Verifying List Swarm Attestation Protocols

Jay Le-Papin
Brijesh Dongol
Helen Treharne
Stephan Wesemeyer
jay.le-papin@surrey.ac.uk
b.dongol@surrey.ac.uk
h.treharne@surrey.ac.uk
s.wesemeyer@surrey.ac.uk
University of Surrey
Guildford, Surrey, UK

ABSTRACT

Swarm attestation protocols extend remote attestation by allowing a verifier to efficiently measure the integrity of software code running on a collection of heterogeneous devices across a network. Many swarm attestation protocols have been proposed for a variety of system configurations. However, these protocols are currently missing explicit specifications of the properties guaranteed by the protocol and formal proofs of correctness. In this paper, we address this gap in the context of *list swarm attestation protocols*, a category of swarm attestation protocols that allow a verifier to identify the set of healthy provers in a swarm. We describe the security requirements of swarm attestation protocols. We focus our work on the SIMPLE+ protocol, which we model and verify using the TAMARIN prover. Our proofs enable us to identify two variations of SIMPLE+: (1) we remove one of the keys used by SIMPLE+ without compromising security, and (2) we develop a more robust design that increases the resilience of the swarm to device compromise. Using TAMARIN, we demonstrate that both modifications preserve the desired security properties.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**; **Mobile and wireless security**; **Formal methods and theory of security**.

KEYWORDS

Swarm Attestation; Tamarin Prover; Security Properties; Verification; Internet of Things

ACM Reference Format:
Jay Le-Papin, Brijesh Dongol, Helen Treharne, and Stephan Wesemeyer. 2023. Verifying List Swarm Attestation Protocols. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '23), May 29-June 1, 2023, Guildford, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3558482.3581778

1 INTRODUCTION

The pervasiveness of IoT devices in modern applications, including in critical infrastructure, means that ensuring and verifying their security is of vital importance. Because such devices are typically low-powered, resource-constrained machines, they are often incapable of running sophisticated anti-malware programs (such as those found on personal computers). Instead, the integrity of software running on an untrusted device (the *prover*) is often provided via *remote attestation*, which allows a device (the *verifier*) to attest software across a network [32].

Many modern systems comprise *swarms* of devices [6]. Unfortunately, traditional remote attestation protocols scale poorly as the communication and computation is centralised at the verifier. Therefore, starting with SEDA [6], researchers have recently started focusing on *swarm* or *collective* remote attestation (CRA) protocols [4, 27] that generalise remote attestation by distributing the work across the network, allowing a large number of provers to be attested efficiently.

CRA protocols distribute the burden of attestation by having provers communicate with one another to disperse the cost of computation throughout a swarm, and aggregate results so the verifier only needs to communicate with one or a few provers to attest the entire swarm. Some multi-prover schemes have each prover producing an individual report, which is either sent directly to the verifier [1] or forwarded to the verifier by other provers [10]. Most involve varying degrees of report *aggregation*; provers wait for other provers to produce and send reports, which are then collated into a single report before being forwarded onwards [3, 5, 6, 30]. Some protocols only include verifiers and provers as roles, while others separate *aggregators* into their own role [2].

While many new CRA protocols have been developed [4, 27], there are few existing works that address the *formal verification* of such protocols. Aman and Sidkar [1] verify authentication and secrecy in the ATT-Auth protocol using ProVerif [8]. Although ATT-Auth supports multiple node attestation, this does not involve any report aggregation. The verifier communicates directly with each prover to be attested. However, the verifier uses a more efficient algorithm to decide whether provers are healthy, which improves scalability.

In this paper, we address the lack of formal verification around swarm attestation with the following set of contributions.

(1) We formalise a set of generic security properties for a subset of CRA protocols, which, in the context of CRA, have only been informally stated in prior papers [10]. Additionally, we propose a new property, *Result Timeliness*, which has not been considered previously.
(2) Using the SIMPLE+ [5] swarm attestation protocol as a case study, we encode both the protocol and the security properties in the Tamarin Prover (TAMARIN) [31]. This bounded analysis indicates that SIMPLE+ satisfies all its stated security guarantees under its original threat model but fails to satisfy almost all of them in a stronger but arguably more realistic threat model.
(3) As a result of this analysis, we propose two variants of the SIMPLE+ protocol. The first variant shows that the original protocol can be simplified without loss of security under its original threat model. The second variant of SIMPLE+ preserves the important security properties in the stronger threat model.

As far as we are aware, this is the first paper to tackle formal verification of CRA involving report aggregation at the protocol level, as well as the first formal definitions of security guarantees for a general subset of CRA protocols.

Overview. In Section 2, we provide some background to this paper via an introduction to CRA protocols using the SIMPLE+ swarm attestation protocol as an example. In Section 3, we define a framework for defining security properties for a subset of CRA protocols and describe its instantiation in terms of SIMPLE+. In Section 4, we describe the SIMPLE+ TAMARIN model, different threat models and the verification results. In Section 5, we present our modifications to SIMPLE+ that a) reduces the number of keys that it relies on and b) makes it more resilient to stronger attackers. Using TAMARIN, we model these protocol changes to SIMPLE+ as well as verifying their correctness and security, showing the strength of our approach. Related work and conclusions are described in Section 6 and Section 7, respectively.

Auxiliary Materials. The TAMARIN models for the SIMPLE+ protocol and its two variants are supplied as auxiliary material [28].

2 BACKGROUND

In this section, we define what is meant by a swarm attestation protocol (Section 2.1) and describe the SIMPLE+ protocol [5] (Section 2.2).

2.1 Swarm Attestation

A *swarm attestation protocol* is a protocol that allows a remote verifier to measure the software state of devices (referred to as *provers*) in a swarm [5, 6].

If a prover's software state (alternatively called software configuration) is as expected, the prover is described as *healthy*; having an unexpected software state makes a prover *unhealthy*. In this paper, we refer to the condition of being healthy or unhealthy as a prover's *status*. A CRA protocol can either attest the *overall* software integrity of the swarm, integrity of *individual* devices, or something in between. While some swarm protocols may also attest other aspects, e.g., the *network topology* [10], in this paper, we study protocols that exclusively attest the software state of provers in a swarm, and in which the verifier learns which provers were successfully attested. To the best of our knowledge, most CRA protocols attest software state [3, 5, 6, 11, 23, 30], with other elements sometimes being considered as an addition. For example, SARA [17] attests software state *and* historical information about a prover's interaction with other IoT services.

We define different aspects of swarm attestation that serve to differentiate protocols. Firstly, information about the software state of provers may be aggregated to various degrees. Carpent et al. [10] define *Quality of Swarm Attestation (QoSA)*, which describes how much information about the swarm a verifier receives. These are: Binary QoSA (B-QoSA), List QoSA (L-QoSA), Intermediate QoSA (I-QoSA), and Full QoSA (F-QoSA). In this paper, we focus on L-QoSA protocols, which let a verifier identify which provers in the swarm are healthy and which provers failed to be successfully attested, e.g., SIMPLE+ [5], LISA-α [10], SANA [2], and SeED [23].

Ammar et al. [5], give the following L-QoSA-centric description of a CRA protocol [1]: "*The goal of any swarm attestation scheme is for a verifier v to distinguish between healthy and compromised devices in a swarm S, where limited false positive cases are accepted but not vice versa.*"

Secondly, CRA protocols may differ in whether they are interactive or non-interactive [4]. In an *interactive protocol*, provers attest in response to a request from a verifier, such as the $Attest_{req}$ message in SIMPLE+ discussed below. In a *non-interactive protocol*, provers decide themselves when to attest. This could, for example, be done with a secure clock to trigger attestation at set points in time as with PADS [3].

Finally, attestation protocols can either be self-attestation protocols, or not [4]. In a traditional attestation protocol, a prover sends a digest of its software state to a verifier and the verifier determines whether or not it is healthy. In a self-attestation protocol, a prover determines whether or not it is in a healthy state and informs the verifier of this.

