Principles of Remote Attestation *

George Coker[1], Joshua Guttman[2], Peter Loscocco[1], Amy Herzog[2], Jonathan Millen[2], Brian O'Hanlon[2], John Ramsdell[2], Ariel Segall[2], Justin Sheehy[2], and Brian Sniffen[2]

[1]National Security Agency
[2]The MITRE Corporation

Abstract. Remote attestation is the activity of making a claim about properties of a target by supplying evidence to an appraiser over a network. We identify five central principles to guide development of attestation systems. We argue that (i) attestation must be able to deliver temporally fresh evidence; (ii) comprehensive information about the target should be accessible; (iii) the target, or its owner, should be able to constrain disclosure of information about the target; (iv) attestation claims should have explicit semantics to allow decisions to be derived from several claims; and (v) the underlying attestation mechanism must be trustworthy. We illustrate how to acquire evidence from a running system, and how to transport it via protocols to remote appraisers.
We propose an architecture for attestation guided by these principles. Virtualized platforms, which are increasingly well supported on stock hardware, provide a natural basis for our attestation architecture.

1 Introduction

Much economic activity takes place on heterogeneous networks of computers, involving interactions among autonomous principals, including individuals, retail companies, credit card firms, banks, and stock brokerages. Because the amount of money in these activities is large and increasing, the networks are attractive targets for criminals.

In many attacks, the adversary inserts software remotely, without physical access to the devices, and this software compromises secrets. For instance, in March 2008, an attack was announced against the large American grocery store chain Hannaford Brothers. Unauthorized code had been inserted on the servers in each of the company's 300 stores. This code retained the credit card information for each transaction occurring at a store and periodically transmitted the information to a third party. As a consequence, over 4,200,000 credit and debit cards were compromised. At least 2,000 fraudulent transactions have been identified as results. Even though Hannaford's systems were designed not to store customer payment details and to adhere to compliance standards of the credit card companies, changes to their application software led to large disclosures [20].

* MITRE's work on this paper was supported by the National Security Agency through US Army CE-COM contract W15P7T-05-C-F600.

An even larger case led to indictments in August 2008. Over 40 million card numbers were stolen from US companies such as TJX, a clothing distributor and retailer, and other large firms. According to the indictment papers, eleven criminals collaborated in this group of attacks. Members were located in the US, Estonia, Ukraine, Belarus, and China. In these attacks, wireless access points were the initial entry point. Newspapers described the inserted software as sniffers. However, the indictments mention that an insecure wireless access point at a Marshall's retail store in Florida allowed the defendants to compromise data stored in servers at TJX, located in Massachusetts [36].

There are several characteristics of these attacks.

1. The attacks are executed remotely, apparently without physical access to the computers attacked;
2. The computers are often standard, general purpose systems, rather than specialized devices such as automated teller machines;
3. The networks involve transactions among independent entities, such as a retailer, a distributor, a customer, and the credit card firms. No one organization controls the software configurations on all the relevant systems.

The ubiquitous attacks that insert malware onto individually owned computers, to sniff for bank account and password information, share these characteristics. The bank cannot control the configurations of its customers' computers. Nevertheless, there would be benefits shared by the bank and its customers if the bank could ascertain that the customer's computer was free of malware before allowing the customer to enter the account number and password. The client software on a customer's computer could also check software on the bank or retailer computer to which it was connected.

To limit the success of attacks with these three characteristics, one needs a way to provide reliable evidence about the state of software executing on a system, so as to determine whether (1) has occurred, while preserving great flexibility, so that the strategy can succeed despite the potential lack of specialized hardware capability (2). Moreover, the diversity of trust relations reported in point (3) offers improved resilience, since remote appraisal of the evidence can be incorporated into the distributed architecture. However, it also means that the evidence delivered must be tailored—not every participant will allow other participants to know all about the strengths and weaknesses of their configuration.

A principal goal of trusted computing is to provide reliable evidence about the state of software executing on a system. This evidence is intended to ensure that targets will not engage in some class of misbehaviors. To this end, the Trusted Computing Group (TCG) introduced the Trusted Platform Module (TPM) and the associated concept of *remote attestation* [3].

Remote attestation may be used to address a number of trust problems including guaranteed invocation of software, delivery of premium content to trusted clients, assuaging mutual suspicion between clients, and more. The range of applicability is clearly much broader than just the financial area featured above. As the requirements of all such applications cannot be known *a priori*,

attestation systems and measurement systems alike must be flexible, providing for privacy, completeness of measurement, and trust in the basic collection and reporting mechanisms.

Many existing attestation proposals, including those put forth by the TCG, are aimed at specific use-cases. They therefore lack the flexibility to address many attestation problems, including the cross-organization challenges for general purpose devices that we have just described. Indeed, many existing definitions of attestation primarily focus on describing the particular properties [29] desirable in those use-cases. For example, in [11], the author uses the term attestation to specifically mean the process of transmitting a sequence of hashes of certain system components and a digital signature of that sequence; in Microsoft's "NGSCB" [7] it refers to identification and authentication of known code via digital signatures; Copilot [18] makes use of direct hashes of kernel memory, and so on. We prefer a general definition of platform attestation that abstracts from specific desired properties, an approach that matches a promising line of work on "property-based attestation" [4].

In this paper, we describe a flexible attestation architecture, based on a few guiding principles. Systems built according to this architecture can be configured to carry out a variety of attestation scenarios. We believe that this attestation architecture provides the mechanisms needed for systems to interrogate each other before sensitive interactions, so as to ensure that those interactions will be safe.

In Section 2, we introduce some fundamental attestation notions, and codify terminology for them. Section 3 stipulates five principles to which, we will argue, attestation architectures should adhere. In Section 4, we enumerate some architectural ingredients needed to satisfy the principles, which suggest the type of platform presented in Section 5, the section describing the main mechanisms that can be used to satisfy our principles. Sections 6 and 7 discuss open problems and related work.

2 Terminology

In this section we define the process of attestation and provide context for the understanding of security properties achievable via that process.

Our approach to system attestation departs significantly from the notion put forth by the TCG, in great part due to increased flexibility. Emphasis is placed on attestation based upon *properties* of the target, useful in a variety of scenarios, rather than solely on attestation based upon *identity*.

Terminology. *An* appraiser *is a party, generally a computer on a network, making a decision about some other party or parties. A* target *is a party about which an appraiser needs to make such a decision.*

We refer to the decision-making party as an appraiser rather than a verifier to emphasize that the decision-making process is a matter of evaluating the target against the appraiser's internal standards, rather than proving the target's

correctness. The trust decision made by an appraiser often supports an access request made on behalf of the target, and is usually a decision about the expected behavior of that target. To make a decision on this basis, a diligent appraiser needs a significant amount of information about the target's hardware, software, and/or configuration—essentially, the knowledge that the state of the target is such that it will not transition into an unacceptable state while the appraiser still continues to trust it. There is some inevitable tension between the human organizations behind the appraiser and target, as the appraiser's owner wishes to have complete and correct information about any given target while the target's owner wishes to give up no more than the minimal information necessary for the success of its request (and perhaps even less).

Terminology. Attestation *is the activity of making a claim to an appraiser about the properties of a target by supplying evidence which supports that claim. An* attester *is a party performing this activity. An appraiser's decision-making process based on attested information is* appraisal.

Our definition of attestation encompasses the wide variety of definitions found in the related literature. In the most commonly addressed class of attestations, each attestation provides a means for appraisers to infer that the target of the attestation will not engage in a class of misbehaviors. For example, if the target reports its kernel is unmodified, the attester has reason to trust reports from the target, and some appraiser trusts information provided by the attester, then that appraiser can infer that the target will not engage in misbehaviors that might have occurred had the target's kernel been corrupted at the time of its measurement. Not all attestations are about lack of misbehaviors, even though most of the commonly discussed use cases are in that class. For example, an attestation could also be used to assist in the selection and use of settings or software for future interactions, or as a means of proving prior authorization for some purpose.

This broader point of view makes a rich understanding of the related concepts of system measurement, attestation protocols, and system separation vital to successful attestation. Here there is a distinction between the measurement of a target system (the evidence) and the attestation itself.

Terminology. *To* measure *a target means to collect evidence about it through direct and local observation of it.*

Attestation about a target system will report measurements, or will report conclusions inferred using measurements and possibly also other attestations. Measurements can be as simple as a SHA-1 hash of code on the target, but can also be more complex. In this paper, measurement is discussed only as necessary to support our architecture for attestation, but cf. [23, 37, 22, 27] for more on measurement strategies.

Terminology. *An* attestation protocol *is a cryptographic protocol involving a target, an attester, an appraiser, and possibly other principals serving as trust proxies. The purpose of an attestation protocol is to supply evidence that will be*

considered authoritative by the appraiser, while respecting privacy goals of the target (or its owner).

Evidence may be attested to in a number of equivalent but semantically different forms depending on the attestation protocol. For example, the attestation may report raw evidence as directly observed, as reduced evidence (e.g. a hash of the raw evidence), or by substitution with a credential provided by a third party evaluator of the raw evidence. For example, an SSL certificate authority may consume many attestations as to the identity and practices of a target before producing one certificate attesting to the quality of a target [4].

A given target may wish to provide different information to different appraisers depending on the current trust relationships it has with those parties. A worthwhile desire in developing an attestation system is to resolve the mutual tension between the appraiser's desire for disclosure and the target's desire for privacy as well as possible given the contradictory nature of the parties' interests. One approach to defusing this tension is for the appraiser to demand frequent repeated (but perhaps less detailed) attestations, re-evaluating its trust decisions often. It may be possible to determine that a party will be sufficiently trustworthy for the 15 minutes after performing a given attestation, but not feasible to determine that it will be so for a day.

The word *trust* is used many ways in English, and in many ways even within this paper. However, one special usage is important enough to define here:

Terminology. *Principal B* trusts *principal A* with regard to the statement ϕ *if and only if, from the fact that A has said ϕ, B infers that ϕ was true at a given time.*

In many cases, the result of an attestation protocol depends on a complicated mixture of facts that the appraiser can check directly, such as cryptographic signatures, and those he cannot. B must decide to trust A (or not) with respect to uncheckable facts.

