Design and Formal Verification of a Copland-based Attestation Protocol

Adam Petz
Information and Telecommunication Technology Center,
The University of Kansas
Lawrence, KS, USA
ampetz@ku.edu

Grant Jurgensen
Information and Telecommunication Technology Center,
The University of Kansas
Lawrence, KS, USA
gajurgensen@ku.edu

Perry Alexander
Information and Telecommunication Technology Center,
The University of Kansas
Lawrence, KS, USA
palexand@ku.edu

ABSTRACT

We present the design and formal analysis of a remote attestation protocol and accompanying security architecture that generate evidence of trustworthy execution for legacy software. For formal guarantees of measurement ordering and cryptographic evidence strength, we leverage the Copland language and Copland Virtual Machine execution semantics. For isolation of attestation mechanisms we design a layered attestation architecture that leverages the seL4 microkernel. The formal properties of the protocol and architecture together serve to discharge assumptions made by an existing higher-level model-finding tool to characterize all ways an active adversary can corrupt a target and go undetected. As a proof of concept, we instantiate this analysis framework with a specific Copland protocol and security architecture to measure a legacy flight planning application. By leveraging components that are amenable to formal analysis, we demonstrate a principled way to design an attestation protocol and argue for its end-to-end correctness.

CCS CONCEPTS

• **Security and privacy** → **Logic and verification**; **Trust frameworks**; **Domain-specific security and privacy architectures**.

KEYWORDS

remote attestation, formal methods, verification

ACM Reference Format:
Adam Petz, Grant Jurgensen, and Perry Alexander. 2021. Design and Formal Verification of a Copland-based Attestation Protocol. In *19th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE '21), November 20–22, 2021, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3487212.3487340

This work is funded in part by the NSA Science of Security initiative contract #H98230-18-D-0009 and Defense Advanced Research Project Agency contract #HR0011-18-9-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

MEMOCODE '21, November 20–22, 2021, Beijing, China

ACM ISBN 978-1-4503-9127-6/21/11…$15.00
https://doi.org/10.1145/3487212.3487340

1 INTRODUCTION

A common goal for communicating systems is *trust* where we say that principal B trusts A if it can strongly identify A and either directly observe A behaving as expected or indirectly observe A through a trusted third party. Semantic remote attestation [4, 5] is one mechanism supporting trust establishment. In remote attestation an *appraiser* (B) requests an *attestation* from a remote *target* (A). The target responds by performing the requested attestation and returning *evidence* of its state to the appraiser. The appraiser assesses the evidence and chooses whether to trust the remote system.

This work centers on hardening a legacy system into one where trust can be established and maintained over time. Remote attestation is added to a UAV control system to establish trust between the UAV and a ground station providing waypoints for navigation. Formal methods–proof and model finding–are leveraged throughout the retrofit process to ensure correctness in the resulting system. The goal being demonstrating the use of formal methods in system design involving multiple components, requirements, and constraints.

Working with colleagues at MITRE, JHUAPL and NSA we have developed Copland [17] for representing, reasoning about, and executing remote attestation protocols. Copland specifies atomic measurements for gathering evidence, mechanisms for sequencing measurements, and remote measurement requests for layering attestation. Copland has a well-defined formal semantics for execution specifying the structure of cryptographic evidence produced and the precise order measurement activities are performed.

In prior work we defined Copland's formal semantics [17], defined and verified an execution environment for Copland that abides by the formal semantics [15], and illustrated usage patterns for layered attestation [6]. In this work we introduce a novel workflow for the formal analysis of Copland-based attestation protocols and demonstrate that workflow on a specific Copland protocol and security architecture for a UAV/Groundstation demonstration platform.

2 COPLAND

The primary aims of the Copland framework [1, 6, 7, 14–17, 19] are to support the specification, execution, and analysis of layered attestation protocols. Protocols are specified as language terms called *phrases*, which serve as input to both execution and analysis alike. During execution phrases are interpreted to invoke local measurement routines, request remote measurements, and bundle evidence results cryptographically. During analysis phrases denote a precise

measurement ordering and cryptographic evidence structure that, if upheld by an execution, constrains an adversary by the attacks it can perform and go undetected by attestation.