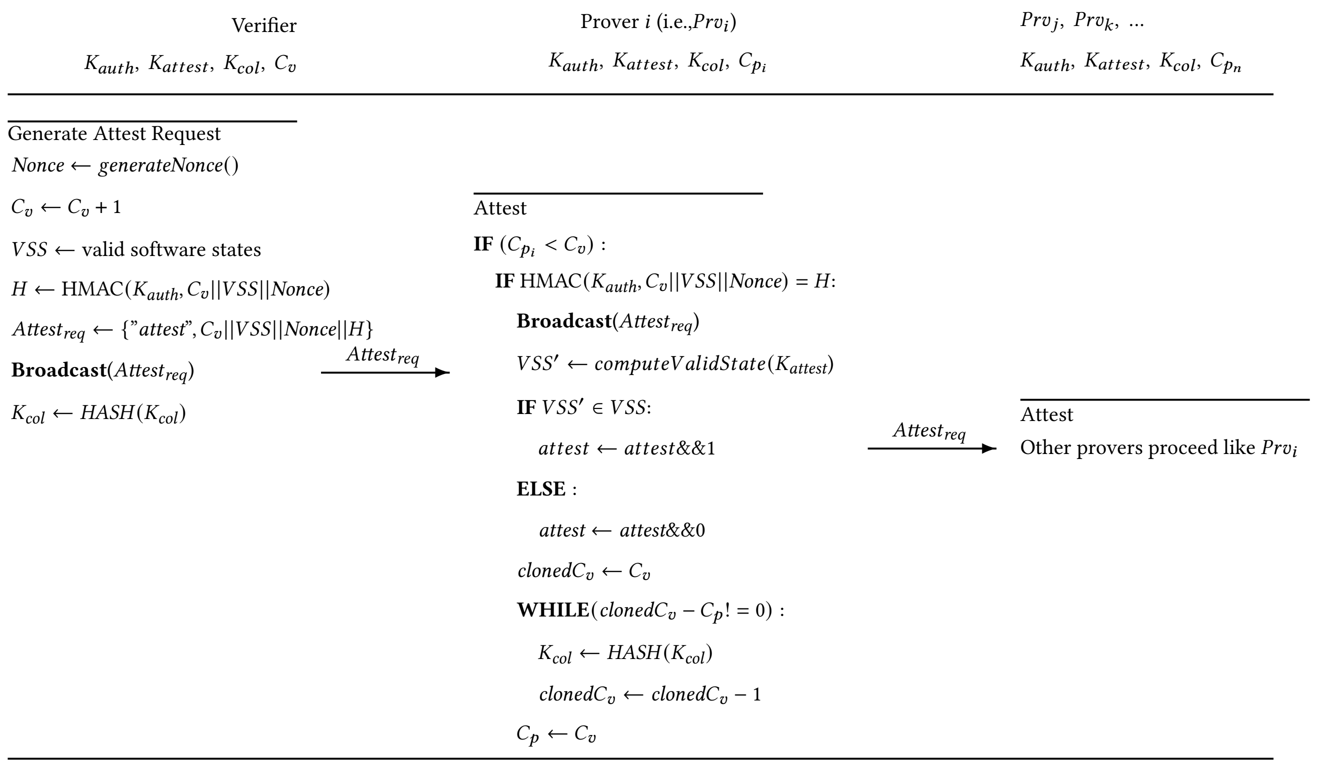
2.2 The SIMPLE+ Protocol

Ammar et al. [5] propose SIMPLE, a remote attestation protocol that is implemented using the $S\mu V$ [13] microvisor to provide strong security guarantees without imposing any hardware requirements. $S\mu V$ isolates a portion of memory and prevents any programs from being loaded which could allow unauthorised access to this region of memory. Attestation code can reside within this region of memory instead of Read-Only Memory (ROM) as used by SMART [18] or TrustLite [25].

SIMPLE+ extends SIMPLE to collective remote attestation, where a single verifier attests an arbitrary number of provers. The protocol is split into three phases as follows:

Phase one – *initialisation*, loads keys and sets initial values.
Phase two – *attest* phase, is triggered by the verifier by broadcasting an attestation request message, $Attest_{req}$, to the swarm. Each prover then rebroadcasts this message before computing (attesting) its own software state and checking it w.r.t. the expected software state(s).

[1]We say this is L-QoSA centric as it does not apply to all CRA protocols. For example, SEDA [6] does not allow a verifier to distinguish between healthy and compromised devices. Instead, it gives the verifier an indication of the collective status of the entire swarm.



Verifier: K_{auth}, K_{attest}, K_{col}, C_v

Prover i (i.e., Prv_i): K_{auth}, K_{attest}, K_{col}, C_{p_i}

Prv_j, Prv_k, ...: K_{auth}, K_{attest}, K_{col}, C_{p_n}

Verifier:

```
Generate Attest Request
$Nonce \leftarrow generateNonce()$
$C_v \leftarrow C_v + 1$
$VSS \leftarrow$ valid software states
$H \leftarrow \mathrm{HMAC}(K_{auth}, C_v||VSS||Nonce)$
$Attest_{req} \leftarrow \{"attest", C_v||VSS||Nonce||H\}$
Broadcast($Attest_{req}$)   $\xrightarrow{Attest_{req}}$
$K_{col} \leftarrow HASH(K_{col})$
```

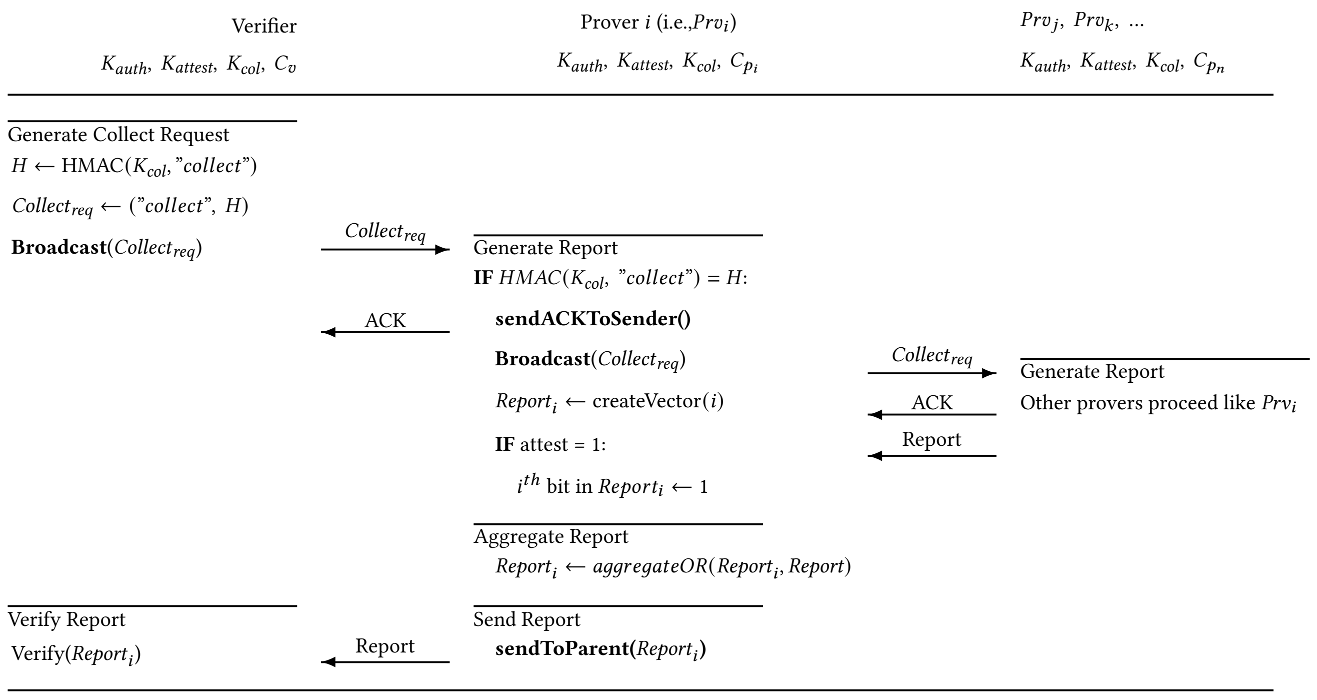
Prover i:

```
Attest
IF ($C_{p_i} < C_v$) :
  IF HMAC($K_{auth}, C_v||VSS||Nonce$) = $H$:
    Broadcast($Attest_{req}$)   $\xrightarrow{Attest_{req}}$
    $VSS' \leftarrow computeValidState(K_{attest})$
    IF $VSS' \in VSS$:
      $attest \leftarrow attest\&\&1$
    ELSE :
      $attest \leftarrow attest\&\&0$
    $clonedC_v \leftarrow C_v$
    WHILE($clonedC_v - C_p != 0$) :
      $K_{col} \leftarrow HASH(K_{col})$
      $clonedC_v \leftarrow clonedC_v - 1$
    $C_p \leftarrow C_v$
```

Prv_j, Prv_k, ...:

```
Attest
Other provers proceed like $Prv_i$
```

Figure 1: SIMPLE+ attest phase



Verifier: K_{auth}, K_{attest}, K_{col}, C_v

Prover i (i.e., Prv_i): K_{auth}, K_{attest}, K_{col}, C_{p_i}

Prv_j, Prv_k, ...: K_{auth}, K_{attest}, K_{col}, C_{p_n}

Verifier:

```
Generate Collect Request
$H \leftarrow \mathrm{HMAC}(K_{col}, "collect")$
$Collect_{req} \leftarrow ("collect", H)$
Broadcast($Collect_{req}$)   $\xrightarrow{Collect_{req}}$
                             $\xleftarrow{ACK}$

Verify Report
Verify($Report_i$)           $\xleftarrow{Report}$
```

Prover i:

```
Generate Report
IF $HMAC(K_{col}, "collect") = H$:
  sendACKToSender()
  Broadcast($Collect_{req}$)   $\xrightarrow{Collect_{req}}$
  $Report_i \leftarrow$ createVector($i$)   $\xleftarrow{ACK}$
  IF attest = 1:   $\xleftarrow{Report}$
    $i^{th}$ bit in $Report_i \leftarrow 1$

Aggregate Report
  $Report_i \leftarrow aggregateOR(Report_i, Report)$

Send Report
  sendToParent($Report_i$)
```

Prv_j, Prv_k, ...:

```
Generate Report
Other provers proceed like $Prv_i$
```

Figure 2: SIMPLE+ collect phase

Phase three – *collect* phase, is triggered by the verifier by broadcasting a $Collect_{req}$ message to the swarm. Upon receiving $Collect_{req}$, each prover sends an acknowledgement back to the sender of $Collect_{req}$ before rebroadcasting $Collect_{req}$. The prover then generates its own report and waits for reports from each prover that it received an acknowledgement from. Once these reports are received, an aggregated report is generated and sent to the prover's parent, i.e., the originator of the $Collect_{req}$ message it received (which may be the verifier), alongside an HMAC to ensure the integrity of the report.

