Attestation in Trusted Computing: Challenges and Potential Solutions

Andrew Lee-Thorp

Technical Report
RHUL–MA–2010–09
31st March 2010

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England

http://www.rhul.ac.uk/mathematics/techreports

Attestation in Trusted Computing: Challenges and Potential Solutions

Andrew Lee-Thorp

March 25, 2010

Acknowledgements

I would like to thank Professor Kenny Paterson and Shane Balfe, both of the Information Security Group at Royal Holloway, University of London. Shane Balfe suggested the topic and Professor Paterson steered me along the path to completing this report. I would also like to thank my wife, Julia for looking after our family of two boys, Matthew and Michael, for the entire duration of this course.

Abstract

This report examines the state of play in TCG attestation. It asks the question: how practical is the attestation specification and does it meet the needs of designs that propose to take advantage of trusted computing functionality? It is shown that, broadly speaking, both specification and implementation falls short of its stated goals. Application designs expect different semantics. Straightforward application of attestation to a running system does not provide adequate assurance nor does it scale. It is argued that extending the TCG architecture and reworking application designs are the most viable routes to making attestation a practical proposition.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Trusted computing is designed to protect data from software attack [66] and a modicum of hardware attack. Trusted platforms are a response to today's infrastructure needs and the challenges facing existing security solutions[5]: secure computers have no way of proving their integrity; secure computing solutions tend to be expensive; security equipment tends to fall foul of government export restrictions on cryptography.

The specification set defined by the Trusted Computing Group aims to address these inherent weaknesses whilst retaining some of the benefits of commodity computing platforms: low cost, flexibility and openness. Fundamental to trusted computing is a low-cost, tamper-evident computing chip which is intended to be mounted on a computer motherboard.

Moreover, trusted platforms sport a distinguishing feature: *trusted platforms can vouch for their own integrity.* This is significant when one considers the Trusted Computing Group (TCG) definition of trust:

> **Trust** is the expectation that a device will behave in a particular manner for a specific purpose.

How do Trusted Platforms support Trust? Trusted platforms collect and provide evidence of behaviour. Endorsements of the trusted platform by third parties allows that evidence to be trusted. A core capability of a Trusted Platform therefore, is to directly support trust mechanisms in secure hardware. These capabilities are named *roots of trust*: the hardware is *trusted* because it must be [5] - all functions derive their security from this assumption and this trust is *rooted* because it is the basis for all further reasoning about a platform's security. There are three roots of trust: root of trust for measurement (RTM), root of trust for storage (RTS) and root of trust for reporting (RTR). These must resist software attack and some forms of physical attack[5].

Balacheff et al.[5] envisage three generations of trusted platforms. In the short-term, the main benefit will be due to protected storage functionality in which the Trusted Platform Module (TPM) acts as a "portal to encrypted data" [5]. The medium-term generation of platforms will leverage the measurement of integrity metrics which represent the software state of the platform. This adds a new dimension of access control to protected storage [66]. The longer-term vision

is of more pervasive trusted inter-connectivity in which for example, organisations can expose their computer systems and remain confident that data is used only in the way that is intended.

The longer-term is enabled by a mechanism known as reporting (of integrity metrics) or simply **(remote) attestation**. In essence it allows a third party to obtain confidence in the identity and trustworthiness of a target before it interacts with the target.

Investigators claim numerous applications for trusted computing, from the enhancement of digital signatures, security in P2P networks, supporting single sign-on as well as futuristic applications such as distributed trusted third party (TTP) services and more[60]. The extension of attestation to the semantically rich domain of user-land applications presents a major obstacle to the widespread adoption of trusted computing[6].

In its older cousin, sometimes referred to as outbound authentication [79], the trust chain is implemented as a certificate chain that terminates at a code entity. In TCG, the trust chain carries a very different semantic. Each component extends the trust chain by **measuring** software in the next layer of the platform stack. The subtle difference therefore is that the TCG trust chain does not identify a code entity but a measured platform configuration. This means that secrets are not associated with entities but with platform configuration.

It is not surprising therefore that the subject of practical attestation has proven to be an active area of research. Extending TCG attestation concepts to user-land applications remains problematic and is usually based on assumptions about the requirements. Moreover, there appears to be a dearth of literature on how integrity information can be verified practically.

This report examines the state of play in TCG attestation. It asks the question: how practical is the attestation specification and does it meet the needs of designs that propose to take advantage of trusted computing functionality? It is shown that attestation falls short of its stated goals. Application designs tend to rely on owner semantics and open, heterogeneous environments present a significant challenge to the large scale adoption of trusted computing. The models and implementation of real (and proposed) attestation systems are contrasted. Extending standard measurement semantics to a running system does not adequately capture platform state nor does it scale. Attempts have been made to address some of the deficiencies of attestation but these have promised much and delivered little. It is speculated that (i) the TCG architecture needs to be expanded to provide greater assurance of system integrity and (ii) that designers need to rework the way that applications are structured to minimise the amount of code that needs to be trusted.

The report is structured as follows. Chapter 2 gives a broad overview of trusted computing features including the newer features specified in version 1.2. In chapter 3 TCG attestation is described in detail, including its underlying trust model and inherent limitations. Chapter 4 introduces and uses a simple threat model to investigate applications that use attestation. Chapter 5 discusses a reference model for reasoning about attestation and discusses four broad attestation architectures. Chapter 6 describes a number of attestation schemes and chapter 7 concludes.

Chapter 2

Trusted computing: an overview

Trusted Computing refers to a platform of the type specified by the Trusted Computing Group (TCG)[1] as well as the next generation of hardware [43, 81, 4] and operating system [63, 49, 9] designed to provide trusted features and hardware-enforced isolation. A **trusted platform** (TP) is a platform that has a trusted component, probably in the form of built-in hardware [5]. The TCG specification set covers mobile phone [39], server [33], PC [32], peripheral and trusted network components [40] but it is the three elements: the TPM [36], RTM [38] and TSS [35] that constitute the core of a trusted platform.

Trusted Platform Module (TPM) The TPM is a low-cost hardware chip, designed to provide resistance to software attacks and a modicum of hardware attacks. The TPM is the hardware *root of trust* in a Trusted Platform. The specifications for a TPM include: secure volatile and non-volatile memory, a computing engine, input and output components, asymmetric key generation, encryption/decryption, digital signature operations, (internal) symmetric encryption engine, random number generation, and a SHA-1 engine. The chip, however, is not considered to be a TPM until an **endorsement key** has been installed and an endorsement certificate has been created [5].

The TPM is physically or cryptographically bound to the RTM. On a PC, the TPM is likely to be soldered onto the motherboard in a tamper-evident way. The properties and functions of the TPM are described in more detail in the following section.

Root of Trust for Measurement (RTM) The RTM is a computing engine which can measure at least one integrity metric, record the integrity metric to a Platform Configuration Register (PCR), and write the measured value to the Storage Measurement Log (SML). On a PC, the RTM is a part of the platform and is initiated by instructions in the Bios Boot Block (BBB) known as the Core Root of Trust for Measurement (CRTM). The CRTM is assumed to be immutable and it is trusted to execute only those programs that it is intended to execute (and vouched for by the Platform Entity).

[1] www.trustedcomputinggroup.org

TCG Software Stack (TSS) The TCG software stack [37] is platform support software for interfacing the TPM. It includes a device driver, core services, and a common trusted platform services. The TSS does not need to be trusted, but a challenger should examine the TSS integrity metrics to determine whether the platform can be trusted.

2.1 TPM specifications

This section describes the features of a version 1.2 capable TPM.

2.1.1 Protected capabilities and shielded locations

TCG defines, in abstract terms, locations in the TPM that must be free from interference or prying (shielded locations) and functional capabilities whose operation must be trusted (protected capabilities). Only protected capabilities have access to shielded locations. Shielded locations are intended to store secret or sensitive data. Of particular importance to attestation is a set of Platform Configuration Registers (PCRs) which are designated as shielded and a protected capability on a PCR called *extend*. Each PCR has a 20-byte length, equal to the length of a SHA-1 digest. A TPM must support at least 16 PCRs, the first 8 of which have their purpose specified by TCG. Their purpose is to store evidence of platform behaviour which can later be presented in a platform attestation (Chapter 3).

2.1.2 Entities and credentials

In order to establish trust in a Trusted Platform, TCG defines a number of entities that are used to build trust in a platform via the issuance of credentials designed to vouch for a particular aspect of each Trusted Platform.

The **Trusted Platform Module Entity (TPME)**, usually the manufacture or OEM, is an entity that vouches, in an **endorsement credential**, for a (collection of capabilities in a) TPM instance. An endorsement credential is a digitally signed attestation of the binding between a Trusted Platform instance, its public endorsement key (EK) from its endorsement key pair, TPM type information, TPM security properties and a reference to the TPME.

The **Conformance Entity (CE)** vouches, in a **conformance credential**, that a particular design and implementation of a TPM and its trusted building blocks [38] , meets the TCG specifications. A conformance certificate is issued by the CE (typically an evaluation facility) if the component manufacturer's security target satisfies a particular TCG protection profile [5].

The **Platform Entity (PE)** vouches, in a **platform credential**, that a TPM has been correctly incorporated into a Trusted Platform and that this instance is accurately described by its purported conformance credentials. In particular it guarantees the immutability of the CRTM and the secure coupling of the CRTM and TPM.

The **Validation Entity (VE)** vouches for expected platform state, i.e. the correspondence between integrity measurements and correctly functioning trustworthy components.

A **Privacy-CA (P-CA)** is a certification authority that vouches that a given platform is a Trusted Platform possessing the claimed properties. Through the issuance of an **Attestation Identity Credential** it provides an assurance that an identity name and **Attestation Identity Key** belong to a genuine Trusted Platform

A **DAA Issuer** is a TTP that fulfils a similar function to a P-CA through the issuance of **DAA Credentials** in a manner that does not allow the EK and AIK to be linked. **DAA Verifiers** consume DAA credentials.

2.1.3 Keys and key management

Authorisation and mobility attributes serve to constrain access to keys but not to control how they are used[5]. Key mobility falls into one of the following categories:

- *Migratable* keys are trusted only by their originator and can be moved to another platform.
- *Certified Migratable* keys (CMKs) are migratable keys having properties that the TPM can certify. The key creator designates a Migration Authority or a Migration Selection Authority, to have the authority to migrate the CMK. Provided the MA (or MSA) can be trusted then this allows CMKs to be migratable but remain secure.
- *Non-migratable* keys are known to, and used exclusively by, the TPM that owns them[2] Non-migratable keys are trusted by everyone as everyone can be sure that (i) they are created within a TPM, (ii) they never appear outside a TPM and (iii) that all operations using the private key are performed within the TPM.

Amongst the key classes[15] (storage, signature, identity, binding, legacy) TCG defines two distinguished keys to have a well-defined purpose: the endorsement key and the storage root key. Each TPM has exactly one, statistically unique, asymmetric key pair : a platform **Endorsement Key (EK)** pair that is installed by the TPME. The private EK is used for decryption only and is securely held in a TPM shielded location. It is never revealed outside of, nor is it exported from, the TPM.

The first key to be created when an entity assumes ownership of a TPM is the **Storage Root Key (SRK)**. The SRK is a non-migratable key, permanently loaded in the TPM, that is seated at the root of a key hierarchy. In this hierarchy each parent node is used to protect the next layer of child objects. Intermediate (i.e. non-leaf) nodes are asymmetric storage key pairs where the public and private key is used to respectively wrap (encrypt) and unwrap (decrypt) child objects. A parent key must be "loaded" into the TPM to unwrap a child object. In this way the key hierarchy may need to be traversed by successively loading a parent key to unwrap the next child object, starting with the SRK and ending with the target object. Therefore the time to access a protected object depends on its position in the tree relative to the root. To

[2]This is usually the TPM in which the non-migratable key was created but TCG defines a *maintenance* mechanism for moving an entire protected storage tree, including non-migratable keys, to another TPM.

mitigate the delay in accessing a node, keys can be swapped out to non-TPM storage and later swapped in using key session caching. Keys are cached outside of the TPM as an opaque blob, encrypted using a temporary key know only to the TPM and valid only for the current *caching session*.

An **attestation identity key** (AIK) or simply an identity serves to identify a Trusted Platform as genuine. In the context of attestation an AIK is used for the following:

1. Sign the current PCR values with the private AIK using a TPM operation known as *quote*.
2. Sign the the public part of a signature key pair with the private AIK which is then used to sign the current PCR values [5].

2.1.4 Identities

TCG makes a distinction between proving that a platform is a *genuine* Trusted Platform and proving that a platform is a *particular* Trusted Platform instance[5]. In the latter case, the endorsement mechanism associates a unique secret with a single Trusted Platform instance. In the former case, TCG specifies two mechanisms for obtaining an **identity credential** for a **TPM identity**, also known as an **attestation identity**. Each Trusted Platform has only one TPM and this binding is attested in a platform credential. It follows that the identity of the TPM is also the identity of the Trusted Platform[5].

In an abstract sense, an attestation identity is simply a chosen Trusted Platform name plus an identity key. An identity credential is a statement by a TTP (P-CA or DAA Issuer) that an entity with the chosen name and knowledge of how to correctly use an identity secret, has previously proven to the TTP that it is a Trusted Platform. Whenever the platform wants to use its identity to present data to a third party, it supplies the data signed by its identity key (or by a non-migratable signature key signed by the private identity key) plus the identity credential.

TCG defines two mechanisms by which a Trusted Platform may obtain a virtually unlimited number of identities.

Privacy CA In this approach a Privacy Certificate Authority attests to the binding between a chosen identity name, public AIK and the Trusted Platform instance.

In summary, the TPM is asked to create a new identity, which it returns as a self-signed (using the newly generated signing AIK) *identity binding*. A platform agent encrypts the signed identity binding and credentials (endorsement, platform, conformance) using the Privacy CA public key. The Privacy CA examines the evidence generated by the Trusted Platform plus the associated credentials. If it is satisfied as to the validity of the TPM, and agrees to vouch for the trusted platform description, it creates an (attestation) *identity certificate*. Then the identity certificate is encrypted with a symmetric key and the symmetric key plus a digest of the public AIK is encrypted with the platform's public endorsement key. The TPM decrypts

the symmetric key and public AIK digest using the private endorsement key. If the decrypted digest matches the digest of the requested identity then it releases the symmetric key to the untrusted platform software which can then recover the identity certificate.

To attest integrity metrics, the platform supplies a signature generated by the signing AIK plus the identity certificate which vouches for the public AIK.

Direct Anonymous Attestation Direct Anonymous Attestation is a group signature scheme with the property that the group manager, i.e. the DAA Issuer, cannot open the group signature and identify the signer [14]. An anonymous revocation method is provided using a feature dubbed *user-controlled link-ability* that enables the signer to create self-certified anonymous revocation evidence that is usable only by the target verifier.

In the DAA *Join* phase a DAA Issuer issues credentials in the form of a Camenisch-Lysyanskaya (CL) signature on a non-migratable, unique TPM secret, f, without learning the value of the secret. In the DAA *Sign* phase the DAA credentials and the TPM secret are used to produce a signature proof of knowledge on the public AIK.

2.1.5 Integrity measurement and storage

Trusted Platforms can vouch for their own integrity. This requires one of several adaptations to existing platform architecture beginning with the **Root of Trust for Measurement (RTM)** and several **measurement agents**. The RTM is a computing engine entrusted with reliably measuring integrity events, i.e. events that alter the trusted state of a platform. The simplest interpretation of an integrity altering event is that software is about to loaded and executed. The basic idea is to instrument each layer of platform software with a measurement agent. Each measurement agent, starting with the CRTM[3], measures the local platform environment and the next measurement agent. It records these measurements and then hands control to the next measurement agent. The chain of measure-record-handoff terminates with the final measurement agent which is part of the software that is loaded and running once the Trusted Platform has booted (typically the OS). The final measurement agent continues to record integrity altering events. This process is also referred to as *authenticated boot*.

Each integrity measurement is recorded in a two stage process that must be atomic.

1. The value of the measured integrity event and the PCR index is stored outside the TPM in the **Storage Measurement Log (SML)** which has append-only semantics.

2. A digest of the measured integrity event value is stored in the TPM under the jurisdiction of the **Root of Trust for Storage (RTS)**, using an exposed operation know as *extend*: $PCR_i \leftarrow SHA1(PCR_i||measurement\_digest)$

Amongst the uses for PCRs are the following:

[3]The CRTM is not required to measure itself.

1. **Integrity reporting** - the reporting of evidence of platform behaviour.

2. **Sealing** - as a form of access control for *protected storage* wherein the TPM will reveal a secret only when the platform is in an acceptable state.