While complete descriptions of the Copland language and its semantics can be found elsewhere [1, 17] we describe here only what is relevant to understand the phrases presented in this work. The most primitive term in Copland concrete syntax is a *measurement* and is a triple of the form: (S Q T) where S and T are symbols representing the measurement agent and target of measurement, respectively, and Q the place where T resides. As an example, the phrase:

```
(comp_hash Q dir)
```

is the composite hash, `comp_hash`, measurement performed over the contents of directory `dir` residing at place `Q`.

A simple extension to this protocol is:

```
(comp_hash Q dir) -> !
```

where the resulting hash evidence is sent to the signature operator, `!`, that signs it with a cryptographic key local to place `Q`. In Copland, a *place* is an abstract identifier for an attestation manager, and each place has an associated key used for signing evidence. The protocol where place `P` asks place `Q` to perform and sign a composite hash has the form:

```
P: @Q [(comp_hash Q dir) -> ! ]
```

The notation `P:` indicates `P` is the top-level place where this protocol is orchestrated and appraised. The `@Q` operator indicates a request to `Q` to execute measurements on `P`'s behalf. The result at `P` is a composite hash of `dir` at `Q`, both performed and signed by `Q`. This evidence can be appraised by checking the signature with `Q`'s public key and comparing the hash with a known golden value.

3 COPLAND ANALYSIS FRAMEWORK

In prior work we developed a formally-verified execution environment for Copland phrases called the Copland Virtual Machine (CVM) [13, 15]. A crucial property of CVM execution is that it emits measurement event traces and cryptographic evidence which are both faithful to the Copland reference semantics. Here we also consider the security architecture of the platform hosting a CVM, and how formal properties of the CVM and its architecture bolster the soundness of a higher-level automated analysis.

Figure 1 shows our general framework for analyzing and executing an arbitrary Copland phrase t. The Appraiser platform takes t as input and relies on a stateful execution environment introduced in Petz et al. [15] to generate and remember a random nonce, construct an attestation session around that nonce, and perform appraisal over the evidence result. The Target platform leverages the CVM to carry out the request and emits an Event Trace guaranteed to respect a strict partial ordering of attestation-relevant events called an *Event System*, derived statically from t.

Alongside the CVM on the Target platform, measurement components m_1, m_2, ..., and target applications a_x, a_y, ..., operate within a Security Architecture depicted in Figure 1 by dotted green lines around components. Our analysis framework remains parameterizable over such *means of isolation* that protect trusted measurers from their potentially-untrusted targets. This supports dropping in alternative isolation mechanisms as deemed appropriate by the

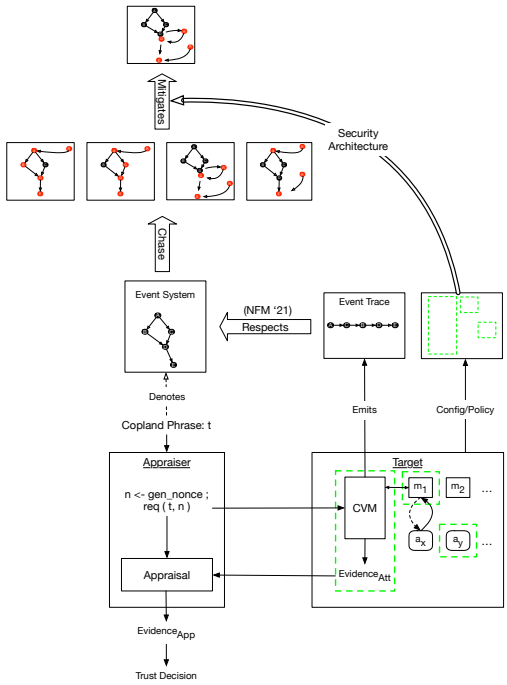

Figure 1: Copland Verification Architecture

consumer of a particular attestation. In Section 4 we instantiate a concrete measurement architecture that leverages the formally-verified seL4 microkernel [8] for component separation.

Our higher-level analysis leverages a model-finding tool developed by our collaborators at MITRE instrumented explicitly to analyze Copland-based protocols [19]. Given a Copland phrase t and an indication of the measurement target(s) of interest, the tool produces an initial set of models that describe all distinct ways an active adversary could corrupt the target and go undetected by measurement. This initial analysis relies on existing axiomatizations of Event Systems and an adversary that are encoded as first-order logical statements understood by the underlying Chase [18] model-finder. The encoding of Event Systems characterizes honest measurements derived directly from t, while the adversarial model encodes rules of a capable attacker that can arbitrarily corrupt and repair the very same measurement components.