The second and third phase are illustrated in Fig. 1 and Fig. 2, respectively. We explain these phases in detail below.

A prover is initialised with three secrets. K_{auth} and K_{col} are symmetric keys shared with all provers in the swarm, while K_{attest} is a device specific symmetric key. These keys are all shared with the verifier. These keys are stored in a secure region of memory, alongside a counter C_p, an *attest* variable (initially 1), and the prover's identifier p_i. A secure region of memory is one that is protected by a secure architecture [13, 18, 25] that prevents access by any software, which could be malware, other than the attestation protocol.

The attest phase starts with the verifier broadcasting an attest request $Attest_{req}$ message to all neighbouring provers, which authenticate it using their K_{auth} before rebroadcasting. Subsequently, each prover computes a digest of its own software state VSS' and checks that this digest is a member of the set of valid software states VSS sent by the verifier. If it is, the value of the *attest* variable is unchanged. If $VSS' \notin VSS$ then *attest* is set to 0. Once a prover's *attest* value is set to 0, there is no way in this protocol for it to ever be set back to 1 again.

The key K_{col} is updated during every round of the attest phase. This means that there is a unique paring[2] between the round number and the value of K_{col} for that round. If a prover missed one or more rounds of attestation, K_{col} will be hashed multiple times to bring it up to date.

In the collect phase, the collect request $Collect_{req}$ is broadcast, authenticated, and rebroadcast similar to the behaviour of $Attest_{req}$. K_{col} is used for authentication here, meaning that only provers which participated in the most recent round of attestation will be able to participate in this phase.

In the collect phase, every prover instantiates a report as a bitstring of length n, where n is at least equal to the number of provers in the swarm. The i-th bit is set by a prover p_i to be equal to its *attest* value and all other bits are set to 0. When a prover receives a report, it is aggregated with its own report using OR. This process is performed by all provers in the swarm, resulting in a single bitstring that ideally contains the attestation result for every prover. As the default value and the value indicating an unhealthy prover are both 0, SIMPLE+ is not able to distinguish between unhealthy and unresponsive provers.

In SIMPLE+, it is assumed that the attest and collect phases are each executed *atomically*. This means that an attacker cannot infect a device with malware while it is executing the code for either of these phases.

[2]assuming the hash function never collides

3 SECURITY PROPERTIES

In this section we provide a set of security properties that we require of CRA protocols. We present four main properties: *Prover Authentication*, *Verifier Authentication*, *Legitimate Response*, and *Result Timeliness*. Later in Section 4, we evaluate these properties against two threat models, based on a Dolev-Yao attacker: $\mathcal{M}_{noreveal}$ (in which no prover may leak its keys) and $\mathcal{M}_{otherreveal}$ (in which only the considered prover may not leak its keys).

It is clearly important that a CRA protocol allows both a verifier and the provers to authenticate the sender of a message that was received to ensure that an attacker cannot "fool" a verifier by replaying prover messages nor induce a prover to attest unnecessarily. We detail authentication properties in Section 3.2.

Moreover, it is also critical that the result sent by any prover is an accurate reflection of the state or status of that prover at the time the report is generated. Additionally, a verifier should be able to verify the freshness of an attestation result. Properties of this nature are covered in Section 3.3.

In Section 3.4 we discuss the minimum security guarantees we would expect an L-QoSA CRA protocol to provide.

To define a generic set of properties that can apply to many L-QoSA CRA protocols, we first propose a formalisation to describe these protocols in Section 3.1. In Section 3.5, we show how our formalisation applies to a real CRA protocol, SIMPLE+. In Appendix A we do this for three other L-QoSA CRA protocols.

3.1 Formalising Swarm Attestation Protocols

In this section, we lay the groundwork for defining the security properties of CRA protocols. We present several functions that model the different steps of a CRA protocol from generating digests to concluding whether a prover is healthy or unhealthy.

3.1.1 Defining a Swarm. As we have discussed, in a CRA protocol, there are two roles: the prover, prv, and the verifier, vrf. CRA protocols are characterised by the participation of multiple provers in a single run of the protocol; we define the set of all provers that *may* participate in the protocol together as P, and we represent a swarm as the pair (vrf, P). For the purposes of this paper, we focus on CRA protocols with a single verifier.

3.1.2 Defining State and Status. We let Σ denote the set of all possible software states that a prover can be in. The purpose of an attestation protocol is to determine the *Status* of the provers. If a prover is in a state that the verifier considers acceptable, it has the *healthy* status. Otherwise, it is *unhealthy.*

$$Status \mathrel{\widehat{=}} \{healthy, unhealthy\}$$

While the real software state of a prover at any given time must either be a *healthy* or an *unhealthy* one, the conclusion drawn by the verifier about the status of a prover can be different. For example, PADS [3] and LISA-α [10] both allow the status of a prover to be *unknown.* Conversely, SIMPLE+ does not allow *unknown*, and all unreachable provers will instead be recorded as *unhealthy*. We refer to this conclusion as the observed status.

3.1.3 Defining Digest. At the core of a CRA protocol is the generation of a *digest* from a prover's state. The set of all digests for a software state is given by

$$Digest \mathrel{\hat{=}} \{h(\sigma) \mid \sigma \in \Sigma\}$$

where we assume that h denotes the function to produce the digest for a given state σ. Depending on the protocol, h may be a hash function [10], or a function producing a binary output, such as 0 signifying that prv is unhealthy and 1 signifying that prv is healthy [5]. This idea of leaving some components such as h abstract is inspired by Asokan et al. [6], who state that a CRA protocol should "be independent of the underlying integrity measurement mechanism used by devices in [the swarm]".

3.1.4 Representing Time. Since a prover may be attested more than once, we must record the time at which attestation occurs. A CRA protocol may enable a verifier to know this time precisely, e.g., using real-time clocks, or it may be a more abstract notion such as round numbers. In our framework, we use T for the protocol's representation of time, while an instant in real time is a member of $\mathbb{R}_+$. The exact meaning of T is left abstract, and instantiated on a per-protocol basis, as shown in Section 3.5.

3.1.5 Remaining Aspects of Framework. With Σ, $Digest$, P, T, and the representation of real-time described, we now present the remaining aspects of our formalisation.

The software configuration $\sigma \in \Sigma$ of a prover $prv \in P$ at a given real time $\tau \in \mathbb{R}_+$ is a member of the set:

$$\Gamma \mathrel{\hat{=}} P \times \Sigma \times \mathbb{R}_+$$

An attestation result is generated (using the function μ below) from a configuration at a particular time:

$$Result \mathrel{\hat{=}} P \times Digest \times T$$

The abstract time $t \in T$ may be used by an agent to determine an interval in which the real attestation time $\tau \in \mathbb{R}_+$ lies, using the function:

$$bound : (vrf \cup P) \times T \longrightarrow Intv$$

where $Intv \subseteq \mathbb{P}(\mathbb{R}_+)$ is the set of all intervals over the non-negative reals. We assume that a result for a prover, software state, and time $(prv, \sigma, \tau) \in \Gamma$ using the following function:

$$\mu : \Gamma \longrightarrow Result$$

where $\mu((prv, \sigma, \tau)) = (prv, h(\sigma), t)$, and $\tau \in bound(prv, t)$

Finally, a result can be judged to indicate a status using

$$\rho : Result \longrightarrow Status$$

3.2 Authentication Properties

Authentication is typically defined using Lowe's hierarchy [29]. Our definitions for *Prover Authentication* and *Verifier Authentication* are CRA-specific instantiations of standard authentication properties.

We assume that all agents are initialised (e.g., with their private keys) before the protocol begins. In our properties we repeatedly refer to an *honest* agent. An honest agent is one that operates faithfully to the protocol specification and does not leak any secrets. It is important to emphasise for CRA that an honest agent does not necessarily have a healthy software state. So long as the root of trust (such as SMART [18]) is not compromised, an unhealthy prover should still execute the attestation protocol according to the specification.

