Orchestrating Layered Attestations

John D. Ramsdell[1(✉)], Paul D. Rowe[1], Perry Alexander[2], Sarah C. Helble[3], Peter Loscocco[4], J. Aaron Pendergrass[3], and Adam Petz[2]

[1] The MITRE Corporation, Bedford, USA
ramsdell@mitre.org
[2] The University of Kansas, Lawrence, USA
[3] John Hopkins University Applied Physics Laboratory, Laurel, USA
[4] National Security Agency, Fort Meade, USA

Abstract. We present Copland, a language for specifying layered attestations. Layered attestations provide a remote appraiser with structured evidence of the integrity of a target system to support a trust decision. The language is designed to bridge the gap between formal analysis of attestation security guarantees and concrete implementations. We therefore provide two semantic interpretations of terms in our language. The first is a denotational semantics in terms of partially ordered sets of events. This directly connects Copland to prior work on layered attestation. The second is an operational semantics detailing how the data and control flow are executed. This gives explicit implementation guidance for attestation frameworks. We show a formal connection between the two semantics ensuring that any execution according to the operational semantics is consistent with the denotational event semantics. This ensures that formal guarantees resulting from analyzing the event semantics will hold for executions respecting the operational semantics. All results have been formally verified with the Coq proof assistant.

1 Introduction

It is common to ask a particular target system whether it is trustworthy enough to engage in a given activity. Remote attestation is a useful technique to support such trust decisions in a wide variety of contexts. Fundamentally, remote attestation consists in generating *evidence* of a system's integrity via *measurements*, and *reporting* the evidence to a remote party for appraisal. Depending on their interpretation of the evidence, the remote appraiser can adjust their decision according to the level of risk they are willing to assume.

Others have recognized the insufficiency of coarse-grained measurements in supporting trust decisions [8,10,20,22]. Integrity evidence is typically either too broad or too narrow to provide useful information to an appraiser. Very broad evidence—such as patch levels for software—easily allows compromises to go undetected by attestation. Very narrow evidence—such as a combined hash of the complete trusted computing base—does not allow for natural variation across systems and over time.

F. Nielson and D. Sands (Eds.): POST 2019, LNCS 11426, pp. 197–221, 2019.
https://doi.org/10.1007/978-3-030-17138-4_9

An alternative approach is to build a global picture of system integrity by measuring a subset of system components and reasoning about their integrity individually and as a coherent whole. This approach can give an appraiser a more nuanced view of the target system's state because it can isolate integrity violations, telling the appraiser exactly which portions of the system can or cannot be trusted. We call this approach *layered attestation* because protected isolation frequently built into systems (e.g. hypervisor-enforced separation of virtual machines) allows the attestation to build the global picture of integrity from the bottom up, one layer at a time. A layered attestation whose structure mimics the layered dependency structure of a target system can provide strong trust guarantees. In prior work, we have formally proved that "bottom-up" strategies for layered attestation force an adversary to either corrupt well-protected components or work within small time-of-check-time-of-use windows [17,18].

The "bottom-up" principle has been embodied in many attestation systems (e.g. [2,6,7,10,22]). A common tactic in these papers is to design the target system and the attestation protocol in tandem to ensure the structure of the attestation corresponds to the structure of the system. This results in solutions that are too rigid and overly prescriptive. The solutions do not translate to other systems with different structures.

In previous work, members of our team have taken a different approach. Maat is a policy-based measurement and attestation (M&A) framework which provides a centralized, pluggable service to gather and report integrity measurements [16]. Maat listens for attestation requests and can act as both an appraiser and an attester, depending on the needs of the current scenario. After a request for appraisal is received, the Maat instance on the appraiser system contacts and negotiates with the attesting system's Maat instance to agree upon the set of evidence that must be provided for the scenario. Thus Maat provides a flexible set of capabilities that can be tailored to the needs of any given situation. It is therefore a much more extensible attestation framework.

In early development of Maat, the negotiation was entirely based on a set of well-known UUIDs and was limited in flexibility, especially when Maat instances did not share a core set of measurement capabilities. We discovered that this approach to negotiation severely limited the extensibility of Maat. It is not sufficient to have a flexible set of attestation mechanisms—a flexible language for specifying layered attestations is crucial. This paper introduces such a language.

Contribution. We present Copland, a language and formal system for orchestrating layered attestations. Copland provides domain specific syntax for specifying attestation protocols, an operational semantics for guiding implementations, and a denotational semantics for reasoning and negotiation. We designed Copland with Maat in mind aiming to address three main requirements.

First, it must be flexible enough to accommodate the wide diversity of capabilities offered by Maat. Copland is parametric with respect to the basic actions that generate and process evidence (i.e. measurement and bundling). Since we cannot expect all platforms and architectures to have the same set of capabilities, Copland focuses instead on specifying the ways in which these pieces fit

together. COPLAND programs, which we call *phrases* or *terms*, are built out of a small set of operators designed to orchestrate the activities of measurement agents across several layers of a target system.

Second, the language must have an unambiguous execution semantics. We provide a formal, operational semantics allowing a target to know precisely how to manage the flow of control and data throughout the attestation. This operational semantics serves as a correctness constraint for implementations, and generates traces of events that record the order in which actions occurred.

Finally, it must enable static analysis to determine the trust properties guaranteed by alternative phrases. For this purpose we provide a denotational semantics relating phrases to a partially ordered set of events. This semantics is explicitly designed to connect with our prior work on analytic principles of layered attestation [17,18]. By applying those principles in static analysis, both target and appraiser can write policies determining which phrases may be used in which situations based on the trust guarantees they provide.

Critically, we prove a strong connection between the operational execution semantics and the denotational event semantics. We show that any trace generated by the operational semantics is a linearization of the event partial ordering given by the denotational semantics. This ensures that any trust conclusions made from the event partial order are guaranteed to hold over the concrete execution. In particular, our previous work [17,18] characterizes what an adversary must do to avoid detection given a specific partial order of events, identifying strategies to force an adversary to work quickly in short time-of-check-time-of-use windows, or dig deeper into more protected layers of the system. This connection is particularly important in light of the flexibility of the language. Since our basic tenet is that a more constrained language is inherently of less value, it is crucial that we provide a link to analytic techniques that help people distinguish between good and bad ways to perform a layered attestations. We discuss this connection to our previous work in much more detail in Sect. 7.

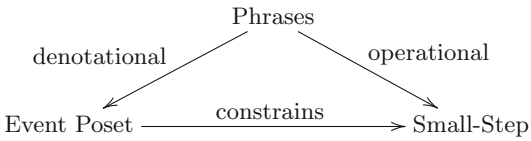

Fig. 1. Semantic relations

Figure 1 depicts the connections among our various contributions. It also provides a useful outline of the paper. Section 3 describes the syntax of COPLAND corresponding to the apex of the triangle in Fig. 1. Section 4 introduces events. Events are the foundation for both semantic notions depicted in Fig. 1. Each semantic notion constrains the event ordering in its own way. The denotational semantics of the left leg of the triangle is presented in Sect. 5, and the operational

semantics of the right leg is given in Sect. 6. The crucial theorem connecting the two semantic notions is sketched in Sect. 7.

All lemmas and theorems stated in this paper have been formally verified using the Coq proof assistant [1]. The Coq proofs are available at https://ku-sldg.github.io/copland/. The notation used in this paper closely follows the Coq proofs. The tables in Appendix B link figures and formulas with their definitions in the Coq proofs.

Before jumping into the formal details of the syntax and semantics of Copland, however, we present a sequence of simple examples designed to give the reader a feel for the language and its purpose.

2 Examples of Layered Attestations

Consider an example of a corporate gateway that appraises machines before allowing them to join the corporate network. A simple attestation might entail a request for the machine to perform an asset inventory to ensure all software is up-to-date. For purposes of exposition, we may view this as an abstract userspace measurement USM that takes an argument list $\bar{a}_1$ of the enterprise software to inventory. We can express a request for a particular target p to perform this measurement with the following Copland phrase:

$$@_p\, \mathsf{USM}\ \bar{a}_1 \tag{1}$$

This says the measurement capability identifiable as USM should be executed at location identified by p using arguments $\bar{a}_1$. The request results in evidence of the form $\mathsf{U}_p(\xi)$ indicating the type of measurement performed, the target of the measurement p, and any previously generated evidence (in this case the empty evidence ξ) it received and combined with the newly generated evidence.

If the company is concerned with the assets in the inventory being undermined by a rootkit in the operating system kernel, it might require additional evidence that no such rootkit exists. This could be done by asking for a kernel integrity measurement KIM to be taken of the place p in addition to the userspace measurement. The request could be made with the following phrase:

$$@_p\, (\mathsf{KIM}\ p\ \bar{a}_2\ \overset{(\bot,\bot)}{\sim}\ \mathsf{USM}\ \bar{a}_1) \tag{2}$$

In this notation, $\mathsf{KIM}\ p\ \bar{a}_2$ represents a request for the KIM measurement capability to be applied to the target place p with arguments $\bar{a}_2$. The symbol $\overset{(\ell,r)}{\sim}$ indicates the two measurements may be taken concurrently. The annotation ℓ defines how evidence accumulated so far is transformed for use by the phrase on the left, and r for the one on the right. In the case of $(\bot, \bot)$, no evidence is sent in either direction. The evidence resulting from the two composed measurements has the form $\mathsf{K}^p_p(\xi) \parallel \mathsf{U}_p(\xi)$, where $\parallel$ indicates the measurements were invoked concurrently.