One of the goals of a well-designed attestation system is that it should place a *high as possible* burden on the attacker[20, 21]. Towards this goal, our analysis framework incorporates additional constraints to reflect properties of the security architecture, eliminating certain classes of attack from consideration. More concretely, we encode architectural properties as logical assumptions to the model finder to indicate corruptibility based on the integrity of individual components and their contextual dependencies. With measurement protocol and architectural assumptions incorporated, the attack models that remain must be acceptable to the consumer of attestation. Otherwise, an iteration to refactor the Copland protocol and accompanying security architecture may be in order.

4 DEMONSTRATION PLATFORM

Our demonstration platform is an Un-piloted Air Vehicle (UAV) system taken from our work on the DARPA CASE program. We start with a pre-existing legacy implementation, extend it to support layered attestation built with Copland Attestation Managers, and formally analyze the resulting system. The transformed architecture exhibits desirable security properties that we can leverage as sound assumptions in the analysis framework from Figure 1.

The scenario features two communicating systems, a ground station and a UAV. The UAV accepts flight plans from the ground station in the form of waypoints and attempts to navigate within security constraints. Both the ground station and UAV run pre-existing UxAS [2] software, running in Linux. In the un-hardened system the UAV has no assurance that the ground station it accepts directions from is trustworthy. The UAV has no means of distinguishing a compromised or fake ground station from a genuine, trustworthy one. To address these concerns, we extend the two systems to support Copland attestation protocols.

4.1 Transformation of the UAV

In the hardened architecture, the UAV is augmented with a filter, paired with a Copland Attestation Manager (AM). The filter intercepts communication between the ground station and the UAV flight planner, forwards messages from trusted ground stations, and drops messages from un-trusted ones, as determined by the attestation manager. For a ground station that is neither trusted nor un-trusted, the attestation manager will send it a Copland attestation request. If the ground station responds with corresponding evidence, then the attestation manager will appraise the evidence and provide a trustworthy/non-trustworthy decision to the filter. This extended UAV system is presented in Figure 2.

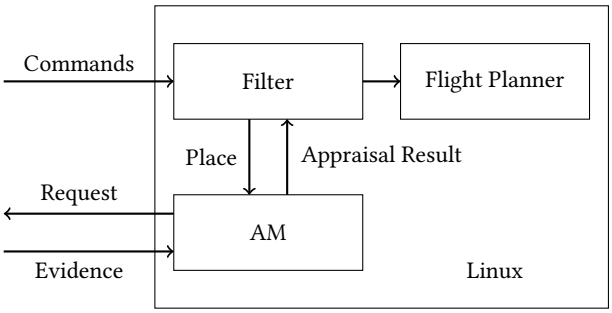

Figure 2: Hardened UAV Architecture

4.2 Transformation of the ground station

Where the UAV had to be extended to support appraisal, the ground station must be extended to support attestation, requiring more substantial architectural changes to the system. First, we add a Copland Attestation Manager to the Linux environment running UxAS. This AM is outfitted with a number of Linux measurement procedures and UxAS-specific measurers to support attestation. As such, it is primarily responsible for monitoring the authenticity of the legacy ground station software and its outgoing messages.

Unfortunately, trust in this AM is necessarily limited by its position in the Linux environment. It is running in the same Linux userspace as its target, UxAS. It cannot be trusted to unearth deeply embedded threats such as a rootkit from the Linux userspace because we cannot guarantee sufficient separation between the AM and the equally privileged malicious actor. Such an actor could tamper with the AM's measurements, or even steal its private key and impersonate it outright.

For this reason, we virtualize the Linux environment, and add another layer to the system. In this new model, the ground station runs the seL4 microkernel, with two components. One is a virtual machine manager, hosting the aforementioned Linux environment. The other component is an additional attestation manager. To distinguish the two, we refer to the attestation manager running in the Linux environment as the *UserAM*, and the attestation manager running at the seL4 level as the *PlatformAM*, both depicted in Figure 3.

