TapCon: Practical Third-Party Attestation for the Cloud

Yan Zhai [⋆] Qiang Cao[†] Jeffrey Chase[†] Michael Swift [⋆]
[⋆]University of Wisconsin Madison [†]Duke University

Abstract

One way to establish trust in a service is to know what code it is running. However, verified code identity is currently not possible for programs launched on a cloud by another party. We propose an approach to integrate support for code attestation—authenticated statements of code identity—into layered cloud platforms and services.

To illustrate, this paper describes *TapCon*, an attesting container manager that provides source-based attestation and network-based authentication for containers on a trusted cloud platform incorporating new features for code attestation. *TapCon* allows a third party to verify that an attested container is running specific code bound securely to an identified source repository. We also show how to use attested code identity as a basis for access control. This structure enables new use cases such as joint data mining, in which two data owners agree on a safe analytics program that protects the privacy of their inputs, and then ensure that only the designated program can access their data.

1 Introduction

Services are often composed of multiple pieces of software, managed by a variety of frameworks. In the cloud, infrastructure-as-a-service (IaaS) providers launch virtual machine instances (VMs) with specified program images. Within those VMs, a platform service based on a container manager such as Docker may further launch microservices from yet another set of images.

This paper addresses the question of how to reason about security properties of service applications running in this complex environment. For example, clients of a service may want assurances about the service's code before trusting it with sensitive data. We want to enable reasoning about the whole stack of software embodied in a service by considering the properties of each component and how they are combined.

If we trust a program P to have certain properties, then we can infer that a running instance of P also has those properties. But how to be sure a cloud instance is running a trusted program P? In general, it is possible only if one controls the infrastructure P runs on, the build chain that produces an executable image for P, and any other software that is present at runtime to interpret or assist it—e.g., an operating system and runtime libraries.

In a cloud setting, clients of a service know little about any of the links in this trust chain, because the service is hosted by another party (an IaaS provider) and often managed by yet another party, e.g., a SaaS service provider, who may be a tenant of the IaaS service. Today it is common to rely instead on *social trust* in the service provider to maintain the advertised properties.

Software identity can be a stronger basis for trust when clients have knowledge of the software. For example, clients may have trust in installations of untampered open-source IaaS implementations such as OpenStack, container management services such as Docker, and data analytics platforms such as Spark. These systems are in wide use and their source code is open to public inspection. As noted by the Nexus OS authors, trust in code may also come from analysis, synthesis, or fiat [19]. Program analysis can verify security properties of source code [15, 6, 12, 11], and trusted code generators may assure various properties of generated code [13]. Software trust could also derive from "fiat" endorsement by a trusted authority, such as a curator of open-source code, or a private auditor (for proprietary code).

When software is trusted, we can provide a stronger basis for trust in services running in the cloud through *attestations of code identity*—authenticated statements by trusted parties about the code that runs in a service instance. These attestations may be combined with endorsements of the code or its security properties to infer trust in an instance. In particular, we focus on *third-party attestation*, in which a third party—other than the owner of a tenant service or of the host machine—consumes the attestations and endorsements. For example, a customer can trust a cloud-hosted service if it trusts the program that the service runs and the platform that hosts it.

Recent works in cloud attestation focus on a trusted hardware platform to attest to hashes of running binary images (e.g., using a TPM [10] or Intel SGX [8]). Haven [5] and SCONE [4] run service instances in hardware-protected SGX enclaves [2] and attest to a hash of the code so that the instance owner can verify that it was launched correctly. However, binary hashes are of limited value to a third party who cannot verify which software produced the image. Other works show that SGX and TPMs can be a basis for third-party attestation [14, 9, 19] given trusted program layers whose hashes are widely known.

We propose a logical attestation approach for layered platform stacks anchored in a trusted IaaS cloud, and show how to use it for end-to-end scenarios that incorporate attested code identity into an access control scheme. To illustrate, we describe TapCon[1], an attesting container manager built on Docker. TapCon publishes *source-based attestations* about the code identity of container instances that it launches. To reduce reliance on expensive asymmetric cryptography, we leverage the IaaS provider's managed network to block spoofed packets, so that network addresses may be used directly as authenticated instance identifiers that bind an instance securely to its attested code.