If the enterprise has configured their machines to have two layers of different privilege levels (say by virtualization), then they may wish to request that the

kernel measurement be taken from a more protected location q. This results in the following request.

$$@_q\,(\mathsf{KIM}\ p\ \bar{a}_2\ \overset{(\bot,\bot)}{\sim}\ @_p\,\mathsf{USM}\ \bar{a}_1) \tag{3}$$

Notice the kernel measurement target is still the kernel at p, but the request is now being made of the measurement capability located at q. The kernel measurement of p taken from q and the request for p to take a userspace measurement of its own environment can occur concurrently. The resulting evidence has the form $\mathsf{K}_q^p(\xi) \parallel \mathsf{U}_p(\xi)$, where the subscript q indicates the kernel measurement was taken from the vantage point of q, and the superscript p indicates the location of the kernel measurer's target. The subscript p in the second occurrence of the @ sign indicates that the userspace measurement is taken from location p.

Finally, consider two more changes to the request that makes the evidence more convincing. By measuring the kernel at p *before* the userspace measurement occurs, the appraiser can learn that the kernel was uncompromised at the time of the userspace measurement. This bottom-up strategy is common in approaches to layered attestation [17,22]. Additionally, an appraiser may wish each piece of evidence to be signed as a rudimentary chain of evidence. These can both be specified with the following phrase.

$$@_q\,((\mathsf{KIM}\ p\ \bar{a}_2 \rightarrow \mathsf{SIG})\ \overset{(\bot,\bot)}{\prec}\ @_p\,(\mathsf{USM}\ \bar{a}_1 \rightarrow \mathsf{SIG})) \tag{4}$$

In this phrase, the $\prec$ symbol is used to request that the term on the left complete its execution before starting execution of the term on the right. The $\rightarrow$ symbol routes data from the term on the left to the term on the right, similar to function composition. In this case evidence coming from KIM and USM is routed to two separate instances of a digital signature primitive. Since these signatures occur at two different locations, they will use two different signing keys. The resulting evidence has the form $[\![\mathsf{K}_q^p(\xi)]\!]_q$;; $[\![\mathsf{U}_p(\xi)]\!]_p$, where ;; indicates the evidence was generated in sequence, and the square brackets represent signatures using the private key associated with the location identified by the subscript.

Copland provides a level of flexibility and explicitness that can be leveraged for more than the prescription of the evidence to be gathered. Using this common semantics, appraisers and attesters have the ability to negotiate *specific* measurement agents and targets to utilize to prove integrity. For example, if the measurement requested is computationally intensive, an attester may prefer to provide a cached version of the evidence. The appraiser may be willing to accept this cached version, depending on local policy. In this scenario, a negotiation would take place between the two systems to determine an agreeable set of terms. The appraiser could begin by requesting that Eq. (4) be performed by the target, which would then counter with a different phrase specifying cached instead of fresh measurement. Depending on the implementation, this difference could utilize an entirely separate measurement primitive (e.g., C_USM instead of USM) or merely a separate set of arguments to the primitive. The ability to specify the collection of previously generated evidence is especially important when gathering evidence created via a measured boot.

The actions taken to appraise evidence can also be defined by phrases and negotiated before the attestation takes place. If the target is willing to perform a measurement action but doesn't trust the appraiser with the result, the two parties could agree upon a mutually trusted third party to act as the appraiser.

3 Phrases

We begin with the basic syntax of phrases in Copland. Figure 2 defines the grammar of phrases (T) parameterized by atomic actions (A) and the type (E) of evidence they produce when evaluated. Figure 3 defines phrase evaluation. Each phrase specifies what measurements are taken, various operations on evidence, and where measurements and operations are performed. Phrases also specify orderings and dependencies among measurements and operations.

$$
\begin{aligned}
A &\leftarrow \mathsf{CPY} \mid \mathsf{USM}\ \bar{a} \mid \mathsf{KIM}\ P\ \bar{a} \mid \mathsf{SIG} \mid \mathsf{HSH} \mid \cdots \\
T &\leftarrow A \mid @_P\, T \mid (T \to T) \mid (T \overset{\pi}{\prec} T) \mid (T \overset{\pi}{\sim} T) \\
E &\leftarrow \xi \mid \mathsf{U}_P(E) \mid \mathsf{K}^P_P(E) \mid [\![E]\!]_P \mid \#_P\, E \mid (E\ ;;\ E) \mid (E \parallel E) \mid \cdots
\end{aligned}
$$

where $\pi = (\pi_1, \pi_2)$ is a pair of splitting functions.

Fig. 2. Phrase and evidence grammar

The atomic phrases either produce evidence via measurement, or transform evidence via computation. Some actions, like $\mathsf{USM}\ \bar{a}$, perform measurements of their associated place, while others, such as $\mathsf{KIM}\ q\ \bar{a}$, measure another place. A *userspace measurement*, $\mathsf{USM}\ \bar{a}$, measures the local environment. The term $@_p\, \mathsf{USM}\ \bar{a}$ requests that place p perform some measurement $\mathsf{USM}\ \bar{a}$ of its userspace. Such measurements may range from a simple file hash to complex run time analysis of an application. A *kernel integrity measurement*, $\mathsf{KIM}\ q\ \bar{a}$, measures another place. The term $@_p\, \mathsf{KIM}\ q\ \bar{a}$ requests that p perform a kernel measurement on place q. Such measurements measure one place from another and perform integrity measurements such as LKIM [14]. Starting from a trusted place p, $@_p\, \mathsf{KIM}\ q\ \bar{a}$ can gather evidence for establishing trust in q and transitively construct chains of trusted enclaves.

The Copland phrase $@_p\, t$ corresponds to the essential function of remote attestation—making a request of place p to execute a protocol term t. Places correspond with attestation managers that are capable of responding to attestation requests. Places may be as simple as an IoT device that returns a single value on request or as complicated as a full SELinux installation capable of complex protocol execution.

Evidence produced by $@_p\, \mathsf{USM}\ \bar{a}$ and $@_p\, \mathsf{KIM}\ q\ \bar{a}$ have types $\mathsf{U}_p(e)$ and $\mathsf{K}^q_p(e)$ respectively where p is the place performing measurement, q is the target place, and e is the type of incoming evidence. Place p is obtained from context specified by the $@_p\, t$ phrase invoking $\mathsf{KIM}\ q\ \bar{a}$. Notice that we work with dependent types.

The phrases $(t_1 \rightarrow t_2)$, $(t_1 \overset{\pi}{\prec} t_2)$, and $(t_1 \overset{\pi}{\sim} t_2)$ specify sequential and parallel composition of subterms. Phrase $(t_1 \rightarrow t_2)$ evaluates two terms in sequence, passing the evidence output by the first term as input to the second term. The phrase $(t_1 \overset{\pi}{\prec} t_2)$ is similar in that the first term runs to completion before the second term begins. It differs in that evidence is not sent from the first term as input to the second term. Instead, each term receives some filtered version of the evidence accumulated thus far from the parent phrase. This evidence is split between the two subterms according to the splitting functions $\pi = (\pi_1, \pi_2)$ that specify the filter used before passing evidence to each subterm. The resulting evidence has the form $(e_1 \mathbin{;;} e_2)$ indicating evidence gathered in sequence. Finally, $(t_1 \overset{\pi}{\sim} t_2)$ specifies its two subterms execute in parallel with data splitting specified by $\pi = (\pi_1, \pi_2)$. The evidence term $(e_1 \parallel e_2)$ captures that subterm evaluation occurs in parallel.

Two common filters are identity and empty. $id\, e = e$ returns its argument, producing a copy of the filtered evidence while $\bot\, e = \xi$ always returns empty evidence regardless of input. For example, $\pi = (\bot, \bot)$ passes empty evidence to both subterms, $\pi = (\bot, id)$ sends all evidence to the right subterm, and $\pi = (id, id)$ sends all evidence to both subterms.

A collection of operator terms specifies various operations over evidence. SIG, HSH, and CPY generate a signature, a hash and a copy of evidence previously gathered. The evidence forms generated by SIG and HSH are $[\![e]\!]_p$ and $\#_p\, e$, respectively. A place identifies itself in a hash by including its identity in the data being hashed. Unlike a cryptographic signature, this serves only to identify the entity performing the hash. It does not provide protection against forgery. Our choice to use hashes in this way is not critical to achieving the COPLAND design goals. Replacing it with more standard hashes would cause no problem. Other operator terms are anticipated, but these are sufficient for this exposition and for most phrases used in our examples.

$$
\begin{aligned}
\mathcal{E}(\mathsf{CPY}, p, e) &= e \\
\mathcal{E}(\mathsf{USM}\ \bar{a}, p, e) &= \mathsf{U}_p(e) \\
\mathcal{E}(\mathsf{KIM}\ q\ \bar{a}, p, e) &= \mathsf{K}^q_p(e) \\
\mathcal{E}(\mathsf{SIG}, p, e) &= [\![e]\!]_p \\
\mathcal{E}(\mathsf{HSH}, p, e) &= \#_p\, e \\
\mathcal{E}(@_q\, t, p, e) &= \mathcal{E}(t, q, e) \\
\mathcal{E}(t_1 \rightarrow t_2, p, e) &= \mathcal{E}(t_2, p, \mathcal{E}(t_1, p, e)) \\
\mathcal{E}(t_1 \overset{\pi}{\prec} t_2, p, e) &= \mathcal{E}(t_1, p, \pi_1(e)) \mathbin{;;} \mathcal{E}(t_2, p, \pi_2(e)) \text{ where } \pi = (\pi_1, \pi_2) \\
\mathcal{E}(t_1 \overset{\pi}{\sim} t_2, p, e) &= \mathcal{E}(t_1, p, \pi_1(e)) \parallel \mathcal{E}(t_2, p, \pi_2(e)) \text{ where } \pi = (\pi_1, \pi_2)
\end{aligned}
$$

Fig. 3. Evidence semantics

4 Events

Events are observable effects associated with phrase execution. For example, a userspace measurement event occurs when a USM term executes; a remote request event occurs when $@_p\, t$ begins executing; and a sequence of split and join events occur when the various sequential and parallel composition terms execute. The events resulting from executing a phrase characterize that phrase.