Indeed, the importance of Trusted Platform Modules lies exactly here. Trusted Platform Modules (TPMs) are chips on a platform's motherboard. Information, such as measurements of the platform's software state, may be placed into the TPM's Platform Configuration Registers (*PCRs*). A *quote* is a digitally signed report of PCR contents. When B finds that a TPM quote is a valid digital signature over particular PCR values, B may decide to trust that platform with regard to certain statements. A TPM thus provides cryptographically signed evidence of facts about the state of the platform to which it is attached. The TPM residing on the target may be considered a principal in an attestation protocol.

This notion of trust differs from the Trusted Computing Group's usage. They emphasize the *predictable* behavior of a device or component [41]. In our terminology, this is the purpose of *measurement*. Measurement is useful when the direct evidence it collects supports behavioral predictions about the target of measurement. Trust in our sense is what motivates one principal to accept conclusions, based on the observations and assertions of another.

Terminology. *A* root of trust *is a hardware device with known behavior, which a certificate asserts to be present on a particular platform.*

A root of trust for measurement *is a hardware device (or some functionality provided by hardware) that can reliably prepare certain measurements on the software state of a device. A* root of trust for reporting *is a hardware device (or some functionality provided by hardware) that can reliably attest to the result of a measurement. A* root of trust for storage *is a hardware device (or some functionality provided by hardware) that ensures that certain data such as cryptographic keys will be stored in a way that will always preserve their secrecy.*

Originally, the Trusted Computing Group proposed that the Basic I/O System (BIOS) should perform the earliest measurement of the software obtaining control of the CPU at the beginning of the boot process. This had significant limitations, because the BIOS is complex and frequently imperfectly reliable. In recent architectures, a CPU instruction provides an independent root of trust for measurement.

The Trusted Platform Module provides a root of trust for reporting, since the hardware architecture ensures that only the roots of trust for measurement can write to certain platform configuration registers. Moreover, digital signatures supported by a certificate infrastructure allow a third party to treat messages prepared by the TPM as attestations about these values. The TPM also uses encryption to serve as a root of trust for storage for keys including the private *Attestation Identity Keys* used to sign attestations.

The hardware TPM on a platform may also provide a root of trust for believing that certain software entities on the system behave like TPMs. In this case we refer to those entities as *virtual TPMs* or *vTPMs*.

Terminology. *A platform provides* measured boot *if hashes of the code controlling successive stages of its boot process are deposited in PCRs in its TPM.*

A platform offers secure late launch *if its CPU has an instruction that atomically (1) constructs a hash of the contents of a stipulated region of memory, (2) deposits the hash into a PCR, and then (3) transfers control to the instruction at the start of the stipulated region.*

Secure late launch (also simply called *late launch*) is *late* in the sense that it occurs after the normal boot sequence completes, and possibly long after it. It is desirable because it allows the system to transfer control to known code after hardware specific activities—such as device drivers—have initialized. In particular, the code that gains control from late launch may be a hypervisor that places the code of device drivers and also their memory buffers into non-privileged memory. Both Intel and AMD have developed products offering late launch. When combined with memory management controls on device Direct Memory Access, secure late launch provides excellent protection against some of the least well controlled code running in traditional OS kernels.

3 Principles for Attestation Architectures

Five principles are crucial for attestation architectures. While an ideal attestation architecture would satisfy all five, in real systems, only an approximation of the ideals is possible. Thus, attestation mechanisms may provide better implementations of some features than others. The five principles motivate the architecture presented in Section 4.

Principle 1 (Fresh information) Assertions about the target should reflect the *running system*, rather than just disk images. Some measurement tools provide only start-up time information about the target in the expectation that its security-relevant properties still hold. Other tools can inspect the current state of an active target, as discussed in Section 5.1. □

The architecture cannot predict the uses to which appraisers will put the information it delivers. Appraisers may need to make very different decisions, and—to justify them—need to make different predictions about the future behavior of the target. This suggests the next principle.

Principle 2 (Comprehensive information) Attestation mechanisms should be capable of delivering comprehensive information about the target, and its full internal state should be accessible to local measurement tools. □

With comprehensive information come worries about the consequences of disclosure. Disclosure may cause loss of privacy for a person using the target platform. It can also subject the platform to attack, for instance if the attestation discloses an unpatched vulnerability to an adversary.

Principle 3 (Constrained disclosure) A target should be able to enforce policies governing which measurements are sent to each appraiser.

Hence, an attestation architecture must allow the appraiser to be identified to the target. Policies may distinguish kinds of information to be delivered to different appraisers. The policy may be dynamic, relying on current run-time information for individual disclosure decisions. For instance, a target may require that the appraiser provide an attestation of its own state, before the target discloses its own. □

Attestations make *assertions* about a piece of the real world, i.e. the state of the target at a particular time. The appraiser can make decisions because there are conventions about what different attestations mean, which is to say that the attestations have semantics. The appraiser's work depends essentially on this semantics, since the resulting appraisals are intended to be true summaries of the degree of reliability of the target's future behavior. This semantics must be compositional, so that the appraiser can combine the meanings of a number of attestations in order to provide more comprehensive summaries.

Principle 4 (Semantic explicitness) The semantic content of attestations should be explicitly presented in logical form.

The identity of the target should be determined by this semantics, so an appraiser can collect attestations about it. The appraiser should be able to infer consequences from several attestations, e.g. when different measurements of the target jointly imply a prediction about its behavior. Hence, attestations should have uniform semantics, and be composable using valid logical inferences. □

Principle 5 (Trustworthy mechanism) Appraisers should receive evidence of the trustworthiness of the attestation mechanisms on which they rely. In particular, the attestation architecture in use should be identified to both appraiser and target. □

There will be a good deal of natural variation in how different systems meet these principles, and in the choices they make when some principles are only partially satisfied. In specific situations, it may be sufficient to satisfy these principles only partly. For instance, limited forms of evidence about the target may suffice for an appraiser, or evidence that has aged to an extent may be accepted. When different degrees of adherence to the principles are designed into a system, then the variation is static. When the system adjusts at runtime to provide different degrees of evidence in different situations, or when different peers are the appraiser, then the variation is dynamic.

4 An Attestation Architecture

There are five main constraints, imposed by the principles of Section 3, that provide the content for the proposed architecture. In this section, each constraint is briefly described in the context of how it is motivated by the principles. A system designed according to this architecture must have the following abilities:

1. To *measure* diverse aspects of the target of attestation;
2. To *separate domains* to ensure that the measurement tools can prepare their results without interference from the (possibly unreliable) target of attestation;
3. To *protect itself*, or at least a core *trust base* that can set up the domain separation mechanism, ensuring that it cannot be weakened without this fact being evident from the content of attestations;
4. To *delegate attestation* so that attestation proxies can collect detailed measurements and convincing evidence, and summarize them to selected peers, when the target would not permit the full facts to be widely shared;
5. To *manage attestation* to handle attestation queries by invoking suitable measurement tools, delivering the results to the appraiser or a proxy as constrained by policies.

These constraints are discussed in turn. One might envision the constraints as suggesting an architecture of the form shown in Fig. 1. The relationships among components in the figure are discussed in the subsections below.

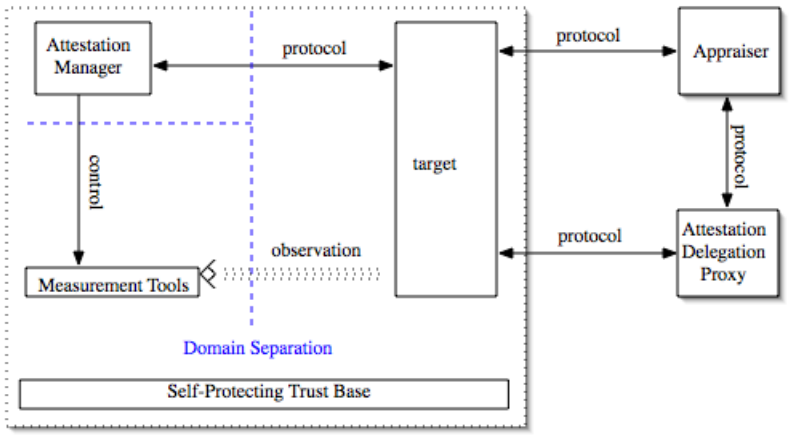


Fig. 1. Attestation Architecture

4.1 Measurement Tools

Providing comprehensive information about a system (satisfying Principle 2) requires the ability to provide a collection of tools that (jointly) comprehensively measure the target.

Comprehensive measurement of a system requires more than simply the ability to read all of the data contained in that system. It also means that some measurement tools must understand the structure of what they are measuring. For example, just being able to scan and hash the memory used by an operating system kernel may not suffice to provide useful measurements of it. *Usefulness*, here, is in the eye of the appraiser, and typically involves evidence about the past or future *behavior* of the target. A hash of the memory of a program is not sufficient to provide a prediction about its future behavior, since its states during execution may yield an unbounded set of hashes. Rather, predictions may be based on the hashes of some portions of memory (which take only a limited number of values) together with other information about the remaining parts of memory. Measuring complex system components requires knowing the structure of the targets. Some trust decisions require these structure-sensitive measurements; ideas about how to provide such measurements are included in Section 5.1.

As a result of this, there cannot be a "one size fits all" measurement capability for attestation. Different measurement tools must be produced for measuring components with different structure. Further, no complete set of tools can be known ahead of time without restricting the target systems from ever adding new applications. Thus, our architecture must support a collection of specialized measurement tools, and in order to be able to provide evidence for arbitrary

future attestations it must also support adding new tools to that collection over time.

In addition to measurement capacity being comprehensive, freshness is also a goal. (Principle 1) This means that our measurements cannot always be performed a priori – one must be able to measure various parts of a system on demand. These demands are made from the point of view of an appraiser. A remote party must be able to trigger measurement; it is insufficient to only have runtime measurement occur via periodic automatic remeasurement triggered by the measurement system or tools.

