RZ 3548 (# 99559) 05/10/04
Computer Science 16 pages

Research Report

Property Attestation—Scalable and Privacy-friendly Security Assessment of Peer Computers

Jonathan Poritz, Matthias Schunter, Els Van Herreweghen, and Michael Waidner

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland
{jap,mts,evh,wmi}@zurich.ibm.com

Property attestation — Scalable and privacy-friendly security assessment of peer computers

Jonathan Poritz, Matthias Schunter,
Els Van Herreweghen, Michael Waidner
IBM Zurich Research Laboratory
Zürich, Switzerland
{jap,mts,evh,wmi}@zurich.ibm.com

Abstract

A core security challenge is the integrity verification of the software that is executed on a machine. For example, an enterprise needs to know whether a gateway machine has been infected by malicious code. One prevailing approach is to use directories of configuration check-sums to detect when a configuration has been changed (see `www.tripwire.org`). These software-only solutions have limitations when the operating system itself is compromised. The tamper-resistant Trusted Platform Module (TPM) specified by the Trusted Computing Group (TCG) allows a TPM-enhanced platform to securely attest to a configuration of a machine. Based on such binary attestation, a verifying peer computer can then decide whether or not to trust the verified platform.

In this paper, we argue that the approach of binary attestation is not privacy-friendly, scalable or open and vendor-neutral. The main criticism is that this approach needlessly discloses the complete configuration (i.e., all executed software) of a machine. The focus of binary attestation are the binaries instead of their security. We present a protocol and architecture for property attestation that resolves these problems. With property attestation, a verifier is securely assured of *security properties* of the verified platform's execution environment without receiving detailed configuration data. This enhances privacy and scalability since the verifier needs to be aware of its few required security properties instead of an huge number of acceptable configurations.

1 Introduction

Processing critical information relies on the security of the computing platform. Typical security goals are to prevent such critical information from leaking beyond the realm of machines that are trusted by the user or to prevent corrupted machines from

impacting the integrity of a computation. External verification of platform integrity enables a machine to verify that another machine meets certain security requirements. This is useful, for example, when a grid server wants to assure that a grid node is untampered before delegating a grid process to it.

The Trusted Computing Group (TCG) is an IT industry consortium which has developed a specification of a small, low-cost commodity hardware module, called the Trusted Platform Module (TPM). The TPM can serve as a root of trust in remote (and local) platform verification.[1] The base TCG model of this configuration verification process, which we shall call "binary attestation", aims at measuring all executed code. This is done by mandating that each measured piece of software stores metrics (hash values of the configuration's components) of a sub-component into the TPM before executing it. This is boots-trapped by the BIOS that is trusted by default and measuring and storing the boot loader. The chain of trust can then be extended to the operating system components up to the applications and their configuration files. Once the executables are measured into the TPM, the TPM can reliably attest to the metrics of the executed components by signing them with a TPM-protected key. The signed integrity metrics can then be transmitted to a verifying machine. This verifier can decide whether to consider the machine trustworthy enough to involve it in a subsequent computation. As we will elaborate in Section 3.3, this straightforward approach of binary attestation lacks scalability, privacy, and openness. The main reason is that the whole configuration is transmitted (limited privacy), that the verifier needs to know all configurations of all machines to be verified (scalability), and that the verifier checks binaries that are specific to a vendor and operating system instead of verifying security properties (limited openness).

We propose *property* attestation as an alternative. The idea of property attestation is to use TPM technology to provide a verifier with evidence of well-defined security properties of a remote verified platform without the ability (and need) to know what exact implementation has been used on that platform. We describe an architecture for property attestation where a trusted verification proxy converts a platform's integrity metrics into high-level security properties based on property certification statements of the measured components.

Property attestation improves scalability since a verification proxy can be specific to the verified machine.[2] It resolves the privacy issues since the verification proxy hides the configuration information. It enables openness since the same security property can be provided by many implementations by multiple vendors.

1.1 Outline of this Paper

The outline of the paper is as follows. § 2 summarizes related work. In § 3, we recall the TCG's 'binary attestation' approach focusing on reliably reporting the configuration of a machine. We point out its disadvantages and motivate the need for property

[1] Note that this article focus on the attestation function of the TPM while omitting other useful functions like secure storage that are orthogonal to our research.

[2] We describe various deployment scenarios for such a verification proxy, including how it can be securely implemented on the verified machine itself; the latter scenario allowing for secure self-attestation of properties by a verified platform.

attestation. In § 4, we outline the basic idea and the interactions that are needed for property attestation. In § 5, we describe protocols that achieve property attestation by means of a security proxy. This security proxy performs a configuration-based security evaluation of a given machine and reports the results (high-level security properties) to the verifier. In § 6, we describe various deployment scenarios; in one of these scenarios, the security proxy and the verified system are both deployed on a single machine, achieving the goal of self-verification of a machine. Section 7 concludes the article.