Our work makes the following contributions:

- TapCon is compatible with existing cloud infrastructure services and applications, needing only small callouts to publish and verify attestations.
- TapCon provides attestation of a binary program's provenance back to a trusted source code repository (§3), as an alternative to a binary program hash.
- TapCon illustrates a practical and general architecture for *layered attestation* for extensible cloud platform stacks, from virtual machines to containers to pluggable applications, with trust at each level rooted at the level below.
- TapCon roots its trust in cloud providers and trusted source code repositories, and does not depend on special hardware (e.g., TPM and SGX) or cryptographic protocols, although they can be used.

We first motivate and provide an overview (§2), followed by a description of the TapCon service (§3) and the role of logical attestation (§4). We conclude with a brief evaluation of our current prototype to show that attestations are sufficiently cheap for practical use.

2 Motivation and Overview

As a motivating application scenario, consider the problem of *joint data mining*: suppose that two groups each have a private dataset and want to cooperate by running analytics program P over both datasets (A and B) together. They trust that P produces output that does not expose confidential details of A or B to one another. They choose to leverage cloud infrastructure and common platform frameworks—an open analytics stack in

[1]TapCon is Trusted Attestation Platform for Containers, or a fastener that anchors a structure to a solid foundation.

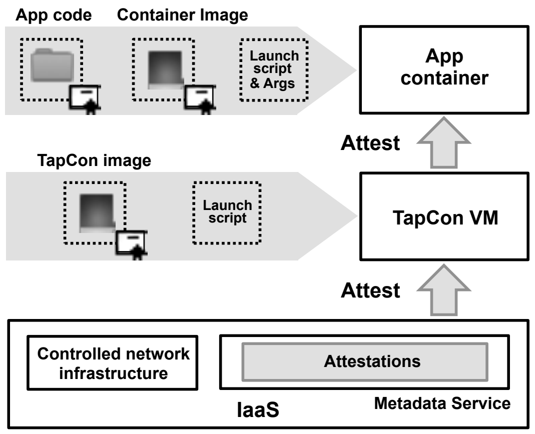

Figure 1: **Layered attestation example: a trusted IaaS service attests a sealed TapCon VM, which launches and attests application containers. The shaded areas represent attested programs and their attestations stored in an IaaS metadata service. TapCon attests to the container's image, application code, and the launch script and parameters.**

Docker containers—to deploy P and grant it access to A and B. How can they ensure that their datasets are accessible by a VM instance only if it runs the correct P?

Attestation-based access control. We propose that each group installs an access policy that permits data access from an instance running P—an example of *attestation-based access control*. The cloud storage service that stores A and B examines attestations of the layered cloud stack to verify that each data requester complies with a policy that the data owner provides or approves: e.g., it grants access if the request originates from a suitably trusted program P running in a secured environment (§2.3, §4).

Sealed software appliances. For cloud attestation to be secure, a trusted program P must be *sealed* so that the instance owner is blocked from subverting it through the management API, e.g., by replacing P after the instance launches, or taking a snapshot that exposes its data (e.g., datasets A and B). Instead, each trusted program P incorporates its own management API that is part of P's definition and respects its advertised security properties. We refer to programs with this property as sealed software appliances.

Layered Attestation. We focus on validating the end-to-end trust chain for application instances running at the top of a layered cloud stack. Each layer of the stack launches a virtualized environment or interpreter for programs it launches above it, and publishes attestations for those programs. Figure 1 depicts layered attestation of an application container by a server running TapCon, which runs as a VM that is itself sealed and attested (§2.2, §3).

2.1 IaaS Root of Trust

We start from the premise that the IaaS cloud provider is trusted to deploy and attest VM instances correctly and to isolate them from one another. Our previous work [20] presents a system called CQSTR ("sequester") that extends OpenStack to support basic attestation of VM instances. CQSTR issues trust statements attesting to an image identity for each VM instance, and includes a trusted *metadata service* to store these statements. Our approach in this paper builds on that foundation.