4.2 Domain Separation

For a measurement tool to provide information about a target of attestation, the measurement tool must be able to deliver accurate results even when the target is corrupted. This is an important consequence of Principle 5.

There are two parts to this. First, it must have access to the target's state so as to be able to distinguish whether that target is corrupted or uncorrupted. This state includes both the target's executable code and also modifiable data structures that determine its future behavior. Second, the measurement tool's state must be inaccessible to the target, so that even if the target is corrupted, it cannot interfere with the results of the measurement.

There are different ways that this separation can be achieved. One is to virtualize the target, so that the measurement tool runs in a separate virtual machine (VM) from the target [17]. The virtual machine monitor must then be able to control cross-VM visibility so that the measurement tool has read access to the target. It must also ensure that the target does not have any control over the measurement tool. There may be a message-passing channel established between them, but the hypervisor/VMM must be able to ensure transparent visibility of the measurement tool into the target and protection of those tools from the target.

Alternatives are possible. For instance, CoPilot (Section 7) uses a form of hardware separation in which the measurement tool runs on a coprocessor and the visibility constraints are expressed via hardware instead of being based on the configuration of a hypervisor.

Given the improved virtualization facilities that new processors from Intel and AMD provide, the VM approach is very attractive: it makes minimal requirements beyond standard commodity hardware.

4.3 Self-Protecting Trust Base

We have established that domain separation is necessary in order to have trust in attestations and specifically in the integrity of our measurement tools. This raises a question: how to produce assurance for the integrity of the domain separation itself?

The core of our system's trust must come from components which are simple enough or sufficiently evaluated that one can be convinced that they do not

require remeasurement after they have been running. Part of this core must obviously include the hardware used as our Trusted Computing Base. Any other component must either be measurable from a place that it cannot control or must be sufficiently measured via a static startup measurement taken before it begins operating.

Note that what is needed here is more than just a trusted static subset of our system. The difficulty here is that our trust base must be sufficiently simple and static to not require remeasurement, but also sufficiently rich to bootstrap our measurements and attestations. Anything performing measurement and attestation on the platform will in turn require measurement and attestation about itself in order to convince an appraiser of its trustworthiness. It must be ensured that this chain "bottoms out" at something sufficient to perform certain essential measurements and attestations to support the chain above it while being simple enough that static startup-time measurements are sufficient to determine trust.

It is not trivial to determine the content of this static trust base. One of the difficulties arises around the element of domain separation. It is preferable for the domain separation mechanism to be simple and secure enough to belong in this element, but no hypervisor exists today that satisfies those properties and is also rich enough to provide the services desired. We return to this difficulty in Section 6. One possible alternative is that a hardware component provides runtime measurements of the domain separation mechanism.

4.4 Attestation Delegation

In practice, the appraiser will need to delegate many aspects of determining the quality of the target to specialists called *attestation proxies*. There are two essential reasons for this.

First, Principle 2 contains an intrinsic conflict with Principle 3. The former states that comprehensive insight into the state of the target must be available. The latter says that the target should be able to choose and enforce a policy on the disclosure of information about its state.

A natural way to reconcile these two principles is to allow appraiser and target to agree on an attestation proxy that is partially trusted by each [4]. The target trusts the proxy to disclose only information about its state which is of limited sensitivity. The appraiser trusts the proxy to make statements only when they are warranted by appropriately fresh and comprehensive information about the target.

The second reason why attestation proxies are important is that they can function as *specialists*. Enormous expertise is needed to interpret detailed measurements, such as those needed to predict behavioral properties about an operating system. An appraiser may get more reliable information and more usable information from an attestation proxy than it would be able to extract on its own from the comprehensive data. The maintainers of an attestation proxy can ensure that it has up-to-date information about the strengths and weaknesses of specific system versions or configurations.

Naturally, these delegations require protocols that allow the principals to ensure they are communicating with appropriate proxies. These protocols must supply the principals with messages that unambiguously answer the principals' questions. The design of such *attestation protocols* may follow the methods of the strand space theory [14], and may use the strand space/trust management connection from [16, 15]. An attestation protocol that we call CAVES developed these ideas, and is discussed in Section 5.4.

These delegations, combined with attestations satisfying Principle 4, enable a powerful new capability: an appraiser may compose separate layered or orthogonal attestations, involving different proxies, in order to make a judgment. Another approach, "Layering Negotiations [19]," has been proposed for flexible and composable attestation. We have used many of the same tools as this work, such as Xen and SELinux. The layered negotiations have a fixed two-level structure and are intended to enable distributed coalitions. Our approach is intended to enable general, arbitrarily flexible composability regardless of application usage model.

4.5 Attestation Management

A central goal of our architecture is flexibility. It is essential that our system be able to respond meaningfully to different requests from different appraisers without having pre-arranged what every possible combination of attestations might be.

One way to support this notion is with an architectural element referred to as the *Attestation Manager*. A component realizing this idea contains a registry of all of the measurement and attestation tools currently on the platform, and a description of the semantic content produced by each. As a consequence of Principle 4, this component can select at runtime the appropriate services needed to answer any query which could be answered by some subset of the measurement and attestation capabilities currently on the system. The attester in the CAVES protocol (see Section 5.4) is implemented by the attestation manager together with the components that provide services to it, including measurement tools and the TPM itself.

As an Attestation Manager will clearly be involved in nearly every remote attestation, it is also a natural place to enforce some of the constrained disclosure called for by Principle 3. It can restrict the services it selects based on the identity of the party the information would be released to, according to locally-stored access policies. In order to defend this capability from both the untrusted target of attestations and also from potentially-vulnerable measurement tools, the Attestation Manager must be protected via domain separation.

Our attestations will have to use cryptography in order to protect communications from adversaries. This same protection, taken together with domain separation, means that the target can be safely used as an intermediary for communication with appraisers or proxies. This leads to the very beneficial result that an Attestation Manager can be a purely local service; it does not need to be directly accessible by any remote parties.

4.6 The Principles and this Attestation Architecture

We regard these architectural elements as a consequence of the principles, in the sense that—we believe—any system that meets the principles will contain elements much like these components. They do not ensure that the principles will be met. For instance, none of these architectural elements ensures that the semantics of attestations are well-defined and composable (Principle 4). For instance, an attestation manager is effectively required if comprehensive information about the target (Principle 2) is to be prepared freshly (Principle 1) in response to a query. Attestation delegation is the key to reconciling Principle 2 with Principle 3. Domain separation and a self-protecting trust base are required to achieve a trustworthy mechanism (Principle 5) that can provide comprehensive information (Principle 2) about the target.

5 A Composable Attestation Platform

In this section, we illustrate how a platform could be instantiated to achieve our attestation principles and mechanisms. Through our descriptions, we highlight how the techniques described here could be used to prevent future attacks similar to those carried out against Hannaford Brothers (as described in the introduction). Detection and prevention of such attacks requires work on the part of the enterprise: One way to integrate attestations is to add additional capabilities to existing configuration management activities. Another is to add them to existing transactions within the enterprise: Suppose that each Hannaford store transmits all transactions to a central server at the end of the day. If the server required an attestation from the machines at individual stores, and the store clients required the same from the server, rogue software could be detected before any credit card numbers were transmitted.

Constructing such a system is challenging on several levels. In describing a candidate system in this section, we focus on the five constraints listed in Section 4. Our candidate platform must be capable of:

1. *Measuring* diverse aspects of the target of attestation;
2. *Separating domains* to ensure that the measurement tools can prepare their results without interference from the (possibly unreliable) target of attestation;
3. *Protecting itself*, or at least a core *trust base* that can set up the domain separation mechanism, ensuring that it cannot be weakened without this fact being evident from the content of attestations;
4. *Delegating attestation* so that attestation proxies can collect detailed measurements and convincing evidence, and summarize them to selected peers, when the target would not permit the full facts to be widely shared;
5. *Managing attestation* to handle attestation queries by invoking suitable measurement tools, delivering the results to the appraiser or a proxy as constrained by policies.

We explore a possible implementation supporting these five constraints through the grocery store server example. In the following sections, imagine an architecture like that pictured in Figure 2.

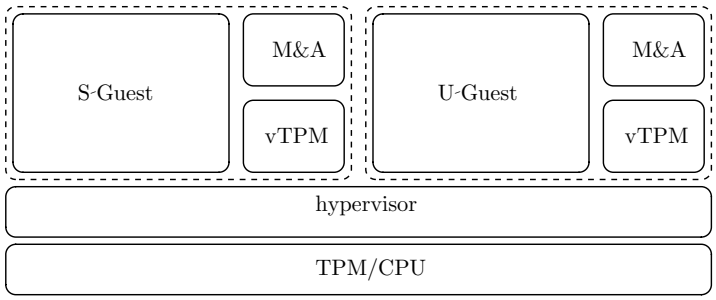


Fig. 2. A Generic Composable Attestation Platform

The self-protecting trust base here is composed of the hypervisor[1], CPU, and TPM. This representation is abstract–the implementation is not tied to a specific virtual machine monitor or microprocessor. The hypervisor does play a crucial role by providing domain separation and sharing services.

The supervisory virtual platform (S guest) contains support functionality for the system: For example, TPM virtualization services, specialized measurement tools, and so forth. The "User" platform (U guest) runs the server's "normal" operating system and data collection processing software.

TPM virtualization allows parties interacting with the User platform to seamlessly make use of TPM facilities without requiring knowledge of the full system architecture. The measurement and attestation functionality included in the platforms provide specialized measurement functions and attestation capabilities for varied target environments. They form the externally-visible face of the increased capabilities for the server.

We first turn our attention to the measurement capabilities of this system.

5.1 System Measurement

As mentioned in Section 4.1, providing comprehensive information about a system (satisfying Principle 2) requires the ability to provide a collection of tools that (jointly) comprehensively measure the target.