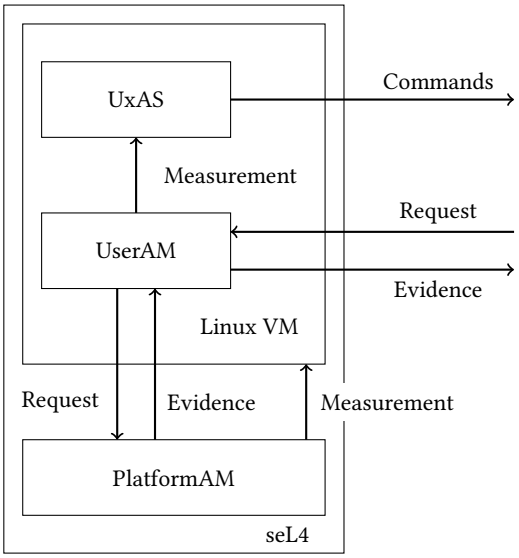

Figure 3: Hardened Ground Station Architecture

Introduction of the seL4 layer allows the PlatformAM to externally conduct measurements of the vulnerable Linux environment. Crucially, seL4 possesses strong memory separation capabilities which prevents a compromised and malicious Linux kernel from interfering with the PlatformAM's measurements. These memory access controls are part of the specification of seL4 that has been formally verified, allowing for provably-separate components [9].

The two attestation managers work together in a layered attestation paradigm. The UserAM can make granular measurements easily from within the Linux environment, while the PlatformAM can attest to the healthiness of the Linux VM as a whole. In this sense, the PlatformAM extends its increased trustworthiness to the UserAM by measuring its Linux environment.

Finally, we must consider what supports the trustworthiness of the PlatformAM and the seL4 layer. The memory separation properties of seL4 are static guarantees, thus we trust this layer

so long as the correct seL4 image boots. To accomplish this, we need to anchor our trust in some hardware-based root-of-trust. This evidence will necessarily be hardware-specific. For our design purposes, we assume that the boot process leaves some evidence token available to the PlatformAM which convincingly indicates the proper seL4 image booted.

5 TRANSFORMED UAV PLATFORM ANALYSIS

In this section we instantiate the Copland Verification Architecture of Figure 1 to analyze the UAV attestation scenario. We begin by describing a Copland phrase and its components, then derive its honest measurement events from the CVM semantics that serve as input to analysis. For analysis we encode increasingly strong properties of the groundstation security architecture, inspect the generated attack models, and finally consider tradeoffs between alternative measurement strategies that differ in depth and frequency.

5.1 Copland Phrase Description/Components

A Copland phrase to measure the transformed ground station target platform appears in Figure 4. The phrase begins with: ∗heliAM, n : which designates heliAM (the UAV AM component from Figure 2) as the *appraising place* and also specifies a nonce n be passed as initial evidence with the attestation request. Next, @userAM[...] specifies that the terms inside [...] be executed at the userAM place. userAM and platAM are place identifiers that represent distinct attestation domains both running on the remote ground station (UserAM and PlatformAM from Figure 3).

As soon as userAM receives the initial request, @platAM[...] specifies that it initiate a request to the (more privileged) Platform AM. platAM starts by invoking query_img to read the contents of the seL4 image img loaded at boot-time into protected memory. This image serves as evidence of the static configuration of all components on the ground station platform at startup. The abstract nature of places in Copland allows us to represent this protected memory region as its own place identifier bootMem that will be incorporated into analysis.

After querying the image, platAM performs cross-domain measurements of components at userAM. One is (kim userAM ker) that specifies an integrity measurement of the linux OS kernel running at userAM. The other is (uim userAM uam) that specifies an integrity measurement of the uam (Userspace Attestation Manager) component that itself performs more specialized measurements of UxAS. The +~+ operator specifies that both of its subterms may execute in parallel, whereas -> requires strict linear sequencing.

After receiving evidence of platform integrity from platAM, userAM proceeds to perform specialized measurements of the target application. These measurements perform dynamic monitoring of the execution context of the UxAS flight planning software and UxAS itself. After completing its measurement, userAM signs the accumulated evidence bundle and sends it in a response back to heliAM.

