authentication of mobile devices,” in *International Conference on Ubiquitous Computing*. Springer, 2007, pp. 253–270.
- [5] D. Perito and G. Tsudik, “Secure code update for embedded devices via proofs of secure erasure,” in *15th European Symposium on Research in Computer Security (ESORICS’10)*, Athens, Greece, September 20–22., 2010, pp. 643–662.
- [6] M. Ammar, W. Daniels, B. Crispo, and D. Hughes, “Speed: Secure provable erasure for class-1 iot devices,” in *8th ACM Conference on Data and Application Security and Privacy*, ser. CODASPY ’18. ACM, 2018, pp. 111–118.
- [7] R. Trujillo-Rasua, “Secure memory erasure in the presence of man-in-the-middle attackers,” *Journal of Information Security and Applications*, vol. 57, 2019.
- [8] D. A. Basin, S. Capkun, P. Schaller, and B. Schmidt, “Let’s get physical: Models and methods for real-world security protocols,” in *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17–20.*, 2009, pp. 1–22.
- [9] S. Mauw, Z. Smith, J. Toro-Pozo, and R. Trujillo-Rasua, “Distance-bounding protocols: Verification without time and location,” in *39th IEEE Symposium on Security and Privacy, SP, 21–23 May 2018, San Francisco, California, USA*, 2018, pp. 549–566.
- [10] T. Chothia, J. De Ruiter, and B. Smyth, “Modelling and analysis of a hierarchy of distance bounding attacks,” in *27th \$SUSENIX\$ Security Symposium (\$SUSENIX\$ Security 18)*, 2018.
- [11] A. Debant, S. Delaune, and C. Wiedling, “A symbolic framework to analyse physical proximity in security protocols,” in *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- [12] —, “Proving physical proximity using symbolic models,” Univ Rennes, CNRS, IRISA, France, Tech. Rep., 2018.
- [13] I. Boureau, T. Chothia, A. Debant, and S. Delaune, “Security Analysis and Implementation of Relay-Resistant Contactless Payments,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.
- [14] G. Avoine and et al., “Security of distance-bounding: A survey,” *ACM Computing Surveys*, vol. 51, no. 5, 2018.
- [15] C. Cremers, K. B. Rasmussen, B. Schmidt, and S. Capkun, “Distance hijacking attacks on distance bounding protocols,” in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 113–127.
- [16] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, “The TAMARIN prover for the symbolic analysis of security protocols,” in *CAV’13*, 2013.
- [17] C. Meadows, R. Poovendran, D. Pavlovic, L. Chang, and P. Syverson, “Distance bounding protocols: Authentication logic analysis and collusion attacks,” in *Secure localization and time synchronization for wireless sensor and ad hoc networks*. Springer, 2007, pp. 279–298.
- [18] A. Debant and S. Delaune, “Symbolic verification of distance bounding protocols,” in *Principles of Security and Trust - 8th International Conference*, F. Nielson and D. Sands, Eds., 2019.
- [19] M. A. Alturki, T. B. Kirigin, M. Kanovich, V. Nigam, A. Scedrov, and C. Talcott, “A multiset rewriting model for specifying and verifying timing aspects of security protocols,” in *Foundations of Security, Protocols, and Equational Reasoning*. Springer, 2019.
- [20] R. Corin, S. Etalle, P. H. Hartel, and A. Mader, “Timed model checking of security protocols,” in *Proceedings of the ACM workshop on Formal methods in security engineering*, 2004.
- [21] G. Jakubowska and W. Penczek, “Modelling and checking timed authentication of security protocols,” *Fundamenta Informaticae*, vol. 79, no. 3–4, 2007.
- [22] S. Szymoniak, O. Siedlecka-Lamch, and M. Kurkowski, “Timed analysis of security protocols,” in *Proceedings of 37th International Conference on Information Systems Architecture and Technology*. Springer, 2017.
- [23] C. Cremers and S. Mauw, *Operational Semantics and Verification of Security Protocols*. Springer, 2012.