3.2.1 Prover Authentication. In a CRA protocol, authentication of provers ensures the verifier, vrf, knows that a result it receives for any prover, $prv \in P$ matches the result that prv sent. We therefore require vrf to have at least Non-Injective Agreement [29] with provers, resulting in the following definition:

Definition 3.1 (Prover Authentication (status) (PA(s))). Let (vrf, P) be a swarm and (prv, x, t) an attestation result. We say that a CRA protocol for (vrf, P) satisfies PA(s) iff, whenever an honest verifier vrf completes an attestation procedure with an honest prover prv, believing the status of prv to be s at t, then there is an execution of prv producing the result (prv, x, t) and $\rho((prv, x, t)) = s$.

Prover Authentication is parameterised by the status s. Some protocols may provide *Prover Authentication* for any status s. Unfortunately, many protocols, such as SIMPLE+, do not differentiate between unresponsive provers and unhealthy provers. In this case, *Prover Authentication* (*unhealthy*) can be violated by a verifier that thinks a prover is unhealthy, but where the prover is actually healthy, and the response was simply lost.

3.2.2 Verifier Authentication. The computation involved in attestation is usually quite expensive. For example, Carpent et al. [5] state that, for SIMPLE+, computing an HMAC of the entire flash memory of a MicroPnP IoT platform[3] takes 17.41 seconds. If an attacker can induce provers to repeatedly attest themselves by replaying attestation requests, they can easily perform a DoS attack without needing much bandwidth (a single request can cause the entire swarm to attest). To prevent this, some CRA protocols seek to provide *Verifier Authentication*, which requires Injective Agreement [29].

Definition 3.2 (Verifier Authentication (VA)). Let (vrf, P) be a swarm. We say that a CRA protocol for (vrf, P) satisfies *Verifier Authentication* iff for all provers $prv \in P$, whenever prv completes a run of the protocol, apparently having received one or more messages $m_1, ..., m_k$ from the verifier vrf, then vrf has previously been running the protocol, sending messages $m_1, ..., m_k$ and that prv and vrf agree on the contents of the message(s) $m_1, ..., m_k$. Additionally, for each $m_i \in \{m_1, ..., m_k\}$, prv accepts m_i exactly once.

For example, as shown in Section 4.4, in SIMPLE+, *Verifier Authentication* is achieved in the attest phase through the shared K_{auth} key and a monotonically increasing counter. This prevents a network adversary from forcing a prover to attest itself more times than the verifier intends.

3.3 Result Properties

Authentication alone is insufficient for ensuring that a remote attestation protocol achieves its aim. We detail two more properties in this section: *Legitimate Response* and *Result Timeliness*.

3.3.1 Legitimate Response. While *Prover Authentication* ensures that the verifier and prover agree on an attestation result, it does not say that the prover sent the result it was supposed to; the status indicated by a prover's attestation result should be in accordance with its actual software state. To capture this, we define *Legitimate Response*.

[3]With an 8-bit AVR ATmega 1284p microcontroller running at 10MHz with 16kB of SRAM and 128kB of Flash

Like *Prover Authentication*, *Legitimate Response* is also parameterised by the status of the prover.

Definition 3.3 (Legitimate Response (status) (LR(s))). Let (vrf, P) be a swarm. A CRA protocol for (vrf, P) satisfies LR(s) if, for all provers $prv \in P$, whenever an honest prv sends a report (prv, x, t) and $\rho((prv, x, t)) = s$, then prv has a software state σ at some $\tau \in \mathbb{R}_+$ such that $\mu((prv, \sigma, \tau)) = (prv, x, t)$.

In other words, there is a time τ at which the prover prv computed its report, $\mu((prv, \sigma, \tau)) = (prv, x, t)$. Unless the verifier, *vrf*, and prv have perfectly synchronised clocks, t will not be precise enough to derive τ from. For example, t could be a round number, from which you can derive a range of times $bound(vrf, t)$, giving the interval where *vrf* is in round t. It could also be a timestamp from a real-time clock, such as in SeED [23], in which case the range of $bound(vrf, t)$ will be relatively small, limited only by the accuracy of the real-time clock used (see Appendix A.3).

3.3.2 Result Timeliness.

Prover Authentication ensures that any prover prv and the verifier *vrf* agree on the value of t in any given result. For this to be meaningful, the prover and verifier must have some level of agreement over what t means. We define *Result Timeliness*, where it must be the case that for any result (prv, x, t) that *vrf* accepts, $\tau \in bound(vrf, t)$. In other words, the attestation result was produced some time during the interval that *vrf* expects. This is not necessarily guaranteed by the definition of μ, which only states that $\tau \in bound(prv, t)$.

Definition 3.4 (Result Timeliness (RT)). Let (vrf, P) be a swarm. A CRA protocol for (vrf, P) satisfies *Result Timeliness* if, for all honest provers $prv \in P$, whenever *vrf* accepts an attestation result $(prv, h(\sigma), t)$, produced at τ, i.e., by $\mu((prv, \sigma, \tau))$, then we have $\tau \in bound(vrf, t)$.

3.4 Result Accuracy

For a verifier to be able to trust an attestation result it receives, indicating a status s, the protocol must satisfy all of $PA(s)$, $LR(s)$, and RT. Without RT, the verifier cannot be assured of when the received attestation result was produced. Without $LR(s)$, an honest prover can incorrectly report its state or digest. And, without $PA(s)$, the previous two properties are meaningless as an adversary can replace the result with whatever values it likes. We therefore define *Result Accuracy* as the conjunction of these properties.

Definition 3.5 (Result Accuracy(status) (RA(s))). Let PA(s) be *Prover Authentication* and LR(s) be *Legitimate Response*. A protocol satisfies RA(s) if it satisfies both PA(s) and LR(s), i.e.,

$$RA(s) \equiv PA(s) \wedge LR(s) \wedge RT$$

A protocol which allows false positives, i.e., one that allows *healthy* provers to be recorded as *unhealthy*, but not false negatives, i.e., one that does not allow *unhealthy* provers to be recorded as *healthy*, will satisfy *Result Accuracy* (*healthy*) but not *Result Accuracy* (*unhealthy*). While it would be ideal for neither false positives nor false negatives to be accepted, a CRA can be regarded as being secure if it allows false positive results, but never allows false negative results, i.e., *an L-QoSA CRA scheme may be considered secure iff it satisfies Result Accuracy (healthy).*

3.5 Instantiation of Our Framework

In this section, we demonstrate how the functions defined in Section 3.1 apply to a real-world CRA protocol.

SIMPLE+ is a self-attestation protocol, which relies on the provers deciding for themselves what status their software state has, and then sending this status to the verifier. The statuses *healthy* and *unhealthy* in SIMPLE+ are represented with a 1 bit and a 0 bit, respectively.

- $Digest \mathrel{\hat{=}} \{0, 1\}$

As is common amongst CRA protocols, SIMPLE+ uses a monotonically increasing counter to uniquely identify rounds of attestation. Every prover and the verifier stores a counter value, C_p and C_v respectively, as identified in Fig 1 and Fig 2. Here, in our formal notation C_p is denoted by C_{prv}.

- $T \mathrel{\hat{=}} \mathbb{N}$
- $\mu((prv, \sigma, \tau)) \mathrel{\hat{=}} (prv, h(\sigma), C_{prv})$, where C_{prv} is prv's counter value at τ, $h(\sigma) = 1$ if σ is considered a valid state in round C_{prv}, and $h(\sigma) = 0$ otherwise.
- $bound(a, C_a)$ returns the interval in which agent a's counter is equal to C_a, where a can be a prover or the verfier.

When the verifier receives an attestation result, it can examine it to determine whether it corresponds to a *healthy* or *unhealthy* status. Firstly, any results where the counter does not match *vrf*'s counter are discarded. Then *vrf* examines the digest to determine whether it corresponds to a healthy or unhealthy state, as shown.

- $\rho((\_, 1, C_v)) = healthy$
- $\rho((\_, 0, C_v)) = unhealthy$

4 MODELLING SIMPLE+ IN TAMARIN

In Section 3 we introduced *Prover Authentication*, *Verifier Authentication*, *Legitimate Response*, *Result Timeliness*, and *Result Accuracy*. In SIMPLE+, we are interested in all these properties.

The full list of properties we analyse in SIMPLE+ is *Prover Authentication* (Healthy), *Prover Authentication* (Unhealthy), *Legitimate Response* (Healthy), *Legitimate Response* (Unhealthy), *Result Timeliness*, *Result Accuracy* (Healthy), *Result Accuracy* (Unhealthy), and *Verifier Authentication*. These are abbreviated as *PA(H)*, *PA(U)*, *LR(H)*, *LR(U)*, *RT*, *RA(H)*, *RA(U)*, and *VA*, respectively.

In this section, we use TAMARIN [31] to verify whether these properties hold for SIMPLE+. We motivate our modelling choices in Section 4.2, the threat models in Section 4.3 and show our analysis of the security properties in Section 4.4.

