Research Workshop on Challenges for Trusted Computing

at:

3rd European Trusted Infrastructure Summer School (ETISS 2008)

Tuesday 2nd September 2008: 16:00-18:00

Programme and contributed papers

Overview of Research Workshop

Trusted Computing refers to the collection of interrelated and interoperating technologies, which, when combined, help to establish a more secure operating environment on commodity platforms. A fully-realised Trusted Computing platform will allow users to reason about the behaviour of a platform, as well as providing standardised mechanisms to protect sensitive data against software attack.

Trusted Computing has been proposed as a means of enhancing the security of numerous applications. For example, it has been promoted as an adjunct to the digital signature process, to enable secure software download, to support secure single sign-on solutions, to secure peer-to-peer networks, to improve the security and privacy of biometric user authentication, to harden mobile devices, and to facilitate identity management. A number of authors have also considered Trusting Computing's applicability to the agent paradigm, grid security, e-commerce transaction security, and to defend against the ever-growing threat posed by crimeware.

Despite its many potential beneficial applications, Trusted Computing is not without its detractors. Privacy concerns relating to trusted platforms have been raised. The extent to which Trusted Computing could be used to enable and enforce digital rights management, and, more generally, the possible expropriation of platform owner control, are contentious issues. Concerns have also been expressed that Trusted Computing could be used to support censorship, stifle competition between software vendors, facilitate software lock-in, and hinder the deployment and use of open source software, thereby potentially enabling market monopolisation by certain vendors.

The aim of this workshop is not to engage in this debate, but rather to highlight some of the key challenges that we believe need to be addressed in order to accelerate the widespread adoption of Trusted Computing.

Workshop Organisers

Shane Balfe, Royal Holloway, University of London

Eimear Gallery, Royal Holloway, University of London

Chris Mitchell, Royal Holloway, University of London

Kenny Paterson, Royal Holloway, University of London

Programme

16:00-16:10 Opening remarks

16:10-16:40 *Challenges for trusted computing*: Shane Balfe and Eimear Gallery (ISG, Royal Holloway, University of London, UK)

16:40-17:10 *Advances on PrivacyCAs*: Martin Pirker, Ronald Toegl and Daniel Hein, Peter Danner (IAIK, Graz University of Technology, Austria)

17:10-17:40 *Attacking the BitLocker Boot Process*: Sven Tuerpe, Andreas Poller, Jan Steffan, Jan-Peter Stotz and Jan Trukenmueller (Fraunhofer-Institute for Secure Information Technology, Darmstadt, Germany)

17:40-17:55 Discussion

17:55-18:00 Wrap-up

Contributed papers

Challenges for Trusted Computing

S. Balfe, E. Gallery, C.J. Mitchell and K.G. Paterson

Technical Report
RHUL–MA–2008–14
26 February 2008

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
`http://www.rhul.ac.uk/mathematics/techreports`

Abstract

This article identifies and discusses some of the key challenges that need to be addressed if the vision of Trusted Computing is to become reality. Topics addressed include issues with setting up and maintaining the PKI required to support the full set of Trusted Computing functionality, the practical use and verification of attestation evidence, and backwards compatibility, usability and compliance issues.

1 Introduction

A trusted platform, as discussed in this article, refers to a platform of the type championed by the Trusted Computing Group (TCG). That is, a trusted platform is one which will behave in a particular manner for a specific purpose. Trusted computing refers to the collection of interrelated and interoperating technologies, which, when combined, help to establish a more secure operating environment on commodity platforms. A fully-realised trusted computing platform will allow users to reason about the behaviour of a platform, as well as providing standardised mechanisms to protect user private data against software attack [36]. A trusted platform should be able to reliably gather and provide evidence of its current operating state, or any subcomponent thereof. Any divergence from an intended operating state can be reported to interested parties, allowing them to make informed decisions as to whether to continue to interact with the platform in question.

Trusted Computing functionality has been proposed to enhance the security of numerous applications. For example, it has been promoted as an adjunct to the digital signature process [6, 49], in enabling secure software download [19], in support of secure single sign-on solutions [29], in securing peer-to-peer networks [8, 23, 45, 43], in improving the security and privacy of a biometric user authentication process [14], in hardening mobile devices [18] and to facilitate identity management [27, 28]. A number of authors have also considered trusting computing's applicability to the agent paradigm [15, 30, 31, 35], grid security [25, 26], e-commerce transaction security [10, 9], and to defend against the ever-growing threat posed by crimeware [7]. Further applications are described in [5, 20, 24, 62].