2 Related Work

Various approaches have been described for using secure coprocessors in combination with system measurements to preserve or prove the integrity of computing platforms. *Secure boot* implies that a system can measure its own integrity and terminate the boot process if an integrity failure is detected. Yee [15] describes a secure boot process where the secure coprocessor verifies loaded modules prior to their execution against securely stored cryptographic checksums of their images. The AEGIS system described by Arbaugh et al. [1] describes a multilevel secure bootstrap process for an IBM PC system also allowing recovery from integrity failures. Again, cryptographic hashes of loaded modules are compared with stored signatures before being executed; each layer validating the integrity of the subsequent layer, starting from a trusted (assumed uncompromised) portion of the BIOS.

Authenticated or *trusted* boot implies that a system can convince a remote verifier of its integrity and security. The IBM 4758 secure coprocessor [3] realizes both secure and authenticated boot; *outbound authentication* as described by Smith [10] allows coprocessor applications to authenticate themselves to remote parties. The bootstrapping of trust and the linking of each layer to its predecessor is realized by the predecessor generating a signature over the next layer's measurement result (cryptographic hash) as well as its public signature key.

The Trusted Platform Module (TPM) specified by the Trusted Computing Group [11, 13, 12, 14] supports a trusted boot through its cryptographic functionality and protected storage. The TPM provides the necessary interfaces for the various layers in the boot process to securely store measurements (cryptographic hashes) of subsequent layers. A remote verifying platform can be convinced of the correctness of these measurements based on a challenge-response authentication mechanism where the TPM signs the requested measurements.

The TCG specifications only define the trusted boot process up to the bootstrap loader. Maintaining the chain of trust up to the application layer requires additional support by the operating system. Sailer et al. [7] describe a TCG-based integrity measurement architecture for Linux.

3 The TCG Approach: Binary Attestation

The TCG specifications [11, 13] define mechanisms for a TPM-enabled platform to reliably "report its current hardware and software configuration to a local or remote

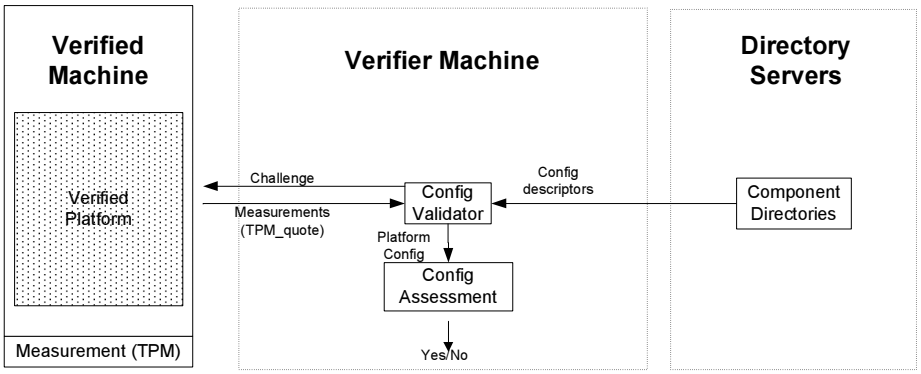

Figure 1: TCG Attestation Architecture

challenger" [2]. This 'binary attestation' (based on measurements of binary executables) is based on (1) the platform building a chain of trust from the hardware up to the operating system (and, potentially, including applications) by measuring integrity metrics of modules and storing them in the TPM, and (2) the TPM being able to report on these metrics in an authenticated way. A verifier obtaining such authenticated metrics can then match them against the values of a known configuration and decide whether the verified machine meets her security requirements or not.

3.1 Binary Attestation Architecture

Figure 1 represents a modularized architecture corresponding to the TCG concept of binary attestation.

The following entities are involved in the attestation and verification process:

Verified Machine A machine that has a Trusted Platform Module (TPM) and a computing base that may execute untrusted code.

Verifier Machine The machine of the verifier. All modules on this machine are known and trusted by the verifier.

Directory Servers Servers that provide additional authenticated information about components in signed component directories. Examples include Tripwire directories `www.tripwire.com`.

We make certain definitions to speak about this architecture:

Verified Platform The computing environment on the verified machine.

TPM The Trusted Platform Module (TPM) on the verified machine. It is used by the verified platform to store measurements of code executed in the verified platform.