4.1 The Tamarin Prover

TAMARIN is a state-of-the-art protocol verification tool for symbolic modelling. It supports unbounded verification, mutable global state, and flexible user-defined equational theories. TAMARIN models are transition systems over a multi-sorted term algebra, operating on the semantics of multiset rewriting logic [16]. Security properties to be analysed are expressed using lemmas in a guarded first-order logic quantifying over variables inside facts declared in the model. The tool offers automatic verification succeeding in many cases, as well as an interactive verification mode for manual proof tree traversal. The tool provides both proofs, and disproofs by counterexample. Moreover, TAMARIN supports modelling all communication using the standard Dolev-Yao (DY) [15] adversary.

When processing a model, TAMARIN operates under the *perfect cryptography assumption.* In other words, the DY adversary can only decrypt messages for which she knows the key, she cannot brute-force a key or mathematically derive it, nor can she "undo" a hash of a message or find a hash collision. However, she can manipulate terms symbolically, i.e., build up new terms by combining existing ones or computing new ones using her knowledge of the protocol.

Due to its support for unbounded verification, TAMARIN proofs are not guaranteed to terminate or can take a long time to complete. TAMARIN thus supports various heuristics to cover the search-space yielded by the constraint-solving problem underlying the analysis to improve its performance. These heuristics determine which rules should be prioritised during the proof search. When these built-in heuristics are not sufficient, a user can also create a file to prioritise different rules yielding a bespoke search heuristic. This is called an *oracle* and can be passed to TAMARIN [33] to automate these user-guided proofs.

Writing an oracle is non-trivial process, though, and there is no guarantee of finding an optimal oracle with certainty. Restrictions are another way of limiting the search-space of a model and thus increasing the chances of a proof in TAMARIN terminating more quickly. However, a model with restrictions might also limit the potential attacks and thus requires careful reasoning to ensure its results continue to be applicable to the unrestricted case.

All the TAMARIN models for the SIMPLE+ protocol and its variants and their respective oracles can be found at [28].

4.2 Modelling Choices

We faithfully represent the SIMPLE+ protocol specification [5] in our TAMARIN models with a statically defined number of rounds, provers, and topology. The naming convention used in these models follows closely the names of the commands as well as the variable names and labels used in Fig. 1 and Fig. 2. In this section, we highlight our key modelling decisions, such as modelling the topology of the swarm, software state, and bounding the number of provers.

The main challenges we faced in writing a TAMARIN model were: how to model the swarm topology when using a DY attacker; encoding *Result Timeliness* as a lemma, because there is no direct translation of the property into first order logic; and dealing with the unbounded nature of CRA protocols w.r.t. the number of rounds of attestation and the number of provers being attested.

4.2.1 Model Bounding. The number of provers in a swarm is theoretically unbounded. If a naive model of the protocol is pursued, then when searching for a counterexample to lemmas, TAMARIN will endlessly add new provers and never terminate. In line with other works that deal with a theoretically unbounded number of protocol participants [7, 20], we statically bound the number of provers and only explore models with two provers in this paper [4].

Similarly, the maximum value of rounds must also be restricted. In this paper, we focus on models restricted to one or two rounds. Rounds in SIMPLE+ are indicated by the counter value, which we model using a multiset of constants as in Wesemeyer et al. [34]. We add a restriction to prevent this counter from growing larger than a value indicating one or two rounds, depending on the model.

[4]It may be possible to generate proofs for an unbounded number of provers using TAMARIN's inductive proof mode but this is beyond the scope of this paper.

4.2.2 Topology. In SIMPLE+'s attest phase, a verifier broadcasts the $Attest_{req}$ message, then provers which receive the request re-broadcast it. In our model, communication range is not modelled. So, the verifier broadcasts an attest request, then the adversary can deliver it to all provers, meaning we do not need provers to rebroadcast the request.

In the collect phase, the (re-)broadcast of collect requests, combined with ACK messages, serves a purpose beyond ensuring all provers receive the message; it is used to establish parent-child relationships amongst entities in the swarm, deciding which provers aggregate reports from whom. While in the protocol this topology is established dynamically in each collect phase, we fix it in our model for performance reasons. Therefore, all that is necessary is for provers to receive the collect request, and it does not matter who from, just like in the attest phase. Fixing the topology means that the topology unfortunately cannot change between rounds. In our model, the prover with id '0' will always be the aggregator.

4.2.3 Timeouts. In SIMPLE+, a parent would be expected to wait a certain amount of time for a response from its children, before proceeding with the protocol regardless of whether they have all yet responded. In our model, we implicitly model timeouts by providing a non-deterministic choice between receiving a report to aggregate or proceeding to the next step of the protocol.

4.2.4 Modelling State and Atomicity. We represent the state of a prover with a constant. This is either the value 'healthy' or 'unhealthy'. While in reality there may be multiple states that correspond to either 'healthy' or 'unhealthy', we do not think that a more fine-grained modelling than this is necessary. What is important in the protocol is for provers and the verifier to agree that 'healthy' corresponds to a healthy status, while 'unhealthy' corresponds to an unhealthy status.

We have a rule, malware_infection, to change the state of a 'healthy' prover to 'unhealthy'. This rule cannot be applied to a prover which is in the middle of its attest or collect phase, reflecting the atomicity assumption.

4.2.5 Reports. We choose to faithfully represent the report in TAMARIN as a sequence of ones and zeroes representing the bit-string used in SIMPLE+. To model aggregation, we define two constants `zeroo` and `onee` and a function `or`, which we use to define a bitwise OR operation. Assuming `in_1, in_2, old_1, old_2` $\in$ {`onee`, `zeroo`}, we have:

```
functions: or/2, zeroo/0, onee/0

report_in = <in_1, in_2>
old_report = <old_1, old_2>
new_report = <or(in_1, old_1),or(in_2, old_2)>
```

4.2.6 Key Reveals. Our model has an arbitrary key reveal rule which reveals a prover's keys to the adversary. The authors of SIMPLE+ assume only a software attacker which is prevented from accessing any secret keys by the trusted architecture used by the protocol. The presence of a key reveal could, however, reflect an attack outside of this assumption, such as a physical attack. Due to the lack of any secure hardware, the authors of SIMPLE+ state that $S\mu V$ is particularly vulnerable to physical attacks, which therefore makes this an interesting scenario to explore. We go into further detail in Section 4.3.

4.2.7 *Lemmas in TAMARIN.*

We encode our properties defined in Section 3 as TAMARIN lemmas (e.g., Fig. 3). We also have a collection of executability lemmas to ensure the model functions as expected by checking that certain traces are possible. For example, checking a complete run of the protocol exists, or that it is possible for any branch in the model to be taken.

The (Prover and Verifier) Authentication lemmas are simple variations of the (non-)injective agreement lemmas given in the TAMARIN manual. *Legitimate Response* is straightforward and essentially states "If a prover says it is healthy, then the rule to corrupt that prover's software state has not fired at any point in the past".

More interesting is the lemma, shown in Fig. 3, that guarantees Result Timeliness. For Result Timeliness, we need to show that if *vrf* accepts an attestation result produced at τ, then (1) τ is not less than the lowest value in $bound(vrf, t)$, and (2) τ is not greater than the greatest value in $bound(vrf, t)$. (1) is shown by conjunct (A), which shows that *prv*'s counter when attesting can never be greater than *vrf*'s counter. This puts a lower bound on the attestation time, showing that an honest prover will never produce an attestation result for round c before *vrf* begins round c. (2) is shown by conjunct (B), which shows that a verifier will not accept results claiming to be for previous rounds. This implies the upper bound, as an attestation result for round c can only be registered up until the moment that *vrf* proceeds to round $c + 1$.

4.2.8 *Other.*

We use restrictions to ensure that provers and the verifier have unique identities with no overlap. Lastly, SIMPLE+ is divided into an attest and collect phase. We include both in the same model and they are executed sequentially. In the real protocol, it is possible for multiple executions of the attest phase to occur before an execution of the collect phase.

4.3 Threat Models

We consider two threat models using a Dolev-Yao adversary. In the first, $\mathcal{M}_{noreveal}$, no arbitrary key reveals are permitted. This represents the threat model that Ammar et al. [5] consider. The only way an attacker would be able to gain access to a key is if it was leaked as part of the protocol. The second, $\mathcal{M}_{otherreveal}$, allows arbitrary key reveals from any prover thus allowing us to verify if a swarm's security properties hold for an honest prover even if one or more other provers are dishonest and leak their keys to the attacker.

SIMPLE+ is designed to withstand software-only attacks and relies on the $S\mu V$ [13] to prevent key leakage to a software-only attacker. Therefore, the leakage of keys presumes either a failure of the $S\mu V$ or a non-software-only attack. Physical attacks are considered by CRA protocols [22, 26], and is discussed by Ammar et al. [5], so we consider this a reasonable scenario to investigate.