The events associated with a subphrase t_1 within phrase t_0 is determined by the position in t_0 at which the subphrase occurs. For example, the term $(t \to t)$ has two occurrences of t that will be associated with some events. It is essential that the set of events associated with the left occurrence is disjoint from the set of events associated with the right occurrence. For this reason, each event has an associated natural number that is unique to that event.

$$
\begin{array}{ll}
[t]_i^{i+1} \in T_i^{i+1} & \text{if } t \text{ is atomic} \\
[@_p\, t]_i^{j+1} \in T_i^{j+1} & \text{if } t \in T_{i+1}^{j} \\
[t_1 \to t_2]_i^{k} \in T_i^{k} & \text{if } t_1 \in T_i^{j} \text{ and } t_2 \in T_j^{k} \\
[t_1 \overset{\pi}{\prec} t_2]_i^{k+1} \in T_i^{k+1} & \text{if } t_1 \in T_{i+1}^{j} \text{ and } t_2 \in T_j^{k} \\
[t_1 \overset{\pi}{\sim} t_2]_i^{k+1} \in T_i^{k+1} & \text{if } t_1 \in T_{i+1}^{j} \text{ and } t_2 \in T_j^{k}
\end{array}
$$

Fig. 4. Annotated terms

Annotated terms enable the generation of a unique number for each event in the Coq proofs. An annotated term, $[t]_i^j$, adds bounds, i and j to term t, where i and j are natural numbers. By construction each event related to $[t]_i^j$ has a unique natural number k such that $i \leq k < j$. The set of all annotated terms is defined by $\bar{T} = \bigcup_{i,j=0}^{\infty} T_i^j$, where T_i^j is defined in Fig. 4. The number of events associated with $[t]_i^j$ is $j - i$.

As examples, two terms from $\bar{T}$ are:

$$[[\mathsf{KIM}\ p\ \bar{a}]_0^1 \to [\mathsf{SIG}]_1^2]_0^2 \qquad\qquad [@_p\, [\mathsf{USM}\ \bar{a}]_1^2]_0^3$$

The annotations on KIM and SIG indicate that the event associated with KIM is numbered 0 while the event associated with SIG is numbered 1. The entire sequence term includes numbers for both KIM and SIG. Similarly the $@_p\, \mathsf{USM}\ \bar{a}$ term allocates the number 1 for USM, and adds 0 and 2 for a request and reply event respectively associated with $@_p\, t$. For details of annotation generation, see Fig. 9 in Appendix A, which presents a simple function that translates terms into annotated terms.

Figure 5 presents event syntax while Fig. 6 relates phrases to events. The relation between annotated term t, place p, evidence e, and the associated event v, is written $t \diamond_e^p v$. Given some term t and current evidence e in place p, $t \diamond_e^p v$ relates event v to t in p. Note that each event has a natural number whose purpose is to uniquely identify the event as required by the Coq proofs.

$$
\begin{aligned}
V \leftarrow\ & \mathsf{CPY}(\mathbb{N}, P, E) \mid \mathsf{USM}(\mathbb{N}, P, L, E, E) \mid \mathsf{KIM}(\mathbb{N}, P, L, E, E) \\
\mid\ & \mathsf{SIG}(\mathbb{N}, P, E, E) \mid \mathsf{HSH}(\mathbb{N}, P, E, E) \mid \mathsf{REQ}(\mathbb{N}, P, P, E) \\
\mid\ & \mathsf{RPY}(\mathbb{N}, P, P, E) \mid \mathsf{SPLIT}(\mathbb{N}, P, E, E, E) \mid \mathsf{JOIN}(\mathbb{N}, P, E, E, E)
\end{aligned}
$$

Fig. 5. Event grammar

$$
\begin{aligned}
[\mathsf{CPY}]_i^{i+1} &\diamond_e^p \mathsf{CPY}(i,p,e) \\
[\mathsf{USM}\ \bar{a}]_i^{i+1} &\diamond_e^p \mathsf{USM}(i,p,\bar{a},e,\mathsf{U}_p(e)) \\
[\mathsf{KIM}\ q\ \bar{a}]_i^{i+1} &\diamond_e^p \mathsf{KIM}(i,p,\bar{a},e,\mathsf{K}_p^q(e)) \\
[\mathsf{SIG}]_i^{i+1} &\diamond_e^p \mathsf{SIG}(i,p,e,[\![e]\!]_p) \\
[\mathsf{HSH}]_i^{i+1} &\diamond_e^p \mathsf{HSH}(i,p,e,\#_p\, e)
\end{aligned}
$$

$$
\begin{aligned}
[@_q\, t]_i^j &\diamond_e^p \mathsf{REQ}(i,p,q,e) \\
[@_q\, t]_i^j &\diamond_e^p v \text{ if } t \diamond_e^q v \\
[@_q\, t]_i^j &\diamond_e^p \mathsf{RPY}(j-1,p,q,\bar{\mathcal{E}}(t,q,e))
\end{aligned}
$$

$$
\begin{aligned}
[t_1 \to t_2]_i^j &\diamond_e^p v \text{ if } t_1 \diamond_e^p v \\
[t_1 \to t_2]_i^j &\diamond_e^p v \text{ if } t_2 \diamond_{\bar{\mathcal{E}}(t_1,p,e)}^p v
\end{aligned}
$$

$$
\begin{aligned}
[t_1 \overset{\pi}{\prec} t_2]_i^j &\diamond_e^p \mathsf{SPLIT}(i,p,e,\pi_1(e),\pi_2(e)) \\
[t_1 \overset{\pi}{\prec} t_2]_i^j &\diamond_e^p v \text{ if } t_1 \diamond_{\pi_1(e)}^p v \\
[t_1 \overset{\pi}{\prec} t_2]_i^j &\diamond_e^p v \text{ if } t_2 \diamond_{\pi_2(e)}^p v \\
[t_1 \overset{\pi}{\prec} t_2]_i^j &\diamond_e^p \mathsf{JOIN}(j-1,p,e_1,e_2,e_1\ ;;\ e_2)
\end{aligned}
$$

where $e_1 = \bar{\mathcal{E}}(t_1,p,\pi_1(e))$ and $e_2 = \bar{\mathcal{E}}(t_2,p,\pi_2(e))$

$$
\begin{aligned}
[t_1 \overset{\pi}{\sim} t_2]_i^j &\diamond_e^p \mathsf{SPLIT}(i,p,e,\pi_1(e),\pi_2(e)) \\
[t_1 \overset{\pi}{\sim} t_2]_i^j &\diamond_e^p v \text{ if } t_1 \diamond_{\pi_1(e)}^p v \\
[t_1 \overset{\pi}{\sim} t_2]_i^j &\diamond_e^p v \text{ if } t_2 \diamond_{\pi_2(e)}^p v \\
[t_1 \overset{\pi}{\sim} t_2]_i^j &\diamond_e^p \mathsf{JOIN}(j-1,p,e_1,e_2,e_1 \parallel e_2)
\end{aligned}
$$

where $e_1 = \bar{\mathcal{E}}(t_1,p,\pi_1(e))$ and $e_2 = \bar{\mathcal{E}}(t_2,p,\pi_2(e))$

Fig. 6. Events of terms

Each atomic term has exactly one associated event that records execution details of the term including resulting evidence. Each $@_p\, t$ term is associated with a request event, a reply event, and the events associated with term t. Each $(t_1 \to t_2)$ term is associated with the events of its subterms. Both $(t_1 \overset{\pi}{\prec} t_2)$ and $(t_1 \overset{\pi}{\sim} t_2)$ are associated with the events of their subterms as well as a split and a join event. The evidence function $\bar{\mathcal{E}}$ is the same as $\mathcal{E}$ except it applies to annotated terms instead of terms.

Essential properties of the annotations are expressed in Lemmas 1–3. In each lemma, let ι be a projection from an event to its number.

Lemma 1. *$[t]_i^j \diamond_e^p v$ implies $i \leq \iota(v) < j$.*

Each event associated with a term has a number in the range of the term's annotation. This is critical to the way that subterm annotations are composed to form term annotations.

Lemma 2. *$t \diamond_e^p v_1$ and $t \diamond_e^p v_2$ and $\iota(v_1) = \iota(v_2)$ implies $v_1 = v_2$.*

Event numbers are unique to events. If two events have the same number, they must be the same event.

Lemma 3. *$i \leq k < j$ implies for some v, $[t]_i^j \diamond_e^p v$ and $\iota(v) = k$.*

There is an event associated with every number in an annotation range. There are no unassigned numbers in the range of an annotation.

5 Partial Order Semantics

The previous mapping of phrases to evidence types defines a denotational semantics for evaluation. The $t \diamond_e^p v$ relation defines visible events that result when a phrase executes. Here we add a partial order to define correct orderings of events associated with an execution. In Definition 5, we define strict partial order $\mathcal{R}(t, p, e)$ over the set $\{v \mid t \diamond_e^p v\}$, for some term t, place p, and initial evidence e. It defines requirements on any event trace produced by evaluating t at p with e.

The relation $\mathcal{R}(t, p, e)$ is defined by first introducing a language for representing strict partial orders, then representing semantics of language terms as event partial orders. The grammar defining the objects used to represent strict partial orders is

$$O \leftarrow V \mid (O \rhd O) \mid (O \bowtie O).$$

Events are ordered with the precedes relation. We write $o : v \prec v'$ when event v *precedes* another v' in partial order o. We write $v \in o$ if event v occurs in o.