Despite its many beneficial applications, Trusted Computing is not without its detractors; see for example [2, 3, 4, 38, 39, 40, 46, 51, 63]. Privacy concerns relating to trusted platforms were raised in [37]. The extent to which Trusted Computing could be used to enable and enforce Digital Rights Management (DRM), and, more generally, the possible expropriation of platform owner control, have been contentious issues [46, 60]. Anderson [2] expresses the view that Trusted Computing could be used to support censorship, stifle competition between software vendors, facilitate software lock-in and hinder the deployment and use of open source software, thereby potentially enabling market monopolisation by select vendors.

Our aim in this article is not to engage in this debate, but rather to highlight some of the key challenges that we believe need to be addressed in order to accelerate the widespread adoption of Trusted Computing. Many of the challenges we identify are not purely technical in nature, but rather involve a mixture of technical, policy and management aspects. This could make this article somewhat different in flavour from the majority of current research in Trusted Computing. We trust that our work will bring some useful diversity into the field and that our approach will add to, rather than subtract from, the value of the article for readers.

This article is structured as follows. In order to make the article self-contained, Section 2 provides an overview of Trusted Computing technology. In Section 3, we discuss some of the challenges that we believe may hinder the widespread adoption and use of Trusted Computing. In particular, we focus on: the Public Key Infrastructure (PKI) that is associated with the deployment of Trusted Computing; certificate and Trusted Platform Module (TPM) revocation as well as the impact of hardware attacks on platforms; problems with attestation evidence gathering and verification; backward compatibility requirements; usability issues; and challenges arising from non-compliance with the TCG's technical specifications. We conclude with Section 4.

2 Trusted Computing Concepts

The provision of the full range of Trusted Computing functionality is dependent upon the integration of a number of additional hardware and software components into a computing platform:

A Trusted Platform Module (TPM): The TPM forms the core of a Trusted Computing platform [56, 57, 58]. A TPM is a microcontroller with cryptographic coprocessor capabilities that provides a platform with the following features: a number of special purpose registers called Platform Configuration Registers (PCRs), into which cryptographic digests representing platform state altering events can be recorded; a means of reporting the platform's current state to remote entities; secure volatile and non-volatile memory; random number generation; a SHA-1 hashing engine; and asymmetric key generation, encryption and digital signature capabilities.

Core Root of Trust for Measurement (CRTM): The CRTM should form an immutable portion of a host platform's initialisation code and is responsible for measuring the host platform's state, that is, the collection of operating system and application software components running on the machine. The process of measuring platform state may be much more complicated than simply taking a single hash of a monolithic piece of code, however. For example, for a Personal Computer (PC) client [55], the CRTM code would measure a platform's BIOS and store a hashed representation of the BIOS code in one (or more) of the PCRs within a TPM. The CRTM would then hand execution control to the BIOS, which in turn would measure, store, and transfer control to the next component in a host platform's boot sequence. This process would then continue until all components are measured. Through this succession of measuring, transfer of control and execution, a transitive trust chain from the CRTM to the host Operating System (OS) can be formed. Attestations to the measurements recorded to the TPM's PCRs that make up this chain can subsequently be provided to interested parties. (We discuss the different flavours of Trusted Computing attestation in more detail below.)

Isolation technologies: The introduction of isolation technologies, such as Microsoft's Next Generation Secure Computing Base (NGSCB) [32, 33] or XEN Source's XEN [11], were designed to take advantage of CPU and chipset extensions incorporated in a new generation of processor hardware [22]. Together, these new initiatives will enable a platform to be partitioned into isolated execution environments. An isolated execution environment, independent of how it is implemented, should provide the following services to hosted software [33]:

- protection of the software from external interference;
- observation of the computations and data of a program running within an isolated environment only via controlled inter-process communication;
- secure communication between programs running in independent execution environments; and
- a trusted channel between input/output devices and a program running in an isolated environment.

We note that current deployments of Trusted Computing technologies have tended to focus more on the TPM than on the CRTM or isolation technologies. This naturally limits the trustworthiness of the platform. Even so, a number of proposals [8, 9, 44] demonstrate that even a limited deployment of Trusted Computing can bring security benefits.