5.2 Event Semantics

Figure 5 shows the Event System, a partial ordering on measurement events, determined by the Copland phrase in Figure 4. This

```
*heliAM, n:
    @userAM [
             @platAM [ (query_img bootMem img) ->
                       ((kim userAM ker)
                              +~+
                        (uim userAM uam)) -> !
                      ] ->
             ((uam userAM uxas_ctxt)
                     +~+
              (uam userAM uxas)) -> !
             ]
```

Figure 4: UAV Copland Phrase

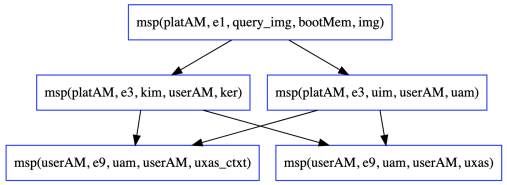

Figure 5: Event System derived from the Copland phrase in Figure 4 above.

```
l(E) = msp(userAM, e, uam, userAM, uxas)
  => prec(E,E2) &
     phi(userAM, uxas, E2) | phi(userAM, uxas_ctxt, E2).
```

Figure 6: Assume adversary avoids detection at main measurement event.

graphical output comes from the model finder tool, but an identical partial ordering can be derived from the Copland formal semantics in Coq. This ordering states that the boot image is queried first, followed by the integrity measurements launched from platAM. The integrity measurements are free to execute in any order, but must complete before the specialized userspace measurements at userAM begin. This event ordering is the first input into the automated attack analysis in the model finder, and its soundness is justified by the Copland Virtual Machine that is formally verified to uphold such measurement orderings.

5.3 Architectural Assumptions

Given the measurement events in Figure 5, the model finding tool requires additional assumptions about the environment in which these measurements are carried out before it can produce a meaningful analysis. The first of these is an indication of the *measurement event(s) of interest*. In other words, the instant(s) during attestation where we would like to determine if the adversary has sufficiently corrupted a specific target while avoiding detection. Figure 6 shows this statement for our UAV attestation scenario, encoded in first-order syntax accepted by the model finder. This logical statement asks for models where either uxas or something in its execution context uxas_ctxt are corrupt (phi predicate) after the event where uam measures uxas.

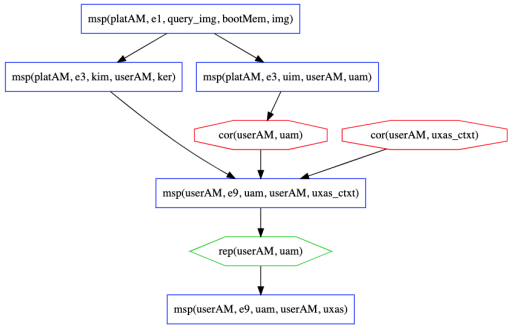

Figure 7: One attack model: corruption and repair of uam

```
% No dependencies for components at bootMem or platAM
ctxt(bootMem, C, C2) => false.
ctxt(platAM, C, C2) => false.
```

Figure 8: Contextual assumptions about "deep" components in the architecture.

Given only honest measurement events and this event of interest, the model finder will generate an exhaustive set of attack models assuming a capable adversary that can corrupt and repair arbitrary components on the system. One such model appears in Figure 7. Here we see the adversary has corrupted the userspace AM (uam) after it was measured, and leveraged it to lie about the corrupted state of uxas_ctxt. It later covers its tracks by repairing uam. In total, the tool generates 74 *essentially distinct* attack models like this. Because many of these attacks are unacceptable, analysis like this early in the design of an attestation protocol is useful to pinpoint parts of the system that may require hardening.

Further assumptions come from properties about dependencies of components and the architecture where they operate. The statements in Figure 8 encode that components at bootMem and platAM do not depend on any other components for their own integrity. Without assumptions like these the analysis conservatively assumes that for each component, there exists an arbitrary component co-resident at its place capable of affecting its integrity. Each of these statements are justified by trust in specific components and their environment: bootMem is a protected storage location; measurement components at platAM run in an isolated, native seL4 environment with limited dependencies and limited-purpose code.

Statements in Figure 9 encode additional assumptions about the corruptibility of components. The first says that only way to corrupt a component at platAM is by corrupting img at bootMem. The final two statements say that the boot image cannot become corrupted, and that the only way for uam to become corrupted is via a corrupted OS kernel. The first of these is justified by protecting the image somewhere like trusted hardware or a dedicated seL4 component, where only highly-priviledged bootloader code has write access. To justify the latter, in our prototype we limit the code of uam to very specific measurement functions and include them as a library packaged with the Copland Virtual Machine at userAM.