Configuration Validator The module that obtains a (TPM-authenticated) measurement and reconstructs the platform's configuration. To do this, it uses a configu-

ration log file (see Section 3.2) additionally provided by the verified platform as well as configuration descriptors certified in Component Directories.

Configuration Assessment Given a configuration, the assessment determines whether this configuration satisfies the requirements of the verifier. This assessment typically involves matching the configuration against a set of configurations allowed by the verifier.

Component Directory A signed repository of information related to components. An example of a component directory is a software vendor's database providing hash values and associated descriptions of its latest products.

3.2 Binary Attestation and Verification Mechanisms

We now explain the interactions that implement binary attestation. The ability of the TPM reliably to report on the verified platform's computing environment follows from the TPM-enabled measurement and reporting. Our description in the following paragraphs focuses on the PC-platform [2].

The measurement and storage of integrity metrics is started by the BIOS Boot Block (a special part of the BIOS which is believed to be untampered) measuring itself and storing the measurements in a TPM PCR (Platform Configuration Register) before passing control to the BIOS. In the same way, the BIOS then measures option ROMs and the Boot Loader and records these measurements in a TPM PCR before passing control to the Boot Loader. The process continues as the Boot Loader measures and stores integrity metrics of the OS before executing it, the OS in turn measuring and storing integrity metrics of additionally loaded OS components before their execution. If support by the OS is provided, applications can also be measured before being executed.

The measurement and reporting processes are depicted in a simplified manner in Figure 2, in which $\mathcal{H}$ represents the cryptographic hash function SHA-1. During initialization, various PCRs as well as a configuration log file (stored on the platform) are initialized; this log file keeps track of additional information such as descriptions or file paths of loaded components [7]; its integrity need not be explicitly protected by the TPM. During subsequent measurement of components, this log file is extended, while metrics (hash values) of the executables are stored in the TPM using the tpm_extend method replacing the contents of the appropriate PCR register with the hash of the old contents and the new metrics. We do not discuss which metrics are stored in which PCR; it suffices to say that metrics of loaded components are reliably stored in the TPM.

When a remote verifier wants to assess the security of the verified platform, she sends a challenge c to the platform. The platform uses this challenge to query (with a tpm_quote command) the TPM for the value of the PCRs. The TPM responds with a signed message $\mathrm{sign}_{AIK}(\overrightarrow{PCR}, c)$ containing the PCR values and the challenge[3]. The platform returns this signed quote to the challenger (verifier) together with information

[3]The TCG specifications specify a quote over specific PCR values rather than the full set; for simplicity, we assume in this discussion that a quote is always given over all PCR values.

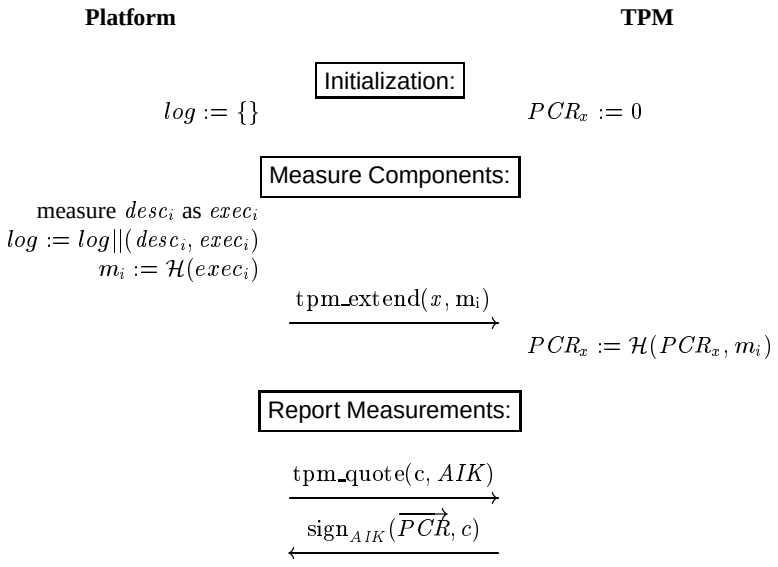

Figure 2: TPM-Enabled Measurement and Reporting Process

from the log file needed by the verifier to reconstruct the verified platform's configuration; the verifier can then decide whether this configuration is acceptable.

The key used for signing the quote, AIK, is an "Attestation Identity Key" of the TPM; as a TPM may have multiple AIK s, the key or its identifier has to be specified in the tpm_quote request. An Attestation Identity Key is bound to a specific TPM; its public part is certified in an Attestation Identity Key Certificate by a *Privacy-CA* as belonging to a valid TPM.[4] The verifier of a quote signed with a (correctly certified) AIK believes that the quote was produced by a valid TPM, more specifically, by the unique TPM owning that AIK. This belief is, of course, based on the assumption that the TPM is not easily subject to hardware attacks and that effective revocation mechanisms are in place dealing with compromised keys.