In our example, a conscientious customer will desire an attestation containing multiple kinds of evidence to gain assurance that the theoretical grocery chain central server will not store or re-transmit credit card data to a third party. A

[1] We hope that over time, hypervisors will be built to far stricter standards than has happened yet today. A hopeful sign from this point of view is the recent work of NICTA on a verified L4 variant [21].

measurement of the data processing software must be included in the attestation at a minimum. Since the underlying operating system is often the target of network-based attacks, a measurement of the OS kernel would also be appropriate. Finally, a certificate describing the hypervisor and S guest configuration may be important as well.

To continue with our example, assume that the client expects an enterprise-signed configuration certificate, an OS-measurement report, a hash of the data processing software, and a vTPM quote from the U guest vTPM containing hashes of the two measurement reports in the PCR mask. This need for evidence connected across different virtual machines can only be satisfied with semantically explicit attestations as described by Principle 4.

Note in Figure 2 that both the S and U environments possess specialized measurement and attestation services, running in dedicated virtual machines called M&A domains. This separation provides isolation between the measurement target and the measurement tool. Upon receiving an attestation request, the server's software directs the incoming request to its M&A domain and proxies responses from that M&A back out to the client. The M&A domain is broken out into components that can satisfy the different portions of an attestation request (see Fig 3).

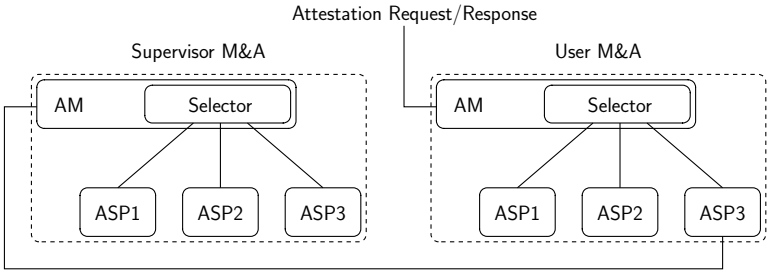


Fig. 3. Generic Measurement and Attestation (M&A) Architecture

There are three main enabling technologies for an attestation: The appropriate evidence collector must be selected; the evidence must be collected; and the data must be moved between all relevant parties in a trustworthy manner. We explore each of them in more detail as we break down a single attestation.

An attestation manager (AM) manages the attestation session. Attestation Service Providers (ASPs) provide specialized attestation services and are called by the AM. All measurement tools on the system fall into this category.

Our grocery store server chains an attestation across the platform by the use of ASPs that make attestation requests to other M&A domains and relay the attestation responses. Fig. 3 shows a possible set of components that might be used in an attestation, including an ASP in the User M&A domain (perhaps a certificate handling program) which makes an attestation request to the Su-

pervisor M&A domain for the underlying system certificate. This enables the grocery store server to make an attestation satisfying Principle 5 (Trustworthy Mechanism).

By abstracting key attestation services into ASPs, and abstracting over ASPs using the AM, we gain an extensible system with the ability to add new services and protocols without the need to redesign or re-evaluate the entire system for each addition. Should new data processing software be installed on the server, a new measurement tool could also be installed without requiring any change to the client attestation request. Procedures for adding and removing ASPs raise infrastructure issues that are not addressed in this paper.

This extensible system offers the opportunity for extremely sophisticated measurement capabilities. New capabilities are primarily focused on VM-to-VM inspection within a virtualized system.

Evidence of Security State Fresh and comprehensive information about a particular component (Principles 1 and 2) may be prepared in a virtualized environment if another component can inspect its memory, delivering evidence about its current state. This current state evidence is intended to establish some desired property of its future behavior.

Clearly, there are different cases, depending on the complexity of the behavior that the VM needs to engage in. We discuss two main groups, but regard this as an area where much future work will be needed.

Essentially finite state components. Some virtual machines are essentially finite state devices, in the sense that they engage in some transaction, and then return to a predictable state. This property makes them easy to measure, since they need to be inspected only when they have returned to one of the known stable states. An appraiser can check the hash of the state (or the memory that it occupies) against a practicably short list. A small VM supplying a trusted system service can be designed to satisfy this property.

A VM that has been implemented in a garbage-collected language can be essentially finite state. A garbage collection should occur at the end of each transaction. Then the measurement activity should hash the VM's code and also the live memory accessible from registers and pointers in the code. The result should be a hash in a small known set. This measurement activity may visit live memory using the same algorithm that the garbage collection itself would, because the hashing activity is like the garbage collection. However, instead of copying live memory to a fresh heap (as a stop-and-copy GC would do), it simply accumulates the contents of memory via hashing. Other variants of this idea are possible.

Components with crucial invariants. Most components are not essentially finite state. For instance, the Linux kernel maintains linked lists of data structures representing the inodes for files that are currently open. The Linux Kernel Integrity Monitor, a research prototype measurement mechanism for Linux [23], walks this

linear list, asking certain questions about each entry. For instance, depending on the filesystem type on which an inode resides, its pointers should reference particular groups of functions implementing the fundamental file system operations. Although inodes for files on different filesystem will have pointers with different values, there is a limited set of possibilities. Thus, a procedure that walks the list and for each inode structure checks its filesystem type and the targets of its pointers is reasonable. In fact, various attacks would be thwarted or detected by this procedure. In some cases, two data structures should be inspected together to check their consistency, for instance, the process list should contain all processes on the list to be scheduled. Hackers often prefer to make the processes they run not appear in the process list.

Much of this evidence-gathering can be viewed as algorithmically akin to garbage collection [37]. The measurement process needs to traverse pointers repeatedly, starting from pointers that reside in static program variables or on the stack. As each data structure is encountered, some checking procedure should determine whether it contains an invariant-respecting value, or whether it has reached a state that the code can never legitimately produce. However, some aspects—e.g. correlating multiple representations of the collection of processes within a kernel—have a different structure.

Naturally, the code itself will probably need to be checked via some opaque process like comparing its hash to a list of permissible values. Some data structures like the Linux system call table—which translates numeric operations codes to pointers to functions—will simply be checked to make sure that they have not altered since start time.

5.2 Domain Separation: A Trustworthy Base

As mentioned above, suppose that before forwarding the day's transactions to a central server, a grocery store client system requires and evaluates an attestation. If that attestation included evidence constructed in an appropriate way, the client could be confident that the server will not transmit its credit card numbers to a third party, *before* transmitting any information.

Recall that we envision the central server having an architecture like that pictured in Fig. 2. The hypervisor is the most crucial part of the trust base of the system. It is responsible for providing domain separation and sharing access (for measurement tools).

TPM virtualization allows parties interacting with the User platform to seamlessly make use of TPM facilities without requiring knowledge of the full system architecture. The measurement and attestation functionality included in the platforms provide specialized measurement functions and attestation capabilities for varied target environments. They form the externally-visible face of the increased capabilities for the server.

Of course, booting such a system in a trustworthy fashion requires both care and planning, to enable later attestations about the system support structure. Attesting to a correct boot process relies heavily on the TPM quote.

5.3 Self-Protection: The Boot Process

A central source of evidence about a platform is the array of Platform Configuration Register (PCR) values stored in the TPM. An attestation protocol may deliver a requested subset of those values to an appraiser. But what conclusions may the appraiser draw from inspecting those values?

The TCG vision for proper use of a TPM is based on a *chain of trust* rooted in the TPM hardware and some firmware. The firmware, referred to as the "root of trust for measurement" (RTM) kicks off a series of measurements at boot time.

The chain of trust is simple in principle but complex in fact. The simple view is that the boot process traverses a sequence of firmware or software system components, each of which measures the next component in the sequence before transferring control to it. At boot time, the measurements are just hashes of the code in each component, and each hash value is stored in a designated PCR. If there is an unbroken series of measurements that match the expected values, each measured software component should be correct.

This conclusion depends on a belief that if the system software has not been corrupted, it performs its measurement duties as specified. There may be bugs in the original software, of course, but at least the threat of malicious modification or other alteration has been reduced.

The chain of trust also depends significantly on details of the TPM design. For example, it is not possible, in a single command, to simply rewrite a PCR to a given value. PCRs may be either reset or *extended*, where an extend operation combines the existing and input values by hashing their concatenation. In earlier versions of the TPM, PCRs could be reset only on a hardware reboot. Thus, a maliciously modified component could be prevented from rewriting the PCR containing its measurement, and thus replacing a telltale bad hash with the expected good value.

In the more recent version 1.2 of the TPM [40], some of the responsibility for trusted operations is delegated to external firmware in the associated chipset, as provided by the CPU manufacturers. New "dynamic" PCRs were added, which could be reset at any time. However, reset and extension operations on these PCRs are subject to access control based on *localities*. The locality of a command may be determined by which system component originated it. This permits some flexibility, such as a partial reboot with new system software, called a "late launch". It also complicates the argument supporting the integrity of the stored measurements.

The more complex TPM version 1.2 design led us to ask if we could check the argument for validity of measurements formally, using an abstract model of the TPM and associated chipset features. A particular concern was to model the possibly malicious misbehavior of contaminated system software. We assumed a measurement architecture that is consistent with the Intel view in [11], in which the CPU has an SENTER instruction that loads and measures an initialization routine SINIT, which in turn measures previously loaded images of the hypervisor and a few basic supervisory VMs, before launching the hypervisor.

Our model [26] of this architecture was written first in SMV [24], and then rewritten in SAL [8]. The SAL version of the model is not yet publicly available. These languages support automatic model checking of system properties. We checked invariants stating that if certain PCR values were as expected, then they were in fact the correct measurements of system components nominally assigned to them. This way, we could confirm exactly which subset of the PCRs had to be reported to confirm measurements of various components, including all measured supervisory VMs. Other verified properties included an expected property of late launch, namely, that even if the BIOS is compromised, the late launch process still permits trusted system software to be loaded and confirmed; and the inference that if a certain good PCR value is observed, then SENTER has been executed.

5.4 Attestation Transport

Principles 3 and 5 require that the target be able to control the propagation of information about its state, and that a verifier be able to trust the underlying attestation mechanisms. It is especially challenging to satisfy these principles in a distributed setting where evidence is transported over a public network. The natural approach to addressing these challenges is with the use of cryptographic protocols.