	Threat Model	
Property	$\mathcal{M}_{noreveal}$ (T_1, T_2)	$\mathcal{M}_{otherreveal}$ (T_1, T_2)
LR(H)	✓ (13.4, 483.8)	✓ (15.5, 1292.1)
LR(U)	✓ (2.7, 77.6)	✗ (2.7, 7.1)
PA(H)	✓ (16.1, 10616.2)	✗ (1.5, 5.5)
PA(U)	✗ (29.3, 1126.1)	✗ (1.6, 1.6)
VA (Attest)	✓ (4.6, 228.4)	✗ (1.4, 2.2)
VA (Collect)	✓ (35.6, 1738.3)	✗ (5.0, 9.7)
RT	✓ (4.2, 115.5)	✗ (4.1, 17.0)

Table 1: Verification results of SIMPLE+, where T_k = time taken (in seconds) to verify the property with the maximum number of rounds is bounded by k

4.4 Verification Results for SIMPLE+

In this section we detail our analysis of SIMPLE+ given our two threat models, $\mathcal{M}_{noreveal}$ and $\mathcal{M}_{otherreveal}$, and discuss the security implications. We captured the properties given in Section 3 as first-order logic lemmas in TAMARIN.

We have four lemmas to capture the properties *PA(H)*, *PA(U)*, *LR(H)*, and *LR(U)*. Then, we have two lemmas for *VA*, one for the attest phase and one for the collect phase. These are referred to as *VA* (Attest) and *VA* (Collect), respectively. The verification results of these properties for SIMPLE+ are shown in Table 1. Recall that RA is the conjunction of PA, and LR, so we do not verify it separately.

Under the $\mathcal{M}_{noreveal}$ scenario, the only property to be falsified is *PA(U)*. This is expected, as a verifier cannot distinguish between an unhealthy prover and an unresponsive prover. The adversary can block messages from a healthy prover, forcing the verifier to register a healthy prover as unhealthy.

Conversely, under $\mathcal{M}_{otherreveal}$, every authentication property fails. Because all keys except K_{attest}, which is not used for authentication, are shared by all provers in a swarm, the compromise of keys from any prover allows an attacker to impersonate any agent in the swarm - causing every desired authentication property to be violated (*PA(H)*, *PA(U)*, and *VA*)[5].

The only property that is not violated under $\mathcal{M}_{otherreveal}$ is *LR(H)*. SIMPLE+ is a self-attestation protocol which means that the verifier tells the prover what state(s) it expects the prover to be in, and the prover responds by confirming whether or not it is in that state. The expected software state is cryptographically tied to K_{attest}, which is a key unique to each prover and shared only with the verifier. Even under $\mathcal{M}_{otherreveal}$, an attacker cannot forge an expected software state that an uncompromised prover will accept, maintaining the *Legitimate Response* (*healthy*) property.

The violation of *LR(U)* follows from the violation of authentication properties. While the adversary cannot generate a VSS that will be seen as legitimate due to the unique K_{attest} key per prover, it can replace VSS with garbage and produce a valid HMAC for this garbage using the leaked swarm-wide K_{auth}. This means that even if the prover is in a state that the verifier would consider healthy, its state won't match any member of VSS, causing the prover to believe it is unhealthy.

[5]Recall that properties parameterised by (H) apply when a healthy prover is under consideration, and properties parameterised by (U) apply when an unhealthy prover is considered.

```
lemma correct_result_timeliness:
  "All V P c att #t01 . Commit_ColRes(V, P, att, c) @ #t01 ==>
  ( // (A) prv's c when attesting can never be less than vrf's c (assuming c never decreases,
    //     which it does not)
      (All #t02 . Commit_AttReq(P, V, c) @ #t02 ==>
      (Ex #t03 . Running_AttReq(V, P, c) @ #t03 & #t03 < #t02)) &
    // (B) vrf will never accept a result for round lower than its current round
    //     (as indicated by c)
      (All #t02 . Commit_ColRes(V, P, att, c) @ #t02 ==>
      not(Ex #t03 . Running_AttReq(V, P, c+'1') @ #t03 & #t03 < #t02))
  )
| (M_noreveal: Ex prv2 K #t02 . Key_Reveal(vrf, prv2, K) @ #t02 & #t02 < #t01
   M_otherreveal: Ex K #t02 . Key_Reveal(vrf, prv, K) @ #t02 & #t02 < #t01)"
```

Figure 3: Result Timeliness

The use of unique keys for K_{attest} does not appear to improve the security of the protocol. While *LR(H)* holds thanks to K_{attest}, it alone is not a useful security guarantee. Even if a prover *prv* is guaranteed to send the correct result (iff the correct result is '1'), this result will not necessarily reach *vrf* as the adversary can impersonate *prv* and send a forged result in its place, so long as another prover's keys have been compromised.

Our analysis suggests that SIMPLE+ is a secure protocol, according to the properties we have defined, under the assumptions that it was designed for. However, it is a very brittle protocol and, should those assumptions ever not hold, and an attacker is able to access the keys from a prover, the compromise of one prover will compromise the entire swarm.

5 MODIFYING SIMPLE+

The TAMARIN model developed in Section 4 provides a basis for modelling and verifying variants of SIMPLE+. First, in Section 5.1, we investigate the purpose of K_{attest}, showing that it can be removed without compromising the security of the protocol. Second, we observe that SIMPLE+ is insecure against a threat model that allows even a single dishonest prover since the keys K_{col} and K_{auth} are shared by all devices in the swarm. This is problematic for any scenario in which the physical security of provers cannot be guaranteed, which is a trade-off made by the authors [5]. Therefore, in Section 5.2, we propose a stronger protocol and evaluate its advantages and disadvantages w.r.t. SIMPLE+.

5.1 Removing K_{attest}

A curious aspect of the SIMPLE+ protocol is that the key K_{attest} does not seem to have a practical purpose. To test this hypothesis, we developed a variant of SIMPLE+ (called SIMPLE∗), which is SIMPLE+ with K_{attest} removed and analysed its security guarantees.

The negligible impact of K_{attest} is demonstrated in Table 2. The only difference in security guarantees is the lack of *LR(H)* under $\mathcal{M}_{otherreveal}$. Alone, *LR(H)* is not a useful property as, without *PA(H)*, the adversary can forge a response for the prover which lies about its software state. The key security goals for remote attestation, Healthy Report Correctness and *VA*, are unaffected.

	Threat Model	
Property	$\mathcal{M}_{noreveal}$ (T_1, T_2)	$\mathcal{M}_{otherreveal}$
LR(H)	✓ (11.0, 431.1)	✗ (6.0, 22.5)
LR(U)	✓ (2.0, 64.5)	✗ (2.3, 5.8)
PA(H)	✓ (12.1, 10091.0)	✗ (1.2, 1.4)
PA(U)	✗ (5.1, 16.1)	✗ (1.3, 1.4)
VA (Attest phase)	✓ (3.9, 204.7)	✗ (1.1, 2.3)
VA (Collect phase)	✓ (29.3, 1686.1)	✗ (4.4, 9.6)
RT	✓(3.2, 115.6)	✗(3.1, 13.9)

Table 2: Verification results for SIMPLE∗

5.2 Resilience via Asymmetric Cryptography

As discussed above, SIMPLE+ is insecure whenever there is the possibility of *any* prover leaking either K_{col} or K_{auth}. In this section, we explore a second variant of SIMPLE+ (that we refer to as SIMPLE$^*_{BLS}$ [6]), which, inspired by SARA, uses asymmetric cryptography to address this vulnerability. Additionally, using our observation in Section 5.1, we leave out K_{attest} in SIMPLE$^*_{BLS}$, though it would be trivial to re-introduce it if desired. We show that, under both $\mathcal{M}_{noreveal}$ and $\mathcal{M}_{otherreveal}$, SIMPLE$^*_{BLS}$ guarantees the desired security properties from Section 3.

5.2.1 *Describing SIMPLE$^*_{BLS}$.*

SIMPLE$^*_{BLS}$ is derived from SIMPLE∗ as follows. First, we remove all the symmetric keys (K_{auth} and K_{col}) and replace them with an asymmetric key pair for each agent. We do not go into detail regarding public key infrastructure; however, we assume that all agents have their own private key and are able to authenticate messages from neighbouring provers as well as the verifier. The verifier can authenticate messages from all provers.

Second, the HMACs in the attest and collect phase are replaced with asymmetric signing and verification of the message. As K_{col} is no longer present, the looping in the attest phase is removed. Freshness is provided by including the counter C_v/C_p in messages.

Finally, the bitstring report remains and is accompanied by an aggregate signature. Each prover signs the concatenation of their report value and C_p using their secret key. Provers aggregate their

[6]The "BLS" in SIMPLE$^*_{BLS}$ refers to the fact that it uses Boneh–Lynn–Shacham aggregate signatures [9]. We choose BLS as it has been previously studied in TAMARIN [21]. Alternatively, one could use the Optimistic Aggregate Signature scheme used by SANA [2].