Note that the above measurement process does not prohibit execution of untrusted code, it only guarantees that the measurement of such code will be securely stored in the TPM. Thus, if malicious code is executed, the integrity of the platform may be destroyed; however, the presence of an untrusted (or simply unknown) component will be reflected by the TPM quotes not matching the 'correct' or 'expected' values.

[4] In the remainder of this paper, we will always assume that the verifier of a quote is in possession of, or can obtain, the appropriate certificate certifying the AIK but we will not explicitly represent the transport of such certificate in protocols.

3.3 Limitations of Binary Attestation

In settings where verified platforms have a relatively stable configuration, binary attestation can be a practical and efficient way of verifying the integrity of platforms. An example may be the management of an enterprise network where managed devices (should) have the same or one of only a limited number of configurations. In such a setting, there may also be no privacy or openness problems related to platforms reporting to their full configuration.

In less closed environments, however, binary attestation suffers from limitations of scalability, privacy and openness that can be resolved by property attestation:

Lack of scalability Binary attestation requires the verifier to know all potential hash-values of all (combinations of all) components of any machine that it may be required to verify. Knowing all acceptable configurations is be hard to manage.

Privacy invasiveness The TPM defines protocols called Direct Anonymous Attestation (DAA) to anonymously prove the presence of a valid TPM. Binary attestation limits this anonymity by revealing the configuration information of the platform to the verifier. This is violates best privacy practices by revealing irrelevant (and likely personally-identifying) information.

Lack of openness A verifier expects a particular hash-value of a particular implementation. This fixes the implementation instead of the security properties: The usual verifier will expect a particular implementation from a particular vendor. This will put other implementations from other vendors at a disadvantage even if these implementations provide the same or even a higher level of security.

The ability to attest to a platform's properties rather than metrics can greatly improve scalability of the attestation approach: a verifier need not know every possible acceptable configuration; she rather wants to be assured that the platform will enforce certain security properties (*e.g.*, adherence to privacy policies). At the same time, property attestation protects the privacy of the verified platform's owner and encourages openness and interoperability by focusing on security properties rather than on exact implementations.

4 Property Attestation

Property attestation addresses the privacy, openness and scalability problems associated with binary attestation. With property attestation, a verifier is convinced of high-level security properties of a remote platform without receiving the remote platform's configuration information. Examples of security properties are the absence of certain vulnerabilities or the ability to enforce certain policies; security properties also include privacy and availability statements. The SuSe `www.suse.com` common criteria evaluated Linux enterprise edition can prove that it satisfies Assurance Level EAL2+ for the Controlled Access Protection Profile. A server farm should be able to assure a verifier that it has a high-probability of 24x7 availability. A enterprise can certify that a given set of files belong to it's base installation (while others do not).

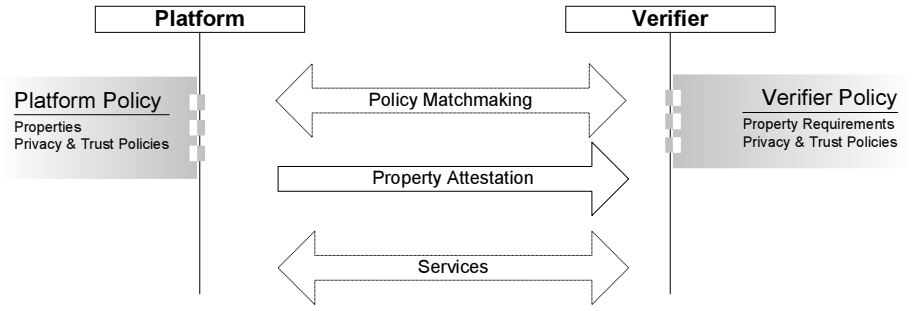

Figure 3: High-Level View of Property Attestation

4.1 High-level View of Property Attestation

Figure 3 depicts property attestation at a high level. The verifier and verified platform engage in a protocol to prove that the platform satisfies the verifier's security requirements. If the verifier is satisfied with the offered properties, they can engage in the exchange of services.

The actual properties offered are determined by a matchmaking process between a *verifier policy* and a *platform policy*. The term *policy* stands for a collection of static and/or dynamic security and privacy requirements, trust assumptions and properties:

- The *verifier policy* includes the verifier's property requirements as well as the trust policy describing which entities she trusts for signing or certifying certain property-related statements. A verifier may trust a software distributor to state correct product information in a component directory (*e.g.*, which binaries belong to which product), but may not trust the distributor for certifying security properties about the software. In our architecture in Section 4.2, security property certification will be performed by *property certifiers*. In addition to having a trust policy, a verifier may have a privacy policy specifying, for example, to whom she wants to disclose security requirements and trust policy contents.