A cryptographic protocol is a mechanism to produce *controlled agreement* among principals in a distributed system. The particular goals of a cryptographic protocol determine which parameters participants should agree on as well as additional constraints on those parameters. For instance, a key agreement protocol is intended to produce agreement on a shared session key, and possibly additional parameters such as the identities of the peers receiving the key. Furthermore, communication is controlled—via the use of cryptography or randomly chosen nonces—to ensure, for instance, that no other principal can learn the session key, or that certain responses were freshly generated. A cryptographic protocol achieves its goals when all executions, even in the presence of malicious parties, lead to the desired controlled agreement.

In order to implement remote appraisal it is not enough for a principal V, making a decision about some evidence, to trust a principal A—in the sense of *trust* stipulated in Section 2, page 5—with regard to the correctness of the evidence A provides. V should apply the trust inference only if V knows that A has made the statement. That requires agreement on A's identity and on the parameters embedded in the message in which A reported the evidence. Similarly, in order for A to assess properly whether or not it is safe to send the evidence, it should know the correct identity of the V that can interpret its message. It must rely on the transport mechanism to prevent any other principals from learning the content of the evidence. In other words, sound decisions by both A and V require controlled agreement of certain relevant parameters. Thus a well-designed protocol which is known to achieve its goals will allow both A and V to make well-informed decisions.

A Two-Party Attestation Protocol. A simple protocol for attestation using a TPM is given by Sailer, et. al. [31], as shown in Fig. 4, using terminology adapted to our conventions. At boot time, measurements of OS code are deposited into some PCRs. Subsequently, the system makes measurements M, typically when loading code into processes, and deposits the hashes $h(M)$ of these measurements into other PCRs. They do not work in a virtualized framework.

In a protocol run, a Verifier V sends a challenge containing a nonce N_V to a client's Attestation Service.[2] The Attester responds with the measurement list M and a TPM quote result containing PCR values from the target. The quote result has a PCR vector P, with a hash that binds N_V to it, signed with an identity key I. Confidentiality and authentication for the exchange is provided via an underlying SSL session.

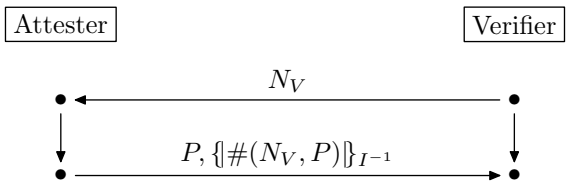


Fig. 4. Sailer Protocol

The ideas behind this protocol are reasonable. However, finer trust analysis is desirable in a virtualized framework. Moreover, a client C may want to interact with a server S, but would not permit S to view all its measurements (Principle 3). Thus, we must define how S asks an acceptable V to perform the appraisal, and how S receives the answer from V. We also think it important to separate the actions of the client C from those of its Attester A. Given this multiparty structure, it is important to make the cryptographic protection on the protocol explicit, rather than relying on SSL. For these reasons, we have designed a new protocol involving four main parties and a certificate authority E.

CAVES, a Multiparty Attestation Protocol. CAVES is designed to support attestation in a context in which measurements are of the sort described in Section 5.1, and evidence evaluation is delegated to another principal V, a specialist used to reconcile the comprehensiveness proposed in Principle 2 with the constrained disclosure of Principle 3.

The CAVES protocol was named after its five participants (see Fig. 5). The additional principals are a client C, a server S, and the enterprise Privacy Certification Authority E responsible for the certificate covering the Attestation Identity Key I.

[2] We call the latter the Attester A, and we retain the term "Verifier" V rather than "Appraiser" to avoid a collision in using the letter A.

The protocol begins with a request by C to receive some resource R, which includes a session key K to use to deliver the challenge from the verifier V. Server S delegates to some acceptable specialist appraiser or "verifier" V the task of appraising C. V uses information in R to select an appropriate query J and a PCR selection mask M, and delivers that, forwarded through S and C, to C's local measurement and attestation agent A. In our platform architecture, the attestation manager in the M&A domain selects and launches an ASP as introduced in Section 5.1 and as described in more detail in Section 5.5. That ASP may invoke other ASPs to perform specialized measurements in accordance with the query J, and to access a TPM.

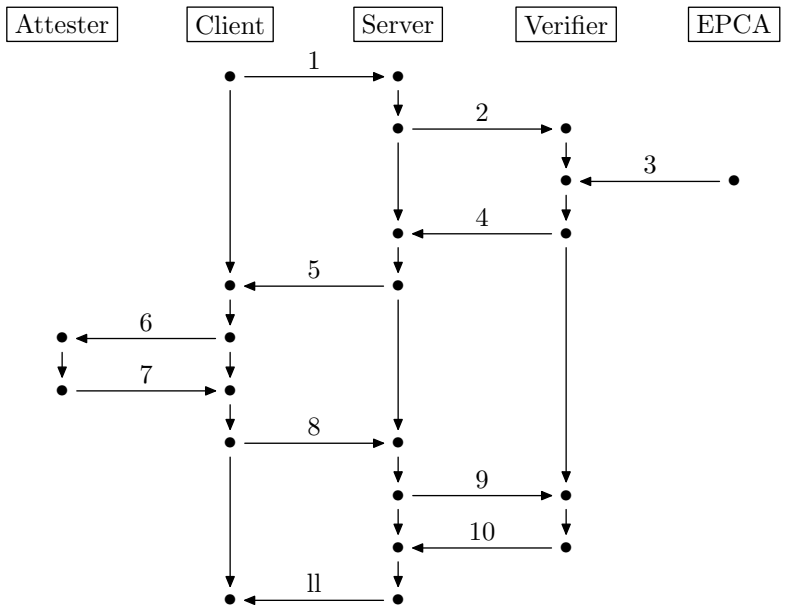
A retrieves the evidence requested in J in the form of J's output J_O, e.g. using the methods described in Section 5.1. It packages this output together with the current PCR values P for the registers selected by V in the PCR mask M. This information is guaranteed using a TPM quote. The TPM quote uses the hash $\#(A, V, R, N_V, J, J_O)$ of some parameters as a nonce-like seed to be included in a digital signature. The digital signature is prepared using I^{-1}, a TPM-resident Attestation Identity private key.

This information is returned to V in a blob B encrypted with V's public encryption key, to ensure that C and S cannot read it. V selects an Attestation Identity Key I, certified by an Enterprise Privacy Certification Authority E, to validate the quote. V obtains the attestation identity key I to validate the quote by looking in a credential (shown in message 3) for the attester A specified in message 2.

The keys K_S and K_V are public (asymmetric) encryption keys, for which the participants may have PKI certificates. We assume that participants use these PKI certificates to decide whether to regard the matching private parts as uncompromised. The key K is a symmetric session key created by C for this interaction with S. N_S, N_V are randomly chosen nonces that S, V generate for this session. The key K' is a symmetric key created by the attester, and used to protect the delivery of the data.

The channel between the attester and the client is protected by the hypervisor and is not encrypted. Our method for modeling the channel protection is by regarding the messages between the attester and the client as encrypted with an artificial long term symmetric key $\mathsf{ltk}(A, A)$.

We have analyzed this protocol in detail using CPSA. CPSA is a Cryptographic Protocol Shapes Analyzer [9], and it was extremely useful in the process of refining CAVES. In particular, CPSA allowed us to develop this version of the protocol in which C is essentially anonymous: No other participant has a name for C, and no PKI certificates refer to it by any name. Thus, all that V and S know about C comes from the evidence provided by A. This evidence consists of the measurement output J_O together with the PCR values that are signed in the quote. Verifier V must decide whether this evidence is adequate to justify a positive appraisal, and indeed this is exactly the specialized expertise that A and S had in mind, when taking V as a mutually acceptable verifier.



$$C \to S : \{|R, A, K|\}_{K_S} \quad (1)$$
$$S \to V : \{|S, R, A, N_S|\}_{K_V} \quad (2)$$
$$E \to V : \{|\mathsf{cert}, A, I, E|\}_{K_E^{-1}} \quad (3)$$
$$V \to S : \{|N_V, J, V, N_S, M|\}_{K_S} \quad (4)$$
$$S \to C : \{|N_V, J, V, M|\}_{K} \quad (5)$$
$$C \to A : \{|S, N_V, J, V, M, R|\}_{\mathsf{ltk}(A,A)} \quad (6)$$
$$A \to C : \{|K', N_V, B|\}_{\mathsf{ltk}(A,A)} \quad (7)$$
$$C \to S : B \quad (8)$$
$$S \to V : B \quad (9)$$
$$V \to S : \{|\mathsf{valid}, N_S, K'|\}_{K_S} \quad (10)$$
$$S \to C : \{|\mathsf{data}, D|\}_{K'} \quad (11)$$

$$B = \{|K', S, J_O, M, P, \{|\#(\#(A, V, R, N_V, J, J_O), M, P)|\}_{I^{-1}}|\}_{K_V}$$

C:	Client	R:	Request	N_S:	Server Nonce
A:	Attester	D:	Data	N_V:	Verifier Nonce
V:	Verifier	M:	PCR Mask	K:	Client Key
E:	EPCA	P:	PCR Vector	K':	Attester Key
S:	Server	J:	Query	I:	Identity Key
		J_O:	Measurement	B:	Blob

Fig. 5. CAVES Protocol

A strength of analysis with CPSA is that it helps to clarify the interplay between trust assumptions and protocol structure. In particular, the CAVES analysis relies on one central trust assumption. This is:

Appraisal and behavior. If verifier V approves a run with attester A, and receives acceptable values J_O, P, then the underlying platform of A protects communication between A and its local target, a client C. Moreover, this C's behavior in runs of CAVES will be in accordance with the requirements of the protocol.
The implication of this inference for CPSA usage is to declare the virtual channel key $\mathsf{ltk}(A, A)$ to be uncompromised.