Figure 10 lists four final assumptions that eliminate most of the remaining feasible attack models. The first three make explicit the

```
% platAM components only corrupted via a corrupt boot image
l(E) = cor(platAM, C) => phi(bootMem, img, E).

%% img in bootMem cannot be corrupted
phi(bootMem, img, E) => false.

% user AM (uam) only corrupted via a corrupt kernel
l(E) = cor(userAM, uam) => phi(userAM, ker, E).
```

Figure 9: Assumptions about the "corruptibility" of components.

```
% All components at userAM (except ker itself)
% depend on ker
ctxt(userAM, C, uam) => C = ker.
ctxt(userAM, C, uxas_ctxt) => C = ker.

% In addition to ker, uxas depends also on uxas_ctxt
ctxt(userAM, C, uxas) => C = ker | C = uxas_ctxt.

% Ignore attacks that corrupt ker
l(E) = cor(userAM, ker) => false.
```

Figure 10: Final Architecture Assumptions

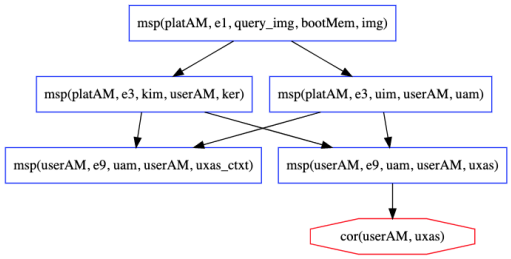

Figure 11: Final Attack Model

context of the remaining components in userspace. The final assumption ignores attacks on the kernel. Because dynamic attacks on OS kernels are feasible in practice, one could remove this assumption to explore the implications of such an attack. However, for our final analysis we assume that the (kim userAM ker) measurement lauched from platAM gives sufficient evidence that the kernel will run uncorrupted long enough for the other measurements that depend on it to complete.

Given all of the above assumptions, the attack model in Figure 11 characterizes one of the two remaining ways to corrupt uxas and go undetected (an analogous model exists for corrupting uxas_ctxt). Specifically, the adversary must corrupt uxas (or its context) *after* they are measured and before the flight planning services are consumed. This is an example of a recent or time-sensitive attack that is in theory more difficult to execute[21]. In what follows, we discuss strategies for making these types of attacks more difficult still by repeated measurement, thus justifying them as acceptable attack models.

5.4 AM Monad alternatives

From the appraiser's perspective a Copland protocol is executed atomically before evidence bubbles back to its environment. While it is possible to craft standalone phrases that are sufficient for simple, static attestation scenarios, there are cases where an appraiser might desire more flexibility to orchestrate the execution of multiple Copland phrases and compose intermediate evidence results. Pure Copland has no persistent state and no error handling mechanism to account for failed or divergent attestations. To address these issues we define the *Attestation Manager (AM) Monad* as a standard state monad with exceptions with a formal definition in Coq [13] and prototype implementations in CakeML and Haskell[7, 16].

An example computation in the AM Monad called `attest_gs` that prepares, executes, and appraises a Copland phrase from the UAV scenario is as follows:

```
attest_gs t :=
  do {n  <- generateNonce;
      ev <- run_cvm(n,t);
      b  <- appraise n t ev;
      update_filter(b)}
```

For flexibility, we parameterize `attest_gs` by the Copland phrase used to measure the groundstation target. Assuming the Copland phrase from earlier in Figure 4 is assigned the name `case_cop`, we could instantiate this AM Monad computation via function application: `attest_gs case_cop`.

Recall from earlier that even with architectural protections and bottom-up measurement strategies, an adversary can escape detection by performing timely attacks on components after they are measured. One way to make these attacks less effective is to perform periodic re-measurement of the system:

```
do_while(true) (
  attest_gs case_cop;
  sleep(s))
```

Here `s` is within some time interval chosen by the appraiser to make attack and repair of the target difficult for the adversary. Depending on the scenario and capabilities of the attacker, `s` could be bounded under a certain threshold, or even randomly generated within an acceptable range.