- [24] J. A. Goguen and J. Meseguer, “Order-sorted algebra i: equational deduction for multiple inheritance, overloading, exceptions and partial operations,” *Theoretical Computer Science*, vol. 105, no. 2, 1992.
- [25] F. Pfeiffer, K. Finkenzeller, and E. Biebl, “Theoretical limits of iso/iec 14443 type a rfid eavesdropping attacks,” in *Smart SysTech 2012; European Conference on Smart Objects, Systems and Technologies*, 2012, pp. 1–9.
- [26] S. Čapkun, L. Buttyán, and J.-P. Hubaux, “SECTOR: secure tracking of node encounters in multi-hop wireless networks,” in *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, 2003.
- [27] D.-Z. Sun, Y. Mu, and W. Susilo, “Man-in-the-middle attacks on Secure Simple Pairing in Bluetooth standard V5. 0 and its countermeasure,” *Personal and Ubiquitous Computing*, vol. 22, pp. 55–67, 2018.
- [28] G. Lowe, “A hierarchy of authentication specifications,” in *CSF’97*, 1997, pp. 31–43.
- [29] C. Cremers and D. Jackson, “Prime, order please! Revisiting small subgroup and invalid curve attacks on protocols using Diffie-Hellman,” in *IEEE 32nd Computer Security Foundations Symposium*, 2019.
- [30] S. Brands and D. Chaum, “Distance-bounding protocols,” in *EUROCRYPT’93*, 1993, pp. 344–359.

APPENDIX

A. Timeless Semantics

The timeless semantics is obtained by removing the time-stamps from the rules used in the time-based model. This means that traces obtained from this semantics are just a sequence of events. The resulting rules are as follows.

$$\frac{s \in Q, R \in R, a \in \text{Agent}, id \in_s Id, a \in \text{Honest} \implies R \in \text{roles}(\mathcal{P})}{s \xrightarrow{(\text{create}_a(R))^{id}} s \cup \{(id, \epsilon, R, a)\}} \text{Create}_{\mathcal{P}\pi},$$

Template rule for the timeless semantics:

$$\frac{s \in Q, (id, \tau, R, a) \in s, \tau = e_1 \cdots e_i, e_1 \cdots e_i e_{i+1} \in_{s \setminus \{(id, \tau, R, a)\}} \text{insts}(R), \quad []}{s \xrightarrow{(e_{i+1})^{id}} (s \setminus \{(id, \tau, R, a)\}) \cup \{(id, \tau \cdot e_{i+1}, R, a)\}},$$

$$\begin{aligned} &\rightarrow [e_{i+1} = \text{send}_a(m), a \vdash_{\{e_1, \dots, e_i\}} m] && \text{Send}_{\pi} \\ &\rightarrow \left[\begin{array}{l} e_{i+1} = \text{recv}_a(m), \text{send}_b(m') \in \text{labels}(s), \\ m =_E m' \end{array} \right] && \text{Recv}_{\pi} \\ &\rightarrow [e_{i+1} = \text{equal}_a(\langle m_1, m_2 \rangle), m_1 =_E m_2] && \text{Equal}_{\pi} \\ &\rightarrow [e_{i+1} \in \text{SignalEvent}] && \text{Signal}_{\pi} \end{aligned}$$

B. Proofs

Proof of Lemma 1. From the definition of \rightsquigarrow_{τ} we deduce that if $(t_x, e_x) \rightsquigarrow_{\tau} (t_y, e_y)$ we have:

$$t_y - t_x \geq \frac{d(\text{actor}(e_x), \text{actor}(e_y))}{c}$$

Lets assume $(t_i, e_i) = (t_{i_1}, e_{i_1}) \rightsquigarrow_{\tau} \dots \rightsquigarrow_{\tau} (t_{i_k}, e_{i_k}) = (t_j, e_j)$. The property above and the triangle inequality lead to the required result:

$$\begin{aligned} t_j - t_i &= t_{i_k} - t_{i_1} = \sum_{j=1}^{k-1} t_{i_{j+1}} - t_{i_j} \\ &\geq \sum_{j=1}^{k-1} \frac{d(\text{actor}(e_{i_j}), \text{actor}(e_{i_{j+1}}))}{c} \\ &\geq \frac{d(\text{actor}(e_{i_1}), \text{actor}(e_{i_k}))}{c} = \frac{d(\text{actor}(e_i), \text{actor}(e_j))}{c} \quad \square \end{aligned}$$

Proof of Lemma 2. By the previous lemma, we have that:

$$\begin{aligned} \frac{2\delta}{c} &\geq t_j - t_i = (t_j - t_k) + (t_k - t_i) \geq \\ \frac{d(a, \text{actor}(e_k)) + d(\text{actor}(e_k), a)}{c} &= \frac{2 \cdot d(a, \text{actor}(e_k))}{c} \\ \implies d(a, \text{actor}(e_k)) &\leq \delta \quad \square \end{aligned}$$

Proof of Lemma 3. By symmetry, it is only necessary to prove that $\tau \in \llbracket \mathcal{P} \rrbracket_d \implies \tau' \in \llbracket \mathcal{P} \rrbracket_d$.