- The *platform policy* includes the properties that can currently be assured by the platform, as well as privacy and trust policies specifying which information (properties or configuration) can be disclosed to whom.

The matchmaking and negotiation process between verifier and platform policies can differ depending on the entities involved. The question is what portions of the local policies are communicated? Individuals are often reluctant to reveal their privacy policies while enterprises are often reluctant to reveal their trust policies and property requirements. We envision the following scenarios:

B2C If a business platform wants to prove its properties to a consumer, it will reveal what properties can be offered under what trust policy. The consumer then locally decides whether this satisfies his requirements. The consumer does not reveal any information.

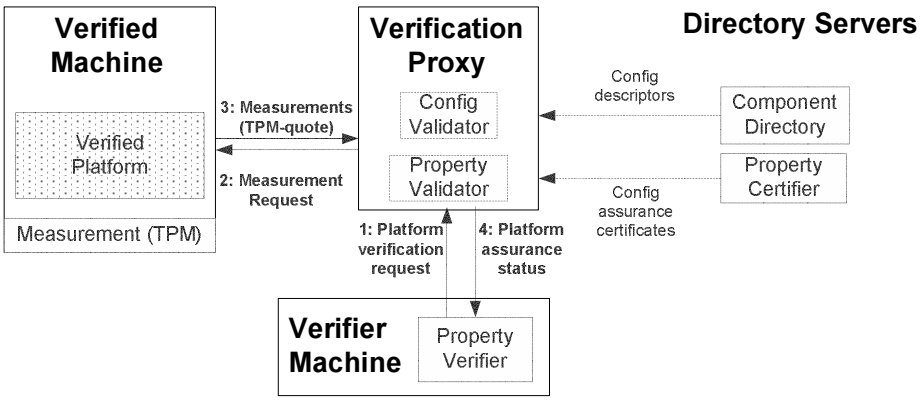

Figure 4: Property Attestation Architecture: Actors and Basic Message Flows

C2B If a business verifies a consumer platform, it will send its trust policy while the consumer platform then proves the corresponding properties.

B2B If a business verifier verifies a business platform, it will reveal its trust policy while the business platform responds with the properties that can be guaranteed under this trust policy.

In practice, proving properties and revealing (parts of) policies can be an iterative process where parties gradually build up trust as in [8, 9, 16, 17].

In the following sections, we will focus on the C2B scenario with simple privacy policies: the verified platform's privacy policy simply forbids that the verifier receives actual PCR measurements; and the verifier is willing to reveal her trust policy but not her specific security requirements. The verifier is thus willing to send her trust policy and the verified platform attests to the properties it can assure under that trust policy. These assumptions will allow us to illustrate the core concepts of property attestation; a description of a generic matchmaking process for complex privacy and trust policies is not within the scope of this paper. An example of a more complex platform privacy policy may be not to attest to any property if the verifier's trust policy is too restrictive; *e.g.*, if a verifier's trust policy specifies trust in only a single software vendor, then attesting to a high-level property under that restrictive trust policy reveals that the verified platform is running only software from that vendor.

4.2 Property Attestation Architecture

Figure 4 shows the component architecture of a property attestation system. New parts of the property attestation architecture are:

Property Certifier An agent that describes (and certifies) which security properties are associated with which component. Example include manufacturers that certify properties of their products (such as offering certain services), evaluation

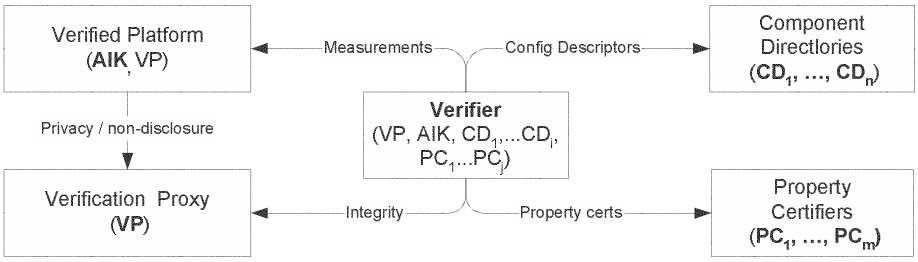

Figure 5: Trust Model for Property Attestation: Entities and Keys (bold identifiers denote key-pairs)

authorities that certify their evaluation results (such as common criteria assurance level for a given protection profile), or enterprises or other owners of the machines that self-certify the code that they deem acceptable.