2.1 Trusted Computing Credentials and Hardware Attestation

In addition to the integration of additional hardware and software components, certification and accreditation play an important role in building a trusted platform. In this regard, there are a number of interrelated credentials that must be present before a platform can be considered to be a *trusted* platform:

An endorsement credential: Each TPM is associated with a unique asymmetric encryption key pair called an *Endorsement Key* (EK) pair. An endorsement credential binds the public component of this key pair to a TPM description and vouches that a TPM is genuine. This credential is typically generated by the TPM manufacturer, with the binding taking the form of a digital signature created using a signing key of the manufacturer.

One or more conformance credentials: Conformance credentials vouch that a particular type of TPM and associated components (such as a CRTM and the connection of the CRTM and TPM to a motherboard) conform to the TCG specifications. These credentials are issued by entities with sufficient credibility to evaluate platforms containing TPMs, typically conformance testing facilities.

A platform credential: A platform credential proves that a TPM has been correctly incorporated into a design which conforms to the TCG specifications. This credential is typically generated by a platform manufacturer. In order to create a platform credential, the platform manufacturer must examine the endorsement credential, the conformance credentials relevant to the trusted platform, and the platform to be certified.

2.2 Identity Attestation

Since an EK is unique to a platform, it may act as a “super-cookie” in identifying subsequent platform actions across multiple domains. To help allay concerns that an EK may become associated with personally identifiable information, the TCG [56] introduced the ability for a TPM to generate and use an arbitrary number of pseudonyms, in the form of *Attestation Identity Key* (AIK) key pairs. Unlike an EK, AIKs serve to identify a platform as trusted without uniquely identifying it as a *particular* trusted platform. However, in order for a relying party to have assurance that an AIK represents a trusted platform, a platform must obtain AIK certification from a mutually trusted third party. Two approaches to AIK certification have been proposed by the TCG.

In the first approach, a trusted third party, referred to as a *Privacy-Certification Authority* (P-CA), provides assurance that an AIK is bound to a trusted platform in the form of an *AIK credential*. When a platform requests an AIK credential from a P-CA, it must supply an AIK public key as well as its endorsement, conformance and platform credentials. The P-CA verifies these credentials, thereby obtaining assurance that the trusted platform is genuine, and creates an AIK credential attesting to this fact. This credential takes the form of a digital signature on the AIK public key. However, this approach has attracted a certain amount of criticism. A P-CA is capable of linking all AIK credentials issued to a specific platform via the platform's EK, putting the P-CA in a position where it is able to defeat the anonymity protection provided by the use of AIKs. Moreover, it is unclear what business model might support the development of commercial P-CAs.

Direct Anonymous Attestation (DAA) has been proposed as an (optional) alternative to the P-CA model, and offers strong anonymity guarantees through advanced cryptographic techniques. These prevent the corresponding issuing authority from being able to link AIK credentials with a single platform identity (EK). A DAA Issuer produces a certificate on a blinded TPM secret. The TPM-host platform later uses this certificate and TPM secret to produce a special type of digital signature (called a signature proof of knowledge) on an AIK public key corresponding to the TPM secret. A verifier of this signature does not actually get to see the certificate, but instead receives an assurance (via a zero-knowledge protocol) that the platform possesses such a certificate on the DAA public key corresponding to the TPM secret.

2.3 Object Attestation

2.3.1 State Attestation

A platform's state is represented by a set of integrity measurements. When a platform component (i.e. a piece of software executing on a platform) is "measured", a hash of the component is recorded to one of a set of PCRs within the host platform's TPM (where the PCR to which a particular component is recorded is platform-specific). Subsequent measurements to this register are recorded by overwriting its current value with a hash of the concatenation of the new measurement and the existing contents of the register. In this way, the cumulative contents of a TPM's registers reflect the current software state of the host platform, as well as the states through which the platform has transitioned.

Platform attestation is the process by which a platform can reliably report evidence of both its identity (as a valid trusted platform) and its current state. A TPM signs the contents of (one or more of) the TPM's PCRs which reflect (all or part of) the current host platform's state. The private component of an AIK key pair is used to create the signature. This signed bundle is communicated to an external entity in conjunction with a corresponding AIK public key credential and a record of the platform components which have been measured. The receiving entity validates the AIK credential, verifies the AIK signature, and compares the signed PCR values against a set of reference integrity measurements. The TCG envisages that reference values and associated credentials will typically be created during software development [59].