With this principle in mind, we analyzed CAVES in the following steps.

1. Under reasonable assumptions, CPSA tells us that a complete run of the server S role is possible only if the verifier V and attester A have also had complete runs; moreover, the parties agree on all the parameters they should share.
2. Since V will send its last message only if it approves the values from the attester A, this justifies the additional assumption that $\mathsf{ltk}(A, A)$ is uncompromised.
3. With this assumption added, CPSA determines that no adversary can simulate the behavior of a client C. Hence, if S sends the requested data D, then D goes only to an approved C.
4. We also check that an adversary can never receive the platform's sensitive data J_O and P.

CPSA Results for CAVES. Expanding this outline slightly, we analyze CAVES from the point of view of the server S. We will make two steps. First, we will use CPSA to discover what can have happened, given that S has had a full local session of the protocol. In this first step, we will assume that the private decryption key K_V^{-1} of the selected verifier V is uncompromised. We will also assume that S's own private decryption key K_S^{-1} is uncompromised.

We will assume that each regular, compliant principal, when selecting a nonce or session key, will choose one that is fresh and unguessable. By "fresh," we mean that no other regular participant will have chosen this same value for any purpose. By "unguessable," we mean that the adversary, if trying to create useful values, will never be lucky enough to select this same one.

Subject to these assumptions, CPSA tells us that in CAVES, every execution in which a server S has a local run has at least the behavior shown in Fig. 6. The dotted arrows $n \overset{\prec}{\cdots\cdots\succ} m$ indicated that some message ingredients sent at node n must be available in order to deliver the message received at node m. In this run, the participants agree on all of their parameters, meaning that the protocol has forced a sufficient amount of agreement among them. However, the client C is absent. Nothing in our current assumptions forces the involvement of a run of a regular, compliant C.

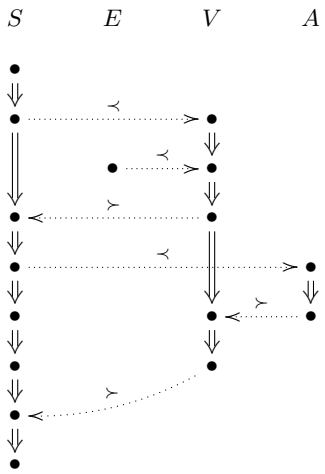


Fig. 6. Behavior Required if Server S Completes

However, our principle on appraisal and behavior justifies adding *another* assumption, namely that the key $\mathsf{ltk}(A, A)$, representing the channel between appraiser A and the client (target of attestation) C is used only in accordance with the protocol. In part, this represents a conclusion about the platform underlying C. Because V can appraise the PCR values P, V can ascertain that the underlying platform implements a safe channel between A and its target of attestation. Moreover, the measurement output J_O provides evidence allowing V to infer that the behavior of the target of attestation will meet expectations. Our principle allows us to put these two conclusions together, to justify the additional assumption that $\mathsf{ltk}(A, A)$ will be used only in accordance with the CAVES protocol.

If we restart the analysis with the behavior shown in Fig. 6, augmented by this new assumption, CPSA reports that every possible execution contains also a client session, as shown in Fig. 7. Again, the participants agree on all expected parameters.

One other key fact is a limitation on the agreement of principals on a parameter. In particular, it is a goal of the attester A that no party other than V should learn the values P, J_O. This means that no other participant will learn its internal state. It is evident from the protocol that C, S do not learn them. However, could an adversary? To test this, we ask CPSA two questions. The first, shown in Fig. 8, shows a run of A and a disclosure of J_O, indicated by the node at the top right that receives the values J_O unprotected by any encryption. Subject to the assumptions that K_V^{-1} is uncompromised, and that J_O and K' are fresh and unguessable, CPSA reports that this diagram is incompatible with any possible execution. The same remains true if J_O is replaced with P.

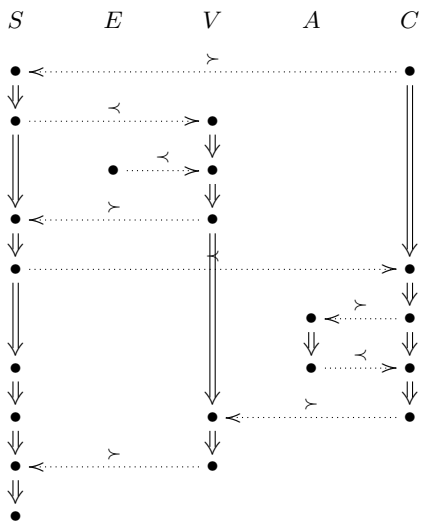


Fig. 7. Behavior Required if Server S Completes, and the Channel between C and A is used only in Accordance with CAVES

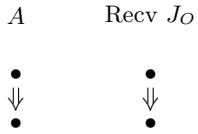


Fig. 8. Unsatisfiable Query, Disclosing J_O

5.5 Managing Attestation

Recall that in Figure 2 both the S and U environments possess specialized measurement and attestation services, running in dedicated virtual machines called M&A domains. This separation provides isolation between the measurement target and the measurement tool.

Upon receiving an attestation request, the server's software directs the incoming request to its M&A domain and proxies responses from that M&A back out to the requester. The M&A domain is broken out into components that can satisfy the different portions of an attestation request (see Fig 3).

As mentioned above, there are three main enabling technologies for an attestation: The appropriate evidence collector must be selected; the evidence must be collected; and the data must be moved between all relevant parties in a trustworthy manner. We explore each of them in more detail as we break down a single attestation.

An attestation manager (AM) manages the attestation session. This includes listening for requests, breaking them into calls for evidence, and implementing any privacy policies the server might have, as specified in Principle 3. (For example, the server may only be willing to share configuration information with a small subset of peers.)

Attestation Service Providers (ASPs) provide specialized attestation services and are called by the AM. In the case of our grocery store server, there are four components to the request: A certificate, an OS measurement report, a software measurement report, and a vTPM quote. One AM implementation currently in existence uses the *call by contract* [25] method; each ASP provides a set of *contracts* describing what actions it will take and what input it requires. Again following our running example, the vTPM may offer a quote given a PCR mask and nonce as input.

The ASP and AM concepts are intentionally general, so as to allow for the widest possible field of attestation scenarios. Internal ASP structure will depend on function. We expect most ASPs to fall into one or more of the following categories:

- Computation of attestation-relevant information, such as specialized OS measurement,
- Calls to other system resources, such as the vTPM, and
- Calls to its parent AM, requesting the use of another ASP

A particular M&A domain will likely include several ASPs covering all of our functional categories in order to support our Principles 1–5 for the platform. The grocery store server might include specialized OS and data collection software measurement agents, the vTPM for the U guest, the hardware TPM, and perhaps a hash computation/checking function used by both the Attestation Manager directly and the OS measurement agent.

The choice of one or more ASPs for measurement may be informed by all of our principles; different ASPs may provide information that is fresh, comprehensive, and/or semantically explicit to varying degrees, and the attestation manager may choose not to call the most comprehensive or semantically explicit service providers in order to constrain the disclosure of system information. Different ASPs may also be more or less trustworthy; a simple measurement provider might provide less useful information but allow easier verification of correct measurement behavior. The mechanism that matches a request to the right ASP is called the *Selector*.

Contracts and Selection As explained in [25], the AM will interpret its request as a *contract* expressible in a logical form. A contract has a *precondition*, a *postcondition*, and a confidentiality requirement called a *nondisclosure agreement (NDA)*.

Preconditions and postconditions are logical formulas. In the request contract, these formulas are expressed using logical terms formed from predicate

symbols and formal parameters. For example, a request for a TPM quote might have a precondition

$$\mathsf{nonce}(n) \wedge \mathsf{mask}(m) \wedge \mathsf{idkey}(k)$$

(where "$\wedge$" is a logical "and") and a postcondition

$$\mathsf{tpmQuote}(n, m, k, p, q).$$

The request contract also has a mapping of input parameters (the ones in the precondition) to values specified by the caller. The semantics of most of the predicate symbols used is application-specific; we will say more about this shortly.

A possible form for the NDA is a mapping of formal parameters to conditions under which the corresponding parameter may be released. A simple instance of this scheme is to assign either a condition of true, meaning that the parameter is not confidential, or false, meaning that the parameter value is shared only by the service and the caller. More complex instances could specify particular principals who may see certain parameter values.

Given a request contract, the Selector must find an ASP that can satisfy it. To do this, it uses a registry, which is a table of entries corresponding to the available ASPs. Each entry expresses a service contract and also contains a link to the code implementing the service. The form of a registry entry is shown in Fig. 9.

Precondition	Postcondition	NDA	Link

Fig. 9. Attestation Service Registry Entry

Although the conditions in the service contracts have the same form as those in request contracts, the conditions in an acceptable service contract do not have to match the request conditions exactly. It is enough if the request precondition implies the service precondition, though the former may be stronger; and the service postcondition implies the request postcondition, though the former may be stronger; and the NDA conditions also relate in such a way that the confidentiality commitments of the service imply the confidentiality requirements of the request. Obviously, this means that ASPs must be trusted to keep their promises. It is the responsibility of the administrator who fills in registry entries to do whatever is prudent about checking ASP claims.

In order to find an acceptable ASP, the Selector must map formal parameters properly, and also recognize relationships between predicate symbols. For example, if a request guarantees a precondition $\mathsf{positive}(x)$ and an ASP relies upon a precondition $\mathsf{nonnegative}(x)$, a well-informed Selector would invoke a built-in rule that *positive* implies *nonnegative*, and would consider that ASP as a candidate. A good Selector should have an extensible knowledge base with rules like this. Some caution is needed here, because the caller and ASP must agree on

any such rules, otherwise spurious selections might be made. An implementation intended for wide usage would have to set up naming and definition standards for condition predicates to prevent misuse.