Trust in the IaaS provider could be social, or based on auditing and endorsement by a trusted third party, or on hardware-based attestation of the IaaS hypervisor and cloud control system. In this paper we focus on how to build practical cloud attestation upward to higher layers, taking this foundation as a given.

2.2 TapCon Service

To illustrate how to build an environment for layered cloud attestation, we developed a simple tenant-managed platform-as-a-service (PaaS) layer. TapCon is a virtual appliance image that implements a sealed Docker container service running as a tenant VM over the IaaS layer. TapCon supports a novel combination of features for practical cloud attestation.

Source-based attestation and certified builds. Attesting to a binary hash is sufficient for first-party attestation: it proves to the instance owner that the instance was launched securely from an image built by the owner. But for third-party attestation we also need assurance of the provenance of the binary. TapCon attests that a container is launched from a trusted code repository—*source-based attestation*. It generates the corresponding binary through a known and trusted build chain that is part of its trusted computing base (certified builds).

Authentication. Any code attestation system must provide some way to authenticate communications with the attested instance, so that insecure software cannot masquerade as a secure instance. Hardware-level attestation—and many software attestation systems—bind a software identity to a keypair whose private key is held by the attested service. This approach introduces various overheads and practical challenges to manage keypairs safely, and may require substantial changes to software to sign and/or encrypt communications.

As an alternative, our system leverages the managed network of the IaaS provider to block any spoofed packets. On a secure network, any communications from an assigned network address are known to have originated from the instance that owns the source address (§3). This property enables us to use network addresses as secure identifiers for instance endpoints [3, 16].

2.3 Logical trust

Trust in an attested service is based on a chain of statements extending across layers in the system. The chains are rooted in one or more trust anchors: for example, a relying party may trust the IaaS provider and authorities who endorse source code or binary images. In the joint data mining scenario, the data server validates the attestations for the instance running *P*, and checks each data request for compliance with access control policies of the owners for datasets *A* and *B*. The chain is rooted in the IaaS layer and extends through the TapCon PaaS layer (Figure 1).

Our approach to attestation-based access control uses standard Datalog logic to represent attestation statements, access policies, and other statements of trust (§4). For example, logical trust is also useful to represent endorsements of trusted programs. An access policy may specify trusted programs directly—e.g., by hashes over source or binary artifacts—but a more general approach is to grant access to programs with *security properties* of interest. Then, for example, if a new version of a program *P* is endorsed as having the relevant properties, there is no need to change the policies to reference the new version. For example, an endorser of the program *P* in the joint data mining scenario might assert that *P* does not leak its *A* and *B* inputs. The owners of *A* and *B* must trust the endorsers a priori (e.g., by social trust) or based on statements about the endorser by another trusted party (e.g., the owner's employer, or a consortium).

Logic gives us a powerful language to represent the statements of the endorsers and attesters, and also any delegations that provide the basis for trusting them. The logical trust system can expose all such trust assumptions and reason from them rigorously to infer security properties for attested services.

2.4 Related Work

Our approach is complementary to SCONE [4], which runs container code in SGX enclaves (following Haven [5]). We could use SCONE to ground attestation chains in a hardware root of trust. The TapCon alternative avoids the performance costs and limitations of enclaves. We also show how to use logical trust to check third-party attestations across multiple layers.

There is a close parallel between our work and systems that implement mandatory access control by decentralized information flow control (DIFC), which enforce end-to-end security for data flows based on security properties (tags) of trusted programs [18]. Our work can be used to attest a DIFC-enforcing layer in a secure cloud stack, among other uses.

Our use of logical trust is similar to *logical attestation* in Nexus [19]. We extend logical attestation to a cloud setting. We also use standard Datalog as a trust language; in contrast, Nexus introduces a new authorization language that is intractable.

3 The TapCon Layer

TapCon implements a simple tenant-managed PaaS service with features for layered attestation. Any IaaS tenant can instantiate a TapCon cluster using a *TapCon base image*. Our modified IaaS service launches a TapCon VM instance, and attests it as running the TapCon image. It stores the attestation in its CQSTR metadata service.