Verification Proxy Towards the verified platform, the verification proxy acts as a verifier of binary attestations; towards the verifier, it acts as the verified platform in the high-level property attestation view of Figure 3. When receiving a platform verification request by the verifier, it challenges the verified machine for integrity measurements. These measurements are then transformed into a platform configuration through configuration validation, and subsequently into platform properties through property validation. The property validation is based on property certificates (binding components and configurations to properties) issued by property certifiers.

Property Verifier This module engages with the property prover in the property attestation exchange. Its requirements are based on the verifier policy (property requirements and trust policy) that it requires as an input.

4.3 Property Attestation Trust Model

We outline certain deployment-dependent security assumptions that are made by our design. In section 6 we show how to guarantee that they are satisfied.

The verification proxy is a core component of the design. The verified platform (or its user/owner) needs to trust in its integrity (correct operation and authenticated channel) and confidentiality (confidential channel and no information leakage) in order to guarantee privacy. The verifier needs to trust in the integrity of the verification proxy in order to believe the properties that the verification proxy outputs. In addition, the verifier needs to know a verification proxy signature key (public/private key pair) that are used by the verification proxy to authenticate its verification results.

Figure 5 depicts the trust model for property attestation. Each entity is shown together with the public signature verification keys that it needs to know. Bold identifiers represent key-pairs of the entity. The arrows in the figure represent trust relations between entities (or, in fact, trust policies associated with public keys): The *Verified Platform* owns an attestation identity key AIK and knows the verification proxy's (public)

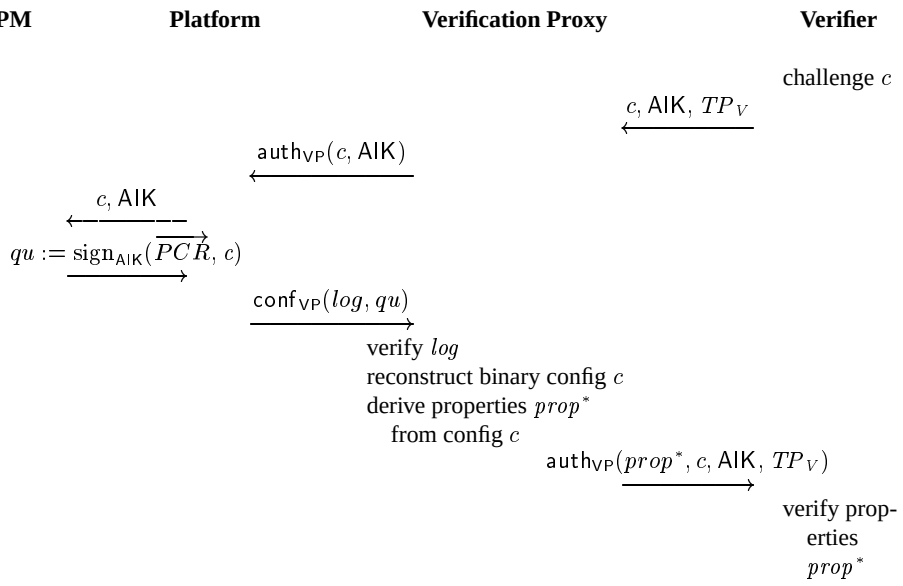

Figure 6: Property Attestation Protocol

key VP. It trusts the owner of VP to protect the confidentiality of its measurements. In the simplified privacy policy model discussed in § 4.1, the verification proxy is thus the single entity to which the verified platform wants to send configuration information. The *Verification Proxy* owns its signature key-pair VP. Each *Component Directory* i owns a key-pair CD_i with which it certifies configuration descriptors. Each *Property Certifier* i owns a key-pair PC_i with which it certifies properties related to (sets of) components. The *Verifier* knows knows the platform identity (public) key AIK of the platform about which it wants to receive property attestation; it trusts that measurements authenticated with that key correctly represent the configuration of the platform based on the TPM certified with AIK (even though he does not see them). The verifier also knows VP and trusts the integrity of property attestations with that key. The verifier trusts configuration descriptions authenticated with $CD_{1\ldots i}$ and property certificates authenticated with $PC_{1\ldots j}$.

5 Property Attestation Protocols

We now describe the protocol for property attestation based on the above trust model; it is represented in Figure 6. The exchange is triggered by the verifier who requests to receive property attestation about the platform associated with AIK. We name the protocol steps corresponding to the names of basic message flows and components in Figure 4.

Platform Verification Request The verifier sends a message to the verification proxy which contains a randomly generated 160-bit challenge (nonce) c, the attestation identity key AIK about which she wants property attestation, and her trust policy TP_V. As mentioned in § 4.1, we assume that the verifier does not protect the privacy of her trust policy; we also assume that the verifier receives all the properties the verified platform can guarantee under this trust policy.