3. **Wrapping** - a TSS function, in which for example an externally created key is encrypted (wrapped) under a TPM parent bind key. Release of the key is contingent upon the platform being in an acceptable state.

4. **Secure Boot** - in which, at each stage of the boot process the next stage is measured and matched against a set of known good measurements or otherwise aborted. Secure boot [47, 3] is not defined by TCG although PCRs could be used for this purpose.

For the reasons outlined below the scheme described so far is known as a Static Root of Trust for Measurement (SRTM). The assumption underlying SRTM is that there is a single root of trust which becomes active when the platform is initialised. The S-CRTM must be immutable since the trust in all measurements depends on the integrity of this component [32]. SRTM has some weaknesses (Section 3.6). Next-generation hardware platforms support a dynamic secure partition reset [81] that sets the hardware into a known state such that future behaviour of a platform cannot be affected by its previous history. TCG introduced the concept of a *locality* in version 1.2 of the specifications to facilitate the Dynamic Root of Trust for Measurement (DRTM) method.

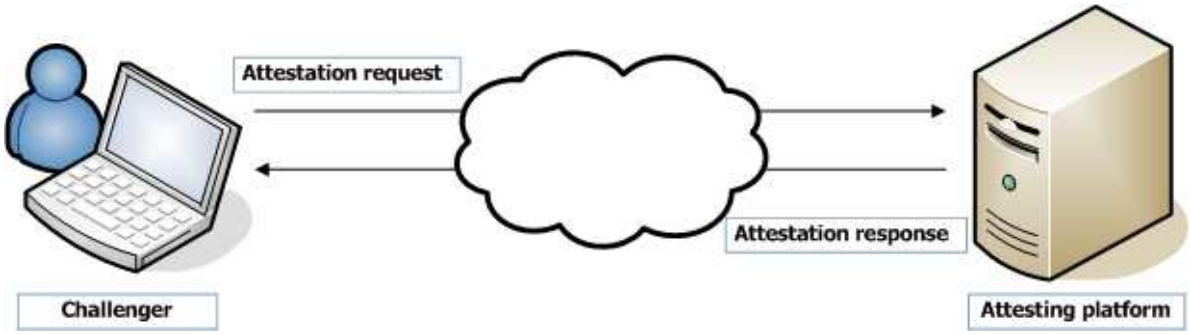
A *trusted process* can assert a locality in way that is not spoofable since this requires hardware support. In the context of DRTM for example, trusted platform software causes the locality 4 PCRs (and localities 2 and 3) to be reset using the `TPM_PCR_Reset` command before sending the **Secure Loader (SL)** to the TPM for measurement which in turn is used to *extend* the default PCR value for locality 4. Using the DRTM method, separate execution partitions may be cleared down, instantiated and measured (in a trusted manner), without affecting future instantiations of separate execution environments.

2.1.6 Integrity reporting and attestation

The third and last platform adaptation is to embed a **Root of Trust for Reporting (RTR)** - the ability to present evidence of (i) behaviour and (ii) authenticity (as a trusted platform) to a remote party.

In Figure 2.1 the remote party presents a nonce and a selection of PCR indices which represent a portion of the target platform configuration. The TPM signs the nonce and the selected PCR values using the private AIK (or signing key signed by the private AIK) and a platform agent returns the signature, the event history for the selected PCRs and the platform credentials. The remote party verifies the signature and validates the event history. At this point it is able to decide whether to trust the platform.

Figure 2.1: Remote attestation. The individual steps are described in the text.



2.1.7 TCG 1.1 and TCG 1.2

The TPM specification attracted the largest number of additions in moving from 1.1 to 1.2 of the TCG specifications[15]. Some of these have already been discussed (Certified Migratable Key, Direct Anonymous Authentication). A selection of the remaining changes is mentioned here briefly.

Delegation Delegation breaks up the all-or-nothing privileges of the owner by creating a form of secondary authorisation to use either a key or an owner-authorised function. This secondary authorisation is said to be *delegated* to trusted software or a trusted user [15]. A delegated authorisation can be revoked.

Locality Locality is a new TPM authorisation mechanism [15]. For example, a TPM protected object can now be sealed to a locality (in addition to PCRs and authorisation data). There are six locality states[15]: none, 0, 1, 2, 3, and 4, whose usage is similar to protection rings with locality 4 being the most trusted. The TPM maintains a locality state based on the locality modifier asserted in hardware or by a trusted process.

PCRs With the introduction of the Locality concept the meaning and usage of PCRs has been altered. A new structure, `TPM_PCR_ATTRIBUTES`, specifies additional attributes that can be set at manufacture time to allow each PCR to have a different behaviour. The three attributes are:

- `pcrReset` is a flag that allows the PCR to be reset at times other than startup,
- `pcrExtendLocal` controls which localities can extend the PCR,
- `pcrResetLocal` controls which localities can reset the PCR at times other than startup.

For example, a `TPM_PCR_INFO_LONG` structure is introduced to specify additional fields for the locality under which a sealed (or wrapped) object was created (`localityAtCreation`) and may be released (`localityAtRelease`) respectively.

Monotonic Counter The TPM is required to support at least four monotonically incrementing counters[25]. The monotonic counter is necessary for auditing functions[15].

Tick Counter The tick counter is a logical clock with starting value of 0 set every time the TPM is turned on [15]. This is used with a tick session nonce to provide a time-stamping service[25].

Chapter 3

Attestation (in TCG)

This chapter presents the end-end process of TCG attestation. There are three phases in the attestation lifecycle: the first (obtaining an identity) is performed once for each identity the platform assumes; the second (integrity measurement and storage) is an active process in which the platform undertakes to maintain a complete history of relevant state altering events; the third (integrity reporting) is a challenge-response protocol executed each time a third party wants to establish trust in a platform. Following this is a discussion of the semantics and inherent limitations of TCG attestation.

3.1 Obtaining an identity

In version 1.1b of the TCG specifications the *Privacy CA* approach is defined. Version 1.2 added the Direct Anonymous Attestation method. Owner authorisation is required to create a new TPM identity.

3.1.1 Privacy CA

The Privacy CA approach was introduced in Section 2.1.4 as a means of certifying a platform identity using a mutually trusted third party, and is described in more detail in Section A.1. Once the (private) AIK is installed it can be used to sign platform integrity metrics.

3.1.2 DAA

The Privacy-CA approach has some undesirable properties [14, 13]: the issuer needs to be trusted not to correlate a TPM's EK with its AIKs; not to collude with other issuers or verifiers which could reveal a TPM's transactions and the inference of personally identifying information; the issuer is a bottleneck since it is involved in every transaction and must be highly available; the issuer combines the distinct roles of a *verification authority* (that normally operates off-line) with that of a *certificate authority* (that needs to operate securely).

Overall, the DAA mechanism (Section 2.1.4) is quite complex (Section A.2). Recent debate has focussed on whether an issuer and verifier can collude to link DAA identities [70, 56]. One proposed mitigation method involves the introduction of yet another CA [56].

3.1.3 Identity revocation

Owing to the multitude of CAs and series of implicit dependencies, a Trusted Computing PKI (TC-PKI) [6] is likely to require a "combination of organisational, policy-oriented, procedural, and legislative approaches"[6]. For example, the revocation of a single credential can result in the cascading revocation of all dependant TPM credentials in a way that should involve multiple CAs being located and contacted in a timely manner [6].

The ease of detection of a rogue TPM and subsequent remedial action depends on the methods used to provide identities [66]. If the extraction of the EK from a TPM was detected then to prevent the further issuance of AIK credentials the Privacy-CA could use a blacklisting approach or a risk-management policy. One policy example is the enforcement of a threshold on the number of identity requests appearing to originate from the same endorsement credential [13].

3.2 Integrity measurement and storage

Integrity measurement and storage was introduced in the previous chapter.

3.2.1 Transitive trust chain

In summary, for SRTM, a process of measure-record-handoff begins with the S-CRTM, cycles through each subsequent measurement agent in the platform stack and terminates at the final measurement agent that is part of the software that has loaded once the boot is complete. The final measurement agent continues to record events that alter or subvert the so-called trusted computing base[1][5]. In this way a transitive chain of trust is built from the S-CRTM to the operating system.

This is best illustrated with an example. The CRTM is seated in the first instructions of the Bios Boot Block (BBB). The boot process begins with the BBB which measures[2] the integrity of the BIOS and embedded Option ROMs, recording these to the SML and reporting the integrity metric(s) to the TPM in PCR_0. The BBB then hands control to the BIOS which measures the configuration of the motherboard including hardware components and how they are configured, reporting the summary to PCR_1. The BIOS then measures the option ROMs, adds to the measurement history and reports the summary to PCR_2. The BIOS hands control to each option ROM to perform necessary initialisation. Each option ROM reports its configuration data to PCR_3. The BIOS then measures the OS loader, stores the measurement value, reports the summary to PCR_4 before handing control to the OS loader. The OS loader reports its

[1] See Gollmann[30] for a definition of trusted computing base

[2] The specification allows the CRTM to measure itself.

configuration to PCR_5 Similarly, after initialisation, the OS loader measures and stores the OS values. Finally, the OS measures and stores OS components and any additional software that forms part of the Trusted Computing Base (TCB). The final measurement agent remains active, measuring additional software that is loaded onto the platform[5].

Suppose that an adversary replaces the OS with an evil OS. This fact will have been noticed by the OS loader when it measured the OS and hence can be detected when the transitive trust chain is reported to a third party. In order to avoid this the adversary must subvert the OS loader but this fact will have been measured by the BIOS. It follows by induction that the only the CRTM needs to be trusted.

The *extend* operation is defined as $h' \leftarrow SHA1(h_1||h_2)$ where h', h_1 and h_2 are digest-sized quantities for the new, old and measured values respectively. The new value h' represents h_1 but modified in a way that is only possibly by h_2. To defeat this mechanism an adversary would have to defeat the second-preimage resistance property of the underlying hash function or find a suitable h3 such that $SHA1(h_1||h_2) = SHA1(h_1||h_3)$ still holds.

3.2.2 Integrity metrics and integrity measurement

Two types of information are stored [5]: (i) an *integrity measurement* which is the measured value of the event itself and (ii) an *integrity metric* which is the condensed value of the history of integrity measurements. Integrity measurements are recorded to a platform (i.e. off-TPM) measurement store known as the Storage Measurement Log (SML). The measurement value is hashed on the platform and then used to (`TPM_Extend`[3] one of the TPM PCRs in such a way that the resulting integrity metric is a digest of the ordered sequence of all the integrity measurements that have been presented thus far. A TPM must provide at least 16 PCRs. Each register has a length equal to 20 bytes, the length of a SHA-1 digest. A PCR is set to a default value (all zeroes) at initial boot but keeps its existing value during platform sleep states[5].

Balacheff et al.[5] speculate that in the post-boot environment the unit of event to measure is the loading of software and that the value of the integrity measurement is a summary of the software in the form of a reference (software name, supplier name, version, URL) that allows the software to be identified.

Apart from a few PCRs, however, the specifications do not prescribe which measurements should be reported to the PCR suite. For example, a measurement could be that of a software image, configuration file, public key or a random value.

3.2.3 Static locality

PCR[0] to PCR[7] is specified to represent the host's static RTM (for Locality 0) for boot-time integrity metrics. The assignments are as follows[32].

[3] `TPM_SHA1Start`, `TPM_SHA1Update` and `TPM_SHA1CompleteExtend` allows the hash to be performed and logged in the TPM for pre-OS environments [5]

[3] A sleep event could be recorded to a PCR.

Table 3.1: Locality and usage assignments for S-RTM PCRs

PCR Index	Locality	PCR Usage
0	0	CRTM, BIOS, and Host Platform Extensions
1	0	Host Platform Configuration
2	0	Option ROM Code
3	0	Option ROM Configuration and Data
4	0	IPL (Initial Program Loader) Code
5	0	IPL Code Configuration and Data (for use by the IPL Code)
6	0	State Transition and Wake Events
7	0	Host Platform Manufacturer Control (Reserved for OEM)
8-15	0	Defined for use by the static OS (OS specific)
16	Any	Debug
23	Any	Application support (OS specific)

3.2.4 Trusted hardware locality and DRTM

The usage of Localities 1-4 is undefined but the hardware must enforce the usage of PCRs based on their locality assignments. Of particular importance is Locality 4 which is the Dynamic Core Root of Trust (D-CRTM) or "Trusted hardware"[4] locality [32] for which PCR[17] to PCR[20] may be used.

The Dynamic Core RTM (D-CRTM) must be an immutable component of a trusted platform but unlike the S-CRTM it is not required to begin when the platform is reset[32]. The implementation is left to the platform manufacturer, for example `SKINIT` (AMD) and `GETSEC [SENTER]` (Intel). In contrast to the SRTM scheme in which the S-CRTM *may* be measured, the D-CRTM *must* be measured (to PCR[17]).

DRTM simplifies integrity measurement in a number of ways.

- If trust is lost at some point, a new chain can be started without rebooting the platform.
- The trust chain is reduced as it contains fewer measurements. This makes the evidence of platform integrity and behaviour more useful. The measurement log will typically include only two entries[81]: the hash of the Secure Loader and the hash of the TCB code.

3.3 Integrity reporting

Integrity measurements allows a Trusted Platform to reliably create an accurate chain of trust from an initial, known state to its current operating state. Integrity reporting is the mechanism which then allows a third party (known as the challenger) to obtain fresh evidence of the current platform configuration, in a trusted manner.

[4]The Locality 4 modifier cannot be spoofed by any untrusted software including the Trusted Operating System[32]

There are five parts to a challenge [5].

1. **Challenge request**. The challenger selects a target platform, and sends a nonce (to obtain an assurance of freshness) and a selection of PCR indices.

2. **Authorisation**. In an optional step, the use of the AIK may need to be authorised.

3. **Execution and response**. The target's platform agent coordinates the supply of integrity measurements to the challenger by harvesting the SML for the event measurement values used to generate the requested PCR values and invokes the `TPM_Quote` TPM command [36]. The `TPM_Quote` command is processed as follows. The TPM validates the input, fills in a `TPM_PCR_COMPOSITE` structure with the selected PCR indices and PCR values which is then hashed. Then, a `TPM_QUOTE_INFO` structure is created and filled with the new hash and the external nonce. This is then hashed and signed, and the resultant blob is returned to the caller. The platform agent collates and dispatches the TPM signature, event measurement history and the public AIK credential. The history information is plain-text data that describes every event that has been reported to the TPM's PCRs since the platform's last boot.

4. **Validation**. The challenger validates the target platform's response to test whether (i) the identity corresponds to an authentic Trusted Platform and (ii) whether the attested history is accurate.

   The challenger performs the following steps [5, 38]:

   (a) The digital signature is checked. Depending on the platform credentials, either the TPM identity certificate mechanism or the DAA-Verify algorithm[14] may be used. The underlying assumption is that the P-CA (or DAA Issuer) is mutually trusted.

   (b) The values that should be reported for each PCR are recomputed by walking the event history and simulating the sequence of `TPM_Extend` events. The following information is needed for each entry in the event history: (i) the index of the PCR and (ii) the hash value of the measured event.

   (c) Each re-computed PCR value is compared to the corresponding reported value.

   Before assessing the software state of the challenged platform the following cases must be considered [5].

   - If the signature could not be validated then the reported information cannot be trusted.
   - If the signature is validated but the re-computed PCR values do not match the signed PCR values then this indicates that something has gone awry and that the reported information is incorrect. For example [5], if the signed PCR values used to measure firmware components do not match the re-computed values but PCRs that represent

measurements in higher layers of the platform stack do match, then this indicates that the chain of trust has been broken.

- If the signature is validated and the re-computed PCR values match, then the challenger can trust that the hash values in the reported event history accurately reflect the integrity metrics that were reported to the TPM. In other words, provided that the challenger trusts the P-CA (or DAA Issuer) to have properly checked the platform's TCG conformance then it can have confidence in the integrity of the method used to collect and report the PCR values.

5. **Interpretation of Integrity Measurements**. Having established that the TPM is genuine and that the hash values reported in the event history are accurate, the challenger needs to make sense of the evidence to decide whether the target platform can be trusted. This is described next.

3.4 Interpretation of integrity measurements

3.4.1 Validation credentials

At this point it remains to be determined what each event entry actually represents. The TCG envisages that software and hardware manufacturers will provide reference measurement values for measurable components[38]. These reference measurements can be compared to the reported measurements during an attestation. A validation credential is a digitally signed document issued by a validation entity (typically the manufacturer) that describes a correctly functioning component and its digest. Current guidance is that validation credentials are issued predominantly for security-sensitive components.