The TapCon image contains a standard Linux stack with a Docker daemon to launch containers. We modified the daemon to issue attestation statements for its containers, identifying the launched image (source code and data), along with parameters and launch scripts.

Authentication by network address. An attested instance is a security principal whose identity in TapCon is authenticated by a network address. In doing this, we presume that the cloud provider configures ingress filtering and host-side virtual networks to block any packets with a spoofed source IP address from entering the network. Public clouds (e.g., Amazon/AWS [1]) support this property.

In our TapCon prototype, the OpenStack IaaS platform initially assigns an unspoofable IP address to each TapCon VM; all containers launched within the VM use its IP address. To authenticate containers by network address, TapCon partitions the VM's network ports among the local container instances. At container launch, TapCon adds a SNAT rule to the VM-local firewall, confining the container to use a delegated range of ports. It then issues an attestation statement that binds the code identity to the container's network address and assigned port range. The metadata service records the TapCon instance—also authenticated by its network address—as the "speaker" of the attestation.

Layering. This architecture extends naturally to higher layers. For example, if software in a container is itself capable of launching secure instances (e.g., processes running within a trusted interpreter/sandbox) and subdelegating port ranges to them, it may publish an attestation for an instance to the metadata service. These statements are authenticated as spoken by the container because it sends them from a port that it controls. Another party—e.g., a client or a storage service—may accept the attestation if it trusts the container to issue such statements. For example, it may trust that the container is running a trusted interpreter program, based on a TapCon attestation about the container itself. In this way the sequence of statements about the layers of the software stack form a logical *attestation chain* in which trust in each layer derives from statements by the layer below. These chains are validated by recursive logic rules that can accept chains of arbitrary depth (§4).

Certified builds. TapCon binds each attested container instance back to specific source code: its trusted build service publishes attestations about the source code and other elements of the build. We modify Docker's build system to provide tighter control over the sources of build information. First, the build service accepts only a Git URL to specify the source to build, which embeds a self-certifying Merkle hash over a source code version. Second, all files downloaded during the build are hashed and attested along with the Git URL. Thus another party can verify that code was built with trusted tools, libraries, and packages, and anyone may audit the build environment by reproducing the certified build locally and comparing the results. To provide a practical starting point, we enable the use of a few official base container images, e.g., Ubuntu and Debian. Other external base images are blocked. We plan to extend this scheme with an attestable Docker registry to avoid recompilation of images on each TapCon host, and to allow launching externally endorsed images.

4 Logical Trust for TapCon

Logical trust offers a natural formalism to represent the statement types summarized in §2—layered attestations, program endorsements, network address delegations—and also validation rules, access policies, and local trust anchors. Our approach follows our work on SAFE [7]: it uses pure Datalog as the trust language, uses scripted linking to connect related statements in DAGs that match the delegation structure, and publishes linked trust statements in an indexed put/get store—the IaaS cloud metadata service. To use Datalog as a trust logic, we extend its syntax to add a "says" operator ("**:**") that attributes each statement to a speaker—the authenticated principal that asserts it, e.g., an instance bound to a network address.

Compliance checking. Any running program may use an off-the-shelf logic engine (Styla) to query and validate an attestation chain according to its locally accepted trust anchors and policy rules. We refer to the program and its logic engine as a relying party or *authorizer*. The authorizer's logic engine and policy rules consider only statements whose authenticated speakers are locally trusted to make those statements. The logic engine infers this trust by recursive application of the policy rules.