Definition 4 (Precedes). *$o : v \prec v'$ is the smallest relation such that:*

1. *$o = o_1 \rhd o_2$ implies $v \in o_1$ and $v' \in o_2$ or $o_1 : v \prec v'$ or $o_2 : v \prec v'$*
2. *$o = o_1 \bowtie o_2$ implies $o_1 : v \prec v'$ or $o_2 : v \prec v'$*

The set of events associated with o is the set $\{v \mid v \in o\}$, and o represents the poset that orders that set.

If o_1 and o_2 represent disjoint posets, then $o_1 \rhd o_2$ represents the poset that respects the orders in o_1 and o_2 and for which every event in o_1 is before every event in o_2. Therefore, $\rhd$ is called the *before* operator. Additionally, $o_1 \bowtie o_2$ represents the poset which simply retains the orders in both o_1 and o_2, and so $\bowtie$ is called the *merge* operator. When applied to mutually disjoint posets, $\rhd$ and $\bowtie$ are associative.

Definition 5 (Strict Partial Order)

$$\mathcal{R}(t,p,e)(v,v') = \mathcal{V}(t,p,e) : v \prec v'$$

where $\mathcal{V}(t,p,e)$ *is defined in Fig. 7.*

The definition of $\mathcal{V}(t,p,e)$ is carefully crafted so that the posets combined by $\triangleright$ and $\bowtie$ are disjoint.

For the phrase $@_q\,\mathsf{USM}\ \bar{a}$, the strict partial order term starting with 0 is

Example 6. $\mathcal{V}([@_q\,[\mathsf{USM}\ \bar{a}]^2_1]^3_0, p, e) = \mathsf{REQ}(0,\ldots) \triangleright \mathsf{USM}(1,\ldots) \triangleright \mathsf{RPY}(2,\ldots)$.

$$
\begin{aligned}
\mathcal{V}([\mathsf{CPY}]^{i+1}_i, p, e) &= \mathsf{CPY}(i,p,e)\\
\mathcal{V}([\mathsf{USM}\ \bar{a}]^{i+1}_i, p, e) &= \mathsf{USM}(i,p,\bar{a},e,\mathsf{U}_p(e))\\
\mathcal{V}([\mathsf{KIM}\ q\ \bar{a}]^{i+1}_i, p, e) &= \mathsf{KIM}(i,p,\bar{a},e,\mathsf{K}^q_p(e))\\
\mathcal{V}([\mathsf{SIG}]^{i+1}_i, p, e) &= \mathsf{SIG}(i,p,e,[\![e]\!]_p)\\
\mathcal{V}([\mathsf{HSH}]^{i+1}_i, p, e) &= \mathsf{HSH}(i,p,e,\#_p\,e)\\
\mathcal{V}([@_q\,t]^j_i, p, e) &= \mathsf{REQ}(i,p,q,e) \triangleright \mathcal{V}(t,q,e) \triangleright \mathsf{RPY}(j-1,p,q,\bar{\mathcal{E}}(t,q,e))\\
\mathcal{V}([t_1 \to t_2]^j_i, p, e) &= \mathcal{V}(t_1,p,e) \triangleright \mathcal{V}(t_2,p,\bar{\mathcal{E}}(t_1,p,e))\\
\mathcal{V}([t_1 \overset{\pi}{\prec} t_2]^j_i, p, e) &= \mathsf{SPLIT}(i,p,e,\pi_1(e),\pi_2(e)) \triangleright \mathcal{V}(t_1,p,\pi_1(e)) \triangleright \mathcal{V}(t_2,p,\pi_2(e)) \triangleright\\
&\quad\ \mathsf{JOIN}(j-1,p,e_1,e_2,e_1\ ;;\ e_2)\\
&\quad \text{where } e_1 = \bar{\mathcal{E}}(t_1,p,\pi_1(e)) \text{ and } e_2 = \bar{\mathcal{E}}(t_2,p,\pi_2(e))\\
\mathcal{V}([t_1 \overset{\pi}{\sim} t_2]^j_i, p, e) &= \mathsf{SPLIT}(i,p,e,\pi_1(e),\pi_2(e)) \triangleright (\mathcal{V}(t_1,p,\pi_1(e)) \bowtie \mathcal{V}(t_2,p,\pi_2(e))) \triangleright\\
&\quad\ \mathsf{JOIN}(j-1,p,e_1,e_2,e_1 \parallel e_2)\\
&\quad \text{where } e_1 = \bar{\mathcal{E}}(t_1,p,\pi_1(e)) \text{ and } e_2 = \bar{\mathcal{E}}(t_2,p,\pi_2(e))
\end{aligned}
$$

Fig. 7. Event semantics

The $\mathcal{R}(t,p,e)$ relation is verified to be both irreflexive and transitive, demonstrating it is a strict partial order.

Lemma 7 (Irreflexive). $\neg\mathcal{V}(t,p,e) : v \prec v$.

Lemma 8 (Transitive). $\mathcal{V}(t,p,e) : v_1 \prec v_2$ *and* $\mathcal{V}(t,p,e) : v_2 \prec v_3$ *implies* $\mathcal{V}(t,p,e) : v_1 \prec v_3$.

Evaluating t is shown to include v if and only if v is associated with t. This ensures that all events associated with t are accounted for in the evaluation relation and that the evaluation relation does not introduce events not associated with t. Thus $\mathcal{R}(t,p,e)$ is a strict partial order for the set $\{v \mid t \diamond^p_e v\}$.

Lemma 9 (Correspondence). $v \in \mathcal{V}(t,p,e)$ *iff* $t \diamond^p_e v$.

Figure 7 defines event semantics in terms of the term being processed, the place managing execution, and the initial evidence. Measurement terms and evidence operations trivially translate into their corresponding atomic events whose output is the corresponding measurement or calculated result.

Simple sequential execution $t = (t_1 \rightarrow t_2)$ is defined using the canonical method where output evidence from the first operation is used as input to the second. The before operator ($\triangleright$) ensures that all events from t_1 complete in the order specified by $\mathcal{R}(t, p, e)$ before events from t_2 start. Note the appearance of evidence semantics in the definition to calculate event output in the canonical fashion.

Sequential execution with data splitting $t = (t_1 \overset{\pi}{\prec} t_2)$ is defined by again using the before operator to ensure t_1 events complete as specified by $\mathcal{R}(t, p, e)$ before events from t_2 begin. The distinction from simple sequential execution is using π_1 and π_2 from π to split evidence between t_1 and t_2. The SPLIT event routes evidence to t_1 and t_2 while JOIN composes results indicating sequential execution.

Parallel execution with data splitting $(t_1 \overset{\pi}{\sim} t_2)$ is defined using split and join events. Again π_1 and π_2 determine how evidence is routed to the composed posets. The merge operator ($\bowtie$) specifies parallel composition while respecting the orders specified for t_1 and t_2. The final $\triangleright$ operator ensures that both posets are ordered before JOIN.

The $@_p\, t$ operation responsible for making requests of other places is defined using communication events. The protocol term $@_q\, t$ evaluated by p results in an event poset where: (i) p and q synchronize on a request for q to perform t; (ii) q runs t; (iii) p and q synchronize on the reply back to p sending the resulting evidence. The before operator ($\triangleright$) ensures that each sequential step completes before moving to the next.

Definition 10. *The* output evidence *associated with an event is the right-most evidence used to construct the event.*

Lemma 11. *$\mathcal{V}(t, p, e)$ always has a unique maximal event e_{max}, and the output of e_{max} is $\bar{\mathcal{E}}(t, p, e)$.*

Lemma 11 shows that evaluating a term with the evidence semantics of Fig. 3 produces the same evidence as evaluating the same term with the event semantics of Fig. 7. Every annotated term has a unique maximal event as defined by $\mathcal{V}(t, p, e)$ implying that each finite sequence of events must have a last event. The evidence associated with that maximal event represents evidence produced by any event sequence satisfying the partial order. Additionally, that evidence is equal to the evidence produced by $\bar{\mathcal{E}}(t, p, e)$ for the same term, place and evidence. Lemma 11 proves that evaluating t in place p results in the same evidence using both the evidence and event semantics. Specifically, that $\bar{\mathcal{E}}(t, p, e)$ and $\mathcal{V}(t, p, e)$ are weakly bisimilar, producing the same result.

6 Small-Step Semantics

The small-step semantics for Copland is defined as a labeled transition system whose states represent protocol execution states and whose labels represent events interacting with the execution environment. The single-step transition relation is

$s_1 \overset{\ell}{\rightsquigarrow} s_2$, where s_1 and s_2 are states and ℓ is either an event or τ denoting a silent transition. The transition $s_1 \overset{\ell}{\rightsquigarrow} s_2$ says that a system in state s_1 will transition in one step to state s_2 engaging in the observable event, v, or no event when $\ell = \tau$. The relation $s_1 \overset{c}{\rightsquigarrow}{}^* s_2$ is the reflexive, transitive closure of the single-step relation. c is called an event trace and is the sequence of events resulting from each state transition. The transition $s_1 \overset{c}{\rightsquigarrow}{}^* s_2$ says that a system in state s_1 will transition to state s_2 in zero or more steps engaging in the event sequence c.

The grammar defining the set of states, S, is

$$\begin{aligned} S \leftarrow\ & \mathcal{C}(\bar{T}, P, E) \mid \mathcal{D}(P, E) \mid \mathcal{A}(\mathbb{N}, P, S) \mid \mathcal{LS}(S, \bar{T}) \\ \mid\ & \mathcal{BS}^{\ell}(\mathbb{N}, S, \bar{T}, P, E) \mid \mathcal{BS}^{r}(\mathbb{N}, E, S) \mid \mathcal{BP}(\mathbb{N}, S, S), \end{aligned}$$

where P is the syntactic category for places, E is for evidence, and $\bar{T}$ is for annotated terms. The transition relation for phrases is presented in Fig. 8.