The TCG speculate also that that the validation credential mechanism be flexible. For example it may refer to a model, series of components or an individual component, that it be distributed electronically and easily consumed and processed by automated tools [38]. It is also proposed [38] that third party agents may provide a service to prove accurate measured event data by comparing PCR values to validation credential digests.

3.4.2 Interpreting a validation credential

If a challenger is to use a validation certificate then it must first trust the validation entity. This could be anyone willing to attest to measured values [38]. If validation credentials are stored in a platform repository on the challenged platform then these can be supplied to the challenger as an aid to interpret the reported configuration [5]. An entry in the event history might be expanded to contain the following fields [5].

- The index of the PCR to which the event was reported.
- The digest of the measured event value reported to the PCR.

- Identifiable information about the event such as the software component name and version or configuration information.
- The validation certificate for the measured component or a reference to the certificate.

Unfortunately software (or component) identities captured in validation credentials do not confer semantic information nor is it clear how these relate to the actual platform state. The latter is a complex aggregate of the states of all of its components.

The challenger must traverse the chain of events, and in doing so, evaluate the trustworthiness of each component identity and the effects of each event to arrive at a conclusion about the trustworthiness of the attested configuration. Balacheff et al.[5] suggest a set of tests based on the trustworthiness of the validation entity and/or component.

In order to overcome the difficulties with verification, Balacheff et al.[5] suggest that (i) in a constrained environment the end states could be certified directly or that the task could be delegated to a trusted intermediary that would either (ii) sample the target state for compliance at specific checkpoints as determined by some policy or (iii) provide an outsourcing service for verification.

3.5 Trusting attestation

This section summarises the key trust elements that are required to establish trust in an integrity challenge.

The principle behind the TCG Trusted Platform is to expand a chain of trust from a single trust foundation. This foundation is subdivided into three roots of trust which are realised in a concrete implementation: the TPM and CRTM. An important observation is that for all other components in the trust chain the relying party only needs to have confidence in the component's ability to *maintain* the trust chain.

Clearly for attestation to "work", the relying party should be able to trust a platform even though she is not in physical vicinity of the platform [5]. This trust is engendered via the conformance, endorsement and platform credentials.

An identity credential is the instrument which separates the relying party from the manufacturer-specific details. The owner must trust the Privacy-CA (or DAA Issuer) to represent, in an identity credential, a true description of the platform (and perhaps not to correlate identities). The output of an integrity challenge is an identity signature over the challenge and integrity metrics. The relying party trusts the integrity metrics are an accurate statement of what was reported to the TPM because she believes the identity signature is from a genuine TPM. The relying party must explicitly trust the Privacy-CA to have performed due diligence in the process of assessing the authenticity of the platform. In trusting the Privacy-CA, the relying part must also trust the entities (TPME, CE, PE) that vouched for the platform.

Finally, the relying party must also trust (or not) the validation entities to sign predicted good values for trustworthy components [5].

- If the challenger does not trust the validation entity or no validate certificate is available, then the component cannot be identified, and is untrusted.
- If a validation certificate is available, but the challenger does not believe that this component will preserve the chain of trust then any following history information can not be trusted.
- If the component is identifiable but the challenger's trust policy rejects it, then the component is untrusted.
- In the event that an untrusted component is identified, the challenger must decide whether the attested configuration still meets its trust policy.

3.6 Properties and limitations of TCG attestation

Having described attestation in detail we are in a position to summarise some inherent properties and weaknesses of what has become to be known as load-time, binary attestation.

- An attestation is a declaration, conveyed in a trusted manner, *at a point in time*, and says nothing about the running state. The attestation and subsequent transaction are not "indivisible" [11] in a number of ways. Firstly, the platform state may change between the attestation and start of the transaction. A challenger can request an attestation before and after a transaction [5] but opportunities for time-of-check-time-of-use attacks remain. In particular the challenger needs to ensure that the attestations are within the same *boot-cycle*. Gallery [25] describes a method in which an exclusive transport session is created between the challenger and the TPM such that any change in software state (due to a `TPM_Extend`) would destroy the session.
- Integrity metrics are intrinsically low-level representations, for example the hash of Option ROM code, a software image or configuration file. Whilst these so-called *code identities* (and their counterpart - the validation credential) serve to identify software[66] they do not confer any semantic information. In particular it does not prove that a program *binary* does what it is supposed to do. To put it another way: "identity is no substitute for behaviour" [81]. This leads to a large *semantic gap*. Extending TCG attestation to applications is problematic.
- Attestation is not a *capability query*. Without additional building blocks it cannot on its own, answer questions about the functional capabilities of a platform, for example whether there is support for an isolated execution environment.
- Software entity or code identity? This question does not pose an issue in a single application system such as the IBM4758 where there is a trust chain directly to a (code) entity[79]. In contrast, the Trusted Platform trust chain is to a platform configuration and does not

identify an entity. Similarly, there is no chain of trust to a live entity. To take a slightly contrived example, if two instances of the Secure Shell Daemon - `sshd` - are started simultaneously, using the configuration files, `/etc/sshd.conf` and `/etc/sshd.evil.conf`, then a relying party cannot use the measurement history to identify a particular `sshd` entity[5].

- Whitelist vs blacklist? These represents two extreme approaches to classifying (un)identifiable code in the measurement history. Both approaches lead to undesirable consequences. In the whitelisting approach to validation, an unambiguous specification defines the acceptable set of valid elements. All else is rejected. In a blacklisting approach the rules define the set that must be rejected and all other elements are considered valid. The problem is that neither measure suffices. Assume for the moment that an integrity measurement system measures all software that is loaded[6]. Then, if a relying party has a policy to trust a set of components it uses whitelisting. However, the relying party cannot know about all software identities on a target platform. Some of these may (or may not) impact the integrity of the set it wishes to trust. The relying party could use blacklisting for this purpose but it cannot predict all rogue software identities. Faced with this dilemma the relying party is left with the unsatisfactory alternative of using policy to decide how to react to unrecognised software.

- The measurement techniques described thus far provide a *load-time* snapshot of the platform configuration. This admits a large array of TOCTOU run-time attacks. The RTM need not be implemented this way. It could instead measure all the integrity altering events [5] so this fact is largely a by-product of the PC stepped boot process in which each layer is instrumented with a measurement function which (by definition) may only measure events in the same (or higher) layer(s). Furthermore given the propensity for vulnerabilities in existing large, feature-rich operating systems a large class of events in the class of "attacks on the layer below" [30] would pass unmeasured and unnoticed.

- In practical terms the TPM can be a significant performance bottleneck. Stumpf et al. [83] suggest a variety of improvements that enables a Trusted Platform to be more performant. The simplest of these is to multiplex a set of integrity challenges into a `HashedNonceList` which is submitted to the TPM in a single challenge. The challenger is responsible for "demultiplexing" the response.

- Integrity measurement and storage is beset by subtle implementation pitfalls. Recording to the event measurement store and the TPM must be atomic. If an application loads

[5] `sshd` and `sshd.conf` could be hashed together [63] to create a unique code identity but this leads to scalability and tractability problems in assessing the state of the platform.

[6] It must do this because it is up to the relying party to reason about the integrity impact of a particular software component

but fails to execute the platform state will be misreported if purely load-time semantics are used.

- TCG attestation has been criticised for being privacy unfriendly by unnecessarily disclosing private configuration information [65, 72]. On the other hand Balacheff et al. [5] argue that this information is not of a secret or private nature.

- A recurring theme is that TCG attestation doesn't scale in open computing environments [65, 72, 81].

  - The frequent patch-upgrade-patch cycles endemic to modern platforms leads to a combinatorial explosion of possible (potentially) trusted and untrusted configurations even for a single suite of named software.
  - In the SRTM approach an extremely long trust chain develops as events are collected since the platform's last reboot.
  - The initialisation sequence, during the OS boot phase, is not necessarily controlled, and the event sequence thereafter is unpredictable. This is important since order matters.

- The SRTM approach brings a unique set of challenges.

  - Firstly, a challenger needs to determine the trustworthiness of every component in the trust chain. This problem becomes intractable [81].
  - Secondly, given the length of the trust chain and the large number of run-time environments (of which the OS is just one) simply identifying the components that contribute to the trust chain is not a trivial task.
  - Thirdly for a static root of trust to be meaningful to a relying party all parts of the trusted computing base needs to be measured. If the TCB is ill-defined then all executable content is inevitably security sensitive and needs to be measured [81].
  - Fourthly, S-CRTM is dependent on the immutability assumption of the the TBB and CRTM. These components are typically implemented on firmware in a flashable storage medium.
  - The DRTM scheme, in contrast, bypasses the SRTM immutability issue: only immutable hardware elements are involved in the D-CRTM and processor patches are disabled during this process [81].

- An inherent assumption is that the challenger's platform is trustworthy [5] and will not lie about the about the verification results in an attempt to elicit personal information[59].

- System attestation or TPM attestation? As systems - networks or even PCs - become increasingly divided into separate parts and become more complex so these become more security critical [54]. Eventually it will become more attractive to attack the peripheral

devices (e.g. if the screen could be attacked to access confidential documents). That an attestation to the platform does not imply an attestation to the system should not be surprising as the scope of the platform is defined by its TCG credentials, and therefore only these elements should be trusted in an attestation.

- Finally, there is the wider issue of liabilities and guarantee of behaviour. Hardware and software manufacturers offer no behavioural guarantees for their components - users are required to agree to licence terms which usually (or always) contain a a disclaimer of purpose and indemnifies the vendor.

Chapter 4

Using attestation

In the previous chapter, TCG attestation was described in detail, as well as its inherent limitations. The purpose of this chapter is four-fold:

- The potential use cases for attestation is summarised in the next section.
- A simple threat model and terminology is introduced to motivate subsequent discussions.
- A simple and practical *rollup* technique is introduced that uses layered encapsulation to achieve separation of concerns and reduce the complexity of predicting integrity metrics.
- Two themes of application for attestation are presented[1]. To illustrate each theme, an application is presented as a case study. A section on related work accompanies each theme. The two themes are (i) local deployment of third party trusted services and (ii) ubiquitous trusted computing.

4.1 Preliminaries

4.1.1 Attestation use cases

A relying party may request an attestation for one of the following reasons.

- To decide whether to interact with a (trusted) platform. This is its *raison d'etre*.
- To decide, after interacting with a platform whether the output is trustworthy.
- To decide, during a long-running transaction whether the platform's real-time state satisfies the relying party's trust policy by obtaining a "continuously running" attestation of state or alternatively via a level of indirection in which an attested monitor process acts as the policy enforcer.

[1]The themes represent common strands encountered during the author's literature survey.

- To test the integrity of a user-land process (i.e. software entity). Note that since the integrity of each layer depends on the layer below, these too must be tested. This is a subcase of the first use case.
- To query a trusted platform state. This is a subcase of the first use case.
- To query the capability (or properties) of a platform. This is not a directly supported use case but the idea has considerable traction in proposals for userland applications and is the motivation for at least two attestation schemes (Chapter 6).

4.1.2 Definitions

The terminology for the rest of this chapter is taken from references [44] and [84].

- **Security Services**. There are five categories of security service [44]: (i) authentication, (ii) access control, (iii) non-repudiation, (iv), confidentiality and (vi) integrity.
- **KEM-DEM**. Key encapsulation mechanism - Data encapsulation mechanism [46] is a hybrid construction in which a symmetric key is used to encrypt (encapsulate) plaintext data and an asymmetric public key is used to protect the data-encrypting symmetric key. KEM-DEM refers here to the technique in general and not the ISO/IEC 18033-2 specification for the method.
- **Vulnerability**. This is a weakness (e.g. absence of a safeguard) or a flaw (e.g. design flaw) in a system.
- **Asset** A thing, concrete or abstract, of interest to an adversary. It is the target of the attack and the thing for which protective measures must be installed (along the attack path(s)). The attacker and owner's view of the value of an asset may be different.
- **Threat**. The adversary's attack goal, or what an adversary might try to do, that poses some form of danger to an asset or to the system at large. An **attack** (or exploit) is the realisation of a threat.
- **Threat tree**. Successful attacks combine an attack point with a number of weaknesses to reach a target. A threat tree (or attack tree) models these relationships as a directed acyclic graph which joins the combinations of weaknesses on a route to a target[2]. The attack goal is the root node and the edges connect each parent node to its sub-goals. Intermediate nodes represent successively less abstract means of realising the (intermediate) goal embodied in its parent. Intermediate nodes are expanded until the point at which a primitive exploit is feasible or an elementary condition is reached in a leaf node.

[2]http://www.matasano.com/log/1032/this-new-vulnerability-dowds-inhuman-flash-exploit/ contains an amusing anecdote on how seemingly disparate weakness can be composed as an attack[21].

- **Attack surface**. The scope and manner of available entry (and therefore attack) points. Any means to interact with the system is an attack surface, e.g. file handles, raw TCP, other IPC, user accounts, file permissions, symbolic links, and email to name a few. Minimising the attack surface restricts the visibility of the system and ultimately reduces the number of attack paths. The by-product of this restriction principle is system simplicity [11].

- **Attack path**. A realisable route (from leaf to root node) through a threat tree. A realisable route means that the resources and incentives of the attacker and probability of attack success outweighs the existing controls on the attack path.

- **Entry point**. An external interface to a system that is simultaneously the element through which an attack begins.

4.1.3 Threat model

Threat modelling has become synonymous with a systematic methodology to assess the security risks of an application. The threat modeling process involves understanding an adversary's goals in attacking a system based on the system's assets of interest. Then, given the assumptions about the resources available to an adversary, the likely attack points in the system can be identified. This information is recycled into the security requirements and finally into security features in both design and implementation.

In a properly defined threat model [84] the following elements need to be specified.

- **System Security**. How the system should and should not be used (i.e. use cases) and information flows within the system. These are (mostly) provided by each case study described in this chapter.

- **Threat Profile**. Threats are enumerated and composed, leading to attack paths and threat trees that affect assets. Risk metrics are derived for vulnerabilities based on impact, probability, and cost. For the purposes of this chapter, we assume that the adversary goals are merely an intellectual curiosity, i.e. how to "break", the system in some way. Secondly, since we are not interested in enumerating all the threats to the system (i.e. as documented by a threat tree) the challenge is to find valid attack paths. We are unconcerned with factors such as likelihood, cost and impact of a successful attack so it remains to describe the properties of our adversary.

- **Adversary**. A description of the adversary is a description of an adversarial view of the system interfaces, assets of interest, the authorisation levels required to access the system's entry points, available resources and adversarial capabilities.

  – The attacker (or attacker's agent), who shall be named Mallory, has physical access to the machine and can behave arbitrarily maliciously (within her capabilities).

– During dormant operation (i.e. when the machine is switched off), Mallory can alter any writable part of secondary storage. This therefore excludes tampering with the TPM. The CRTM, though it may be rewritable, is also excluded.

– During live operation, Mallory can subvert any layer of the software stack, except for an isolation layer (if present).

– Mallory does not execute physical attacks (e.g. cold-boot, malicious DMA) or side-channel attacks but is able to perfectly exploit any weakness in design and implementation (for example reading from or writing to the Linux kernel memory device file `/dev/kmem`).

Is Mallory too powerful? The resultant threat model is far more invasive than an arbitrarily malicious network attacker [20], but it is consistent with the stated aim of trusted computing [5, 66]: to move the trust in a platform away from trust in the owner and into trust in the TCG hardware components.

An attack path for the application described in Section 4.2.1 is discussed next.

Mallory's attack goal is to obtain a plaintext copy of application A_C which is being sent by service provider S to downloader A_D. Her most powerful attack is simply to mount the secondary storage off-line and read the application (assuming that an encrypted filesystem is not in use). She could attempt to dynamically subvert the secure execution environment in which the legitimate receiver, A_D, is executing, but this is too difficult. Mallory could masquerade as the legitimate receiver of the application. This will work if Mallory can fool S into accepting an attestation to a platform configuration that includes both A_D and Mallory. If Mallory can do this then she can load the private key required to decrypt the downloaded application. If S is only concerned that A_D be present in the attested configuration then the attack is complete. If S is using blacklisting then there is still hope. Mallory can mutate her identity until she is no longer blacklisted.

Why is the sub-goal of fooling S in this way a valid vulnerability? In this case it is because the details of reconciling platform state with the identification of the entity at the network endpoint is by design, highly ambiguous, and because the algorithm is left open to implementers.

4.1.4 Layering policies

In this section a small helper pattern is described in high-level terms, to be used later. The initial motivation for the pattern is finding a solution to the following basic problem (see Section 4.2.1): how to *practically* use TCG functionality to securely deliver[3] and then run an application. Balfe and Paterson [7] describe a protocol for the secure delivery of an application.