However, a problem arises if the time required to complete all measurements of the phrase exceeds `s`. Recall that the `case_cop` phrase involves measurements of deep components that might require significant time and system resources to carry out. In real-time embedded systems with hard scheduling requirements, attestation services might be given constrained resources (i.e. CPU time) to complete measurement tasks [3]. This limitation may be at odds with deeper measurements that tend to stall or freeze the system to capture its state. When full appraisals are too costly, it may be that performing one deep measurement of the system during initialization followed by repeated, shallow probes is sufficient to establish a baseline and maintain evidence of integrity. Partitioning the phrase into its deep and shallow portions has the form:

```
case_deep :=
    @userAM [ @platAM [ (query_img bootMem img) ->
                        ((kim userAM ker)
                                +~+
                         (uim userAM uam)) -> ! ]]

case_shallow :=
    @userAM [ ((uam userAM uxas_ctxt)
                      +~+
               (uam userAM uxas)) -> ! ]
```

A final AM Monad computation that performs a deep attestation of the ground station at frequency `s`, and shallow attestations at frequency `r` is as follows:

```
do_while(true) (
  attest_gs case_deep;
  do_for_duration(s) (
    attest_gs case_shallow;
    sleep(r)
  )
)
```

6 RELATED WORK

Nunes et. al. [10] developed VRASED (Verifiable Remote Attestation for Simple Embedded Devices) that uses LTL to verify end-to-end security and attestation soundness by composing properties of Verilog hardware specifications and cryptographic software. APEX [11] extends VRASED's security architecture to support unforgeable remote proofs of execution (PoX) on low-end devices. These efforts achieve convincing end-to-end security guarantees for a fixed embedded platform. In contrast, our Copland-based analysis supports diverse, layered attestation scenarios and alternative protocol/architecture designs with respect to a powerful adversary.

Sardar et. al. formally specify and verify properties of specialized attestation primitives used in Intel's SGX and TDX technologies [22–24]. They encode the protocol primitives as models in the ProVerif tool and perform symbolic analysis over the protocols with respect to a Dolev-Yao adversary.

Rowe's work on analysis of layered attestation protocols characterizes formally how measurement and evidence bundling strategies force an adversary to perform more difficult deep or recent attacks [20, 21]. Besides influencing the design of our UAV attestation protocol and architecture, the formalizations in this work underly the proofs of soundness for both honest and adversarial axiom extensions to the Chase model finder that we rely on for analysis [19]. Helble et. al. [6] demonstrate the complexity of the design space for Copland-based attestation protocols and motivate our frameowrk for analysis. On the implementation side, Maat [12] is of note as an attestation framework that motivated the design of the Copland language[17] and the Copland virtual machine[15].

The model finder tool was recently released publicly as part of the Copland Collection [1] and is described in depth in a recent publication [19]. The tool instruments the more general Chase model finder [18] with specialized axioms encoding the semantics of Copland measurement and the behavior of active adversaries that can corrupt and repair components involved in Copland attestations.

7 CONCLUSION

In this work we describe a prototypical system-level design process that integrates formal methods at critical decision points. The system-level goal is integrating a remote attestation system into a legacy system to establish trustworthiness. The formally verified Copland language and Copland Virtual Machine are used to define an attestation protocol and its semantics. The Attestation Manager Monad situates the protocol in an execution environment providing nonce management, appraisal, and measurement frequency. Leveraging properties of these verified components, MITRE's model finder tool enumerates allowed behaviors of an adversary to help in refining the attestation protocol. The result of this work is not a proof-of-correctness in a single formal tool, but a collection of evidence from multiple tools integrated into a design process that can be revisited as the UAV system changes or applied to alternate designs. The future of Copland and our design tools includes synthesis of implementations from Coq, integration of hardware-based static measurements, and generalizing our approach to other attestation architectures.