Measurement Request Using an authenticated channel, the verification proxy forwards challenge and AIK to the verified platform. The platform decides whether or not to continue based on its policy and trust model. We assume the platform knows VP as the key of a trusted verification proxy and continues by requesting a TPM quote. Note that the challenge used between verification proxy and platform (and TPM) need not be the same as the challenge used between verification proxy and verifier. Indeed, it is up to the verification proxy alone to judge the correctness and freshness of the actual TPM quote.

TPM Quote Request/Response The platform requests and receives the AIK-authenticated quote using the challenge.

Measurements The platform sends the quote and the log-file to the verification proxy using a confidential channel (described below).

Config Validation The verification proxy can now reconstruct the platform's configuration using the authenticated metrics (PCR quote), the log file and (potentially) config descriptors certified by keys within TP_V.

Property Validation The verification proxy derives properties of the platform's components based on property certificates certified by keys within TP_V.

Platform Property Status The verification proxy returns an authenticated message containing the Platform Verification Request and the properties that can be assured. The verifier checks whether this response is authenticated with a key which her policy considers to belong to a trusted verification proxy. If so, she trusts that the properties returned can currently be guaranteed by the platform associated with AIK under TP_V.

Note that the protocol assumes that the security of the verification proxy is guaranteed. In addition, we assume that messages from the verification proxy to the platform and the verifier are authenticated while messages from the platform to the verification proxy are kept confidential (denoted by auth and conf, respectively). How this will be guaranteed depends on the deployment and will be described in Section 6.

Note that more complex privacy policies (e.g., the verified platform also protecting which properties can be proved to which verifiers under which trust policy) may require also authentication by the verifier of the initial request message, as well as confidentiality protection of the verification proxy's response to the verifier.

We assume that high-level security properties about a platform can be guaranteed only if all components on the platform are measured; this assumes that the measurement process as depicted in Figure 2 continues up to the application level. Thus the

verification proxy should not attest to any properties unless it can convince itself that the verified platform's configuration indeed supports that extended measurement.

6 Deployment Scenarios

In previous sections, we assumed the existence of a key pair VP used by the verification proxy for authenticating messages as well as the establishment of a confidentiality-protected channel with the verified platform. Verified platform as well as verifier were assumed to trust this key to belong to an untampered and correct verification proxy.

In this section, we now outline different deployment scenarios achieving the above goals. Each scenario enables the verification proxy to establish an authentic channel and to communicate confidentially with the verified platform and provides guarantees to the verifier and the owner of the verified platform that the verification proxy is untampered.

6.1 Verification Proxy on a Dedicated Machine

The verification proxy can be deployed on a dedicated TPM-enabled machine and convince other parties (verifier and verified platform) of its own integrity through binary attestation.

If we assume that there are only a few approved standard configurations of verifier proxy platforms, we can expect the 'verification proxy verifier' (the platform or the verifier in the property attestation) to know a set of acceptable verification proxy configurations, say $\{\overrightarrow{PCR}_1, \ldots, \overrightarrow{PCR}_n\}$. The verification proxy can now prove its trustworthiness with a TPM quote (using a validly certified AIK) attesting to such an acceptable configuration:

$$\mathrm{sign}_{AIK_{VP}}(\overrightarrow{PCR}, c)$$

The key used for authentication and key distribution in property attestation protocols can then either be AIK_{VP} or another key protected by the TPM and which can be shown to be associated with AIK_{VP}.

For efficiency reasons, it is recommended that the dedicated machine also stores recent copies of the directories with certified material (property certificates and component certificates).

A special case of this deployment is a verification proxy owned by the owner of the verified platform itself. E.g., the platform owner is a company, verified platforms are employee machines, and the verification proxy is the company's firewall hiding details of employee machines' configuration towards company-external property attestation verifiers.

6.2 Self-Attestation: Verification Proxy on the Verified Platform

The idea here is to deploy the verification proxy on the verified platform itself (see Figure 7). This effectively implements a self-verification of the platform. Recent microkernels allow to execute multiple operating system instances on a single machine

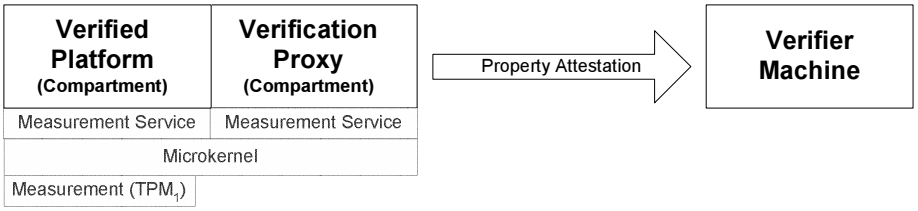

Figure 7: Deployment of Property Attestation on a Microkernel-Enabled Platform

[5, 6]. On such platforms, the TPM can be virtualized such that each compartment has its own virtual TPM [4]. This mechanism can be used to execute two TPM-enabled machines on the same piece of hardware. The deployment is then essentially identical to the two-machine deployment. Since the microkernel usually provides services for secure messaging, authentication and encryption is not needed for messages between the verification proxy and the platform.