The contribution of the pattern presented here is to separate the concerns of downloading and usage. This separation allows the prescription of independent integrity metrics for each

[3]Note that simply appending a key establishment step to a standard attestation is open to a relay attack [82].

concern[4]. In essence the remote party simply double-encrypts[5] the package using the public key counterpart of a signed non-migratable or certified migratable key pair . The outer layer specifies the policy under which it is securely delivered (i.e. that it be integrity-protected) and the encapsulated, inner layer encrypts the application to a second policy that specifies the platform state required to reveal and execute the application.

The solution proceeds as follows.

1. **Key generation**. The TPM generates two storage asymmetric key pairs ($SK1_{pub}$, $SK1_{priv}$) and ($SK2_{pub}$, $SK2_{priv}$) using the $TPM\_CreateWrapKey$ command. As part of the key generation, an independent PCR state is specified for the use of each private decryption key. Each key pair is certified ($TPM\_CertifyKey$) using an identity key or by a signing key [36] which is itself certified by an identity key.

2. **Double encryption**. The remote party uses KEM-DEM to doubly encrypt the package which is then sent to the target. The inner and outer layer are protected under the purview of $SK2_{pub}$ and $SK1_{pub}$ respectively.

3. **Policy enforcement**. On receipt of the package, the platform checks the package integrity , using KEM-DEM to unpack the first layer which is protected by $SK1_{priv}$ (whose usage is bound to the download policy) and stores the application still encrypted under $SK2_{pub}$ (the execution policy). ($SK1_{pub}$, $SK1_{priv}$) can be retained if the download policy is general enough to be reused else it can be destroyed.

Clearly the current platform state needs to match the integrity metrics specified by each policy. To make the solution self-contained some helper glue may be required to allow the application to bootstrap itself. For example, a bootable portion which is used to decrypt (using $SK2_{priv}$) and load the encrypted non-bootable portion. The policy that controls the application execution must include the well-known measurement of the bootable portion.

4.2 Theme 1: Local deployment of third party trusted services

4.2.1 Case study: Secure distribution and local execution of (mobile) code

The suite of services that are being delivered over wireless mobile communications systems is rapidly expanding. 3G systems are able to delivery rich multimedia content to a diverse population of subscribers' mobile devices. The expectation is that the next generation of mobile services will, in partnership with broadcast systems, provide wireless access to controlled video content. This control, known as *conditional access* is specified in a set of standards for

[4]The idea can be easily generalised to a sequence of actions, each rolled up in an encapsulating layer.

[5]See also Proudler's suggestion of double encrypting to realise the enforcement of both hard and soft policies on data[66].

conditional access systems and their interfaces, by the Digital Video Broadcasting (DVB[6]) organisation. The aim of a conditional access system is to ensure that content is accessed only by legitimate subscribers and then only by those subscribers who have in some way subscribed to (or paid for) that particular content. Unfortunately, the DVB set of standards are not well suited to an environment in which mobile users do not have a prior relationship with a broadcaster and in which mobile users wish to view content from many different broadcasters. The reason is that whilst the interfaces and "common scrambling algorithm" are well-defined the keying material, known as the Control Word (CW), is conveyed between the broadcaster and receiver in a proprietary manner. DVB defines two standards for this purpose: (i) by *Simulcrypt*ing at the broadcaster a second conditional access system can be supported and (ii) a separate hardware module per conditional access system at the receiver, which implements the *Common Interface* standard, can be used to release the keying material from the proprietary encrypted control words.

Gallery [26] examined how the latter solution may be achieved by on-demand substitution of a *software-based* conditional access application, for the common interface module, at the mobile receiver. Two fundamental security requirements are defined:

- **Secure download**. The confidentiality and integrity of the downloaded application must be assured.
- **Secure execution**. The application should be protected in secondary storage and during execution.

To realise the former, a helper agent - a download application (A_D) - is defined. A_D has a dual role. Its first responsibility is the secure downloading of the legacy conditional access application, A_C from software provider (S). Then once A_C has been installed it is tasked with providing some level of assurance that during execution, A_C adheres to a specified policy. It is assumed that the download helper (initially) and the downloaded conditional access application (later) both run in a protected execution environment. Of particular interest is the single specified state, specified by integrity metrics `I`, in which both the download protocol is executed, and in which A_C runs.

The download protocol is shown below, abbreviated into four phases.

1. **Handshake and key generation**. A_D requests A_C from S. S responds with a nonce. A_D instructs the TPM to create a new key pair (`TPM_CreateWrapKey`) specifying PCR metrics I required for key release.
2. **Key attestation**. A_D instructs the TPM to certify the newly created key pair. In response the TPM signs the newly public key using a TPM identity key. The TPM also returns information attesting to the mobility constraints and the state I under which the corresponding private key will be released. `S` verifies the signature using the TPM's

[6] www.dvb.org

public identity key, checks that the signature is fresh, and determines whether `I` represents a sufficiently trustworthy state by referring to a local repository or by using the validation credential mechanism.

3. **Key exchange and secure delivery**. `S` generates symmetric keys for confidentiality and encryption, and uses these to send A_C securely to A_D. The symmetric keys are transported under the protection of the new public key.

4. **Application decryption and execution** A_D asks the TPM to load the private decryption key. The TPM does this provided the protected object's `digestAtRelease` field (which was set to I) matches the current platform state. The TPM decrypts and releases the wrapped symmetric keys enabling A_D to decrypt A_C and verify the MAC. A_C is executed or saved to secondary storage in encrypted form.

4.2.2 Analysis

The protocol uses TCG *binary load-time* attestation plus an acceptable set of integrity metrics to verify that the conditional access application is fetched by a download application that is located on a non-malicious, trusted platform. Then, once downloaded, protected storage (of the application decrypting key) is used to maintain the confidentiality and integrity of the (encrypted) resident conditional access application. By making access to the encrypting private key contingent on the same set of integrity metrics, the use of the application is strictly controlled. The download helper is active during execution of the conditional access application to monitor and enforce additional policies.

An analysis of the resulting context follows.

- The attested integrity metrics are used to support a *capability query*, i.e. to identify the presence of the broadcast application and functional support for a protected execution environment. For example: from the integrity metrics S can verify "there is a legitimate broadcast application executing on the mobile host" [26]. The capability query assumption was questioned in the previous chapter.

- A_D is assumed to be executing in a protected execution environment. However, there is no information in the integrity metrics that explicitly binds the executing environment of A_D to the protected environment, nor to the software entity that is the broadcast application. To solve the former, S could attempt to trace the chain of trust. The latter is even less tractable, since the event history is simply a record of events in the order in which they were reported. This illustrates the second pitfall of integrity metrics: the inability to say anything concrete about a *software entity*.

- The attested integrity metrics serve a dual purpose: (i) a statement about the download environment and (ii) to control access to the future running environment. Similarly the download agent serves two purposes. The second of these is the run-time monitoring of

the conditional access application. It can be argued that this is an inappropriate design choice as it fails to separate privileges and complicates the implementation [11].

- The protected execution environment must also host the download agent.
- If the layer (e.g. an isolation layer) that provides the protected execution environment, can be attacked, then no security guarantees can be made.

With a little reorganisation and judicious use of the policy layering approach described earlier, the protocol could be implemented as three layered policies, resulting in a package in which the conditional access application is encapsulated by three layers of encryption. The outer, middle and inner layers reflect the download, monitoring, and application execution policies. The resulting context is slightly more favourable.

- The *capability query* has been removed. The solution is still as secure, however, as the inner policy "seals" the application to the requisite protected execution environment state.
- No reliance is made on the fact that the download and execution environments be the same nor that the integrity metrics identify a software entity.
- Each set of integrity metrics (i.e. policy) is expressed independently. This should make it simpler to specify integrity metrics.
- The monitoring and execution functions are encoded in two distinct policies. This has the benefit that privileges are well separated [11].
- Fewer assumptions are made about the download application which can now execute in an insecure environment.
- If we assume load-time, binary integrity semantics then attacks on the layer below remain. The attack surface, however, is reduced and a modicum of defence in depth is introduced.
- Finally, note that this modification makes it easier to seal the application proper by including an additional layered policy that installs (and seals) the conditional access application (in an automated fashion). Once the installation is complete the policy key pair can be destroyed.

To see that the attack surface has been reduced, first consider that a TOCTOU attack [80] on "the layer below", at the point that the application is downloaded and decrypted by A_D, is thwarted. Next, consider a more sophisticated attack. We recall that an attestation does not easily reconcile a software entity's contribution to the platform state. Now, in the original protocol, consider that A_D is present, inside a protected execution compartment. Also present is an attacker, $Mallory$ running in an insecure environment masquerading as A_D at the client network protocol endpoint. How can S tell from the integrity metrics whether the network endpoint leads to $Mallory$ or A_D? The answer is that it cannot. If $Mallory$ can sufficiently

obfuscate the event history (or if S is using blacklisting and $Mallory$ is not on the rogue list) then she can receive and decrypt the received application. This is because the signed integrity metrics I include $Mallory$. This is a masquerade attack leading to information leakage. To counter this, S might whitelist only acceptable configurations (at the risk of being consumer unfriendly). The alternative, blacklisting, will engage S in a never-ending arms race to identify newer Mallories. This weakness is not present in the modified protocol since $Mallory$ never gets her hands on the unencrypted application. It is transported in a separate policy.

4.2.3 Related work

Single sign-On Pashalidis and Mitchell [62] discuss ways that the Authentication Service (AS) component in a single sign-on (SSO) solution can be locally hosted at the untrusted client. Their work does not consider the additional security offered by a protected execution compartment. Attestation from the trusted platform (which hosts the AS) to the network service is proposed as a means of assuring the service that (i) the AS is hosted on a trusted platform (a software entity identification query), (ii) the platform state is trustworthy and (iii) the AS is running correctly and therefore is behaving as expected. In particular since a well-behaved AS only performs the "SSO protocol" after the user has been authenticated, the attestation guarantees that attacks based on a maliciously modified AS are prevented.

In one attack, the AS is subverted before the attestation is performed (but after it has been measured). The malicious AS then sends the authentication assertion. This is prevented by hashing the concatenation of the credentials that substantiate the authentication assertion, with the challenger's nonce. The new nonce becomes the attestation challenge.

In another attack not prevented, however, an attacker hijacks the AS process and is able to fabricate user authentication assertions (since presumably it is privileged enough to do so if its functionality is trusted).

Distributed TTPs Dent and Price [18] consider the security issues in delegating trusted third party services to a locally distributed "TTP-applet" which runs in a secure execution environment (SEE) on a user's machine. In one example, the user's platform hosts a "CA-applet" which (i) certifies (and revokes) the user's public keys and (ii) acts as a directory service for locally issued certificates. The central CA delegates most CA services to the CA-applet under a strict policy. The challenge is to ensure that the policy is enforced. In order the meet this challenge the security requirements include several attestable elements. The previous chapter discussed the pitfalls for deciding these classes of attestation from TCG integrity metrics.

- Attestation to the authenticity of the SEE and that the SEE is in a state, ready to receive an applet (e.g. "by providing evidence that a proper boot sequence has been executed").
- Attestation to the SEE's secure download capabilities and whether it can meet *CA-central's* confidentiality and integrity policy.

- Attestation to the SEE's ability to execute the applet in a secure manner.
- Attestation to the trustworthy execution and output (see Section 4.1.1).

There are 4 phases in the life of a CA-applet.

1. **Download and Initialisation**. CA-central authenticates the user and the SEE. CA-central checks the policies associated with issuing certificates to the recipient (i.e. the user) of the applet. CA-central generates a CA-applet, generates and personalises the CA-applet with a certificate-generating private key and signs the public verification key. Then CA-central checks the authenticity of the SEE and securely downloads the CA-applet to the SEE.
2. **Registration**. Only the user to which the CA-applet has been issued should be able to register a public key. The user must provide proof of possession of the corresponding private key. Trust is conveyed to the consumer of a CA-applet issued certificate as follows. The relying party (consumer) is provided evidence (PCR metrics) of the CA-applet environment at the time the applet was first downloaded to the SEE. The relying party must adjudge the attested metrics to represent a trustworthy state. The relying party must trust the SEE to operate correctly. If these conditions are fulfilled then it can trust a CA-applet issued certificate.
3. **Directory Service**. The CA-applet should be able to supply a certificate for a public key that it has certified.
4. **Revocation and Renewal**. The CA-applet provides a certificate revocation service (CRLs and OCSP). The CA-applet key has a short lifespan designed to limit the potential abuse of the compromise of the CA-applet private key. This requires a method to renew the user's public key certificates as well as the CA-applet itself.

The goals of an attacker include (but are not limited to) the following: (i) extracting the CA-applet's private signing key, (ii) getting the CA to endorse a more liberal policy, and (iii) getting the CA-applet to certify the attacker's keys as belonging to the user. Some potential improvements can be made to the attestable elements in the above four phases.

- Secure download to an entity's environment involves the question: from the integrity metrics, with *which entity* have I established a secure session? The protocol solution is to cryptographically bind the authenticity checking session to the download session. This ensures that CA-central is downloading to the *same* entity but it doesn't *identify* the entity. For example, *Mallory* can simply execute a man-in-the-middle attack (or masquerade as the SEE) at the start of the protocol and extract the CA-applet's private key. An alternative is to use the policy wrapping technique described earlier.

- Attesting to the download state is stale evidence. The output process (responsible for certificate generation) is not attested. If the time window between CA-applet download and certificate generation is large enough then, with load-time attestation semantics, the system is vulnerable to a time-of-check-time-of-use attack [80].

- The provision of proof (of the private key) by the user is not attested. If *Mallory* can install malware in an insecure environment the she can get the CA-applet to certify as many certificates as she wants[7].

- Mallory can mount the SEE filesystem, off-line and extract the CA-applet private key or modify the delegated policy. To prevent this the central CA can require that the SEE filesystem be encrypted. This is a capability query. Assume for the moment that this capability query is feasible, say by extending a PCR and the event history, with information about the SEE filesystem. Unfortunately, the integrity metrics do not provide the data origin authentication of this attestable fact. It may be that it is Mallory, lying that the SEE filesystem is encrypted (when it isn't). To prevent this the central CA may use an error-prone technique in which it distrusts any attestation that has more than one "similar-looking" attestable fact. A safer option for the central CA is to use the policy layering technique described earlier.

4.3 Theme 2: Ubiquitous trusted computing

Ubiquitous computing is a view of computing in which many computers are both physically available and invisible, in a manner that enhances the activities of everyday life [85]. More recently the term *pervasive computing* has been substituted for ubiquitous computing. The vision remains the same - entities (users and devices) spontaneously and intelligently interact within a networked infrastructure [1, 57, 68]. Examples of such devices include wearable computers, smart rooms and active RFID tags [1]. The enablement of trust is recognised as a key issue in both personal and infrastructure critical applications of pervasive computing [1, 68].

A case for ubiquitous trusted computing has been made from the following four arguments: (i) low cost, (ii) "adequate" security, (iii) applicability to mobile environments, and the most compelling, which is examined briefly in this section - (iv) the facilitation of dynamic, collaborative relationships whilst assuring low levels of risk [5].

4.3.1 Case Study: Enforcing trust in pervasive computing with trusted computing technology

Li et al.[57] investigate the use of trusted computing to enforce trust in a pervasive computing environment. In their scenario, Bob, an external consultant, has been hired to support Alice,

[7]The private keys are uncontrolled anyway since, in this case study, private keys are not generated inside the TPM.

a project manager, in a large organisation. Furthermore the organisation is equipped with its own Privacy-CA for issuing certificates. On arrival at the company's premises, Bob obtains some credentials from Alice, and enters the building.

The security challenge is to allow Bob to enter "smart rooms" and utilise their services in a manner that is mutually trusted. The security requirements and the proposed solutions are summarised as follows [57].

- **Authentication and Authorisation**. In the *Credential Acquirement* phase Bob enrols with the organisational Privacy-CA to obtain an identity credential and an SKAE CA [34] that provides an X.509 *capability certificate* which binds the same identity key to a set of capability attributes to be used later in access control decisions. In the *Access Control* phase Bob attempts to enter a smart room and use its services. An authentication server in the smart space and Bob's device engage in a challenge-response protocol in which Bob's device signs an integrity challenge. The signed integrity challenge plus Bob's capability certificate is returned to the authentication server. The authentication server verifies the signature and the certified capability list.

- **Confidentiality**. The data transmitted and stored should be kept confidential. The mechanism proposed to achieve this is discussed next.