Notice that as $\tau \in \llbracket \mathcal{P} \rrbracket_d$, then τ can be generated inductively by applying the rules defined in the semantics in $\llbracket \mathcal{P} \rrbracket_d$. Furthermore, in this semantics, all time constraints in τ are generated by the application of the Recv_d rule.

We prove that each prefix of τ' can be generated by applying the rules in the semantics. The proof follows by induction in the size of the prefixes. The base case is the empty trace, which by definition is a valid trace of any protocol. For the induction step, assume for some i that $(t'_1, e'_1) \dots (t'_i, e'_i) \in \llbracket \mathcal{P} \rrbracket_d$ and that $a = \text{actor}(e'_{i+1})$. We prove that $(t'_1, e'_1) \dots (t'_{i+1}, e'_{i+1}) \in \llbracket \mathcal{P} \rrbracket_d$ by case analysis:

- e'_{i+1} is a receive event: as τ' is a time valid sequence of timed events, it can be generated by applying the Recv_d rule at time t'_{i+1} .
- e'_{i+1} is not a receive event: notice that

$$\begin{aligned} \pi(\tau_a) = \pi(\tau'_a) &\implies \exists j \in \{1, \dots, n\}: \\ \pi(((t'_1, e'_1) \dots (t'_{i+1}, e'_{i+1}))_a) &= \pi(((t_1, e_1) \dots (t_j, e_j))_a) \\ \wedge e'_{i+1} &= e_j \end{aligned}$$

The same rule that was used to generate (t_j, e_j) can also be used to generate (t'_{i+1}, e'_{i+1}) , given that this rule only depends on the sequence of previous events by agent a .

We conclude all events in τ' can be generated inductively by the rules, as was needed. \square

Proof of Lemma 4. For any trace $\tau^* = (t_1^*, e_1^*) \dots (t_n^*, e_n^*)$, let $K_{\tau^*} \subseteq \{1, \dots, n\}: k \in K_{\tau^*} \iff d(a, \text{actor}(e_k^*)) > \delta \wedge u < k < v$. The set K_{τ^*} thus contains the indices of all timed events between u and v that are executed by an agent outside the vicinity of a .

Let $\tau^0 = (t_1^0, e_1^0) \dots (t_n^0, e_n^0) = \tau$ and $r = |K_{\tau^*}|$. If $r = 0$ then all the necessary conditions are fulfilled for $\tau' = \tau$ and f the identity. Otherwise, there exist τ^1, \dots, τ^r , where $\forall i \in \{1, \dots, r\}: \tau^i = (t_1^i, e_1^i) \dots (t_n^i, e_n^i)$, and a sequence of bijections f_1, \dots, f_r such that:

- $\forall i \in \{1, \dots, r\}: f_i$ is a bijection between $\{1, \dots, n\}$ and $\{1, \dots, n\}$ such that $\forall j \in \{1, \dots, n\}: e_j^{i-1} = e_{f_i(j)}^i$
- $\forall i \in \{1, \dots, r\}, \forall a \in \text{Agent}: \pi(\tau_a^i) = \pi(\tau_a^{i-1})$
- $\forall i \in \{1, \dots, r\}, \exists k \in K_{\tau^{i-1}}: K_{\tau^i} = K_{\tau^{i-1}} \setminus \{k\}$
- $t_{f_1(u)}^1 = t_u^0 = t_u \wedge t_{f_1(v)}^1 = t_v^0 = t_v$
- $\forall i \in \{2, \dots, r\}: t_{f_1 \circ \dots \circ f_i(u)}^i = t_{f_1 \circ \dots \circ f_{i-1}(u)}^{i-1} \wedge t_{f_1 \circ \dots \circ f_i(v)}^i = t_{f_1 \circ \dots \circ f_{i-1}(v)}^{i-1}$
- $\forall i \in \{1, \dots, r\}: \tau^i$ is a time valid sequence.