As described in the two-machine case, the verifier is required to verify the integrity of the verification proxy using binary attestation. In this case, the scope of this verification would be the compartment that executes the verification proxy, while the configuration of the compartment executing the platform is not disclosed. This verification would be based on the services provided by the virtual TPM in the compartment where the verification proxy is executed.

7 Conclusion and Open Problems

We have shown how property attestation can resolve the scalability, privacy and openness issues raised by straightforward binary attestation using TPM hardware. Unlike this binary attestation, property attestation provides an open method to guarantee integrity (or trustworthiness) to a verifier, while enabling many secure implementations.

An open problem is the exact negotiation of the security guarantees that a platform needs to offer. Verification of whether a machine meets certain security requirements under a given trust policy can have have different flavors: The first is that the verifier discloses nothing and asks the prover machine to explain what properties can be guaranteed under what trust policy. The second is that the verifier sends her trust policy (who is trusted to certify what) to the remote platform and the remote platform then self-verifies and returns the security properties that can be guaranteed under this trust policy. The third alternative is to send the security requirements including the trust policy to the remote platform. The remote platform self-verifies and answers whether it can or cannot meet these security requirements.

References

[1] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proc. 1997 IEEE Symposium on Security and Privacy*, pages

65–71. IEEE Computer Society Press, 1997.

[2] B. Balacheff, L. Chen, S. Pearson, and G. Proudler. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall International, 2003.

[3] J. G. Dyer, M. Lindemann, R. Sailer, L. Van Doorn, S. W. Smith, and S. Weingart. Building the IBM 4758 secure coprocessor. *IEEE Computer*, 34(10):57–66, 2001.

[4] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *ACM Symposium on Operating Systems Principles (ASOSP)*, pages 193–206. ACM Press, 2003.

[5] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter. Performance of μ-kernel-based systems. In *16th ACM Symposium on Operating System Principles (SOSP)*, St. Malo, France, October 1997. ACM Press.

[6] B. Pfitzmann, J. Riordan, C. Stble, M. Waidner, and A. Weber. The perseus system architecture. Technical Report RZ 3335 (#93381) 04/09/01, IBM Research Division, 2001.

[7] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. Technical Report RC23064, IBM Research Division, Jan. 2004.

[8] K. Seamons, M. Winslett, and T.Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Proc. 2001 Symposium on Network and Distributed System Security*, San Diego, CA, Apr. 2001. Internet Society.

[9] K. E. Seamons, M. Winslett, T. Yu, L. Yu, and R. Jarvis. Protecting privacy during on-line trust negotiation. In *Worhshop on Privacy-engancing Technologies (PET2002)*, San Francisco, CA, April 2002. Springer LNCS.

[10] S. W. Smith. Outbound authentication for programmable secure coprocessors. In *Proc. 2002 European Symposium on Research in Computer Security (ESORICS)*, pages 72–89, 2002.

[11] The Trusted Computing Group. Main specification version 1.1b, 2003. Available from http://www.trustedcomputinggroup.org.

[12] The Trusted Computing Group. TPM main version 1.2 part 1 design principles, 2003. Available from http://www.trustedcomputinggroup.org.

[13] The Trusted Computing Group. TPM version 1.2 specification changes, Oct. 2003. Available from http://www.trustedcomputinggroup.org.

[14] The Trusted Computing Group. Writing TCG enabled trusted applications, 2003. Available from http://www.trustedcomputinggroup.org.

[15] B. Yee. Using secure coprocessors. Technical Report CMU-CS-94-149, Carnegie Mellon University School of Computer Science, May 1994.

[16] T. Yu, X. Ma, and M. Winslett. PRUNES: an efficient and complete strategy for automated trust negotiation over the internet. In *Proc. 2000 ACM Conference on Computer and Communications Security*, pages 210–219. ACM Press, 2000.

[17] T. Yu, M. Winslett, and K. E. Seamons. Interoperable strategies in automated trust negotiation. In *Proc. 2001 ACM Conference on Computer and Communications Security*, pages 146–155, 2001.