- **Control after distribution**. Each smart space needs to trust that once information has been released to a client it will not be used in an unauthorised manner. The mechanism to achieve this proceeds as follows: Bob's TPM generates and certifies a TPM wrapped key, the public key part is signed by Bob's AIK, the public key and key information is encrypted with the server's public key, and a bundle consisting of the AIK-signed wrapped public key plus the encrypted information is transported to a server in the smart space. The smart space server, after verification of the signature and checking that the purported key information meets its policies, encrypts the document using KEM-DEM under the purview of the wrapped public key and returns this to Bob's device.

- **Trusted services**. In a pervasive computing environment, the client would also like an assurance that the service behaves as expected. For example, Bob's device can request an attestation of platform state from a printer server.

4.3.2 Analysis

Unlike the previous theme which discusses security in attestation, the purpose of this theme has been to set the scene for a discussion of the practical obstacles to pervasive computing. There should be few doubters that the promise of pervasive computing is a consequence of three things: (i) the growth in computer processing horsepower embedded in smaller, cheaper computers, (ii) the advances in technology needed to power these devices and (iii) the ability to interconnect devices so that a networked computing infrastructure becomes pervasive.

The enabler of interconnectivity is the layered protocol paradigm [45]. In a layered network stack, each layer uses a well defined service interface. The service is provided by the layer below. In this model, peer protocol entities "speak" a common language and a layer need not know and should not know, how the underlying services are provided. In the TCG attestation model, however, a relying party *must* know how to verify lower levels in the platform stack if it is to validate the trust chain. It is an inescapable consequence of ubiquitous trusted computing and the concomitant proliferation of device types having new architectures and platform stacks, that the quantity of platform configurations and the complexity of validation (in its current form), will increase substantially. If, as in the scenario of pervasive computing, devices which have no a priori relationship are to interact seamlessly, then a solution is required that breaks this apparent dichotomy: the need to abstract low-level details that reduce complexity with the need to attest all integrity-altering events.

How can Bob know whether to trust his own device? This is treated in the next section.

4.3.3 Related work

iTurtle How can a user know whether a device is trustworthy? McCune et al. [59] present examples of three common problems in adapting TCG attestation to user-based attestation. The source of the problem [59] is the underlying assumption [5] that that relying party's device needs to be trusted.

- There is no authenticated channel between the user and the verifier to notify the user of an unsuccessful attestation outcome.
- In a networked world, the user may interact with any manner of devices - 802.11 access points, home routers and printers. In order to apply TCG attestation to these devices the user needs a trusted verifier. It is unclear, however, why the user should prefer any verification device (laptop, mobile phone, trusted third party) over another.
- Integrity measurements leak information about the challenged platform. If the verifier (or verifiers) is (are) able to correlate this information to uniquely identify the challenged platform then they can impinge on the user's privacy. For example, in an e-commerce scenario, the verifier could link the user to a platform configuration and consequently track future interactions.

The authors argue for an "axiomatically trusted" device which is nicknamed the *iTurtle*. This effectively introduces a new root of trust into the system. The key elements of the device are that (i) it provides user-observable verification, (ii) it is a (single) trusted hardware device, and (iii) it is easy to use (small, simple user interface and has universal physical connectivity). The scale and complexity of verifying a platform configuration is cited as one of the core open challenges for the iTurtle.

It is postulated in the next chapter that, at least logically, the relying party uses two decision functions: a verification function and a trust decision function. The trust decision function is

somewhat nebulous but it captures the "expectation of specific purpose" aspect of deciding trust. The user is presumed here, to believe that her iTurtle makes the correct trust decision. Finally there exists the question of the efficacy of user interface feedback. In the event of a logical verification failure such as happens when the attested event history is incorrect or if the iTurtle does not recognise the attested configuration as "good" what course of action will a user take? Experience [42] suggests that warning signs are frequently ignored.

Demonstrative Ad Hoc Attestation Bangerter et al. [8] make an attempt at building an iTurtle-like solution. Demonstrative ad-hoc attestation is defined be a system in which a user, having no prior relationship with a device (i.e. no pre-shared keys or identifying information is available), physically identifies the device (e.g. by pointing to it) and asks the question: can I trust that device? Ordinarily, the stated scenario is subject to a classic man-in-the-middle attack.

Their solution requires that the relying party establish a prior relationship with a trusted third party to which the verification function is delegated. The modus operandi is as follows.

1. The user launches an ad hoc attestation transaction by starting (pre-installed) ad-hoc attestation software on the computer to be challenged and holding a secure token to the screen.

2. The computer (i.e. ad hoc attestation software) engages in a standard attestation protocol with a trusted third party server.

3. The result is transported securely back to the secure token via the attested computer. This channel needs to be authenticated in order to protect against other man-in-the-middle attacks.

The advantage of this solution is that it moves the verification problem to an expert. The disadvantages, are threefold: (i) it is not quite pervasive - client software must be installed on the attested computer to act as the attestation proxy, (ii) the verification TTP is required to be always online and highly available and (iii) it adds yet another element to the existing, large complex Trusted Computing PKI [6] that is required to support attestation.

4.4 Epilogue

This chapter has barely scratched the surface of the endless array of potential applications for TCG attestation, choosing instead to give a flavour of some of the potential themes for attestation and the open questions that remain.

In summary, the salient points to be drawn are:

- Designers need to carefully incorporate attestation into application protocols. In particular, the semantics and limitations of attestation needs to be well understood.

- In its current form, attested integrity measurements do not provide the kinds of security guarantees that are sometimes imagined. The most common problems are: treating an integrity report as a capability query, and as an attestation to a live software entity.
- The corollary is that TCG integrity reporting is not rich enough to express the kinds of platform state descriptions that designers require to build trust into their applications.
- The seemingly intractable problem of verification of TCG integrity metrics will be a significant impediment to the adoption of TCG functionality in pervasive computing environments.

Chapter 5

Attestation models

5.1 Preliminaries

5.1.1 Elements of attestation

It is postulated here that an attestation scheme must answer the following questions (see also [59]).

- What evidence is being measured, recorded and in turn presented? In TCG that evidence is a chain of trust which is described by a sequence of platform-state altering events. Furthermore the "raw" evidence that is measured and recorded may not be the evidence that is attested [1] (e.g. [72, 41]).
- How is the evidence measured? This question is more relevant to the attesting system because the answer determines the measurement architecture.
- When is this evidence measured? In TCG-style attestation, measurements are accumulated ahead of an attestation request. Just-in-time techniques measure (some subset of) the platform configuration at the time of an attestation request (e.g. SWATT [76], [74])
- How does the relying party verify the attested configuration? Even in a closed, controlled environment (e.g. corporate infrastructure) the number of possible software configurations can be substantial. Ideally, the system should allow the *verification function* to be both deterministic and operate within some bound on computational complexity (e.g. that it be linear [79]) and memory (e.g. that the size of the verification data repository be manageable).
- Finally, how does the relying party use the verification result plus other contextual information (e.g. past experience) to make a trust decision? This is referred to here as the *trust decision function*. Its purpose is to address the semantic gap by translating a

[1] In [41], what is measured is Java bytecode using both static and dynamic source code analysis. What is attested, is instead some notion of behavioural semantics.

good configuration into a notion of trust. Balacheff et al. [5] suggest that this should be somehow automated, perhaps as an OS service to users.

There are two other properties that are desirable in an attestation system.

- **Completeness**. It should be capable of measuring all security-related events that may be relevant to a challenger. This naturally leads to a fundamental design dilemma (see also [30]) - what to measure? The attesting system cannot know, in advance, which trusted state transitions are likely to be of interest to a relying party. A designer is faced with the decision of measuring too many, too fine-grained events and burdening the verification function with a long sequence of events of no semantic meaning. On the other hand there may be too few, too coarse-grained events which are not sufficient to allow the full platform stack to be trusted.

- **Sufficient** The reported evidence should allow relying party to interpret the results in a meaningful way, more specifically to allow the trust decision function to address the semantic gap between machine measurements and the attributes required to form a notion of trust.

5.1.2 System model

For the purpose of this chapter a simple reference model is introduced. There are three views: entity view, layered view and the dynamic (state) view.

- **Entity view**. The entities are the challenged (or attesting) platform, and the challenger (or relying party or target). It is possible to further subdivide responsibilities, for example on the challenged platform - a measuring entity, platform reporting (attestation) entity - and on the relying platform - a verification entity and a trust decision entity. Some attestation proposals do distinguish between these entities, for example in [72] the platform reporting entity is introduced to translate raw TCG measurements into "property configurations". Subdividing and distributing entities implies distinct trust domains.

- **Layered platform view**. The software stacks of the challenger and attesting platform are assumed to be layered, e.g. hardware, kernel, commodity operating system, operating system services and userland applications. Some layers are optional or missing (e.g. high assurance application and Trusted OS). Furthermore a platform may be (vertically) partitioned into isolated, protected compartments under the authority of an isolation layer [63] (e.g. (VMM/hypervisor). The exact layering is less important than the following three observations.

    1. The integrity and state of any layer depends on the integrity and state of the layers below [79].

2. As a consequence, in an open system, a challenger at layer N *must* somehow verify the entire platform stack including layers 0 to N-1. This contrasts with the basic principle of layered design [19].
3. The layering may be far more complex than indicated here, consisting of more than one OS superimposed on a virtual machine service offered by a lower layered OS (see Figure 2.2 in Proudler [66]). In this case an attested trust chain is far harder to trace.

- **State view**. At any layer an integrity-altering event, `e`, might lead to a change in trusted state. For example, $S_i \rightarrow^{e_{i+i}} S_{i+1} \ldots \rightarrow^{e_{i+j}} S_{i+j}$ is a sequence of events e_i, e_{i+1}, $\ldots e_{i+j}$ from known "good" state S_i to some state S_{i+j}.
  - Implicit in the TCG load-time view is that the hash of the software to be loaded is sufficient to describe the state transition which allows state S_{i+j} to be checked as a function of S_i and subsequent load events [12].
  - Together, TCG load-time semantics and the reliance on the integrity preservation of lower layers, makes this state model of little relevance if time-of-check-time-of-use attacks [80] or dynamic attacks on the "layer below" [30] are feasible.

5.1.3 Integrity metrics and state

In the figure below, layers N and N-1 have state S_i and S_j respectively. Now consider events e_{i+1} and e_{j+i} in each layer. If $time(e_{i+1}) < time(e_{j+1})$ then let the resultant composite state be $\bar{S}$, say. If $time(e_{i+1}) >= time(e_{j+1})$ then let the resultant composite state be $\bar{S}'$. Are $\bar{S}$ and $\bar{S}'$ the same state? They might or might not be the same state depending on the platform state machine (for example, depending on whether e_{j+1} has a causal effect on e_{i+1}). However, if layer N and N-1 are recorded to PCR_N and PCR_{N-1} then they will have the same attested configuration. Is this cause for concern? It is a potential pitfall if a third party attempts to use the attested platform configuration to reason about platform state.

Figure 5.1: Both state diagrams yield the same integrity measurements but the end state is not necessarily the same.

	t0	t1	t2	t3
Layer N	S_i	$\rightarrow^{e_{i+i}}$	S_{i+1}	
Layer N-1	S_j		$\rightarrow^{e_{j+i}}$	S_{j+1}
Layer N	S_i		$\rightarrow^{e_{i+i}}$	S_{i+1}
Layer N-1	S_j	$\rightarrow^{e_{j+i}}$	S_{j+1}	

5.1.4 Measurement techniques

In this section we briefly review the set of available techniques available to the designer of an integrity measurement architecture. The source of information comes from the well-studied and practised art of *performance analysis*. The trade-offs are different. For example, the choice of technique for a performance profiler depends critically on the ability to accurately measure entry-exit times and optionally to trace the call stack.

Event callbacks Many modern (interpreted) programming languages (e.g. Microsoft .net, Java) allow a profiling agent to "hook' into the runtime to receive notification of certain events.

Statistical sampling Sampling profilers probe the application at discrete intervals using an interrupt mechanism (e.g. gprof [31]) and are therefore non-intrusive. The output is by definition a statistical approximation. It also leaves open the task of correlating the output to meaningful symbolic information. The *copilot* runtime integrity monitor [64] system is an example of a discrete interval PCI memory sampling solution that must (internally) translate the PCI address space into logical Linux kernel data structures.

Instrumentation Instrumentation is the process of inserting additional measurement collecting instructions into a program. The application designer can manually instrument the system or use a choice of tools (e.g. gprof [31], valgrind [61]) to automate the instrumentation at various stages of the application lifecycle. In the *Linux Integrity Measurement Architecture* (IMA) system [73], application interpreters (such as the `ld.so` dynamic loader) are manually instrumented with a measurement mechanism to measure executable content.

Virtual machine In theory, measurements can be collected by any capable virtual machine. The Java and .net virtual machines use this feature to enforce security policies by walking the call stack at runtime [30]. Use of this technique has been proposed as a means to attest to behavioural semantics of the executing application [41].

5.2 Attestation architectures

A model is an abstraction of a system. In modelling the elements of a concrete attestation solution (Chapter 6) it is easier to study the high-level properties of the model independent of the accidental detail of the concrete instantiation. This report found four broad-based attestation architectures in the literature. The architectures can be distinguished in three ways: (i) the measurement collecting mechanism, (ii) the nature of the trust chain and (iii) the element to which the trust chain leads. We discuss attestation to a platform configuration (Section 5.2.1), attestation to a code entity (Section 5.2.2), probabilistic attestation to application integrity (Section 5.2.3) and attestation to a (computational) process (Section 5.2.4).

5.2.1 Event chaining

In this model, the transitive trust chain is relayed as a sequence of security-related events beginning with an initial known "good" configuration. In TCG S-RTM, the initial configuration is the known default PCR values and the CRTM.

- The advantage of the model is that in its base form it is extremely open - being applicable to most platform architectures.
- This is simultaneously a disadvantage because we can speculate that to be useful, the measured event (e.g. execution value) must relate to the logical structure of the application and consequently to its meaningful interpretation by the challenger.
- The second disadvantage is the combinatorial explosion of potential configurations. If, using TCG load-time semantics, Layers 0, 1 and 2 consist of n, m, and p software entities respectively then the number of configurations is far greater than $n.m.p$. For Layer 2 it is, for any subset of r software entities, the number of ways the r entities can be chosen (i.e. executed in any order) from the set of p installed software entities (Figure 5.2). In reality, this applies only to a platform layer in which the order of software execution is uncontrolled. Even for lower layers (e.g. the operating system boot sequence), however, patches and field upgrades will lead to a combinatorial explosion of potential configurations.

Figure 5.2: The number of possible configurations is a very large number - the sum of the permutations of all subsets of r entities from the set of p possible entities.

$$\sum_{r=1}^{n} \binom{p}{r}$$

- Finally, there is the issue of how to identify an event. According to Proudler [66]:"We know of no practical way for a machine to unambiguously distinguish arbitrary software other than to measure it (create a digest of the software using a hash algorithm) ...". Measuring some event types (e.g. input of configuration data) is questionable even if the event is known to impact system integrity. This is because a challenger cannot predict a value for the event that would preserve integrity [73].

5.2.2 Certificate chaining

In this model the chain of trust is supported by a certificate chain (e.g. NGSCB [63], IBM 4758 [78], Terra [29]) starting with a trusted base and terminating at the attested entity. A certificate here, means a freshly signed statement by a layer N-1 entity attesting to the binding between a public key and a layer N entity. Then, assuming that Layer 0 is trusted (or signed by a trusted root) each subsequent layer can be validated by following the trust chain. A layer

entity is identified by its *code identity* [63] which is a hash of the executable file (and optionally the configuration).

The key properties of certificate chaining are as follows.

- The validation algorithm is linear in complexity, being proportional to the length of the trust chain. Even, if there are n, m, p, ... configurations for Layer 0, 1, 2, ... entities the problem of defining and storing the list of known "good" configurations is proportional to $(n + m + p + \ldots)$. This is a useful feature.
- The scheme supports "owner" semantics and code authentication - an entity can be identified by proof of possession of the corresponding (owned) private key.
- Techniques exist, e.g. transition certificates, to update a lower layer in which "the name of the new version is spoken by the old version" [79].
- Certificate chaining carries two substantial disadvantages.
  - It is an attestation of a static trust chain. The relying party must accept (the risk) that the attested trust chain is not indicative of the current (dynamic) configuration.
  - To draw a reasonable conclusion about the trustworthiness of the (software-based) Layer 0 attestation, an additional hardware mechanism is needed to vouch for its trustworthiness, for example using the TCG attestation method in which the TPM signs the Layer 0 identity (as captured in the TPM PCRs). To ensure that Layer 0 starts the (software) chain of trust a new CPU instruction is needed (see Figure 4.2 in [63]).

5.2.3 Probabilistic integrity

This model is best illustrated with an example.