Trust anchors. All valid inferences are grounded in locally accepted trust anchors, which are the bases for the recursion. For example, an authorizer must trust the IaaS provider (authenticated by keypair) and any TapCon instance as a source of attestation statements; Listing 1 uses an `attester` property (rules **R1** and **R4**) to represent this trust. **R4** presumes that a trusted *endorser* endorses the TapCon binary as a secure container service

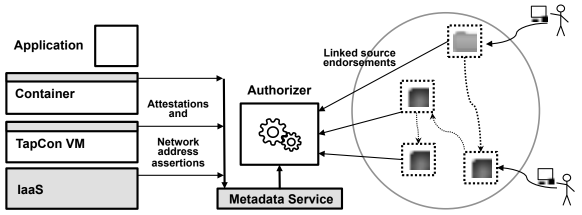

Figure 2: **An application principal that performs attestation-based access control checks is an *authorizer*. It runs an off-the-shelf logic engine to evaluate logical guard conditions according to its local policy rules. The logical rules evaluate code attestations published at each layer, and endorsements of the attested code.**

that issues trustworthy attestations. Given that the TapCon source code and build procedure are open, anyone can verify that it implements a secure container manager and build chain. For simplicity we presume that the endorser is trusted a priori to say so, i.e., it is a trust anchor.

Layered attestation chains. An attestation chain is a linked DAG of authenticated logic statements that establish the code identity, execution integrity, and network address(es) of the program instance at the top of a cloud stack (Figure 2). Layered cloud services, including IaaS providers and the TapCon platform (PaaS) service, issue statements about their child instances at launch time. A TapCon VM that attests a launched container—binding it to an image and a network address range—also issues statements to bind the image with a source code repository. Other parties (*endorsers*) make statements about the security properties of programs (images and/or source repositories). The issuers of all of these statements link them in overlapping DAGs, following [7].

An authorizer may consider all of the statements in an attestation chain together to make access control decisions. It checks compliance with its local policy by evaluating a specified guard condition for access control against a linked set of logic assertions and policy rules governing the authority of speakers to make those assertions. If a request is valid, the engine generates a logical proof of compliance.

Listing 1: Policy rules to validate a layered attestation chain.

```
(R1) attester(H) :- trustedCloudProvider(H).

(R2) runs(Instance, Image) :-
       runsInstance(H, Instance, Image),
       attester(H).

(R3) runsInstance(H, Instance, Image) :-
       AuthNID: attest(Instance, Image),
       bindToID(H, AuthNID).

(R4) attester(Instance) :-
       runs(Instance, Image),
       E: endorseAttester(Image),
       endorser(E).
```

R1. Accept attestations from IaaS cloud providers I trust: grant them the `attester` property.
R2. Believe that a named `Instance` runs a named `Image` if some host *H* attests that it does, and I trust *H* to issue attestations (*H* has the `attester` property).
R3. Believe that a host *H* issued an attestation statement if it was spoken by an authenticated identity `AuthNID` (a public key or secure network address) bound to host *H*.
R4. Trust an instance as an `attester` if it runs software that is endorsed by some trusted `endorser` as implementing a secure platform whose attestations are trustworthy.

Table 1: **Meaning of the recursive logic rules in Listing 1.**

Validation rules. Listing 1 shows simplified rules R1-R4 to validate an attestation chain. Each rule has a *head* on the left, which represents a belief that is implied by (“ :- ”) a list of subgoals in a *body* on the right: the head is true if all subgoals in the body are true, under some assignment of string values to variables (capitalized terms).

R2 allows an authorizer to infer that an instance runs a specific program, if some valid host attests it and is trusted to issue such attestations. The value of *P* is a hash over the code for *P* and its configuration, as shown in Figure 1. **R3** authenticates an attestation statement to a host, e.g., if it was spoken from a network address that is bound securely to that host by other rules for *bindToID* (not shown). It could use other authentication methods (e.g., keypairs) to establish the binding; this is a form of reconfigurable authentication [17]. The recursion in **R4** enables these rules to check layered attestations of any depth. **R1** is a basis for the recursion: it accepts attestations from a trusted IaaS provider.

Validating an attester. **R2** requires that the attesting host *H* at each step is accepted as a valid *attester*: this property captures the belief that *H* is faithful in launching guest instances, binding them to secure network addresses, and attesting their programs. This condition is satisfied if, for example, *H* is a trusted IaaS cloud provider (**R1**), or if *H* is an instance that is itself attested by its own host as running a secure program, and some endorser *E* endorses that program for the `attester` security property (**R4**). Trust in the endorser *E*—and in the IaaS provider—is also derived from authenticated (e.g., signed) logic statements and/or local policies (not shown). An authorizer could, for example, accept endorsers that are approved by its enterprise or by an open-source consortium, or it could choose to trust only itself. Endorsers may endorse other security properties as well; we can adapt the rules to limit the properties that each endorser is trusted to assert.