2.3.2 Data Attestation

Sealing is the process by which sensitive data can be associated with a set of PCR values representing a particular platform configuration, and only released to the platform for use when the current state of the platform is such that the PCR values match those specified at the time of sealing. The sealing and unsealing processes are implemented using encryption and decryption, in tandem with appropriate key management procedures.

Data can also be associated with a string of 20 bytes of authorisation data before being sealed. When unsealing is requested, the authorisation data must then also be submitted to the TPM. The submitted authorisation data is then compared to the authorisation data in the unsealed string, and the object is only released if the values match.

2.3.3 Key Attestation

A TPM can generate an unlimited number of asymmetric key pairs. For each of these pairs, private key usage and mobility can be constrained. A key pair can be generated so that use of its private component is contingent upon the presence of a predefined platform state (as reflected in the TPM's PCRs). Additionally, a private key can be marked as either being migratable or non-migratable. Migratable private keys can be moved from a TPM, whilst non-migratable keys are inextricably bound to a single TPM.

Private keys, like data, can also be associated with a string of 20 bytes of authorisation data. When private key usage is required, the authorisation data must be submitted to the TPM. The submitted authorisation data is then compared to the authorisation data associated with the key, and key use is only permitted if the values match.

To attest to the usage, mobility and authorisation constraints associated with a private key held by a TPM, the TCG has specified the Subject Key Attestation Evidence (SKAE) X.509 extension [53]. SKAE provides a mechanism to allow a verifier to ascertain that an operation involving a private key was performed within a TCG-compliant TPM environment. After obtaining an AIK credential (following the P-CA method outlined in Section 2.2), a platform can sign a data structure containing the public component of a non-migratable TPM key pair and a description of usage attributes of the corresponding private key. An SKAE Certification Authority (CA), after verifying the signed TPM data structure, can then create a new X.509 certificate for the public key. This certificate incorporates an extension field attesting to the TPM-related security properties of the certified key. Such a certificate can then be used as an aid to authentication in protocols such as SSL/TLS [16] or IKEv2 [1].

3 Challenges for Trusted Computing

3.1 Public Key Infrastructure and Trusted Computing

As we saw in Section 2.1, for a platform to be considered trusted, it must first obtain certificates from an endorsement CA, a platform CA, and one or more conformance CAs. Together, these CAs are responsible for issuing the core trusted platform credentials. However, in order to address privacy concerns resulting from overuse of an EK (as contained in an endorsement credential), P-CAs, and later DAA Issuers, were introduced. Subsequently, SKAE CAs were proposed as a means of coping with difficulties in integrating TPM-controlled keys with standard security protocols. Recently, further-PKI-related authorities (notably Migration Authorities (MAs) and Migration Selection Authorities (MSAs) [52]) have been introduced to address issues concerning key migration between TPM-enabled platforms.

Thus the majority of Trusted Computing services depend fundamentally on the deployment and successful inter-operation of a number of PKI elements. We refer to this collection of components as a *Trusted Computing PKI (TC-PKI)*, although it is actually a larger and more complex "eco-system" of elements than would normally be contained in a single PKI. Figure 1 depicts the main types of CA in a TC-PKI. Yet the development of *any* functional PKI requires a sophisticated combination of organisational, policy-oriented, procedural, and legislative approaches. Indeed the challenges and pitfalls of PKI deployment are well-documented [21, 34], and high-profile system and protocol failures that have been blamed on inappropriate deployment of PKI abound, with SET [47] providing one of the most prominent examples. Put simply, providing a PKI is hard.

A TC-PKI not only involves a plurality of CAs, but also a series of implicit dependencies amongst these CAs. In a TC-PKI, a platform CA relies on the due diligence of an endorsement CA and one or more conformance CAs in accrediting components of a trusted platform. Similarly, both privacy CAs and DAA Issuers rely on platform CAs, endorsement CAs and one or more conformance CAs. Furthermore, SKAE CAs rely on the due diligence of Privacy CAs or DAA Issuers in evaluating the accreditation evidence provided by a trusted platform.

Traditionally, Certificate Policies (CPs) (which specify what a certificate should be used for, and the liability assumed by the CA for this use) and Certificate Practice Statements (CPSs) (which specify the practices that a CA employs to manage the certificates it issues) are deployed by CAs in order to define and limit their liabilities with respect to relying parties. CPs and CPSs are, in fact, an essential component in building a successful PKI, since they give a relying party (which could be an end-user or another CA) a means to manage the business risk in pursuing a particular PKI-related course of action. In the past, uncertainty as to where liability lies has driven up the cost of many PKI implementations [34]. In the absence of CPs and CPSs, implicit cross-certification may exist between CAs, which, as noted in [21], implies that CAs are equally trusted. In such a setting the security of a certificate is reduced to that of the least trustworthy CA. Unfortunately, CPs and CPSs are notoriously difficult and costly to create, and so their production may act as a barrier to entities wishing to provide CA services.