Notice $\tau^r = \tau^r$ and $f = f_1 \circ \dots \circ f_r$ fulfill the required conditions in the lemma:

- $\forall a \in \text{Agent}: \pi(\tau_a) = \pi(\tau'_a)$ and $\forall i \in \{1, \dots, n\}: e'_{f(i)} = e_i$ are true by definition of f , given it is the composition of functions that possess the same properties.
- $K_{\tau'} = \emptyset \implies \forall k \in \{f(u), \dots, f(v)\}: d(a, \text{actor}(e'_k)) \leq \delta$
- $t'_{f(u)} = t_u \wedge t'_{f(v)} = t_v$, as $t'_{f(u)} = t_{f_1 \circ \dots \circ f_r(u)}^r = t_{f_1 \circ \dots \circ f_{r-1}(u)}^{r-1} = \dots = t_{f_1(u)}^1 = t_u^0 = t_u$ and the same can be deduced for v . Then $t'_{f(v)} - t'_{f(u)} = t_v - t_u \leq \frac{2 \cdot \delta}{c}$

$\tau' \in \llbracket \mathcal{P} \rrbracket_d$ follows from Lemma 3, as we have $\tau' = \tau^r$ is a time valid sequence by definition, and that $\forall a \in \text{Agent}: \pi(\tau_a) = \pi(\tau'_a)$ as mentioned above. We complete the proof by showing how to construct the trace τ^i and the bijection f_i from τ^{i-1} , for any index i .

To simplify notation, in what follows we will refer to traces τ, τ' (instead of τ^i, τ^{i+1}), bijection f (instead of f_i), integers x and y (instead of $f_1 \circ \dots \circ f_{i-1}(u)$ and $f_1 \circ \dots \circ f_{i-1}(v)$).

Let $k \in K_{\tau}$, then by Lemma 2 $(t_x, e_x) \rightsquigarrow_{\tau}^* (t_k, e_k) \wedge (t_k, e_k) \rightsquigarrow_{\tau}^* (t_y, e_y)$ is false. We analyse two cases:

- $(t_k, e_k) \rightsquigarrow_{\tau}^* (t_y, e_y)$ is false
- $(t_x, e_x) \rightsquigarrow_{\tau}^* (t_k, e_k)$ is false

We will focus on the first case, the other one can be analysed in an analogous way. Let ω be a constant greater than $t_y - t_k$ and $\tau'' = (t_1'', e_1'') \dots (t_n'', e_n'')$ a sequence of timed events defined as follows:

- 1) for each timed event $(t_i, e_i) \in \tau$ such that $i > y$, let $(t_i'', e_i'') = (t_i + \omega, e_i)$
- 2) for each timed event $(t_i, e_i) \in \tau$ such that $i < y$ and $(t_k, e_k) \rightsquigarrow_{\tau}^* (t_i, e_i)$, let $(t_i'', e_i'') = (t_i + \omega, e_i)$. In this case clearly $i \geq k$
- 3) for every other timed event $(t_i, e_i) \in \tau$, let $(t_i'', e_i'') = (t_i, e_i)$

Next, we define a bijection f from $\{1, \dots, n\}$ to $\{1, \dots, n\}$ according to the following property:

- $\forall i, j \in \{1, \dots, n\}: (t_i'' < t_j'') \vee (t_i'' = t_j'' \wedge i < j) \iff f(i) < f(j)$

Notice that f exists and it is unique as it defines a stable order for the values of t_1'', \dots, t_n'' .

At this point we are ready to define the trace $\tau' = (t_1', e_1') \dots (t_n', e_n')$. For all $i \in \{1, \dots, n\}$ let $(t_i', e_i') = (t_{f^{-1}(i)}'', e_{f^{-1}(i)}'')$. Then we deduce $\forall i \in \{1, \dots, n\}: e'_{f(i)} = e_i \wedge (t'_{f(i)} = t_i \vee t'_{f(i)} = t_i + \omega)$

Notice this f and τ' fulfill the required conditions:

- By construction $\forall i \in \{1, \dots, n\}: e_i = e'_{f(i)}$
- Assume $i, j \in \{1, \dots, n\}$ such that $i < j \wedge \text{actor}(e_i) = \text{actor}(e_j)$. Then $(t_i, e_i) \rightsquigarrow_{\tau}^* (t_j, e_j)$ is true, and $t'_{f(j)} = t_j + \omega \implies t'_{f(i)} = t_i + \omega$. We conclude that $\forall a \in \text{Agent}: \pi(\tau_a) = \pi(\tau'_a)$
- By construction $t'_{f(x)} = t_x \wedge t'_{f(y)} = t_y$
- $t'_{f(k)} = t_k + \omega > t_y = t'_{f(y)} \implies f(k) > f(y) \implies \neg(f(x) < f(k) < f(y))$
- τ' is a time valid sequence given that by construction:

$$\begin{aligned} \forall i, j \in \{1, \dots, n\}: (t_i, e_i) \rightsquigarrow_{\tau}^* (t_j, e_j) &\iff \\ (t'_{f(i)}, e'_{f(i)}) \rightsquigarrow_{\tau}^* (t'_{f(j)}, e'_{f(j)}) &\quad \square \end{aligned}$$