All but the simplest ASPs may be expected to require some communication outside the VM in which the ASP was entered. An ASP may be a protocol that uses a network connection to exchange messages with an external principal, or it may make requests involving inter-VM communication. An ASP may also request a service by sending a contract to the AM and asking it to handle the selection, invoking any other ASP that provides the service. We would expect, for example, that an ASP designed to implement the CAVES protocol would ask the AM to find ASPs providing kernel measurement and TPM commands. A call on a TPM command might be either to a vTPM or the hardware TPM, and the requested postcondition should be able to distinguish between those services.

We have implemented a prototype Selector, which has been used with experimental attestation protocols in which TPM commands and other computations are invoked as services. The algorithm it uses is specified in [25]. We use a lightweight Datalog runtime engine to handle the logical manipulation of condition formulas and rules necessary for fully flexible and application-dependent selection.

Had Hannaford customers been able to support attestation as described in this paper, perhaps the problems that were experienced might have been avoided. Client software would have been able to determine that either the server OS or data processing software had been compromised before sending credit card transactions.

6 Open Problems

Even with our architectural constraints and system design, some aspects of the attestation problem remain difficult to solve. The most difficult principles to satisfy with today's technology are establishing **Trustworthy Mechanisms** and gathering **Comprehensive Information**.

The Trusted Platform Module and related technology from Intel (TXT) [6] and AMD (SVM) [5] are useful means for bootstrapping certain aspects of a self-protecting trust base, but a richer trust base is needed than can be provided by this sort of hardware alone. The emergent hardware technologies provide a known origin for any evaluation, the core root of trust for measurement. But ultimately, the integrity of the trust base depends on the assurance of the "hypervisor" implementation. Specifically required is a means to establish domain separation in order to support a trustworthy mechanism for attestation. Our current implementation uses an off-the-shelf virtualization system—but none of those available today offer the desired balance between flexibility and security. Solutions to this problem might be found either by extending traditional separation kernels or possibly by producing a small, assurance-focused virtual machine hypervisor.

The major problem in gathering comprehensive information is that in order to establish trust in application-level software one first needs to establish trust in the operating system on which the software depends. Today's mainstream operating systems were not designed with assurance or measurement in mind. They are large and complex, containing many dynamic features that make them very difficult to analyze even in the absence of a hostile party. It seems unlikely that this situation will improve until there is either a major shift in the structure of mainstream operating systems or the widespread adoption of a new operating system designed from the beginning with measurement and assurance as a design goal.

Another problem in achieving comprehensive information is that, although many researchers are developing specialized tools for measurement and attestation, including [23, 32, 42, 10, 2], these tools have not been integrated into a unifying, multi-purpose attestation architecture. Connecting these disparate viewpoints on a single system into a single comprehensive attestation platform will be a significant challenge.

7 Existing approaches to attestation

Since the introduction of the TPM, research efforts in the area of system attestation have widely expanded. It is a major component and focus of work being done within the Trusted Computing Group [40] [38] [12], Microsoft [7], and multiple independent researchers [18] [33]. Many of these proposals are aimed at specific use cases and thus lack the flexibility to address the more general attestation problems discussed in this paper. Taken in total, however, the efforts aimed at these specific use cases highlight the need for both a broader notion of attestation and a flexible architecture capable of supporting that broader notion.

Trusted Network Connect. Trusted Network Connect (TNC) is a specification from the Trusted Computing Group [38] intended to enable the enforcement of security policy for endpoints connecting to a corporate network [38], and is generally seen as supporting activity at network layers 2 or 3. For this reason, the TNC architecture makes some assumptions about the relationships between parties that make it of limited value for application-level attestations. Once a party has network access, it moves outside the scope of TNC. It should be noted that TNC is not itself a networking or messaging protocol, but rather is intended to be tunneled in existing protocols for managing network access, such as EAP [1].

When considered in light of the principles for attestation outlined in this paper, TNC is most naturally comparable to a specialized attestation manager. Much of the purpose of the TNC Client (TNCC) is to select the appropriate Integrity Measurement Collectors (IMCs) based on requests from Integrity Measurement Verifiers. However, in a TNC-compliant system, no separation exists between measurement tools and either the attestation management function or other measurement tools [39].

TNC allows limited attestation delegation and identification. In particular, before the integrity measurements are taken, "mutual platform credential authentication" [38] can occur. In the TCG context, this means that the two parties can each verify that the other has a valid, unrevoked TPM AIK. However, truly mutual authentication cannot occur in TNC due to its nature as a network access protocol; no attestations from the server to the client other than the initial credential exchange are possible.

Pioneer and BIND. Pioneer and BIND are attestation primitives developed at CMU with very specific design constraints, and could be naturally viewed as particular ASP instantiations.

BIND [34] is a runtime code attestation service for use in securing distributed systems. It centers around a specific measurement capability which binds a proof of process integrity to data produced by that process, and is best-suited for embedded systems without flexible attestation needs.

Pioneer [33] is an attempt to provide a "first-step toward externally-verifiable code execution on legacy computing systems." Here, legacy means systems with no hardware trust base – Pioneer attempts to solve the attestation problem entirely in software. This faces serious challenges in the presence of a malicious OS, and at least one method is known for an OS to fool Pioneer. Also, the success of Pioneer on any given system requires an immense degree of knowledge about (and control of) the underlying hardware. A trusted third party must know the *exact* model and clock speed of the CPU as well as the memory latency. The system must not be overclocked, must not support symmetric multi-threading, and must not generate system management interrupts during execution of Pioneer. Thus, an attacker with sufficient hardware understanding might subvert attestation. At least one such attack, for systems with 64-bit extensions, is mentioned on the Pioneer web site [33].

Schellekens et al. [32] seek to improve the reliability and remove some of the constraints on Pioneer-style attestation systems by integrating hardware support from TPMs in several ways. By using a TPM's tick stamping functionality, a challenger can achieve an approximation of a time measurement made on the appraised platform, reducing the variability resulting from network lag. They also propose modifications to the bootloader, so that a TPM's PCRs could contain information about the machine's CPU identifier or a performance measurement of a test Pioneer challenge. These modifications remove some attack vectors against Pioneer.

Nexus. Nexus [35] is an effort at Cornell to develop an operating system with particular attention to "active attestation." It enables separation via secure memory regions and moves device drivers into userspace. It introduces "labeling functions," a mechanism for providing dynamic runtime data to appraisers. Measurement tools may be sent to the target system by the appraiser and do not need to be pre-integrated with the base system.

As it involves an entirely new microkernel-based operating system, Nexus is most likely to be deployed in a specialized role such as something one might

use for separation purposes instead of a traditional hypervisor. This relationship seems even more relevant in light of the fact that the Nexus project intends to be able to run Linux on top of a Nexus process.

Property Based Attestation. There has been extensive research into the notion of *property-based attestation*, often within the context of particular scenarios. Most of these scenarios can be viewed or adapted to fit within a flexible attestation framework. We mention a brief selection here.

Jonathan Poritz [28] and Sadeghi and Stüble [29] have each pointed out that the end consumer of an attestation ought to care about security properties of the target, as opposed to the specific mechanisms employed to achieve those properties. Poritz suggests that a solution might include virtualization and trusted third parties, but does not propose a concrete approach. Sadeghi and Stüble go farther, suggesting multiple approaches, some of which may require significant changes to the TPM or the TCG software stack.

These authors and several others go on to propose a protocol for performing such property-based attestations [4]. This protocol could be implemented as an ASP in our system.

Kühn et al. [22] propose a mechanism for modifying the boot loader to produce property-based attestations for use in TPM quotes and data sealing. In addition to storing the standard hashes in TPM PCRs, the boot loader also refers to a pregenerated list associating measurements with properties and stores the appropriate properties in another register. A trusted boot loader will only use signed property lists created by a trusted authority whose public key matches a reference stored in the TPM. This effectively moves half of the appraiser's functionality—evaluating hashes for meaning—onto the local machine.

Their solution offers several advantages over earlier property-based attestation proposals, including not requiring extensive changes to the TPM or OS and allowing for data to be sealed to properties rather than hashed measurements. However, their approach assumes a central entity in an IT environment which is responsible for defining property lists, and which is trusted to do so by all recipients of measurements. This is useful in many real-world environments, but would require modification for use in delegated attestation architectures like those we describe. In addition, while hashes of machine components can be hidden, the properties of all software on the machine are extended into a single PCR, providing limited options for constrained disclosure.

Sadeghi et al. [30] also build upon the ideas proposed by Kühn et al. by describing a virtual TPM architecture in which each virtual TPM may contain one or more property providers. Each property provider translates actual hash values as recorded in a PCR into a provider-specific property PCR corresponding to the original. A vTPM policy determines whether actual measurement data or some particular property data is used for a given TPM command, so that, for example, actual hashes might be released for a quote but properties derived from a trusted certificate might be used for sealing and unsealing data. While the proposal in this paper uses vTPM operations to determine which property provider should be used, one could imagine variations with more complexity;

taking appraiser identity into account could allow fine-grained privacy control, while attestation delegation could be achieved by negotiating provider selection based on mutual trusted authorities.

Nagarajan et al. [27] take a related approach to the problem of property-based attestation, suggesting that expert appraisers can provide property certificates to a platform. Rather than having a trusted component on the appraised platform translate PCR values into properties, the signed certificate associating PCR values with a more abstract property is provided to measurement requestors along with a TPM quote. The advantage of this system is that multiple trusted authorities can provide property associations, and properties may cover different abstraction levels. However, this system does not allow for even the limited privacy granted by [22], since any relevant PCR values will still need to be provided to a measurement requestor.

While this is not a complete list of research into property-based attestation, the variety of use-cases outlined here highlights the need for our notion of semantically explicit attestation and a flexible architecture capable of supporting it.

TLS Variants. Gasmi et al. [10] and Armecht et al [2] propose specialized TLS variants that include evaluation of endpoint states, local policy decisions about the trustworthiness of remote communications partners, and TPM-rooted session keys. Central to their goals is the idea that communications partners should know not only what software the remote system was running at the beginning of the communication, but should have the opportunity to cease communication if that state changes at any time in an unacceptable fashion. These protocols are natural candidates for inclusion as ASPs within an architecture like the one we describe.