REFERENCES

[1] Copland website. https://ku-sldg.github.io/copland, 2021.
[2] Balachandran, S., Muñoz, C. A., Consiglio, M. C., Feliú, M. A., and Patel, A. V. Independent configurable architecture for reliable operation of unmanned systems with distributed onboard services. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)* (2018), pp. 1–6.
[3] Clemens, J., Pal, R., and Sherrell, B. Runtime state verification on resource-constrained platforms. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)* (2018), pp. 1–6.
[4] Coker, G., Guttman, J., Loscocco, P., Herzog, A., Millen, J., O'Hanlon, B., Ramsdell, J., Segall, A., Sheehy, J., and Sniffen, B. Principles of remote attestation. *International Journal of Information Security 10*, 2 (June 2011), 63–81.
[5] Haldar, V., Chandra, D., and Franz, M. Semantic remote attestation – a virtual machine directed approach to trusted computing. In *Proceedings of the Third Virtual Machine Research and Technology Symposium* (San Jose, CA, May 2004).
[6] Helble, S., Kretz, I., Loscocco, P., Ramsdell, J., Rowe, P., and Alexander, P. Flexible mechanisms for remote attestation. *ACM Transactions on Privacy and Security* (to appear).
[7] Jurgensen, G., Petz, A., Alexander, P., Barclay, T., Komp, E., Neises, M., and Cousino, A. A copland attestation manager (am) in cakeml. https://github.com/ku-sldg/am-cakeml, 2021.
[8] Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. sel4: formal verification of an operating-system kernel. *Commununications of the ACM 53*, 6 (2010), 107–115.
[9] Murray, T., Matichuk, D., Brassil, M., Gammie, P., Bourke, T., Seefried, S., Lewis, C., Gao, X., and Klein, G. sel4: From general purpose to a proof of information flow enforcement. In *2013 IEEE Symposium on Security and Privacy* (2013), pp. 415–429.
[10] Nunes, I. D. O., Eldefrawy, K., Rattanavipanon, N., Steiner, M., and Tsudik, G. Vrased: A verified hardware/software co-design for remote attestation. In *Proceedings of the 28th USENIX Conference on Security Symposium* (USA, 2019), SEC'19, USENIX Association, pp. 1429–1446.
[11] Nunes, I. D. O., Eldefrawy, K., Rattanavipanon, N., and Tsudik, G. APEX: A verified architecture for proofs of execution on remote devices under full software compromise. In *29th USENIX Security Symposium (USENIX Security 20)* (Aug. 2020), USENIX Association, pp. 771–788.
[12] Pendergrass, J. A., Helble, S., Clemens, J., and Loscocco, P. A platform service for remote integrity measurement and attestation. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)* (2018), pp. 1–6.
[13] Petz, A. copland-avm, nfm21 release. https://github.com/ku-sldg/copland-avm/releases/tag/v1.0, 2020.
[14] Petz, A., and Alexander, P. A copland attestation manager. In *Hot Topics in Science of Security (HoTSoS'19)* (Nashville, TN, April 8-11 2019).
[15] Petz, A., and Alexander, P. An infrastructure for faithful execution of remote attestation protocols. In *NASA Formal Methods* (Berlin, Heidelberg, 2021), A. Dutle, M. M. Moscato, L. Titolo, C. A. Muñoz, and I. Perez, Eds., vol. 12673 of *Lecture Notes in Computer Science*, Springer International Publishing, pp. 268–286.
[16] Petz, A., and Komp, E. haskell-am. https://github.com/ku-sldg/haskell-am, 2020.
[17] Ramsdell, J., Rowe, P. D., Alexander, P., Helble, S., Loscocco, P., Pendergrass, J. A., and Petz, A. Orchestrating layered attestations. In *Principles of Security and Trust (POST'19)* (Prague, Czech Republic, April 8-11 2019).
[18] Ramsdell, J. D. Chase: A model finder for finitary geometric logic. https://github.com/ramsdell/chase, 2020.
[19] Rowe, P., Ramsdell, J., and Kretz, I. Automated trust analysis of copland specifications for layered attestations. In *Principles and Practice of Declarative Programming (PPDP 21)* (Sept. 2021).
[20] Rowe, P. D. Bundling Evidence for Layered Attestation. In *Trust and Trustworthy Computing*. Springer International Publishing, Cham, Aug. 2016, pp. 119–139.
[21] Rowe, P. D. Confining adversary actions via measurement. *Third International Workshop on Graphical Models for Security* (2016), 150–166.
[22] Sardar, M. U., Faqeh, R., and Fetzer, C. Formal foundations for intel SGX data center attestation primitives. In *Formal Methods and Software Engineering - 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1-3, 2021, Proceedings* (2020), S. Lin, Z. Hou, and B. P. Mahony, Eds., vol. 12531 of *Lecture Notes in Computer Science*, Springer, pp. 268–283.
[23] Sardar, M. U., Musaev, S., and Fetzer, C. Demystifying attestation in intel trust domain extensions via formal verification. *IEEE Access 9* (2021), 83067–83079.
[24] Sardar, M. U., Quoc, D. L., and Fetzer, C. Towards formalization of enhanced privacy ID (epid)-based remote attestation in intel SGX. In *23rd Euromicro Conference on Digital System Design, DSD 2020, Kranj, Slovenia, August 26-28, 2020* (2020), IEEE, pp. 604–607.