State $\mathcal{C}(t, p, e)$ is a configuration state defining the start of evaluating t at p with initial evidence e. Its complement is the stop state $\mathcal{D}(p, e')$ defining the end of evaluation in p with final evidence e'. Assertion $\mathcal{C}(t, p, e) \overset{c}{\rightsquigarrow}{}^* \mathcal{D}(p, e')$ represents evaluating t at p resulting in evidence e' and event trace c.

A configuration for an atomic term transitions in one step to a done state containing measured or computed evidence after executing an event. For example, the state $\mathcal{C}([\mathsf{USM}\ \bar{a}]_i^{i+1}, p, e)$ transitions to $\mathcal{D}(p, \mathsf{U}_p(e))$ after the single event $\mathsf{USM}(i, p, \bar{a}, e, \mathsf{U}_p(e))$ performs the represented measurement. Similarly, the state $\mathcal{C}([\mathsf{CPY}]_i^{i+1}, p, e)$ transitions to $\mathcal{D}(p, e)$ after the single event $\mathsf{CPY}(i, p, e)$ copies the evidence.

The state $\mathcal{A}(j-1, p, s)$ occurs while evaluating an $[@_q\, t]_i^j$ term and is used to remember the number to be used to construct a reply event and the place to send the result of evaluating t at q after the reply event. A configuration state $\mathcal{C}(@_q\, t, p, e)$ starts the evaluation of $@_q\, t$ by p and transitions immediately to $\mathcal{A}(j-1, p, \mathcal{C}(t, q, e))$ after executing the request event $\mathsf{REQ}(i, p, q, e)$. The nested state $\mathcal{C}(t, q, e)$ represents remote term execution. Evaluation proceeds with $\mathcal{A}(j-1, p, s)$ transitioning to $\mathcal{A}(j-1, p, s')$ when $s \overset{v}{\rightsquigarrow} s'$. Any event v associated with $s \overset{v}{\rightsquigarrow} s'$ is also associated with the transition $\mathcal{A}(j-1, p, s) \overset{v}{\rightsquigarrow} \mathcal{A}(j-1, p, s')$ and will contribute to the trace. When a state $\mathcal{A}(j-1, p, \mathcal{D}(q, e'))$ results, remote execution completes and the result of q evaluating t as requested by p is $\mathcal{D}(p, e')$ after event $\mathsf{RPY}(j-1, p, q, e')$.

The state $\mathcal{LS}(s_1, t_2)$ is associated with evaluating $(t_1 \rightarrow t_2)$. State s_1 represents the current state of term t_1 and t_2 is the second term waiting for evaluation. The state $\mathcal{C}([t_1 \rightarrow t_2]_i^j, p, s)$ transitions to $\mathcal{LS}(\mathcal{C}(t_1, p, e), t_2)$ representing t_1 ready for evaluation and t_2 waiting. The annotation is ignored in this transition because the transitions are silent. Subsequent transitions evaluate $\mathcal{C}(t_1, p, e)$ until reaching state $\mathcal{LS}(\mathcal{D}(p, e_1), t_2)$ after producing event trace v_1. This state silently transitions to $\mathcal{C}(t_2, p, e_1)$ configuring t_2 for evaluation using e_1 as initial evidence. t_2 evaluates in a similar fashion resulting in e_2 and trace v_2. State $\mathcal{D}(p, e_2)$ is the final state with e_2 as evidence having engaged in the concatenation of v_1 and v_2, $v_1 * v_2$.

For atomic terms:

$$\mathcal{C}([\mathsf{CPY}]_i^{i+1}, p, e) \overset{v}{\rightsquigarrow} \mathcal{D}(p, e) \qquad [v = \mathsf{CPY}(i, p, e)]$$
$$\mathcal{C}([\mathsf{USM}\ \bar{a}]_i^{i+1}, p, e) \overset{v}{\rightsquigarrow} \mathcal{D}(p, \mathsf{U}_p(e)) \qquad [v = \mathsf{USM}(i, p, \bar{a}, e, \mathsf{U}_p(e))]$$
$$\mathcal{C}([\mathsf{KIM}\ q\ \bar{a}]_i^{i+1}, p, e) \overset{v}{\rightsquigarrow} \mathcal{D}(p, \mathsf{K}_p^q(e)) \qquad [v = \mathsf{KIM}(i, p, \bar{a}, e, \mathsf{K}_p^q(e))]$$
$$\mathcal{C}([\mathsf{SIG}]_i^{i+1}, p, e) \overset{v}{\rightsquigarrow} \mathcal{D}(p, [\![e]\!]_p) \qquad [v = \mathsf{SIG}(i, p, e, [\![e]\!]_p)]$$
$$\mathcal{C}([\mathsf{HSH}]_i^{i+1}, p, e) \overset{v}{\rightsquigarrow} \mathcal{D}(p, \#_p\, e) \qquad [v = \mathsf{HSH}(i, p, e, \#_p\, e)]$$

For $@_q\, t$**:**

$$\mathcal{C}([@_q\, t]_i^j, p, e) \overset{v}{\rightsquigarrow} \mathcal{A}(j-1, p, \mathcal{C}(t, q, e)) \qquad [v = \mathsf{REQ}(i, p, q, e)]$$
$$\mathcal{A}(i, p, s_1) \overset{v}{\rightsquigarrow} \mathcal{A}(i, p, s_2) \qquad \text{if } s_1 \overset{v}{\rightsquigarrow} s_2$$
$$\mathcal{A}(i, p, \mathcal{D}(q, e)) \overset{v}{\rightsquigarrow} \mathcal{D}(p, e) \qquad [v = \mathsf{RPY}(i, p, q, e)]$$

For $t_1 \to t_2$**:**

$$\mathcal{C}([t_1 \to t_2]_i^j, p, e) \overset{\tau}{\rightsquigarrow} \mathcal{LS}(\mathcal{C}(t_1, p, e), t_2)$$
$$\mathcal{LS}(s_1, t_2) \overset{v}{\rightsquigarrow} \mathcal{LS}(s_2, t_2) \qquad \text{if } s_1 \overset{v}{\rightsquigarrow} s_2$$
$$\mathcal{LS}(\mathcal{D}(p, e), t) \overset{\tau}{\rightsquigarrow} \mathcal{C}(t, p, e)$$

For $t_1 \overset{s}{\prec} t_2$**:**

$$\mathcal{C}([t_1 \overset{s}{\prec} t_2]_i^j, p, e) \overset{v}{\rightsquigarrow} \mathcal{BS}^\ell(j-1, \mathcal{C}(t_1, p, \pi_1(e)), t_2, p, \pi_2(e))$$
$$[v = \mathsf{SPLIT}(i, p, e, \pi_1(e), \pi_2(e))]$$
$$\mathcal{BS}^\ell(i, s_1, t, p, e) \overset{v}{\rightsquigarrow} \mathcal{BS}^\ell(i, s_2, t, p, e) \qquad \text{if } s_1 \overset{v}{\rightsquigarrow} s_2$$
$$\mathcal{BS}^\ell(i, \mathcal{D}(p, e), t, p', e') \overset{\tau}{\rightsquigarrow} \mathcal{BS}^r(i, e, \mathcal{C}(t, p', e'))$$
$$\mathcal{BS}^r(i, e, s_1) \overset{v}{\rightsquigarrow} \mathcal{BS}^r(i, e, s_2) \qquad \text{if } s_1 \overset{v}{\rightsquigarrow} s_2$$
$$\mathcal{BS}^r(i, e_1, \mathcal{D}(p, e_2)) \overset{v}{\rightsquigarrow} \mathcal{D}(p, e_1 ;; e_2) \qquad [v = \mathsf{JOIN}(i, p, e_1, e_2, e_1 ;; e_2)]$$

For $t_1 \overset{s}{\sim} t_2$**:**

$$\mathcal{C}([t_1 \overset{s}{\sim} t_2]_i^j, p, e) \overset{v}{\rightsquigarrow} \mathcal{BP}(j-1, \mathcal{C}(t_1, p, \pi_1(e)), \mathcal{C}(t_2, p, \pi_2(e)))$$
$$[v = \mathsf{SPLIT}(i, p, e, \pi_1(e), \pi_2(e))]$$
$$\mathcal{BP}(i, S, s_1) \overset{v}{\rightsquigarrow} \mathcal{BP}(i, S, s_2) \qquad \text{if } s_1 \overset{v}{\rightsquigarrow} s_2$$
$$\mathcal{BP}(i, s_1, S) \overset{v}{\rightsquigarrow} \mathcal{BP}(i, s_2, S) \qquad \text{if } s_1 \overset{v}{\rightsquigarrow} s_2$$
$$\mathcal{BP}(i, \mathcal{D}(p, e_1), \mathcal{D}(p, e_2)) \overset{v}{\rightsquigarrow} \mathcal{D}(p, e_1 \parallel e_2) \qquad [v = \mathsf{JOIN}(i, p, e_1, e_2, e_1 \parallel e_2)]$$