1. The **relying party** sends a computational challenge to the target.
2. The **verification entity** is a monitor process on the target host that computes the challenge and in so doing effectively executes a *proof* of its own integrity. It then measures (e.g. hashes) the **target entity** and sends the measurement and proof of integrity back to the relying party
3. The relying party simulates the computational challenge locally to check that the measurement entity is behaving as expected.
4. If the proof by the challenged platform is successful (for example the computational challenge could not have been interfered with because it ran within the expected time) then the relying party trusts the target measurement.

In a pure software implementation of this model the relying party must employ a bag of tricks and assumptions to deal with: (i) man-in the middle attacks, (ii) predictability of execution time (e.g. requiring supervisor mode with interrupts disabled [75]), (iii) processing power of the target [52], (iii) assumptions about platform architecture (e.g. no virtual memory [76]) and (iv) network delays. This places unrealistic constraints on its practical realisation [74].

Schellekens et al. [74] describe a TPM-assisted method in which the TPM time-stamps the start and end of the verification entity's proof of integrity.

5.2.4 Process integrity

The BIND (Binding Instructions aNd Data) proposal [77] is aimed at addressing a number of TCG attestation shortcomings in the context of distributed systems where attestation to the integrity of data processing code is a primary concern. A generic model can be constructed from the BIND system as follows. An attestation service executes in a privileged execution environment. A secure kernel has been touted [77] as the service environment. The attestation service is invoked at the process entry and exit points and dispatches the process code to a secure sandbox.

1. An authenticator, attached to each piece of input data, is checked at the entry point to a layer N attested process. In this context a process is not an OS process, but a logical critical section (or transformation) that is both a consumer and producer of data.

2. The layer N process code is first measured and then cryptographically bound to its input data (e.g. by hashing) to form a *process-input binding*.

3. The attestation service dispatches the process code to a secure sandbox that guarantees its runtime integrity.

4. On and at the time of exit, the output (data) plus the process-input binding is signed to form an authenticator to be consumed by the next process in the distributed chain.

A critique of the general model is presented here. BIND is presented in more detail in Section 6.1.

- Only the layer N critical code section is measured and attested. The platform stack is not measured.

- The chain of trust is (loosely): the secure kernel, attestation service, process code. Therefore the attestation service and service host need to be separately attested (e.g. using TCG load-time attestation). Chapter 4 highlighted the pitfalls of using TCG attestation as an attestation to an entity.

- Under the assumption that the secure kernel cannot be subverted then the **model** provides a specific solution to the following problem: how to attest to a single transaction - input,

transformation and output - and avoid the time-of-check-time-of-use vagaries of TCG attestation.

- The scope of the critical section and the placement of the attestation service represents a fundamental design dilemma on modern architectures.
  - If the scope excludes the callee graph then the process is only partially attested.
  - If the scope includes the callee graph then all code sections in the callee graph need to be measured. This makes a minimalist secure kernel an unsuitable location for an attestation service (e.g. it needs its own virtual memory subsystem to support on-demand paging to measure callee code sections).

5.3 Toward high-order attestation architectures

In order to overcome some of difficulties with TCG attestation a level of indirection can be added. There are two approaches.

- Attest to a policy on the challenged platform (e.g. [2, 48]).
- Layer the attestation mechanism by introducing an additional entity into the system view (e.g. [16, 50, 65]) or by attesting to a higher-level abstraction [72].

This is discussed in the next chapter.

Chapter 6

Attestation in practice

6.1 BIND

The BIND (Binding Instructions aNd Data) attestation service [77] provides fine-grained attestation to data transforming code (Section 5.2.4). In doing so, it aims make the hash verification step more tractable and reduce the scope for time-of-check-time-of-use vulnerabilities.

Figure 6.1: BIND Attestation Service using the services of a TPM and secure execution environment (from [77])

1	PROCESS → SK:	ATTESTATION INIT(input data addresses, size of process code)
2	SK:	disable interrupt
3		verify authenticator on input data
4	SK → TPM:	PCRReset
5	SK → TPM:	PCRExtend(input data addresses, process code)
6	SK → TPM:	h ← PCR Read
7	SK:	set up secure execution environment for the process
8		enable interrupt
9	SK → PROCESS:	yield control
10	PROCESS:	perform transformation on input data
11	PROCESS → SK:	ATTESTATION COMPLETE(output data addresses)
12	SK:	disable interrupt
13		verify that the call comes from the process being attested
14	SK → TPM:	sign(output data, h)
15	TPM → SK:	$\sigma_{signK}(outputdata, h)$
16	SK:	clear protection
17	SK:	enable interrupt
18	SK → PROCESS:	$\sigma_{signK}(outputdata, h)$

The BIND mechanism is depicted in Figure 6.1. BIND executes as a service within a secure

kernel (SK). The simplifying assumption is that the process code is laid out in contiguous memory and the callee graph does not need to be traced. The signing key, $signK$, is signed using a TPM identity key, and "locked" to the secure kernel locality.

The programmer annotates the code entry (ATTESTATION_INIT) and exit (ATTESTATION_COMPLETE) points to invoke and complete the attestation service (steps 1 and 18) via secure kernel interrupts. The SK process the request as follows. The Secure Kernel first verifies the authenticator on the input data (step 3) If this fails an error is returned. In steps 4 and 5 the SK extends a PCR with measurements over the input data and process code. In step 7, the SK creates a secure execution environment, enables the Secure Kernel Interrupt (step 8) and yields control to the process (step 9) to process the input data (step 10). On exit, the process asks the attestation service to sign the output data (step 11). The SK performs sanity checks (step 13) and delegates the signature computation over the output data and the process-input binding (from step 6) to the TPM (step 14). $\sigma_{signK}(\ldots)$ denotes a signature with the TPM signing key over the input data. The SK disables the protection mechanisms established for the attested process (step 16) and and returns control to the process (step 18).

Analysis The model for BIND has been discussed in Section 5.2.4. It fulfils a specific use case (Section 4.1.1). The BIND implementation verifies the authenticator on the input data. In this way an outgoing authenticator vouches for the previous transformation and by induction for the entire chain of transformations. Verification is therefore highly efficient with complexity $O(1)$. The TPM *extend* operations are superfluous since the TPM is not used to *quote* PCR values. The operations could be performed by the secure kernel without loss of security.

6.2 Property-based attestation

Property-based attestation (PBA) was independently proposed in [65, 72] as a means to address some of the deficiencies of TCG attestation. Some examples are [65, 72]:

- **Privacy**. Exact configuration information[1] of (hardware and software) is unnecessarily disclosed. In turn it becomes possible to (i) discriminate (by applying a restrictive business model), (ii) (partially) link transactions and (iii) use attestation as a tool for attack reconnaissance.
- **Scalability**. The number of all potential configurations is simply put, huge (Chapter 5). Secondly, the smallest change to a single component invalidates the entire configuration.
- **Openness**. Binary attestation fixes the configuration to a few popular implementations. Alternative configurations render protected secrets inaccessible.

[1]This is not quite true as the attesting platform can restrict the fields reported for each event in the SML. In this case the relying party would need to take the additional step of identifying the software using, for example, a validation credential.

The underlying idea is that a platform only needs to attest to "properties" which can then be tested against the relying party's security requirements. Multiple hardware/software configurations may still provide the same properties and therefore fulfil the same security requirements. The exact definition of a "property" is somewhat nebulous. It could be "the **absence** of certain vulnerabilities" [65] or built-in measures to enforce privacy laws [72].

Whilst Poritz et al. [65] discuss an architecture, Sadeghi and Stüble [72] focus on mechanisms.

The rest of this section is structured as follows. The proposal in [65] is summarised and analysed first. Then, since the work in [72] is the basis for subsequent investigations [17, 16] it is used to explore the remaining PBA themes.

- In [65] trusted third parties, called *property certifiers* issue credentials that associate security properties with individual **components** (or configurations).
- An inline trusted third party, called a *verification proxy*, is interposed between the attesting and relying platforms. The verification proxy may be hosted on a separate machine or it may be co-located on the attesting platform, in a secure execution environment. The purpose and scope of a verification proxy is as follows.
  - It verifies that binary attestations satisfy the security policy of the relying party.
  - It acts as a binary attestation consumer to the attesting platform and property-based attestation producer to the relying platform.
  - In doing so, it anonymises the attesting platform.
  - Its integrity must be trusted or it must be separately binary-attested.
- In the end-end protocol a property-suite negotiation is envisaged as part of a pre-attestation handshake in which the exchange of verifier and attestor policies leads to a refinement of the actual properties that are attested.

Analysis

- The concept of certifying properties for individual components in isolation is questionable. System vulnerabilities are the the result of combinations of weaknesses which define a potential attack path.
- Since the TPM quotes the PCR values there is no means to "transfer" an AIK signature onto a property attestation. An attestation is reduced to an assurance by a trusted third party.
- The verification proxy is required to be highly available.

- There needs to exist a TTP who is willing to act as an evaluation facility for every possible component. The scalability problem is simply shifted[2] (not reduced) onto a TTP (property certifier).
  - The property certifier (it seems) derives no business benefit but significant cost and liability (see also [6]).
  - The property certifier cannot know in advance which components (or configurations) are likely to be used.
  - The model of a property certification authority is inconsistent with the the fast and loose nature of today's software computing, for example of frequent releases by multiple distributors.

Sadeghi and Stüble [72] describe a PBA scheme as a system in which a trusted third party, called the *certificate issuer*, publicly endorses a certificate that binds a configuration, S_0, to a set of properties $\bar{P} = \{P_0 \dots P_n\}$. The trusted platform, consists of an enhanced component, dubbed TC+, that translates binary PCR metrics into a property attestation. TC+ may be realised in the following ways.

- The TPM is modified to perform the translation in which the property certificate ($cert_{TTP}(S', \bar{P}')$) is used to check whether the current configuration S matches S' and whether the relying party's requested properties $\bar{P}$ matches the properties $\bar{P}'$ endorsed by the certificate issuer.
- An inline Trusted Attestation Service (TAS) is introduced (similar to the aforementioned verification proxy) that runs on a small security kernel at the attesting platform.
- Cryptographic techniques are used to prove that a valid link exists between the binary configuration S_0 and property certificate over $\bar{P}$ without revealing the platform configuration. This is the approach taken in [16].

Analysis The distinguishing feature of Sadeghi and Stüble [72] is the emphasis on certifying whole configuration-property set mappings and the range of mechanisms for achieving PBA. It is acknowledged that some mechanisms introduces a new set problems - for example the trust model introduced by the TAS, the TAS integrity and managing updates to the TAS. It is argued below, however, that there are more far-reaching, non-trivial problems which both the original [72] and subsequent improvements [16, 17] fail to address.

- The solution remains dependant on a TTP evaluation facility.

[2]This is a general problem of PBA and is recognised in [17] but their solution moves the problem back to the relying party.

- The scalability problem remains but it is more fragmented: the property certifier assumes the task of verifying many binary configurations whilst the relying party must be prepared to manage many acceptable property configurations.
- The solution is equally fragile. The smallest component change invalidates the entire configuration and the property certificate.
- It is the author's opinion that pushing high-level abstractions down the software stack is poor design practice.
- The appearance of enhanced privacy is a false economy. PBA replaces one set of identifying information (a binary configuration) with another albeit more common identity (a property-set). Studies (e.g [67]) have cautioned against the illusion of privacy in cases where anonymised datasets can be correlated. If a relying party can link transactions by configuration then she can do so by property configuration. The situation may be even worse, since under PBA, it is possible (in fact proposed) that property configurations will remain stable across software updates.
- TCG attestation allows the relying party to select the PCR indices, for example to attest a portion of the software stack. To overcome this a certificate issuer must predict the likely combinations of requested PCRs and generate multiple property certificates, each with its own set of derived security properties.

The last issue renders PBA fundamentally incompatible with TCG attestation. Incremental improvements have been made to PBA but it can be shown that these do not significantly address the fundamental problems.

In [16] a mechanism is introduced - similar to DAA - based on CL signatures and signature proofs of knowledge. The core idea is that the host proves that there is a valid link between the TPM binary attestation signature and a property certificate (from a certificate issuer) without revealing the platform configuration. This removes the need to trust a TAS but other requirements (e.g. the need for a TTP to certify configuration-property associations) remain.

In [17] a method is introduced based on ring signatures [69] that eliminates the need for the certificate issuer required in earlier schemes [65, 72, 53]. It is assumed that, before the protocol begins, the attestor and challenger have agreed on a set of acceptable (binary) configurations. The TPM then shares a signer role with the host to prove that its configuration is in this set. An analysis of the resulting context follows.

- The certificate issuer has been removed.
- The prover and verifier must agree, in advance, on a set of acceptable configurations. Clearly the size and the elements of the set affect the privacy of the attestation and gives the verifier plenty of scope to cheat. Both cases require an external control to limit the scope for information leakage.

1. The verifier can propose an artificially small set.
2. From a list of known configurations, the verifier can set all but one element to random values, and iterate through the list.

- The end result is to revert to a situation in which the relying party is forced to predict binary configurations.

6.3 TCG-based integrity management architecture (for Linux)

Sailer et al. [73] present an experience report in which they extend TCG binary, load-time attestation to userland applications on Linux. The purpose of the system, called the Integrity Measurement Architecture (IMA), is to allow the relying party to obtain assurance of the integrity of runtime components and the system environment.

The design consists of three elements: a measurement mechanism, integrity challenge mechanism and an integrity validation mechanism.

Measurement mechanism The basic idea is to (i) use a modified BIOS and GRUB bootloader [58] to measure the the initial kernel code, (ii) modify the kernel to measure changes to itself and (iii) modify the runtime system to take integrity measurements when user-level executable con- tent is loaded, but before it is executed. The storage measurement list is maintained within the kernel as an ordered list and the TPM reporting function is used to protect this list.

In their design and implementation the following points are instrumented to perform a measurement call

- Modifications to the kernel are measured by instrumenting the `insmod` program.
- User-level programs are loaded through the user-level loader. If the target is a statically linked binary then this is measured by an instrumented `execve` and nothing else needs to be done.
- Dynamically loadable libraries are loaded by the dynamic loader (`ld.so`) which needs to be instrumented.
- Scripts are loaded via `execve` in which case the loader is the script interpreter (e.g `/bin/perl`) which is measured by `execve`. Script interpreters may load additional code but since these are not instrumented the additional code is not measured.

The measurement function is added to the kernel and exported as a new system call, `measure` that takes as its single argument, the file descriptor of the code to be loaded. From the file descriptor, an inode lookup leads to the data blocks over which a SHA1 hash can be computed. It is possible to change the file descriptor-inode relationship between the `measure` call and the

time at which the code is actually loaded. The authors argue, however, that the code-loading code will have been measured and will be trusted (or not trusted) not to change this relationship.

The function of the storage measurement list is changed to be an ordered set and a caching optimisation is applied such that a new measurement is taken only if the file has not been measured before or might have changed. In other words the system does not record new events that would present the same measurement value[3]. This keeps the size of the measurement list down but is a departure from TCG semantics.

Integrity challenge and validation mechanisms The challenge and validation procedures follow standard semantics (see Chapter 3). For example, each history entry in the measurement list is tested independently. The challenger whitelists known good measurements using a local database. It is expected that a policy exists that specifies the actions for unknown or untrusted fingerprints.

Analysis IMA is a valuable experimental report. It is argued later that the lack of similar reports is a serious shortcoming toward find practical attestation solutions. To summarise:

- The report highlights two issues in the measurement of non-executable data (e.g. configuration files).
    1. It can be difficult to determine when and which file is being used and for what purpose. This is partly due to the fact that the OS (rightly) cannot know the integrity impact of a file load event by a user process.
    2. Unstructured data (for example, data flowing over a network interface) cannot be identified and therefore the challenger cannot predict values that preserve integrity. This leads to an unsatisfactory situation in which run-time behaviour is not captured.
- The measurement scope is the entire system - all entities must be measured irrespective of whether they actually impact the integrity of an application. This requires the verifier to know all measured entities.
- To address the two aforementioned shortcomings the IMA has been modified in the Policy-reduced IMA (PRIMA) system to measure information flow integrity [48]. The benefits gained equals the simplicity lost (for the verifier). For example, the verifier must validate measurements of the MAC policy, code and a list of trusted subjects and code-subject mappings.
- In order to keep the measurement list stable, IMA does not record new load events having the same measurement value.
- IMA assumes that the relying party can attach meaningful semantics to binary measurements.

[3]The storage measurement list and TPM PCR state is unchanged in this scenario.

- To some extent, the relying party must understand the architecture of the attesting system. For example, she must know that the `insmod` is a distinguished entity that can affect kernel integrity and must be able to identify which fingerprints identify `insmod` programs that are instrumented with the `measure` system call.

- In order to fully extend load-time integrity measurements all user-level loaders and script interpreters need to be instrumented.