Other ASPs. Some highly specialized measurement and attestation solutions in the current literature are ideally suited for adoption in architectures like the one we propose, as attestation service providers (ASPs). With their focus on attestation of a limited portion of a system, these projects have the potential to give in-depth and detailed visibility into specific portions of a platform.

CoPilot [18] is a system for detecting root kits in a Linux kernel run on a PCI add-in card, accessing system memory using DMA. It periodically computes hashes over key parts of memory that impact kernel execution, compares against a known value, and reports to an external system that enables manual decisions to be made regarding detected changes. It does not provide a truly *comprehensive* measurement of the target system, because the measurements it produces do not include important information residing in the kernel's dynamic data segment. In addition, since CoPilot does not have direct access to the CPU's registers, it is only able to perform measurements at known memory locations and cannot associate any of its measurements with what is actually running on the processor.

Gu et al. [13] propose a mechanism for remotely attesting to the correctness of program execution. Their solution builds on top of a combination of a TPM and a secure kernel offering process isolation and memory curtaining. In order

to provide meaningful attestation about a program run, they measure not only the program code and initial input, but also additional input provided to the program during its execution in the form of relevant system calls. A dependency analysis of the program is used to determine relevancy. A PCR in the TPM is updated with the most recent measurement whenever a relevant system call is trapped, ensuring that the PCR value accurately reflects the program's execution history at all times.

Software is not the only useful attestation target. One of our core principles for attestation architectures is comprehensive information, so that appraisers can make decisions based on a complete picture of the target system. Van Dijk et al. [42] propose a mechanism by which a trusted timestamp device, such as the TPM's monotonic counter, can be used to prove recency of data in storage devices. Techniques such as this, when implemented using remotely verifiable trustworthy mechanisms, can supplement other software-focused attestation. For example, an architecture for a trusted database server might well have a recency checker as part of its measurement tool suite.

8 Conclusion

Attestation is an area which will see many technological innovations and developments in the near future. In particular, since the major vendors are introducing improved support for virtualized systems, architectures such as ours should be increasingly easy to implement in a trustworthy way. The semantic explicitness and freshness of the attestations that we propose should allow a common vocabulary across many architectures. Constrained disclosure should encourage systems owners to allow their systems to participate in attestations. Comprehensive information should encourage appraisers to place credence in well-supported claims, particularly given underlying trustworthy attestation mechanisms. We have attempted to clarify the way that existing work can be used to contribute to our goals.

References

1. B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol (EAP). RFC 3748 (Proposed Standard), June 2004.
2. Frederik Armknecht, Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, Gianluca Ramunno, and Davide Vernizzi. An efficient implementation of trusted channels based on openssl. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 41–50, New York, NY, USA, 2008. ACM.
3. Boris Balacheff, Liqun Chen, Siani Pearson (ed.), David Plaquin, and Graeme Proudler. *Trusted Computing Platforms: TCPA Technology in Context.* Prentice Hall PTR, Upper Saddle River, NJ, 2003.
4. Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A protocol for property-based attestation. In *STC '06: Proceedings, First ACM Workshop on Scalable Trusted Computing*, pages 7–16, New York, NY, USA, 2006. ACM Press.

5. AMD Corporation. Amd64 architecture programmer's manual volume 2: System programming rev 3.11. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf`, January 2006.
6. Intel Corporation. Intel trusted execution technology. `http://download.intel.com/technology/security/downloads/31516803.pdf`, November 2006.
7. Microsoft Corporation. Ngscb official page. `http://www.microsoft.com/resources/ngscb/default.mspx`, 2007.
8. L. deMoura, S. Owre, and N. Shankar. The SAL language manual. Technical Report SRI-CSL-01-02, SRI International, 2003.
9. Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Searching for shapes in cryptographic protocols. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, number 4424 in LNCS, pages 523–538. Springer, March 2007. Extended version at URL:`http://eprint.iacr.org/2006/435`.
10. Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N. Asokan. Beyond secure channels. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 30–40, New York, NY, USA, 2007. ACM.
11. David Grawrock. *The Intel Safer Computing Initiative*. Intel Press, 2006.
12. TCG Best Practices Group. *Design, Implementation, and Usage Principles for TPM-Based Platforms*, May 2005. Version 1.0.
13. Liang Gu, Xuhua Ding, Robert Huijie Deng, Bing Xie, and Hong Mei. Remote attestation on program execution. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 11–20, New York, NY, USA, 2008. ACM.
14. Joshua D. Guttman. Authentication tests and disjoint encryption: a design method for security protocols. *Journal of Computer Security*, 12(3/4):409–433, 2004.
15. Joshua D. Guttman, Jonathan C. Herzog, John D. Ramsdell, and Brian T. Sniffen. Programming cryptographic protocols. In Rocco De Nicola and Davide Sangiorgi, editors, *Trust in Global Computing*, number 3705 in LNCS, pages 116–145. Springer, 2005.
16. Joshua D. Guttman, F. Javier Thayer, Jay A. Carlson, Jonathan C. Herzog, John D. Ramsdell, and Brian T. Sniffen. Trust management in strand spaces: A rely-guarantee method. In David Schmidt, editor, *Programming Languages and Systems: 13th European Symposium on Programming*, number 2986 in LNCS, pages 325–339. Springer, 2004.
17. Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation – a virtual machine directed approach to trusted computing. In *Proceedings of the Third virtual Machine Research and Technology Symposium*, pages 29–41. USENIX, May 2004.
18. Nick L. Petroni Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *USENIX Security Symposium*, pages 179–194. USENIX, 2004.
19. Yasuharu Katsuno, Yuji Watanabe, Sachiko Yoshihama, Takuya Mishina, and Michiharu Kudoh. Layering negotiations for flexible attestation. In *STC '06: Proceedings, First ACM Workshop on Scalable Trusted Computing*, pages 17–20, New York, NY, USA, 2006. ACM Press.
20. Ross Kerber. Advanced tactic targeted grocer: 'Malware' stole Hannaford data. *The Boston Globe*, page 1, 18 March 2008.
21. Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification

of an OS kernel. In *ACM Symposium on Operating Systems Principles*, Big Sky, MT, October 2009.
22. Ulrich Kühn, Marcel Selhorst, and Christian Stüble. Realizing property-based attestation and sealing with commonly available hard- and software. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 50–57, New York, NY, USA, 2007. ACM.
23. Peter A. Loscocco, Perry W. Wilson, J. Aaron Pendergrass, and C. Durward McDonell. Linux kernel integrity measurement using contextual inspection. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable Trusted Computing*, pages 21–29, New York, NY, USA, 2007. ACM.
24. Ken McMillan. The SMV system, 1992. `http://www.kenmcmil.com`.
25. Jonathan Millen, Joshua Guttman, John Ramsdell, Justin Sheehy, and Brian Sniffen. Call by contract for cryptographic protocol. In *FCS-ARSPA*, 2006. `http://www.mitre.org/work/tech_papers/tech_papers_06/06_0498/index.html`.
26. Jonathan Millen, Joshua Guttman, John Ramsdell, Justin Sheehy, and Brian Sniffen. Analysis of a measured launch. Technical report, The MITRE Corporation, June 2007. `http://www.mitre.org/work/tech_papers/tech_papers_07/07_0843/index.html`.
27. Aarthi Nagarajan, Vijay Varadharajan, and Michael Hitchens. Trust management for trusted computing platforms in web services. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 58–62, New York, NY, USA, 2007. ACM.
28. Jonathan A. Poritz. Trust[ed — in] computing, signed code and the heat death of the internet. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1855–1859, New York, NY, USA, 2006. ACM Press.
29. Ahmad-Reza Sadeghi and Christian Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *NSPW '04: Proceedings, 2004 Workshop on New Security Paradigms*, pages 67–77, New York, NY, USA, 2004. ACM Press.
30. Ahmad-Reza Sadeghi, Christian Stüble, and Marcel Winandy. Property-based tpm virtualization. In *ISC '08: Proceedings of the 11th international conference on Information Security*, pages 1–16, Berlin, Heidelberg, 2008. Springer-Verlag.
31. Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.
32. Dries Schellekens, Brecht Wyseur, and Bart Preneel. Remote attestation on legacy operating systems with trusted platform modules. *Electron. Notes Theor. Comput. Sci.*, 197(1):59–72, 2008.
33. Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–16, October 2005. See also Pioneer Web pages, `http://www.cs.cmu.edu/%7Earvinds/pioneer.html`.
34. Elaine Shi, Adrian Perrig, and Leendert Van Doorn. BIND: A time-of-use attestation service for secure distributed systems. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2005.
35. Alan Shieh, Dan Williams, Emin Gün Sirer, and Fred B. Schneider. Nexus: a new operating system for trustworthy computing. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 1–9, New York, NY, USA, 2005. ACM Press.

36. Brad Stone. 11 charged in theft of 41 million card numbers. *The New York Times*, page B 1, 5 August 2008.
37. Mark Thober, J. Aaron Pendergrass, and C. Durward McDonell. Improving coherency of runtime integrity measurement. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable Trusted Computing*, pages 51–60, New York, NY, USA, 2008. ACM.
38. Trusted Computing Group. *TCG Trusted Network Connect: TNC Architecture for Interoperability*, May 2006. Version 1.1.
39. Trusted Computing Group. *TCG Trusted Network Connect TNC IF-IMC*, May 2006. Version 1.1.
40. Trusted Computing Group. *TPM Main Specification, Design Principles*, version 1.2 edition, 2006. `https://www.trustedcomputinggroup.org/specs/TPM/mainP1DPrev103.zip`.
41. Trusted Computing Group. *TCG Specification Architecture Overview*, revision 1.4 edition, 2007. `http://www.trustedcomputinggroup.org/.../TCG_1_4_Architecture_Overview.pdf`.
42. Marten van Dijk, Jonathan Rhodes, Luis F. G. Sarmenta, and Srinivas Devadas. Offline untrusted storage with immediate detection of forking and replay attacks. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 41–48, New York, NY, USA, 2007. ACM.