	Threat Model	
Property	$\mathcal{M}_{noreveal}$ (T_1)	$\mathcal{M}_{otherreveal}$ (T_1)
LR(H)	✓ (136.8)	✓ (134.9)
LR(U)	✓ (4.6)	✓ (5.5)
PA(H)	✓ (231.1)	✓ (224.4)
PA(U)	✗ (53.2)	✗ (53.5)
VA (Attest phase)	✓ (18.8)	✓ (18.4)
VA (Collect phase)	✓ (121.0)	✓ (122.7)
RT	✓ (1410.8)	✓ (1411.2)

Table 3: Verification results for SIMPLE$^*_{BLS}$

reports with those received from their children, using OR for the bitstring and the BLS scheme to aggregate signatures. The verifier authenticates reports from its children by reading the bitstring and confirming whether or not a signature for each value (along with the current value of C_v) is present in the aggregate signature.

5.2.2 Modelling Decisions. We now detail key decisions we made when modelling this new protocol in TAMARIN.

Hofmeier [21] presents two models of the BLS aggregate signature scheme. The first, the attack finding model, explicitly encodes known attacks on the BLS scheme. The second, the validation model, encodes the computational definition of BLS and can theoretically find any possible attack, known or unknown. We utilise the attack-finding model as it is significantly faster.

5.2.3 Security Analysis. For security properties, we use the same lemmas and threat models as before. The results (given in Table 3) show that by using asymmetric cryptography we can maintain the same properties as SIMPLE+ under a stronger threat model.

However, applying asymmetric cryptography to CRA is not without its disadvantages. First, asymmetric cryptography is more expensive than symmetric cryptography, both in terms of computation cost and bandwidth requirements. This is significant for CRA because the target devices are slow, often battery powered, and CRA protocols are often uninterruptable. This last point is key because, at least in a CRA protocol such as SIMPLE+, any increase in the time it takes to run the protocol on a prover, such as by using more advanced cryptography, is multiplied by the time spent waiting for a prover's children to execute the protocol. In simulations, SIMPLE+, which uses symmetric cryptography, takes over 13 seconds to attest a swarm of 10,000 provers. Many CRA protocols, such as SIMPLE+'s collect phase, are also uninterruptable, meaning the prover can not perform its usual task while executing the attestation protocol.

Secondly, SIMPLE$^*_{BLS}$ requires that provers are able to authenticate their neighbours using asymmetric cryptography, which assumes some form of public-key infrastructure. Therefore, even though SIMPLE$^*_{BLS}$ provides stronger security guarantees, its implementation would not be without disadvantages.

6 RELATED WORK

Outside of L-QoSA protocols, a number of other schemes have been proposed. In B-QoSA, a verifier receives a binary result indicating either that the entire swarm is healthy, or that at least one prover in the swarm is unhealthy, e.g., SEDA [6]. I-QoSA provides a level of information between B-QoSA and L-QoSA. An example of a protocol with I-QoSA is FADIA [30], which partially aggregates results. In FADIA, the swarm is divided into subsets, which the verifier knows, and the verifier registers a binary result for each subset. Carpent et al. [10] also propose F-QoSA, where the topology of the swarm is also attested.

Protocols have used a range of signature schemes in their design. Like SIMPLE$^*_{BLS}$, SANA [2] uses asymmetric cryptography. SANA proposes a novel *optimised aggregate signature* scheme, which produces shorter signatures that can be verified faster when most signers sign an expected message, i.e., the healthy software state[7].

FADIA [30] uses the Eschenauer-Gligor (E-G) scheme [19] for the distribution of *communication* keys. Here, a set of symmetric keys is generated for the swarm (the key pool), and each prover is given a subset of these keys (a key ring). Two provers can communicate if and only if they share a key in their key ring, giving any two provers a probability of being able to communicate based on the key ring and key pool size. However, FADIA still uses unique symmetric keys shared with the verifier to secure attestation results. Meaning that a compromise of E-G keys should not compromise the security of attestation results. Such a design could also be applied to SIMPLE$^*_{BLS}$, but we leave this for future work.

While swarm attestation lacks examples of formal verification, single-prover remote attestation protocols have been verified. Whitefield et al. [35] verify Direct Anonymous Attestation in TAMARIN. The single-prover SIMPLE [5] protocol is accompanied by a formal proof of its software using VeriFast [24]. Nunes et al. [14] use NuSMV [12] to develop a formally verified remote attestation architecture.

7 CONCLUSION

This paper presents the first formalisation of L-QoSA CRA protocols. We define a generic unifying framework that characterises the main aspects of these protocols and the key security properties that they must satisfy. We show that multiple L-QoSA CRA protocols are instantiations of this framework, allowing one to precisely state the properties that are and are not guaranteed by each of these. Our main case study is the SIMPLE+ protocol, whose security has previously not been studied. We instantiate the security properties of SIMPLE+ w.r.t. our framework and analyse it using TAMARIN. We experiment with two variations of SIMPLE+, i.e., SIMPLE$*$ and SIMPLE$^*_{BLS}$, which simplify and strengthen SIMPLE+ w.r.t. stronger threat models. We analyse both SIMPLE$*$ and SIMPLE$^*_{BLS}$ in TAMARIN. Due to the parametric nature of CRA protocols (in number of rounds and number of provers), this is a bounded analysis. Future work could seek to prove the security of CRA protocols in the unbounded case. Future work could also extend the formalisation beyond L-QoSA.

ACKNOWLEDGMENTS

We thank Sharar Ahmadi and Liqun Chen for helpful discussions on this work. Le-Papin, Dongol and Treharne are funded by EPSRC grant EP/V038915/1. Dongol additionally acknowledges EPSRC grants EP/R032556/1 and EP/R025134/2; VeTSS; and ARC Grant DP190102142.

[7]Unlike SIMPLE+, SANA is not a self-attestation scheme

REFERENCES

[1] Muhammad Naveed Aman and Biplab Sikdar. 2018. ATT-Auth: A Hybrid Protocol for Industrial IoT Attestation With Authentication. *IEEE Internet Things J.* 5, 6 (2018), 5119–5131. https://doi.org/10.1109/JIOT.2018.2866623

[2] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. 2016. SANA: Secure and Scalable Aggregate Network Attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 731–742. https://doi.org/10.1145/2976749.2978335

[3] Moreno Ambrosin, Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, and Silvio Ranise. 2018. PADS: Practical Attestation for Highly Dynamic Swarm Topologies. https://doi.org/10.48550/ARXIV.1806.05766

[4] Moreno Ambrosin, Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, and Silvio Ranise. 2020. Collective Remote Attestation at the Internet of Things Scale: State-of-the-Art and Future Challenges. *IEEE Communications Surveys & Tutorials* 22, 4 (2020), 2447–2461. https://doi.org/10.1109/COMST.2020.3008879

[5] Mahmoud Ammar, Bruno Crispo, and Gene Tsudik. 2020. SIMPLE: A Remote Attestation Approach for Resource-constrained IoT devices. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. 247–258. https://doi.org/10.1109/ICCPS48487.2020.00036

[6] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. SEDA: Scalable Embedded Device Attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 964–975. https://doi.org/10.1145/2810103.2813670

[7] David Basin, Saša Radomirovic, and Lara Schmid. 2018. Alethea: A Provably Secure Random Sample Voting Protocol. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. 283–297. https://doi.org/10.1109/CSF.2018.00028

[8] Bruno Blanchet. 2014. Automatic Verification of Security Protocols in the Symbolic Model: the Verifier ProVerif. In *Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures*, Alessandro Aldini, Javier Lopez, and Fabio Martinelli (Eds.). Lecture Notes in Computer Science, Vol. 8604. Springer, 54–87.

[9] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology — EUROCRYPT 2003*, Eli Biham (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 416–432.

[10] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. Lightweight Swarm Attestation: A Tale of Two LISA-s. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (Abu Dhabi, United Arab Emirates) *(ASIA CCS '17)*. Association for Computing Machinery, New York, NY, USA, 86–100. https://doi.org/10.1145/3052973.3053010

[11] Xavier Carpent, Gene Tsudik, and Norrathep Rattanavipanon. 2018. ERASMUS: Efficient remote attestation via self-measurement for unattended settings. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1191–1194. https://doi.org/10.23919/DATE.2018.8342195

[12] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Computer Aided Verification*, Ed Brinksma and Kim Guldstrand Larsen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 359–364.

[13] Wilfried Daniels, Danny Hughes, Mahmoud Ammar, Bruno Crispo, Nelson Matthys, and Wouter Joosen. 2017. SμV - the Security Microvisor: A Virtualisation-Based Security Middleware for the Internet of Things. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track* (Las Vegas, Nevada) *(Middleware '17)*. Association for Computing Machinery, New York, NY, USA, 36–42. https://doi.org/10.1145/3154448.3154454

[14] I. De Oliveira Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik. 2019. VRased: A verified hardware/software co-design for remote attestation. In *Proceedings of the 28th USENIX Security Symposium*. 1429–1446. Cited By :42.

[15] D. Dolev and A. Yao. 1983. On the Security of Public-Key Protocols. *IEEE Trans. Inf. Theory 29* 29, 2 (1983).

[16] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. 1999. Undecidability of Bounded Security Protocols. In *FMSP'99*. ACM.