Proof sketch of Corollary 5. This is a direct application of the Lemma 4 to the events mentioned in the clock events in the same run as the claim.

Notice that as the predicate `correct_time` is true, for all clock events $\text{clock}_a(x, y)$ in the run corresponding to e_i , we get that $t_{i_y} - t_{i_x} \leq \frac{2\delta}{c}$. As such, we can apply Lemma 4 and obtain a new trace in which all actors executing an event between i_x and i_y are near a . Doing this procedure for each clock event, we get the desired trace. This is possible given that the segments events referenced in clock events do not intersect. \square

C. Consistency proofs for standard security properties with our equivalence results

Definition 13 (Non injective agreement).

$$\begin{aligned} \forall \tau \in \llbracket \mathcal{P} \rrbracket_d \tau_i = (t_i, \text{claim}_a(\text{non_in_agree}, \langle b, m \rangle)) : \\ \text{dist_attacker}_{\delta}(\tau, i) \wedge \text{correct_time}_{\delta}(\tau, i) \\ \implies (\exists j < i: e_j = \text{running}_b(\langle \text{role}B, a, m \rangle)) \\ \forall b \in \text{Dishonest} \end{aligned}$$

Definition 14 (Anonymous non injective agreement).

$$\begin{aligned} \forall \tau \in \llbracket \mathcal{P} \rrbracket_d, \tau_i = (t_i, \text{claim}_a(a_non_in_agree, m)) : \\ \text{dist_attacker}_{\delta}(\tau, i) \wedge \text{correct_time}_{\delta}(\tau, i) \\ \implies (\exists b \in \text{Agent}, j < i: e_j = \text{running}_b(\langle \text{role}B, m \rangle)) \end{aligned}$$

Definition 15 (Remote Memory Erasure).

$$\begin{aligned} \forall \tau \in \llbracket \mathcal{P} \rrbracket_d, \tau_i = (t_i, \text{claim}_a(\text{erasure}, \langle b, m \rangle)), : \\ \text{dist_attacker}_{\delta}(\tau, i) \wedge \text{correct_time}_{\delta}(\tau, i) \implies \\ \exists j < i: (t_j, \text{send}_b(m)) \in \tau \vee (t_j, \text{recv}_b(m)) \in \tau \end{aligned}$$

Proposition 9. *The security properties `sec`, `a_sec`, `non_in_agree`, `a_non_in_agree`, and `erasure` are robust security properties.*

Proof. Let $\tau, \tau' \in \llbracket \mathcal{P} \rrbracket_d$ such that the first three conditions in Corollary 5 hold. Then the condition $\forall c \in \text{Agent}: \pi(\tau_c) = \pi(\tau'_c)$ implies that in both traces the knowledge deduced by dishonest agents is the same, so `sec` and `a_sec` are *robust*.

The properties `non_in_agree`, `a_non_in_agree` and `erasure` are *robust* given that both depend on the existence of events before e_i , and by definition $\{e_1, e_2, \dots, e_i\} = \{e'_1, e'_2, \dots, e'_i\}$. \square

Proof of Theorem 8. We prove the left right implication first:

$$\forall d: \llbracket \mathcal{P} \rrbracket_d \triangleright_{\delta} \psi \implies \forall \sim: \llbracket \mathcal{P} \rrbracket_{\pi} \triangleright_{\sim} \psi$$

We will formally prove the contrapositive.

Assuming $\forall \sim: \llbracket \mathcal{P} \rrbracket_{\pi} \triangleright_{\sim} \psi$ is false, we deduce

$$\begin{aligned} \exists \sim, a \in \text{Agent}, p \in \{1, \dots, n\}, \tau \in \llbracket \mathcal{P} \rrbracket_{\pi}: \\ e_p = \text{claim}_a(\psi, m) \wedge \text{dist_attacker}_{\pi}(\tau, p) \wedge \\ \text{correct_time}_{\pi}(\tau, p) \wedge \neg\psi(\tau, p) \end{aligned}$$

We will construct a trace $\tau' = (t'_1, e'_1) \dots (t'_n, e'_n) \in \llbracket \mathcal{P} \rrbracket_d$, for some distance function d , such that $\tau = \pi(\tau')$. To achieve this, we will assign locations to actors such that the feasible values of t'_i exist.