Fig. 8. Labeled transition system

States $\mathcal{BS}^\ell(j-1, s, t, p, e)$ and $\mathcal{BS}^r(j-1, e, s)$ are associated with evaluating the left and right subterms of $[t_1 \overset{\pi}{\prec} t_2]_i^j$ respectively. Recall that $t_1 \overset{\pi}{\prec} t_2$ differs from $t_1 \to t_2$ because the initial evidence for $t_1 \overset{\pi}{\prec} t_2$ is split between t_1 and t_2 and the resulting evidence is the sequential composition

of evidence from t_1 and t_2. The configuration state $\mathcal{C}([t_1 \overset{\pi}{\prec} t_2]_i^j, p, e)$ transitions immediately to $\mathcal{BS}^\ell(j-1, \mathcal{C}(t_1, p, \pi_1(e)), t_2, p, \pi_2(e))$ after the split event $\mathsf{SPLIT}(i, p, e, \pi_1(e), \pi_2(e))$, where $\pi = (\pi_1, \pi_2)$. This state captures the initial configuration of t_1 ready to evaluate with evidence $\pi_1(e)$ along with t_2 waiting to execute with evidence $\pi_2(e)$ after t_1 completes. Evaluation proceeds with state $\mathcal{BS}^\ell(j-1, s, t_2, p, \pi_2(e))$ transitioning to $\mathcal{BS}^\ell(j-1, s', t_2, p, \pi_2(e))$ after event v when $s \overset{v}{\rightsquigarrow} s'$. After one or more such transitions a state $\mathcal{BS}^\ell(j-1, \mathcal{D}(p, e_1'), t, p, e_2)$ is reached after event sequence v_1 indicating that evaluating t_1 has ended and t_2 should begin. This state transitions to $\mathcal{BS}^r(j-1, e_1', s)$ with s initially $\mathcal{C}(t_2, p, \pi_2(e))$ and e_1' being the evidence from t_1. This state will transition repeatedly until a state $\mathcal{BS}^r(j-1, e_1', \mathcal{D}(p, e_2'))$ results after trace v_2 representing completion of t_2. Both t_1 and t_2 are complete with evidence e_1' and e_2' and evidence must be composed. The final state transitions to $\mathcal{D}(p, e_1 \mathbin{;;} e_2)$ after the join event $\mathsf{JOIN}(j-1, p, e_1, e_2, e_1 \mathbin{;;} e_2)$ where $e_n = \bar{\mathcal{E}}(t_n, p, \pi_n(e))$.

State $\mathcal{BP}(j-1, s_1, s_2)$ is associated with parallel evaluation of t_1 and t_2. The configuration state $\mathcal{C}([t_1 \overset{\pi}{\sim} t_2]_i^j, p, e)$ immediately transitions to $\mathcal{BP}(j-1, \mathcal{C}(t_1, p, \pi_1(e)), \mathcal{C}(t_2, p, \pi_2(e)))$ after the split event $\mathsf{SPLIT}(i, p, e, \pi_1(e), \pi_2(e))$. Note that in the state $\mathcal{BP}(j-1, \mathcal{C}(t_1, p, \pi_1(e)), \mathcal{C}(t_2, p, \pi_2(e)))$ configuration states for both t_1 and t_2 can evaluate. More generally in any state $\mathcal{BP}(j-1, s_1, s_2)$ evaluating either s_1 and s_2 may cause the state to transition. When evaluation reaches a term of the form $\mathcal{BP}(j-1, \mathcal{D}(p, e_1'), \mathcal{D}(p, e_2'))$ both term evaluations are complete. This final state transitions to $\mathcal{D}(p, e_1 \parallel e_2)$ after the join event $\mathsf{JOIN}(j-1, p, e_1, e_2, e_1 \parallel e_2)$.

We prove Correctness, Progress, and Termination with respect to this transition system. Correctness defines congruence between the small-step operational semantics and the denotational evidence semantics. Specifically, if the multi-step evaluation relation maps state $\mathcal{C}(t, p, e)$ to $\mathcal{D}(p, e')$ then $\bar{\mathcal{E}}(t, p, e) = e'$.

Lemma 12 (Correctness). *If* $\mathcal{C}(t, p, e) \overset{c}{\rightsquigarrow}{}^* \mathcal{D}(p, e')$ *then* $\bar{\mathcal{E}}(t, p, e) = e'$.

Progress states that every state is either a stop state of the form $\mathcal{D}(p, e)$ or it can be evaluated. With the Progress lemma we know that there exist no "stuck" states in the operational semantics.

Lemma 13 (Progress). *Either* $s_1 = \mathcal{D}(p, e)$ *for some* p *and* e *or* $s_1 \overset{v}{\rightsquigarrow} s_2$ *for some* v *and* s_2.

Termination states that any configuration state will transition to a done state of the form $\mathcal{D}(p, e)$ in a finite number of steps. This is a strong condition that assures evaluation of any well-formed term will terminate.

Lemma 14 (Termination). *For some* n, $\mathcal{C}(t, p, e) \overset{c}{\rightsquigarrow}{}^n \mathcal{D}(p, e')$.

7 Proof Summary

The ordering of events is a critically important property of attestation systems. Even when measurement events properly execute individually, their ordering

is what establishes trust chains. If a component performs measurement before being measured, any trust in that component and subsequent components is lost.

Figure 1 shows phrases denoted as event posets and defined operationally as a labeled transition system. The event posets define legal orderings of events in traces while the LTS defines traces associated with phrase evaluation. The remaining theoretical result is proving that the small-step semantics produces traces compatible with the partial order semantics.

To present event sequences we use the classical notation $\langle v_1, v_2, \ldots, v_n \rangle$ for sequence construction and $c \downarrow i$ to select the i^{th} element from sequence c. The concatenation of c_1 and c_2 is $c_1 * c_2$. Event v is *earlier* than event v' in trace c, written $v \ll_c v'$, iff there exists an i and j such that $i < j$ and $c \downarrow i = v$ and $c \downarrow j = v'$.

The main correctness theorem states that if some term t evaluates to evidence e' after trace c and two events v and v' from c are ordered by the event semantics, then that order is guaranteed in c. Said differently, if the event semantics constrains two events, then the small-step LTS semantics respects that constraint. This theorem is stated formally in Theorem 15.

Theorem 15 (Correctness). *If* $\mathcal{C}(t, p, e) \overset{c}{\rightsquigarrow}{}^{*} \mathcal{D}(p, e')$ *and* $\mathcal{V}(t, p, e) : v \prec v'$, *then* $v \ll_c v'$.

The proof is done in two steps using a big-step semantics defining traces for individual phrases as an intermediary. The inductive structure of the big-step semantics more closely matches the inductive structure of the partial order semantics, easing the proofs about the relation between the two.

The intermediate big-step semantics is specified as a relation between annotated term t, place p, evidence e, and trace c, written $t \, \square_e^p \, c$. The structure of the definition is similar to the structure of the $\diamond$ relation in Fig. 6. Most cases of the definition are straightforward event sequences taken from the small-step semantics.

For atomic actions, the associated sequence is a single event implementing the action. As an illustrative example, $\mathsf{USM}\ \bar{a}$ is associated with

$$[\mathsf{USM}\ \bar{a}]_i^{i+1} \, \square_e^p \, \langle \mathsf{USM}(i, p, \bar{a}, e, \mathsf{U}_p(e)) \rangle.$$

For remote actions, $@_q\, t$, the associated trace starts with a request event followed by the trace c executed remotely and ending with a reply event:

$$[@_q\, t]_i^j \, \square_e^p \, \langle \mathsf{REQ}(i, p, q, e) \rangle * c * \langle \mathsf{RPY}(j-1, p, q, \bar{\mathcal{E}}(t, q, e)) \rangle \qquad \text{if } t \, \square_e^q \, c.$$

For sequential actions, $(t_1 \rightarrow t_2)$, the associated trace starts with the trace c_1 associated with t_1 and ends with the trace c_2 associated with t_2 starting with evidence e_1 from c_1:

$$[t_1 \rightarrow t_2]_i^j \, \square_e^p \, c_1 * c_2 \qquad \text{if } t_1 \, \square_e^p \, c_1 \text{ and } t_2 \, \square_{e_1}^p \, c_2,$$

where $e_1 = \bar{\mathcal{E}}(t_1, p, e)$.

For sequential branching, $(t_1 \overset{\pi}{\prec} t_2)$, the associated trace starts with a split event and continues with trace c_1 associated with t_1 starting with $\pi_1(e)$ followed by trace c_2 associated with t_2 starting with $\pi_2(e)$:

$$[(t_1 \overset{\pi}{\prec} t_2)]_i^j \; \square_e^p \; \langle v_1 \rangle * c_1 * c_2 * \langle v_2 \rangle \qquad \text{if } t_1 \; \square_{\pi_1(e)}^p \; c_1 \text{ and } t_2 \; \square_{\pi_2(e)}^p \; c_2,$$

where

$$\begin{aligned} v_1 &= \mathsf{SPLIT}(i, p, e, \pi_1(e), \pi_2(e)) \\ v_2 &= \mathsf{JOIN}(j-1, p, e_1, e_2, e_1 \mathbin{;;} e_2) \\ e_1 &= \bar{\mathcal{E}}(t_1, p, \pi_1(e)) \\ e_2 &= \bar{\mathcal{E}}(t_2, p, \pi_2(e)). \end{aligned}$$

The case for parallel branching, $(t_1 \overset{\pi}{\sim} t_2)$, requires additional work to capture parallel execution semantics using trace interleaving. We write $il(c, c', c'')$ to assert that trace c is a result of interleaving c' with c''.

Definition 16 (Interleave). *$il(c, c', c'')$ is the smallest relation such that*

1. *$il(c, \langle\rangle, c)$ and $il(c, c, \langle\rangle)$;*
2. *$il(\langle v \rangle * c, \langle v \rangle * c', c'')$ if $il(c, c', c'')$; and*
3. *$il(\langle v \rangle * c, c', \langle v \rangle * c'')$ if $il(c, c', c'')$.*

When c is an interleaving of c' and c'', $v_1 \ll_{c'} v_2$ implies $v_1 \ll_c v_2$ and $v_1 \ll_{c''} v_2$ implies $v_1 \ll_c v_2$, but the order of events in c is otherwise unconstrained.