[17] Edlira Dushku, Md Masoom Rabbani, Mauro Conti, Luigi V. Mancini, and Silvio Ranise. 2020. SARA: Secure Asynchronous Remote Attestation for IoT Systems. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3123–3136. https://doi.org/10.1109/TIFS.2020.2983282

[18] Karim Eldefrawy, Daniele Perito, and Gene Tsudik. 2012. SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust. (01 2012).

[19] Laurent Eschenauer and Virgil D. Gligor. 2002. A Key-Management Scheme for Distributed Sensor Networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (Washington, DC, USA) *(CCS '02)*. Association for Computing Machinery, New York, NY, USA, 41–47. https://doi.org/10.1145/586110.586117

[20] Zuzana Frankovska. 2021. *Modeling and ANalysis of the SCION Dataplane in Tamarin.* Master's thesis. ETH Zurich.

[21] Xenia Hofmeier. 2021. *Formalizing Aggregate Signatures in the Symbolic Model.* Master's thesis. ETH.

[22] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. 2016. DARPA: Device Attestation Resilient to Physical Attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks* (Darmstadt, Germany) *(WiSec '16)*. Association for Computing Machinery, New York, NY, USA, 171–182. https://doi.org/10.1145/2939918.2939938

[23] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Shaza Zeitouni. 2017. SeED: Secure Non-Interactive Attestation for Embedded Devices. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (Boston, Massachusetts) *(WiSec '17)*. Association for Computing Machinery, New York, NY, USA, 64–74. https://doi.org/10.1145/3098243.3098260

[24] Bart Jacobs, Jan Smans, Pieter Philippaerts, Frédéric Vogels, Willem Penninckx, and Frank Piessens. 2011. VeriFast: A Powerful, Sound, Predictable, Fast Verifier for C and Java. In *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6617)*, Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi (Eds.). Springer, 41–55. https://doi.org/10.1007/978-3-642-20398-5_4

[25] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A Security Architecture for Tiny Embedded Devices. In *Proceedings of the Ninth European Conference on Computer Systems* (Amsterdam, The Netherlands) *(EuroSys '14)*. Association for Computing Machinery, New York, NY, USA, Article 10, 14 pages. https://doi.org/10.1145/2592798.2592824

[26] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. 2018. SALAD: Secure and Lightweight Attestation of Highly Dynamic and Disruptive Networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (Incheon, Republic of Korea) *(ASIACCS '18)*. Association for Computing Machinery, New York, NY, USA, 329–342. https://doi.org/10.1145/3196494.3196544

[27] Boyu Kuang, Anmin Fu, Willy Susilo, Shui Yu, and Yansong Gao. 2022. A survey of remote attestation in Internet of Things: Attacks, countermeasures, and prospects. *Comput. Secur.* 112 (2022), 102498. https://doi.org/10.1016/j.cose.2021.102498

[28] Jay Le-Papin, Brijesh Dongol, Helen Treharne, and Stephan Wesemeyer. 2023. Tamarin Models for "Verifying List Swarm Attestation Protocols". (1 2023). https://doi.org/10.6084/m9.figshare.21931191

[29] G. Lowe. 1997. A hierarchy of authentication specifications. In *Proceedings 10th Computer Security Foundations Workshop*. 31–43. https://doi.org/10.1109/CSFW.1997.596782

[30] Mohamad Mansouri, Wafa Ben Jaballah, Melek Önen, Md Masoom Rabbani, and Mauro Conti. 2021. FADIA: Fairness-Driven Collaborative Remote Attestation. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (Abu Dhabi, United Arab Emirates) *(WiSec '21)*. Association for Computing Machinery, New York, NY, USA, 60–71. https://doi.org/10.1145/3448300.3468284

[31] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 696–701.

[32] Rodrigo Vieira Steiner and Emil Lupu. 2016. Attestation in Wireless Sensor Networks: A Survey. *ACM Comput. Surv.* 49, 3, Article 51 (sep 2016), 31 pages. https://doi.org/10.1145/2988546

[33] The Tamarin Team. 2016. Tamarin prover manual. https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf

[34] Stephan Wesemeyer, Ioana Boureanu, Zach Smith, and Helen Treharne. 2020. Extensive Security Verification of the LoRaWAN Key-Establishment: Insecurities & Patches. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. 425–444. https://doi.org/10.1109/EuroSP48549.2020.00034

[35] Jorden Whitefield, Liqun Chen, Ralf Sasse, Steve Schneider, Helen Treharne, and Stephan Wesemeyer. 2019. A Symbolic Analysis of ECC-Based Direct Anonymous Attestation. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. 127–141. https://doi.org/10.1109/EuroSP.2019.00019

A OTHER INSTANTIATIONS OF FRAMEWORK

A.1 LISA-α

In LISA-α, the verifier receives each prover's result as an individual message, and provers only rely upon one another for forwarding these messages to the verifier. LISA-α has observable statuses: the set of healthy provers, the set of unhealthy provers, and the set of provers of unknown status.

- *Digest* $\hat{=}$ the set of possible hashes of attested memory.

Like SIMPLE+, LISA-α uses a single monotonically increasing counter, this time called $CurSeq_{prv}$.

- $T \hat{=} \mathbb{N}$
- $\mu((prv, \sigma, \tau)) \hat{=} (prv, h(\sigma), CurSeq_{prv})$, where $CurSeq_{prv}$ is prv's $CurSeq$ value at τ
- $bound(a, Seq)$ returns the interval in which agent a's current session number is equal to Seq
- $\rho((prv, h(\sigma), CurSeq_{vrf})) = healthy$, iff $h(\sigma)$ corresponds to an expected legal state of prv during the round $CurSeq_{vrf}$, otherwise
- $\rho((prv, \_, CurSeq_{vrf})) = unhealthy$

The verifier has an overall timeout, and if it occurs before results for all provers in P are received, the status of any missing provers is recorded as *unknown*.

A.2 SANA

SANA differs from the previous protocols by having multiple round numbers, each with a unique id. As with the previous counters, these are monotonically increasing, and an agent will only accept a request if it contains a counter value greater than the current value it has for that counter id. Each prover contains a list of counters and their corresponding values. The result is that rounds in SANA are uniquely identified with a counter id, value pair (c_l, v_l).

Therefore, T is the set:

- $T \hat{=} \mathbb{N} \times \mathbb{N}$

with the first $\mathbb{N}$ being the counter id c_l and the second being the counter value v_l. The possible values $\{v_0, ...v_n\}$ for each counter id c_l has a natural ordering.

Each round of attestation is associated with a set of digests $H_{(c_l,v_l)}$, which include all of the digests that represent software states that may be considered healthy during that round.

- *Digest* $\hat{=}$ all possible outputs of $getSoftConf()$
- $\mu((prv, \sigma, \tau)) \hat{=} (prv, h(\sigma), (c_l, v_l))$, where c_l is the counter id being used in the current session and v_l is prv's value for c_l
- $bound(a, (c_l, v_l))$ returns the time interval in which agent a's counter with id c_l is equal to v_l

Attestation results in SANA are aggregated into a single report using an Optimised Aggregate Signature (OAS) [2]. If this report includes results for all provers in P, then the verifier obtains a list of healthy provers (where $h(\sigma) \in H_{(c_l,v_l)}$), and a list of unhealthy provers (where $h(\sigma) \notin H_{(c_l,v_l)}$). If any prover's result is not included in the report, then the entire report is unverifiable (In which case, ρ is not used).

- $\rho((\_, h(\sigma), (c_l, v_l))) = healthy$, iff $h(\sigma) \in H_{(c_l,v_l)}$
- $\rho((\_, h(\sigma), (c_l, v_l)) = unhealthy$, iff $h(\sigma) \notin H_{(c_l,v_l)}$

A.3 SeED

SeED is a single-prover attestation scheme which is applicable to attesting swarms thanks to its low bandwidth and ability to stagger attestation times. It is a non-interactive protocol, meaning the verifier does not send attestation requests to a prover. Instead, the prover is induced to execute the attestation protocol at pre-programmed intervals, and send an attestation report to the verifier.

As seen in previous protocols, the definition of *Digest* is left vague, but SeED is not a self-attestation protocol.

- *Digest* $\hat{=}$ measurement of software

Unlike the previous protocols, SeED does not use a monotonically increasing counter. A real-time clock synchronised with the verifier is a requirement for the prover to know when to attest, so instead of a counter, a timestamp is used. We continue to represent this as $\mathbb{N}$.

- $T \hat{=} \mathbb{N}$
- $\mu((prv, \sigma, \tau)) \hat{=} (prv, h(\sigma), T_{att}$, where T_{att} is the time shown by prv's real time clock at τ
- $bound(a, T_{att})$ returns the interval $[T_{att} - \delta_t, T_{att} + \delta_t + t_{tr}]$ where δ_t is the maximum clock skew between *vrf* and a, and t_{tr} is the maximum allowed time for a message to travel from a to *vrf*.
- $\rho((prv, h(\sigma), T_{att})) = healthy$ if $h(\sigma)$ corresponds to a 'benign' software state, otherwise
- $\rho((prv, \_, T_{att})) = unhealthy$