- IMA exposes a subtle implementation pitfall: if an executable is measured and loaded but then execution fails then the attested configuration is inaccurate. Furthermore since there is no corresponding "unload" event it is not possible to examine the event history to determine whether software is live at the time of reporting.

6.4 Virtual machine based attestation

6.4.1 Semantic remote attestation

Semantic remote attestation is a phrase coined in [41] to describe a language-based, virtual machine (VM) approach to the attestation of program behaviour. The proposal is predicated on the assumption that what is desired (by the relying party) is attestation of sophisticated behavioural properties.

The design is built upon a trusted virtual machine (TrustedVM) and a TrustedVM Attestation Service that acts as a reference monitor. These need to be attested via a separate channel. The TrustedVM hosts platform-dependent[4] code. It is suggested that this creates a favourable context for attestation since the virtual machine can express a range of high-level properties about the interpreted application. Amongst these are: (i) properties of classes, (ii) attestation to runtime behaviour, (iii) attesting to arbitrary properties (e.g. if the challenger supplies a mobile test harness) and (iv) attesting to elements of the system at large.

Semantic remote attestation presents three key challenges.

1. To date, few properties (beyond the constraints of the language) are provable by so-called language-based VMs. For example, the Java Bytecode Verifier proves only that the loaded application obeys the Java language rules.

2. The architecture requires the VM and attestation service to be trusted. These can be attested via a separate channel.

3. If the solution is to be general and open there needs to exist an ontology to reason about the range of behavioural semantics that are envisaged.

[4]The code is described as "platform-independent" but this is inaccurate. For example, Java applications are platform-dependant - being runnable only on the Java Platform. It is the JVM that is ported to different operating systems.

The ideas behind VM assisted attestation are more broadly applicable [22] and this is explored in the following sections.

6.4.2 Terra

The Terra[5] architecture is based on a virtual machine monitor (VMM) which hosts "open-box" and "closed-box" virtual machines (VMs) [29]. The closed-boxed paradigm is an attempt to simulate the benefits of closed hardware platforms by leveraging the VMM's support for isolation and inherently small TCB. Since the Terra VM presents a virtualised hardware interface each VM appliance can be tailored - from a fully-fledged commodity operating system with accompanying application suite to a single high assurance application with no OS support.

The Terra VM is trusted to offer an authentication service to the software running in the VM. The attestation mechanism is a certificate chain from the hardware to the VM [29] (see the method described in Section 5.2.2) which attests to the VM appliance. This limits the flexibility of the approach. It is envisaged that a certificate analogous to a validation credential is used to identify each element in the certificate chain. This creates an unusual (albeit somewhat unrealistic) software model: software vendors release (and sign) entire VMs.

Attesting to the VM makes measurement hard in time and space. To solve the performance problem Terra defines the unit of measurement to be fix-sized 4kB blocks which are hashed separately. A VM descriptor is a hash over all the block hashes. This makes measurement hard in memory - measuring a 4GB entity with block size of 4kB yields 20MB of hashes [29]. To overcome the performance hit, Terra adopts a lazy evaluation approach to verifying a VM image.

Furthermore, because these measurements are representative of blocks, it is difficult to interpret varying measurement values. Finally, due to the block-sized granularity it is difficult to relate varying values to logical entities.

6.4.3 Related work

TPod or "Trusted Platform on demand" [58], is a layered architecture built on TCG hardware using SELinux, OSGi[6] and WS-Security protocols. The BIOS and GRUB bootloader are modified to allow authenticated boot of the SELinux kernel. The platform stack up to the SELinux kernel is TCG-attested. TPod provides pseudo-isolation of non-system applications by assigning each to a domain whose resources (data and execution) are protected from other domains. Java applications are also "domain-separated" within a single JVM instance using the OSGi framework. The TPod designers propose an interoperability layer that facilitate attestation in a heterogeneous environment using the WS-Security protocols. However, it is not discussed how this architecture abstracts the raw integrity measurements in an attestation transaction.

[5] Terra was not tested on TCG hardware

[6] www.osgi.org

vTPM [10] is a Xen implementation of a hypervisor-based architecture that virtualises the TPM (vTPM) for each VM instance. The vTPMs are hosted in a vTPM Manager domain (Xen domain-0). The bootloader, hypervisor and vTPM subsystem are reported to PCRs 0 to 8 (on the real TPM). To attest to a VM instance PCRs 0 to 8 (from the real TPM) are copied to PCRs 0 to 8 in the vTPM (and are marked read only). Higher-numbered vTPM PCRs (9 and onward) are assigned to the guest OS. In this way the entire platform stack is attested. The report identifies three core challenges to fully virtualising a TPM.

- Preserving the TCG semantics of the endorsement, storage root and attestation identity keys is not straightforward. The issuance of these keys is singled out in this respect. For example, in one solution a hardware AIK is used to quote the virtual EK which in turn is used to obtain a virtual AIK.
- VMs support capabilities such as suspend/resume and migration to another platform.
- The trust chain in an attestation needs to be interpreted.
- VMs can be copied but vTPMs must be migrated such that the source instance is destroyed. A protocol based on migratable storage keys is defined for this purpose.

Sadeghi et al. [71] present a proposal to extend the property-based paradigm to TPM virtualisation by using a secure hypervisor. To accomplish this they expand the nature of PCRs by wrapping the register set inside a "provider" and attempt to generalise the extend mechanism. To realise property-based virtualisation a "CertificateProvider" is defined which translates binary measurements of components into properties by "looking up" a property certificate. This is a departure from the original PBA scheme in which a property certificate attests to a whole configuration.

NGCSB [23] is an architecture designed around a cheap cryptographic processor, called the security coprocessor (SCP), a lightweight isolation kernel (called Nexus), a mass market OS (dubbed the left-hand side) and a set of high assurance components (the right-hand side). Attestation (of the high assurance component) follows the certificate chaining model (Chapter 5). The quote operation signs the platform's code identity and an input string using the platform's signing key. The requesting program (high assurance guest) sends this signature plus a supporting certificate chain to the remote party.

6.5 Hardware and software threats

The goal of TCG attestation is to protect information from software attack. Nevertheless, the relative ease at which TCG hardware can be attacked should be a cause for concern.

Kauer[51] demonstrated that by manipulating the Low Pin Count (LPC) Bus a hardware reset could sent to the TPM independent of the rest of the platform. In the same report Kauer used the fact that the BIOS is patchable (on many machines) to change the BIOS TPM driver

and effectively swap in a new CRTM. Kursawe et al. [55] look at a number of passive attacks (for example sniffing the keys used for bulk encryption) but also speculate on active attacks.

- The TPM is subject to a reset attack (as above) which allows an adversary to rewrite the TPM PCRs with any "boot sequence" of her choosing. and thereby to present any platform configuration.
- The TPM could be temporarily disconnected during the boot sequence, for example to hide rogue programs.
- The TPM's PCRs could be manipulated directly.
- The locality assertion of a privileged process is signalled by different addressing modes on the LPC bus. If this can be copied then an adversary could assert special privileges for any command.

Sparks [80] reproduced the TPM reset attack and also provided evidence that the CRT-based RSA engine in the TPM under investigation was susceptible to timing attacks on the RSA private key.

Sparks shows further how load-time TCG semantics (in the Linux IMA [73]) are inadequate under most threat assumptions. The attack assumes that an adversary can run arbitrary code in kernel mode. To see that this assumption is reasonable, first observe that many userland vulnerabilities allow an adversary to elevate their privilege to "superuser" status. The attack works by copying the process page table entry into the kernel page table, modifying the copied entry as writable, and then writing arbitrary data to it, thereby modifying the `txt` segment of the target program.

Bratus et al. [12] argue that since measured memory can be modified at run-time (after the load-time measurement) the trustworthiness of a platform should be enforced with respect to its memory operations. To this end, a hardware-based memory event trapping framework is proposed and prototyped in Xen that allows the TPM to "listen" to unauthorised modifications to page table entries and physical RAM.

Chapter 7

Conclusion

In examining the state of play in TCG attestation this report has looked at the following questions.

1. How is attestation (to be) used?
2. Which are the attestation models and what can we say about their general properties?
3. What do attestation systems look like in practice and are these systems practical?

7.1 Using attestation

As demonstrated in Chapter 4 it is tempting to use TCG attestation in a manner that assumes the semantics of the certificate chaining model - for example, that an entity can be identified by proof of possession of the corresponding private key. Inherent in this assumption is that secrets are bound to an owner entity. This is of course not true since secrets are contingent upon platform configuration. The second and third temptations are to interpret an integrity report as a capability query and as proof of the liveness of a software entity. Finally, in an open computing environment the scale of platform and software heterogeneity makes binary attestation-based pervasive trusted computing implausible.

7.2 The TCG attestation model

In an examination of event-based attestation models (Chapter 5) we see that, unless event measurement is constrained, the combinatorial explosion of potential configurations and the size of the resultant history list means that S-RTM based load-time measurement simply doesn't scale. This is supported by the experience report of the Linux Integrity Measurement Architecture (IMA) which artificially reduced the history list to make integrity validation tractable (in terms of complexity) for the challenger. Secondly, it is difficult to derive an accurate picture of the platform state from the platform configuration.

The classical model - certificate chaining - works in closed systems because the software and hardware platform is (usually) bespoke and is administered in a controlled manner by a single security authority. Under these conditions it is valid to make certain assumptions about behaviour of software, the isolation properties of the system and the integrity of the hardware-rooted trust chain.

In the open architecture targeted by TCG, these trust assumptions do not hold. An open environment must be underpinned by a demonstrable hardware-based root of trust.

7.3 Attestation in practice

Chapter 6 discussed the design and implementation of various attestation schemes. The major flaw in load-time attestation is just that: it is at *load-time*. Given the propensity for userland software vulnerabilities to end up as full (run-time) platform breaks, load-time detection is inadequate. Extending the scope of event measurement to run-time events is an impractical response since verification needs to be tractable and scalable. The issue of tractability arises for two reasons. Firstly, as the event history grows larger it becomes less representative of the current platform configuration. Secondly, the verifier must reason about a large number of potential interactions that impact integrity.

The IMA experience is also valuable for confirming a third major pitfall of load-time attestation. A load-time measurement system cannot predict the integrity impact of a process on the rest of the system and on other processes. It must play "safety" and measure all events but this is not scalable. The alternative is to attest to some enforcement of policy which, in the PRIMA experience, has been used to reduce the number of events measured. With PRIMA, however, the reduction in measurement comes at a price: the system is less open because the relying party must possess more detailed information about implementation, and the verification step is arguably more complicated.

Property-based attestation is an attempt to overcome many of the shortcomings of binary load-time attestation. One of its key goals is to attest to semantics that are more relevant to the abstractions imagined by the user. The first stumbling block to PBA is to reach agreement on the definition of "property" and the definition of "properties". For example, it has been suggested that properties can be used to describe the absence of a thing. PBA too comes at an impractical price: "vanilla" PBA relies on costly evaluations by a TTP. To date only a handful of commodity software has been evaluated to some degree of assurance. Secondly, it is incompatible with the challenge mechanism. The challenger is free to choose which PCRs are quoted but the property certificate issuer fixes the PCR selection. Finally, it is argued in this report that privacy is not significantly improved when compared to standard attestation.

In reaping the benefits of virtualisation designers have a choice: lock the VM to a single physical platform or virtualise the TPM. A virtualised TPM, however, should be strictly bound to its associated VM instance. This has led, inevitably, to an adjustment of the relationship

between a virtual endorsement key, storage key, attestation identity keys and the physical TPM. As a consequence the relying party can no longer base a trust decision on a TCG PKI, but must reason about the relative trustworthiness of the virtualised trust chain.

7.4 Future directions

The sum total of current efforts is that retrofitting attestation to a commodity OS and conventional platform architecture has been unsuccessful.

To motivate the following discussion we make a number of general observations.

- Behavioural characteristics cannot and should not be inferred from the measurement of low-level components since (i) the manufacturer offers no behavioural guarantee [81] and (ii) the purpose of a low-level component (such as an option-ROM) has no equivalent high-level abstraction that is meaningful to a user. This is a direct consequence (and benefit) of layered software.
- Commodity OSes are highly fallible and it is inevitable that attempts to bolster their trustworthiness can only target classes of attacks [12, 80] leaving open the task of plugging other classes of attacks. This piecemeal approach is largely unsatisfactory. For example, the control mechanism itself needs to be attested but is likely to be architecture and operating system specific.
- An attested configuration can only provide an assurance of integrity. It is in the relying party's application of policy that trust in a platform may be inferred.

In order to balance the need for higher assurance with the goal of practical verification in an open architecture there are two complementary approaches which **preserve** the existing TCG effort in binary attestation.

1. Extend the scope of the TCG architecture to encompass protection capabilities at the platform level and allow developers to express security requirements using new primitives that expose these capabilities (see e.g. [12]).
2. Exploit isolation capabilities to reduce the size and scope of the attestable trusted computing base by reorganising applications into distinct parts that do and don't need to be trusted.

Extending TCG architecture This approach is motivated by the observation that whilst TCG attestation is not equipped to attest to a dynamic system, simply expanding the scope of event measurement is impractical. Bratus et al. [12] have implemented a proof of concept that allows the TPM to "listen" to unauthorised integrity altering events. Significant improvements have been realised when application programmers have been provided the tools (e.g. `chroot`) to express security policy [12]. For example, programmers could be given the means to state their security goals and the underlying software/hardware should enforce this policy.

Isolation and new application paradigms Firstly, note that while binary-level integrity assurance does not provide the abstract security properties appropriate to userland applications it is entirely appropriate for low level entities such as the BIOS, boot loader and device drivers which carry no behavioural assurance. Secondly, virtualisation technologies exist and are highly efficient[9]. Thirdly, a considerable degree of simplification can be achieved by dividing an architecture into an untrusted part (the ill-defined "messy") and trusted part (having well-defined security and assurance requirements)[28, 23]. Lastly, the principle of division of privilege is well versed and has led to noticeable improvements in application security[1].

In this paradigm, applications have distinct privileged and unprivileged parts. The privileged part runs in an isolated compartment in a hypervisor. If the hypervisor is an isolation kernel [23] its security can be evaluated. With D-RTM, attesting to the binary integrity of the hypervisor becomes more practical and a layer (possibly within the hypervisor) can support higher-level attestation semantics. In the context of trusted computing, examples of this style of application deployment already exist [24].

7.5 Epilogue

In the author's opinion a potential risk is that privacy concerns, whilst important, should not provide the dominant technical drive toward improving TCG attestation. It has been shown that even anonymised data can be deanonymised and individual persons identified (e.g. [67]).

Researchers should focus on what practical guarantees can be extracted from attestation: sound integrity guarantees seem to be most realistic goal. Experience reports from the research and development community in particular are in thin supply.

[1] Google "privilege separated openssh".

Bibliography

[1] Four grand challenges in trustworthy computing: Second in a series of conferences on grand research challenges in computer science and engineering. Technical report, Computing Research Association, November 2003.

[2] Masoom Alam, Xinwen Zhang, Mohammad Nauman, Tamleek Ali, and Jean-Pierre Seifert. Model-based behavioral attestation. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, New York, NY, USA, 2008. ACM.

[3] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, Washington, DC, USA, 1997. IEEE Computer Society.

[4] ARM. TrustZone Technology Overview. http://www.arm.com/products/security/trustzone/, February 2009.

[5] Boris Balacheff, Liqun Chen, Siani Pearson, David Plaquin, and Graham Proudler. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[6] Shane Balfe, Eimear Gallery, Chris J Mitchell, and Kenneth G. Paterson. Challenges for Trusted Computing. Technical Report RHUL-MA-2008-14, Department of Mathematics, Royal Holloway, University of London, 2008.

[7] Shane Balfe and Kenneth G. Paterson. e-EMV: emulating EMV for internet payments with trusted computing technologies. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable trusted computing*, New York, NY, USA, 2008. ACM.

[8] Endre Bangerter, Maksim Djackov, and Ahmad-Reza Sadeghi. A demonstrative ad hoc attestation system. In *ISC '08: Proceedings of the 11th international conference on Information Security*, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA, 2003. ACM.

[10] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: virtualizing the trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2006. USENIX Association.

[11] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, 2002.

[12] Sergey Bratus, Nihal D'Cunha, Evan Sparks, and Sean W. Smith. TOCTOU, Traps, and Trusted Computing. In *Trust '08: Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies*, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct Anonymous Attestation. Technical Report HPL-2004-93, HP Labs, 2004.

[14] Ernie Brickell, Jan Camenisch, and Liqun Chen. The DAA scheme in context. In Chris J Mitchell, editor, *Trusted Computing*, number 6 in IEE Professional Applications of Computing Series, chapter 5. The Institute of Electrical Engineers, London, UK, 2005.