With interleaving defined, the trace for $(t_1 \overset{\pi}{\sim} t_2)$ begins with a split operation and continues with an interleaving of c_1 and c_2 associated with t_1 and t_2 starting with $\pi_1(e)$ and $\pi_2(e)$ respectively. The trace ends with a join event when both interleaved traces end:

$$[t_1 \overset{\pi}{\sim} t_2]_i^j \; \square_e^p \; \langle v_1 \rangle * c * \langle v_2 \rangle \quad \text{if } t_1 \; \square_{\pi_1(e)}^p \; c', \; t_2 \; \square_{\pi_2(e)}^p \; c'', \text{ and } il(c, c', c''),$$

where

$$\begin{aligned} v_1 &= \mathsf{SPLIT}(i, p, e, \pi_1(e), \pi_2(e)) \\ v_2 &= \mathsf{JOIN}(j-1, p, e_1, e_2, e_1 \parallel e_2) \\ e_1 &= \bar{\mathcal{E}}(t_1, p, \pi_1(e)) \\ e_2 &= \bar{\mathcal{E}}(t_2, p, \pi_2(e)). \end{aligned}$$

The following two lemmas show that every trace in the big-step semantics contains the correct events. Lemma 17 asserts that the right number of events occurs and Lemma 18 asserts that all events do in fact occur in the trace.

Lemma 17. *$[t]_i^j \; \square_e^p \; c$ implies the length of c is $j - i$.*

Lemma 18. *$t \; \square_e^p \; c$ implies $t \; \diamond_e^p \; v$ iff for some i, $v = c \downarrow i$.*

The first step in the proof of Theorem 15 is to show that a trace of the small-step semantics is also a trace of the big-step semantics as shown in Lemma 19. The lemma asserts that any trace c resulting from evaluating t is also related to t in the big-step semantics.

Lemma 19. $\mathcal{C}(t,p,e) \overset{c}{\rightsquigarrow}{}^{*} \mathcal{D}(p,e')$ *implies* $t \,\square_e^p\, c$.

The next step is to show that if c is a trace of the big-step semantics, then that trace is compatible with the partial order semantics.

Lemma 20. *If* $t \,\square_e^p\, c$ *and* $\mathcal{V}(t,p,e) : v \prec v'$, *then* $v \ll_c v'$.

The proof of Theorem 15 follows from a transitive composition of Lemmas 19 and 20.

The real value of Theorem 15 is that it triangulates specifications, implementations, and formal analysis as depicted in Fig. 1. On one hand, the operational semantics is immediately implementable. This allows us to explicitly test and experiment with alternative options as specified in Copland. On the other hand, however, simple testing is not sufficient to understand the trust properties provided by alternative options. It is better to offer potential users the ability to analyze Copland phrases to establish (or refute) desired trust properties. This is the primary purpose of the event poset semantics. Our prior work on the analytic principles of layered attestation [17,18] is based on partially ordered sets of measurement and processing events. That work details how to characterize what an adversary would have to do in order to escape detection by a given collection of events. In particular, it establishes the fact that bottom-up strategies for measurement and evidence bundling force an adversary to perform either recent or deep corruptions. Recent corruptions must occur within a small time window, so it intuitively raises the bar for an adversary. Similarly, deep corruptions burrow into lower (and presumably better protected) systems layers also raising the bar for the adversary.

Although the event posets in Copland's denotational semantics are somewhat richer than those in [17,18], the reasoning principles can easily be adapted to this richer setting. This enables a verification methodology in which Copland phrases are compiled to event posets, then analyzed according to these principles. In this way, the relative strength of Copland phrases could be directly compared according to the trust properties they guarantee. Theorem 15 ensures that any conclusions made on the basis of this static analysis must also hold for dynamic executions conforming to the operational semantics. It essentially transfers formal guarantees into the world of concrete implementations. We are currently exploring methods to more explicitly leverage such formal analysis to help Maat users write local policies based on the relative strength of Copland phrases.

8 Related Work

The concept of adapting an attestation to the layered structure of a target system is not new. The concept is already present in attestation systems like trusted boot [15] and Integrity Measurement Architecture (IMA) [19] which leverage a

layered architecture to create static, boot-time or load-time measurements of system components. Other solutions have designed layered architectures to enable attestation of the runtime state of a system [10,22]. A major focus is on information flow integrity properties since this allows fine-grained, local measurements to be composed without having to measure the entire system [20]. The main contrast between this line of research and our work is that they fix the structure of an attestation based on the structure of the target architecture, whereas in our work, we support extensible attestation specifications that can be altered to suit many different architectures and many different contexts for trust decisions.

Coker et al. [4] present a general approach for using virtualization to achieve a layered architecture, and it presents generic principles for remote attestation suggesting the possibility of diverse, policy-based orchestrations of attestations. These principles have recently been extended in [13] in the context of cloud systems built with Trusted Platform Modules (TPMs) and virtual TPMs [9].

Several implementations of measurement and attestation (M&A) frameworks have been proposed to address the need for a central service to manage policies for the orchestration and collection of integrity evidence. The Maat framework, as described in Sect. 2, is being utilized by the authors as a testing ground for Copland. Maat provides a pluggable interface for Attestation Service Providers (ASPs), functional units of measurement which are executed by Attestation Protocol Blocks (APBs) after a negotiation between an attester and appraiser machine [16]. Another architecture, given in [8], implements a policy mechanism designed to allow the appraiser to ask for different conditions to be satisfied by the target for different types of interactions. The main focus is on determining suitability of the target system to handle sensitive data. Negotiation between systems and frameworks, and the supporting policy specification, are examples of places where Copland can be leveraged to provide a common language and understanding of attestation guarantees.

Another line of research has focused on hardware/software co-design for embedded devices to enable remote attestation on platforms that are constrained in various ways [2,6,7]. For example, the absence of a TPM can increase an adversary's ability to forge evidence. A careful co-design of hardware and software allows them to tailor attestation protocols to the particular structure of a target device. More recently, Multiple-Tier Remote Attestation (MTRA) extends this work with a protocol that is specifically targeted for the attestation of heterogeneous IoT networks [21]. This protocol uses a preparation stage to configure attestations where more-capable devices (those with TPMs, for example) provide a makeshift root of trust for less-capable devices and measurement of the entire network is distributed across the more-capable devices. We believe that Copland would be beneficial in specifying the complex set of actions required of these heterogeneous networks.

Finally, there has been some work on the semantics of attestation. Datta et al. [5] introduces a formal logic for trusted computing systems. Its semantics is similar to our operational semantics in that it works as a transition system on state configurations. The underlying programming language was designed

specifically for the logic, and is considerably more complex than Copland. It was not designed to be used by implementations as part of a negotiation. Also, it seems the logic has only been applied to static measurements such as trusted boot. We also previously developed a formal approach to the semantics of dynamic measurement [17,18]. In this work we characterize the benefit of a bottom-up measurement strategy as constraining the adversary to corrupt quickly or deeply. These results are obtained based on a partial order of events consisting of measurements and evidence bundling. As discussed above, this basis is similar to our partially ordered event semantics. We explicitly provide such a semantics to leverage the formal results that can be obtained by such analysis. While our set of events is richer, we expect the methods of this line of research to apply.

9 Conclusion and Ongoing Work

Copland serves as a basis for discussing the formal properties of attestation protocols under composition. We have described the denotational semantics of Copland by mapping phrases to evidence and to partially ordered event sets describing events associated with a phrase and constraints on event ordering. While the denotational semantics does not specify unique traces, it specifies event orderings mandatory for believing evidence resulting from evaluation.

We have described the operational semantics of Copland by associating phrases with a labeled transition system. States capture evidence and order execution while labels on transitions describe resulting events. The transitive closure of the LTS transition function describes traces associated with LTS execution.

We then show the small-step semantics generates traces that obey partial orderings specified by the denotational semantics. Furthermore, we show those orderings are preserved under protocol composition. This result is vital to the correctness of attestation outcomes whose validity is equally dependent on resulting evidence and the proper ordering of evidence gathering events.

Beyond the correctness proof, the most impactful contribution of Copland semantics is a foundation for testing and experimenting with layered attestation protocols, pushing the bounds of complexity and diversity of application. We are actively exploring advanced attestation scenarios between Maat Attestation Managers (AMs). Recall from the introduction that Maat is a policy-based measurement and attestation (M&A) framework which provides a centralized, pluggable service to gather and report integrity measurements [16]. The Maat team is leveraging Copland to test attestation scenarios involving the configuration of multiple instances of Maat in multi-realm and multi-party scenarios. In addition to its application to traditional Linux platforms, the Maat framework has been applied to IoT device platforms, where different configurations due to limited resources were explored [3]. We believe frameworks such as Maat provide a rich testing ground for the application of Copland as the basis of policy specification and negotiation across many kinds of system architectures, and are feeding the lessons learned in this application back into the on-going Copland research.

The authors are also using Copland as an implementation language for remote attestation protocols in other systems. A collection of Copland interpreters written in Haskell, F# and CakeML [12] running on Linux, Windows 10 and seL4 [11] provide a mechanism for executing Copland phrases. Each interpreter forms the core of an AM that receives phrases, calls the interpreter, and returns evidence. Additionally, the AMs maintain and protect keys associated places and policies mapping USM and KIM instances to specific implementations. Policies are critically important as they describe details of measurers held abstract within a phrase. Policies will eventually play a central role in negotiating attestation protocols among the various AMs implementing complex, layered attestations. A common JSON exchange format allows exchange of phrases and evidence among AMs running on different systems.

Of particular note, the CakeML interpreter targeting the seL4 platform will be formally verified with respect to the formal Copland semantics. CakeML implements a formally verified fragment of ML in the HOL4 proof system while seL4 provides a verified microkernel with VMM support. Verifying the Copland CakeML implementation and individual Copland phrases requires embedding the CakeML semantics in Coq. The Copland implementation will then be verified with respect to the formal semantics. Additionally, the Coq semantics supports proof search techniques for synthesizing Copland phrases. Running the CakeML implementation on the seL4 platform with formally synthesized phrases provides a verified attestation platform that may be retargeted to any environment supporting seL4.