Authorizing data access. In our joint data mining sce-

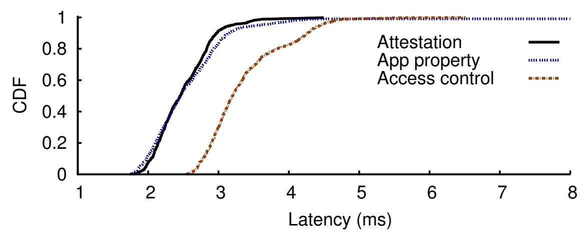

Figure 3: **Latency of queries for three types of compliance checks, measured under concurrency level *C*=10. Latency includes Styla prover cost, related scripting costs, and network latency.**

nario, a policy decision to grant access to dataset *A* or *B* can be based on the identity of the code running in the requesting instance, and the data owner's beliefs about the security properties of that code. As shown in Figure 2, the data owner attaches an access policy to an object. The data owner trusts the authorizer—in this case, a data storage service—to apply its policy faithfully.

Listing 2 shows an exemplary rule that verifies (i) that the *Owner* says that *Endorser* is trusted to endorse *Property* about the program, (ii) that *Endorser* says that *Program* has *Property*, and (iii) *Owner* says the object can be accessed by a program with *Property*.

Listing 2: Policy rule that grants access based on accepted properties of an attested program that the requester is running.

```
hasAccessPrivilege(Program, ObjID, Owner) :-
 Owner: trustEndorserOn(Endorser, Property),
 Endorser: hasProperty(Program, Property),
 Owner: accessPrivilegeByProgramProperty(
                              Property, ObjID).
```

4.1 Prototype Evaluation

We measure the overhead of our TapCon prototype to attest launched containers and for attestation-based access control checks. We use a 3-node cluster on CloudLab. Each node has 16 Intel E5-2630 cores, 128GB memory, and two 10GbE interfaces. The cluster runs Openstack and Docker with extensions for TapCon. We use the Styla logic engine to perform compliance checks.

Boot overhead. The table below compares the latency to launch a container in Docker and in TapCon. The third column shows the time for TapCon to attest a container to the metadata service.

Docker	TapCon	Posting Statements
488ms (±40ms)	497ms (±49ms)	31ms (±8ms)

Overall, TapCon is only 2% slower than Docker, because posting statements can be overlapped with container launch. The overhead to attest VMs is negligible, as each VM takes tens of seconds to launch.

Performance of access control. We measure the latency for three types of attestation-based compliance checks: validation of layered attestation chains, validation of endorsed application properties, and attestation-based access control (Listing 2). We issue to the metadata service statements for 10,000 containers, 100 endorsed properties for each application, and 100 ACL entries for each data object. The test harness performs 100,000 random checks of each type on a hot cache. Figure 3 shows that most of the queries complete within 5 ms. Access control queries rely on ACL entries, the cost of which is linear to the ACL length.

5 Conclusion and Discussion

The paper proposes to incorporate attestation above the standard IaaS cloud abstraction, and shows how to apply it to layered platform services. The paper describes a small set of cloud provider extensions directed at enabling a rich foundation for trustworthy cloud computing. In particular, TapCon serves as an example of how layered attestation makes it possible to deploy *tenant-managed* security services, promoting extensibility of secure cloud platforms.

TapCon is a work in progress, and can benefit from discussion and feedback about several aspects of the project. Does attestation have an important role to play in a more secure cloud future? Code attestation has been around for awhile, but third-party attestation has never quite caught on. What are the key obstacles to its broader adoption? Do the elements of attestation in TapCon—source-based attestation, source code provenance via a certified build chain, endorsements of secure code properties—help to address them? Is it time to decouple attestation from its hardware roots (TPMs and SGX)? Is an IaaS provider suitable as a root of trust for its tenants, or is the risk of compromise or subversion too great? Attestation is a constructive security approach in which subversion of any link in the trust chain is sufficient to invalidate all trust assurances that it provides.