For simplicity we assume the run corresponding to e_p contains only one clock event. Let $\text{run}(\tau, id_p) = (e_{p_1})^{id_p} \dots (e_{p_k})^{id_p}$, $e_{p_i} = \text{clock}_a(x, y)$, $u = p_x$ and $v = p_y$. We assign locations to actors such that $b \sim c \implies d(b, c) = 0$ and $\neg(b \sim c) \implies d(b, c) > \delta$. As `correct_time` $_{\pi}(\tau, p)$ is true, we have $\forall k \in \{u, \dots, v\}: d(\text{actor}(e_k), a) = 0$. We also deduce that $\text{dist_attacker}_{\pi}(\tau, p) \implies \text{dist_attacker}_{\delta}(\tau', p)$.

Let $\gamma > 0$ be an arbitrarily large constant. Then we are ready to define the values of t'_i :

- $t'_1 = 0$
- $1 < i \leq u \implies t'_i = (i - 1) \cdot \gamma$
- $u < i < v \implies t'_i = (i - u) \cdot \frac{2\epsilon}{c} + t'_u$
- $t'_v = t'_u + \frac{2\delta}{c}$
- $v < i \leq n \implies t'_i = (i - v) \cdot \gamma + t'_v$

Notice for sufficiently large γ and sufficiently small ϵ , all necessary conditions are fulfilled (the time stamps are ordered, no message travels faster than the speed of light, and $t'_v - t'_u \leq \frac{2\delta}{c}$ which implies `correct_time` $_{\delta}(\tau', p)$ is true). Then, we deduce τ' can be generated inductively with the rules defined within $\llbracket \mathcal{P} \rrbracket_d$. On the other hand, as $\psi(\tau, p)$ doesn't depend on time, then τ' also represents an attack in $\llbracket \mathcal{P} \rrbracket_d$, as needed.

In what follows we prove the other implication also using the contrapositive:

$$\forall \sim: \llbracket \mathcal{P} \rrbracket_{\pi} \triangleright_{\sim} \psi \implies \forall d: \llbracket \mathcal{P} \rrbracket_d \triangleright_{\delta} \psi$$

Let $\tau = (t_1, e_1) \dots (t_n, e_n) \in \llbracket \mathcal{P} \rrbracket_d$, $p \in \{1, \dots, n\}$ such that τ is an attack trace:

$$\begin{aligned} e_p = \text{claim}_a(\psi, m) \wedge \text{dist_attacker}_{\delta}(\tau, p) \wedge \\ \text{correct_time}_{\delta}(\tau, p) \wedge \neg\psi(\tau, p) \end{aligned}$$

By Corollary 5, a trace $\tau' = (t'_1, e'_1)^{id'_1} \dots (t'_n, e'_n)^{id'_n} \in \llbracket \mathcal{P} \rrbracket_d$ exists such that:

- $\forall c \in \text{Agent}: \pi(\tau_c) = \pi(\tau'_c)$.
- $e'_p = e_p$ and $\{e_1, e_2, \dots, e_p\} = \{e'_1, e'_2, \dots, e'_p\}$
- $\text{run}(\tau', id'_p) = (t'_{p_1}, e'_{p_1})^{id'_p} \dots (t'_{p_k}, e'_{p_k})^{id'_p} \wedge$

$$\begin{aligned} \forall j \in \{p_1, \dots, p_k\}: e'_j = \text{clock}_a(x, y) \wedge \forall z: p_x < z < p_y \\ \implies d(\text{actor}(e'_z), a) \leq \delta \end{aligned}$$

Let \sim be an equivalence relation such that $d(c, a) \leq \delta \iff c \sim a$. We deduce that `dist_attacker` $_{\pi}(\pi(\tau'), p) \wedge \text{correct_time}_{\pi}(\pi(\tau'), p)$ is true by definition, and that $\neg\psi(\pi(\tau'), p)$ is also true given that ψ is *robust*. So $\pi(\tau') \in \llbracket \mathcal{P} \rrbracket_{\pi}$ is an attack trace, as was needed. \square