[15] David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn. *A Practical Guide to Trusted Computing*. IBM Press, December 2007.

[16] Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A protocol for property-based attestation. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, New York, NY, USA, 2006. ACM.

[17] Liqun Chen, Hans Löhr, Mark Manulis, and Ahmad-Reza Sadeghi. Property-Based Attestation without a Trusted Third Party. In *ISC '08: Proceedings of the 11th international conference on Information Security*, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] Alexander W Dent and Geraint Price. Certificate management using distributed trusted third parties. In Chris J Mitchell, editor, *Trusted Computing*, number 6 in IEE Professional Applications of Computing Series, chapter 9. The Institute of Electrical Engineers, London, UK, 2005.

[19] Edsger W. Dijkstra. The structure of the "THE"-multiprogramming system. *Commun. ACM*, 11(5), 1968.

[20] D. Dolev and A. C. Yao. On the security of public key protocols. In *SFCS '81: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, Washington, DC, USA, 1981. IEEE Computer Society.

[21] Mark Dowd. *Application-Specific Attacks: Leveraging the ActionScript Virtual Machine*, April 2008.

[22] Paul England. Practical Techniques for Operating System Attestation. In Peter Lipp, Ahmad-Reza Sadeghi, and Klaus-Michael Koch, editors, *Trust '08: Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies*, Berlin, Heidelberg, 2008. Springer-Verlag.

[23] Paul England, Butler Lampson, John Manferdelli, Marcus Peinado, and Bryan Willman. A Trusted Open Platform. *Computer*, 36(7), 2003.

[24] Sebastian Gajek, Ahmad-Reza Sadeghi, Christian Stuble, and Marcel Winandy. Compartmented Security for Browsers - Or How to Thwart a Phisher with Trusted Computing. In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, Washington, DC, USA, 2007. IEEE Computer Society.

[25] Eimear Gallery. An overview of trusted computing technology. In Chris Mitchell, editor, *Trusted Computing (Professional Applications of Computing) (Professional Applications of Computing)*, chapter 3. IEEE Press, Piscataway, NJ, USA, 2005.

[26] Eimear Gallery. Secure delivery of conditional access applications to mobile receivers. In Chris Mitchell, editor, *Trusted Computing (Professional Applications of Computing) (Professional Applications of Computing)*, chapter 7. IEEE Press, Piscataway, NJ, USA, 2005.

[27] Eimear Gallery. Authorisation Issues for Mobile Code in Mobile Systems. Technical Report RHUL-MA-2007-3, Department of Mathematics, Royal Holloway, University of London, May 2007.

[28] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Flexible OS support and applications for Trusted Computing. In Klaus-Michael Koch Peter Lipp, Ahmad-Reza Sadeghi, editor, *Proceedings of the 9th Workshop TODO*, 2003.

[29] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA, 2003. ACM.

[30] Dieter Gollmann. *Computer Security.* John Wiley and Sons, Chichester, UK, 2nd edition, 2005.

[31] Susan Graham, Peter Kessler, and Marshall McKusick. gprof: a Call Graph Execution Profiler. In *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, 6. ACM SIGPLAN Notices, June 1982.

[32] Trusted Computing Group. *PC Client Specific Implementation Specification For Conventional BIOS Version 1.2*, July 2005.

[33] Trusted Computing Group. *TCG Generic Server Specification Version 1.0 Revsion 0.8*, March 2005.

[34] Trusted Computing Group. *TCG Infrastructure Workgroup Subject Key Attestation Evidence Extension, 1.0 edition*, June 2005.

[35] Trusted Computing Group. *TCG Software Stack (TSS) Specification Version 1.2*, January 2006.

[36] Trusted Computing Group. *TPM Main Part 3 Commands Specification Version 1.2 Level 2 Revision 103*, October 2006.

[37] Trusted Computing Group. *TCG Software Stack (TSS) Specification Version 1.2, Level 1, Errata A. Part1: Commands and Structures*, March 2007.

[38] Trusted Computing Group. *TCG Specification Architecture Overview Specification Revision 1.4*, August 2007.

[39] Trusted Computing Group. *TCG Mobile Trusted Module Specification Version 1.0*, June 2008.

[40] Trusted Computing Group. *TCG Trusted Network Connect TNC Architecture for Interoperability Version 1.3 Revision 6*, April 2008.

[41] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation: a virtual machine directed approach to trusted computing. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*, Berkeley, CA, USA, 2004. USENIX Association.

[42] Amir Herzberg. Why Johnny can't surf (safely)? Attacks and defenses for web users. *Computers and Security*, 28(1–2), February 2009.

[43] Intel. Intel trusted execution technology architectural overview. Technical report, Intel, http://www.intel.com/technology/security, February 2009.

[44] ISO. *Information technology - Open Systems Interconnection – Basic Reference Model – Part2: Security Architecture*, 1989. ISO 7498-2.

[45] ISO/IEC. *Information technology - Open Systems Interconnection – Basic Reference Model: The Basic Model*. International Organization for Standardization (ISO). ISO/IEC 7498-1.

[46] ISO/IEC. *Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers*. International Standards Organisation, 2006. ISO/IEC 18033-2:2006.

[47] Naomaru Itoi, William A. Arbaugh, Samuela J. Pollack, and Daniel M. Reeves. Personal secure booting. In *Proceedings of ACISP 2001, Sydney*. Springer-Verlag, 2001.

[48] Trent Jaeger, Reiner Sailer, and Umesh Shankar. PRIMA: policy-reduced integrity measurement architecture. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, New York, NY, USA, 2006. ACM.

[49] Bernhard Jansen, HariGovind V Ramasamy, and Matthias Schunter. Flexible Integrity Protection and Verification Architecture for Virtual Machine Monitors. In *The Second Workshop on Advances in Trusted Computing (WATC '06 Fall)*. IBM, November 2006.

[50] Yasuharu Katsuno, Yuji Watanabe, Sachiko Yoshihama, Takuya Mishina, and Michiharu Kudoh. Layering negotiations for flexible attestation. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, New York, NY, USA, 2006. ACM.

[51] Bernhard Kauer. Oslo: improving the security of trusted computing. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, Berkeley, CA, USA, 2007. USENIX Association.

[52] Rick Kennell and Leah H. Jamieson. Establishing the genuinity of remote computer systems. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2003. USENIX Association.

[53] Ulrich Kühn, Marcel Selhorst, and Christian Stüble. Realizing property-based attestation and sealing with commonly available hardware and software. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, New York, NY, USA, 2007. ACM.

[54] Klaus Kursawe. The future of trusted computing: an outlook. In Chris J Mitchell, editor, *Trusted Computing*, number 6 in IEE Professional Applications of Computing Series, chapter 11. The Institute of Electrical Engineers, London, UK, 2005.

[55] Klaus Kursawe, Dries Schellekens, and Bart Preneel. Analyzing trusted platform communication. In *In: ECRYPT Workshop, CRASH - CRyptographic Advances in Secure Hardware*, 2005.

[56] Adrian Leung, Liqun Chen, and Chris J. Mitchell. On a Possible Privacy Flaw in Direct Anonymous Attestation (DAA). In *Trust '08: Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies*, Berlin, Heidelberg, 2008. Springer-Verlag.

[57] Shiqun Li, Shane Balfe, Jianying Zhou, and Kefei Chen. *Enforcing Trust in Pervasive Computing with Trusted Computing Technology*, volume 4347/2006 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, January 2007.

[58] Hiroshi Maruyama, Frank Seliger, Nataraj Nagaratnam, Tim Ebringer, Seji Munetho, Sachiko Yoshihama, and Taiga Nakamura. Trusted Platform on demand (TPod). Technical Report RT 0564, IBM Research Tokyo, 2004.

[59] Jonathan M. McCune, Adrian Perrig, Arvind Seshadri, and Leendert van Doorn. Turtles all the way down: Research challenges in user-based attestation. In *Proceedings of the Workshop on Hot Topics in Security (HotSec)*, August 2007.

[60] Chris J Mitchell. What is trusted computing? In Chris J Mitchell, editor, *Trusted Computing*, number 6 in IEE Professional Applications of Computing Series, chapter 1. The Institute of Electrical Engineers, London, UK, 2005.

[61] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, New York, NY, USA, 2007. ACM.

[62] Andreas Pashalidis and Chris J Mitchell. Single Sign-On using TCG-conformant platforms. In Chris J Mitchell, editor, *Trusted Computing*, number 6 in IEE Professional Applications of Computing Series, chapter 6. The Institute of Electrical Engineers, London, UK, 2005.

[63] Marcus Peinado, Paul England, and Yuqun Chen. An Overview of NGSCB. In Chris J Mitchell, editor, *Trusted Computing*, number 6 in IEE Professional Applications of Computing Series, chapter 4. The Institute of Electrical Engineers, London, UK, 2005.

[64] Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2004. USENIX Association.

[65] Jonathan Poritz, Matthias Schunter, Els Van Herreweghen, , and Michael Waidner. Property Attestation : Scalable and Privacy-friendly Security Assessment of Peer Computers. Technical Report RZ 3548 (99559) 05-10-04, IBM Research GmbH, Zurich Research Laboratory, 8803 Ruschlikon, Switzerland, 2004.

[66] Graham J Proudler. Concepts of trusted computing. In Chris J Mitchell, editor, *Trusted Computing*, number 6 in IEE Professional Applications of Computing Series, chapter 2. The Institute of Electrical Engineers, London, UK, 2005.

[67] Naren Ramakrishnan, Benjamin J. Keller, Batul J. Mirza, Ananth Y. Grama, and George Karypis. Privacy Risks in Recommender Systems. *IEEE Internet Computing*, 5(6), 2001.

[68] Kumar Ranganathan. Trustworthy Pervasive Computing: The Hard Security Problems. In *PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Washington, DC, USA, 2004. IEEE Computer Society.

[69] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*. Springer-Verlag, 2001.

[70] Carsten Rudolph. Covert Identity Information in Direct Anonymous Attestation (DAA). In Hein S. Venter, Mariki M. Eloff, Les Labuschagne, Jan H. P. Eloff, and Rossouw von Solms, editors, *SEC*, volume 232 of *IFIP*. Springer, 2007.

[71] Ahmad-Reza Sadeghi, Christian Stüble, and Marcel Winandy. Property-Based TPM Virtualization. In *ISC '08: Proceedings of the 11th international conference on Information Security*, Berlin, Heidelberg, 2008. Springer-Verlag.

[72] Sadeghi, Ahmad-Reza and Stüble, Christian. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*, New York, NY, USA, 2004. ACM.

[73] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2004. USENIX Association.

[74] Dries Schellekens, Brecht Wyseur, and Bart Preneel. Remote attestation on legacy operating systems with trusted platform modules. *Sci. Comput. Program.*, 74(1-2), 2008.

[75] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, New York, NY, USA, 2005. ACM Press.

[76] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. SWATT: Software-based attestation for embedded devices. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2004.

[77] Elaine Shi, Adrian Perrig, and Leendert van Doorn. BIND: A Fine-grained Attestation Service for Secure Distributed Systems. In *IEEE Symposium on Security and Privacy*, 2005.

[78] Sean W. Smith. Outbound Authentication for Programmable Secure Coprocessors. In *ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security*, London, UK, 2002. Springer-Verlag.

[79] Sean W. Smith. *Trusted Computing Platforms: Design and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[80] Evan R. Sparks. A Security Assessment of Trusted Platform Modules. Technical Report TR2007-597, Dartmouth College, Computer Science, Hanover, NH, June 2007.

[81] Geoffrey Strongin. Trusted computing using AMD Pacifica and Presidio secure virtual machine technology. *Information Security Technical Report*, 10, 2005.

[82] Frederic Stumpf, Omid Tafreschi, Patrick Roder, and Claudia Eckert. Robust Integrity Reporting Protocol for Remote Attestation. In *Proceedings of the Second Workshop on Advances in Trusted Computing (WATC'06 Fall)*, 2006.

[83] Stumpf, Frederic, Fuchs, Andreas, Katzenbeisser, Stefan, and Eckert, Claudia. Improving the scalability of platform attestation. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable trusted computing*, New York, NY, USA, 2008. ACM.

[84] Frank Swiderski and Window Snyder. *Threat Modeling.* Microsoft Press, June 2004.

[85] Mark Weiser. Some computer science issues in ubiquitous computing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3), 1999.

Appendix A

Obtaining an identity

A.1 Privacy-CA

The four entities involved in the protocol are: the owner of the TPM, the TPM, the TSS, and a Privacy-CA[5]. The protocol proceeds as follows[5, 27]:

1. The platform owner requests a new identity by calling `Tcsip_MakeIdentity` or `Tcsip_MakeIdentity2` [37] which sends a `TPM_MakeIdentity` command[36] to the TPM. The TPM creates a new AIK pair under the SRK and makes available an *identity binding*: a self-signed signature (using the newly generated signing AIK) that binds the public portion of the AIK, the new identity label, and the identity of the Privacy CA chosen by the TPM owner to attest to the new identity.

2. `Tspi_TPM_CollateIdentityRequest` is called to assemble all the evidence needed by the Privacy CA, i.e. the identity-binding plus the third party credentials.

3. The platform agent encrypts the assembled data using the target Privacy-CA's public key. This is for privacy purposes[5].

4. The Privacy-CA decrypts the received data The Privacy-CA inspects the credentials to determine whether they describe a genuine Trusted Platform. The Privacy-CA inspects the identity-binding signature and checks that it is consistent with the supplied information. The Privacy-CA checks that it (and not some other Privacy-CA) has been asked to attest to the new identity.

5. The Privacy-CA creates an attestation credential, encrypts it with a symmetric key, and encrypts the symmetric key under the TPM public endorsement key. In this way an implicit guarantee of authenticity of the TPM is obtained since the encrypted attestation credential can only be decrypted by a genuine TPM (as the endorsement credential attests). The Privacy-CA also sends an encrypted hash of the public key from the identity binding such that it can only be decrypted by the genuine TPM.

6. The owner authorises the `TPM_ActivateIdentity` command [36] the purpose of which is to obtain assurance that the identity credential is for this TPM and if so to release the symmetric key to the host platform. The TPM processes this command as follows. First it decrypts, using its private EK, the hash of the public key from the identity binding sent by the Privacy-CA. Then, if the decrypted hash matches the hash of one of the TPM's identity keys, it decrypts and releases the symmetric key to the host platform.

7. The release of the symmetric key allows the platform to decrypt the identity certificate.

A.2 DAA

Join In the DAA-join phase a platform enrols as a member of a group to which the issuer has issued credentials.

The platform invokes the `Tspi_TPM_DAA_JoinInit`, `Tspi_TPM_DAA_JoinCreateDaaPubKey` and `Tspi_TPM_DAA_JoinStoreCredential` commands to create and store DAA credentials. The protocol proceeds as follows:

1. The TPM derives a pseudonym N_I of the form ζ_I^f where the TPM secret f, is a function of $DAASeed$, and CK_1. $DAASeed$ is a single secret value stored in the TPM for the DAA scheme. CK_1 is the long-term public key of the issuer. ζ_I, derived from the issuer's long-term basename, provides issuer-specific link-ability. The TPM also makes a commitment to a value U, as a function of f and a fresh secret quantity v'.

2. The issuer checks for a rogue platform by testing whether $N_I \not\equiv \zeta_I^f$, i.e. N_I corresponds to any f on a rogue list.

3. The TPM engages in a proof of knowledge of f and v' whilst authenticating itself to the issuer using its private EK. The issuer does not learn the value of EK in this process.

4. If the proof is successful then the issuer generates a Camenisch-Lysyanskaya signature on f.

Sign The DAA-signing protocol is used by a platform to prove that it possesses an anonymous attestation credential (i.e. DAA Issuer credential) and at the same time to sign the public AIK [14]. To sign a message the `Tspi_TPM_DAA_Sign` command is invoked.

For the purposes of verification, the TPM is identified by a pseudonym $N_V = \zeta^f$ where ζ may be selected by the verifier (as a function of the verifier's basename) or at random by the platform. In the former case, the pseudonym N_V is fixed for the platform-verifier pair and allows transactions to be linked by the verifier. The latter case enables full anonymity and unlink-ability of platform transactions.

Verify The verifier issues the `Tspi_TPM_DAA_VerifyInit` and `Tspi_TPM_DAA_VerifySignature` commands to complete the DAA-Verify procedure.

1. The verifier verifies the signature proof of knowledge on the public AIK.
2. If ζ was selected by the verifier then the verifier checks that N_V is of the correct form[14].
3. The verifier checks whether $N_V = \zeta^f$ for any $f$on the rogue list.