Finally, we recognize that the topic of logical trust—a key element of our approach—is unfamiliar to many in the systems community, despite well-understood rigorous foundations and many case studies showing potential for practical value. We hope that the paper can spark wider exposure and discussion of the power of logical trust and its potential uses for data-centric trust in a cloud setting.

Acknowledgements

We thank the anonymous reviewers and our shepherd, Sharon Goldberg. This material is based upon work supported by the National Science Foundation under grant numbers CNS-1330308, CNS-1330659, and CNS-1419199.

References

[1] Amazon Web Services. Amazon Web Services: Overview of Security Processes, 2014. https://aws.amazon.com/whitepapers/overview-of-security-processes/.

[2] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative Technology for CPU-based Attestation and Sealing. In *the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, volume 13, 2013.

[3] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *the ACM SIGCOMM Conference on Data Communication*, pages 339–350, 2008.

[4] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. OâĂŹKeeffe, M. L. Stillwell, et al. SCONE: Secure Linux Containers with Intel SGX. In *the 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[5] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *the 11th USENIX Symposium on Operating Systems Design and Implementation*, pages 267–283, 2014.

[6] K. Bhargavan, M. Kohlweiss, A. Pironti, P.-Y. Strub, S. Zanella-Beguelin, and C. Fournet. Proving the TLS Handshake Secure (As It Is). In *Advances in Cryptology – CRYPTO 2014*, pages 235–255, July 2014.

[7] Q. Cao, V. Thummala, J. S. Chase, Y. Yao, and B. Xie. Certificate Linking and Caching for Logical Trust. http://arxiv.org/abs/1701.06562, 2016. Duke University Technical Report.

[8] V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[9] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-based Platform for Trusted Computing. In *the 19th ACM Symposium on Operating Systems Principles*, 2003.

[10] T. C. Group. Trusted Platform Module. https://trustedcomputinggroup.org/trusted-platform-module-2-0-brief-introduction/.

[11] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. Setty, and B. Zill. IronFleet: Proving Practical Distributed Systems Correct. In *the 25th ACM Symposium on Operating Systems Principles*, pages 1–17. ACM, 2015.

[12] C. Hawblitzel, J. Howell, J. R. Lorch, A. Narayan, B. Parno, D. Zhang, and B. Zill. Ironclad Apps: End-to-End Security via Automated Full-System Verification. In *the 11th USENIX Symposium on Operating Systems Design and Implementation*, pages 165–181, 2014.

[13] G. Hunt and J. Larus. Singularity: Rethinking the Software Stack. *ACM SIGOPS Operating Systems Review*, pages 37–49, April 2007.

[14] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *the 12th USENIX Symposium Operating Systems Design and Implementation*, 2016.

[15] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal Verification of an OS Kernel. In *the 22nd ACM Symposium on Operating Systems Principles*, 2009.

[16] A. Li, X. Liu, and X. Yang. Bootstrapping Accountability in the Internet We Have. In *the 8th USENIX Symposium on Networked Systems Design and Implementation*, page 155, 2011.

[17] W. R. Marczak, D. Zook, W. Zhou, M. Aref, and B. T. Loo. Declarative Reconfigurable Trust Management. *Computing Research Repository*, Sept. 2009.

[18] A. C. Myers and B. Liskov. A Decentralized Model for Information Flow Control. In *the Sixteenth ACM Symposium on Operating Systems Principles*, 1997.

[19] E. G. Sirer, W. de Bruijn, P. Reynolds, A. Shieh, K. Walsh, D. Williams, and F. B. Schneider. Logical Attestation: an Authorization Architecture for Trustworthy Computing. In *the 23rd ACM Symposium on Operating Systems Principles*, pages 249–264, 2011.

[20] Y. Zhai, L. Yin, J. Chase, T. Ristenpart, and M. Swift. CQSTR: Securing Cross-Tenant Applications with Cloud Containers. In *the 7th ACM Symposium on Cloud Computing*, pages 223–236. ACM, 2016.