As we continue exploring the richness of layered attestation we are also developing type systems and static checkers that determine correctness of specific protocols and protocol interpreters and compilers that produce provably correct results relative to Copland semantics. We are considering extensions to Copland that include nonces, lambda expressions, keys, and TPM interactions to represent a richer set of protocols. Without this formal semantics, it would be impossible to consider the correctness of such extensions.

A Annotated Terms

As noted in Sect. 4, when t is annotated by i and j, we write $[t]_i^j$. The annotations are used in the Coq proofs to construct sequences of unique events associated with collecting the evidence specified by the term.

$$
\begin{aligned}
&\mathsf{anno}(i, \mathsf{CPY}) = (i+1, [\mathsf{CPY}]_i^{i+1}) \\
&\mathsf{anno}(i, \mathsf{KIM}\ p\ \bar{a}) = (i+1, [\mathsf{KIM}\ p\ \bar{a}]_i^{i+1}) \\
&\mathsf{anno}(i, \mathsf{USM}\ \bar{a}) = (i+1, [\mathsf{USM}\ \bar{a}]_i^{i+1}) \\
&\mathsf{anno}(i, \mathsf{SIG}) = (i+1, [\mathsf{SIG}]_i^{i+1}) \\
&\mathsf{anno}(i, \mathsf{HSH}) = (i+1, [\mathsf{HSH}]_i^{i+1}) \\
&\mathsf{anno}(i, @_p\, t) = \\
&\quad \mathsf{let}\ (j, a) \leftarrow \mathsf{anno}(i+1, t)\ \mathsf{in} \\
&\quad \mathsf{anno}(j+1, [@_p\, a]_i^{j+1}) \\
&\mathsf{anno}(i, t_1 \to t_2) = \\
&\quad \mathsf{let}\ (j, a_1) \leftarrow \mathsf{anno}(i, t_1)\ \mathsf{in} \\
&\quad \mathsf{let}\ (k, a_2) \leftarrow \mathsf{anno}(j, t_2)\ \mathsf{in} \\
&\quad \mathsf{anno}(k, [a_1 \to a_2]_i^k) \\
&\mathsf{anno}(i, t_1 \overset{s}{\prec} t_2) = \\
&\quad \mathsf{let}\ (j, a_1) \leftarrow \mathsf{anno}(i+1, t_1)\ \mathsf{in} \\
&\quad \mathsf{let}\ (k, a_2) \leftarrow \mathsf{anno}(j, t_2)\ \mathsf{in} \\
&\quad \mathsf{anno}(k+1, [a_1 \overset{s}{\prec} a_2]_i^{k+1}) \\
&\mathsf{anno}(i, t_1 \overset{s}{\sim} t_2) = \\
&\quad \mathsf{let}\ (j, a_1) \leftarrow \mathsf{anno}(i+1, t_1)\ \mathsf{in} \\
&\quad \mathsf{let}\ (k, a_2) \leftarrow \mathsf{anno}(j, t_2)\ \mathsf{in} \\
&\quad \mathsf{anno}(k+1, [a_1 \overset{s}{\sim} a_2]_i^{k+1})
\end{aligned}
$$

Fig. 9. Term annotation

Terms are annotated using the function displayed in Fig. 9. An annotated term for $t = \mathsf{KIM}\ p\ \bar{a} \to \mathsf{SIG}$ is

$$\mathsf{anno}(0, t) = (2, [[\mathsf{KIM}\ p\ \bar{a}]_0^1 \to [\mathsf{SIG}]_1^2]_0^2),$$

and when $t = @_p\, \mathsf{USM}\ \bar{a}$,

$$\mathsf{anno}(0, t) = (3, [@_p\, [\mathsf{USM}\ \bar{a}]_1^2]_0^3).$$

Lemma 21. $\mathsf{anno}(i, t) \in T_i$.

B Coq Cross Reference

Table 1 matches the contents of a figure with its definition in the Coq proofs. Table 2 does the same for lemmas, definitions, and the theorem.

Table 1. Coq figure cross reference

Fig. 2: `Term.Term`	Fig. 3: `Term.eval`	Fig. 4: `Term.well_founded`
Fig. 5: `Term.Ev`	Fig. 6: `Term.events`	Fig. 7: `Term_system.ev_evsys`
Fig. 8: `LTS.step`	Fig. 9: `Term.anno`	

Table 2. Coq cross reference

Lem. 1:	`Term.event_range`
Lem. 2:	`Term.events_injective`
Lem. 3:	`Term.events_range_event`
Def. 4:	`Event_system.prec`
Lem. 7:	`Event_system.evsys_irreflexive`
Lem. 8:	`Event_system.evsys_transitive`
Lem. 9:	`Term_system.evsys_events`
Def. 10:	`Term_system.out_ev`
Lem. 11:	`Term_system.max_eval`
Lem. 12:	`LTS.steps_preserve_eval`
Lem. 13:	`LTS.never_stuck`
Lem. 14:	`LTS.steps_to_stop`
Thm. 15:	`Main.ordered`
Def. 16:	`Trace.shuffle`
Lem. 17:	`Trace.trace_length`
Lem. 18:	`Trace.trace_events`
Lem. 19:	`Main.lstar_trace`
Lem. 20:	`Trace.trace_order`

References

1. The Coq proof assistant reference manual (2018). Version 8.0. http://coq.inria.fr
2. Carpent, X., Tsudik, G., Rattanavipanon, N.: ERASMUS: efficient remote attestation via self-measurement for unattended settings. In: 2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, 19–23 March 2018, pp. 1191–1194 (2018)
3. Clemens, J., Paul, R., Sherrell, B.: Runtime state verification on resource-constrained platforms. In: Military Communications Conference (MILCOM) 2018, October 2018
4. Coker, G., et al.: Principles of remote attestation. Int. J. Inf. Secur. **10**(2), 63–81 (2011)
5. Datta, A., Franklin, J., Garg, D., Kaynar, D.: A logic of secure systems and its application to trusted computing. In: 2009 30th IEEE Symposium on Security and Privacy, pp. 221–236. IEEE (2009)
6. Eldefrawy, K., Rattanavipanon, N., Tsudik, G.: HYDRA: hybrid design for remote attestation (using a formally verified microkernel). In: Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec 2017, Boston, MA, USA, 18–20 July 2017, pp. 99–110 (2017)
7. Francillon, A., Nguyen, Q., Rasmussen, K.B., Tsudik, G.: A minimalist approach to remote attestation. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, 24–28 March 2014, pp. 1–6 (2014)

8. Gopalan, A., Gowadia, V., Scalavino, E., Lupu, E.: Policy driven remote attestation. In: Prasad, R., Farkas, K., Schmidt, A.U., Lioy, A., Russello, G., Luccio, F.L. (eds.) MobiSec 2011. LNICSSITE, vol. 94, pp. 148–159. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30244-2_13
9. Trusted Computing Group: TPM Main Specification Level 2 version 1.2 (2011)
10. Jaeger, T., Sailer, R., Shankar, U.: PRIMA: policy-reduced integrity measurement architecture. In: Proceedings of the 11th ACM Symposium on Access Control Models and Technologies, SACMAT 2006, Lake Tahoe, California, USA, 7–9 June 2006, pp. 19–28 (2006)
11. Klein, G., et al.: seL4: formal verification of an operating-system kernel. Commun. ACM **53**(6), 107–115 (2010)
12. Kumar, R., Myreen, M.O., Norrish, M., Owens, S.: CakeML: a verified implementation of ML. In: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2014, pp. 179–191. ACM, New York (2014)
13. Lauer, H., Ahmad Salehi, S., Rudolph, C., Nepal, S.: User-centered attestation for layered and decentralized systems. In: Workshop on Decentralized IoT Security and Standards (DISS) 2018, February 2018
14. Loscocco, P., Wilson, P.W., Aaron Pendergrass, J., Durward McDonell, C.: Linux kernel integrity measurement using contextual inspection. In: Proceedings of the 2nd ACM Workshop on Scalable Trusted Computing, STC 2007, Alexandria, VA, USA, 2 November 2007, pp. 21–29 (2007)
15. Maliszewski, R., Sun, N., Wang, S., Wei, J., Qiaowei, R.: Trusted boot (tboot). http://sourceforge.net/p/tboot/wiki/Home/
16. Pendergrass, A., Helble, S., Clemens, J., Loscocco, P.: A platform service for remote integrity measurement and attestation. In: Military Communications Conference (MILCOM) 2018, October 2018
17. Rowe, P.D.: Bundling evidence for layered attestation. In: Franz, M., Papadimitratos, P. (eds.) Trust 2016. LNCS, vol. 9824, pp. 119–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45572-3_7
18. Rowe, P.D.: Confining adversary actions via measurement. In: Kordy, B., Ekstedt, M., Kim, D.S. (eds.) GraMSec 2016. LNCS, vol. 9987, pp. 150–166. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46263-9_10
19. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: Proceedings of the 13th USENIX Security Symposium, San Diego, CA, USA, 9–13 August 2004, pp. 223–238 (2004)
20. Shi, E., Perrig, A., van Doorn, L.: BIND: a fine-grained attestation service for secure distributed systems. In: 2005 IEEE Symposium on Security and Privacy (S&P 2005), Oakland, CA, USA, 8–11 May 2005, pp. 154–168 (2005)
21. Tan, H., Tsudik, G., Jha, S.: MTRA: multiple-tier remote attestation in IoT networks. In: 2017 IEEE Conference on Communications and Network Security (CNS). IEEE, October 2017
22. Xu, W., Ahn, G.-J., Hu, H., Zhang, X., Seifert, J.-P.: DR@FT: efficient remote attestation framework for dynamic systems. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 182–198. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15497-3_12