In the setting of a TC-PKI, such policy statements must be produced by every CA upon which another CA may depend. Currently, these dependencies are only informally defined, and, as a result, there is no clear indication of where any liability will lie. Further, at the time of writing, we are not aware of any TC-specific CPs or CPSs having been created. The picture is further complicated by the fact that all the CAs in a TC-PKI rely (at least to some extent) on the endorsement CA. Therefore, the point in a TPM's life-cycle at which an EK credential is acquired impacts on a platform's ability to obtain platform, AIK, DAA and SKAE credentials. In *early normative* EK credential acquisition, as defined by the TCG [54], a TPM manufacturer generates the EK credential. However, in *post-manufacturing* generation, as defined by the TCG [54], a platform owner is responsible for generating the EK credential. In this instance, the certifying body may not be recognised by other CAs and, as a result, the certified TPM host platform may not be able to obtain further credentials from entities outside the domain of its EK credential issuer. In practice, this may not be an issue, as it seems likely that non-manufacturer supplied EK credentials will not be widely used.

To summarise: Trusted Computing relies on an as yet largely unavailable and unspecified PKI in which multiple CAs (possibly existing in different organisational, procedural and/or jurisdictional domains) are expected to inter-operate. This may pose a significant challenge to the future success of this technology.

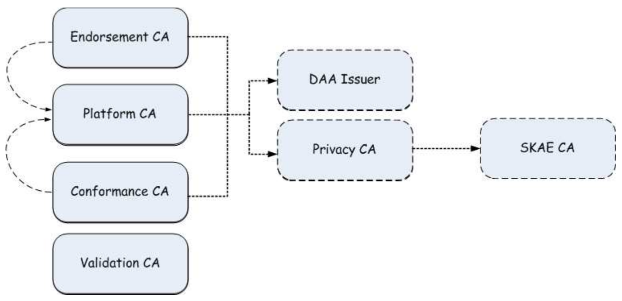

Figure 1: Trusted Computing PKI Components

3.2 Certificate and TPM Revocation

The revocation of credentials within a TC-PKI may introduce further problems. Given the complex dependencies between many of the TC-PKI credentials, the compromise of an individual key and the subsequent revocation of its associated public key certificate will result in a cascading revocation of all dependent TPM credentials. For example, in the event of endorsement key revocation, every AIK associated with the revoked EK must also be revoked. In addition, all SKAE credentials associated with the newly revoked AIKs must also be revoked. This implies that multiple CAs, potentially in independent domains, must be contacted in a timely manner and informed about a revocation decision. This may be a time-consuming and costly endeavour. Further complexity is introduced when attempting to revoke a DAA credential associated with a compromised Endorsement Key pair, because a DAA Issuer cannot link a platform's Endorsement Key pair with a DAA credential.

We next consider revocation of a TPM itself (rather than revocation of its credentials). For cost reasons, the level of tamper-resistance provided by

TPMs is likely to be limited. Moreover, the objective of the mechanisms specified by the TCG is the prevention of information asset compromise through software attack. That is, the software security of the platform is predicated upon the notion that the TPM will maintain an accurate and reliable record of all platform events. Such a focus means that the security of the underlying hardware is assumed and that there is no purely technical driver to promote the development of tamper-resistant TPMs.

Yet it is clear from the example of widespread gaming console modification that, given sufficient incentive, users will actively circumvent hardware-enforced security. In this context, recent demonstrations of a relatively unsophisticated hardware attack [50] through which a TPM's PCRs can be reset without rebooting a platform would appear to pose a significant challenge to Trusted Computing. The ability to reset PCRs effectively destroys the transitive trust chain upon which a remote verifier relies to assess a platform. Once this trust chain is broken, the PCRs can be repopulated with whatever data the platform owner wishes, allowing the owner to misrepresent their platform's current state in a manner that is convincing to a remote verifier. The simple attack of [50] underlines the need for any verifier to consider the "quality" of the platform when assessing the state of a trusted platform. That is, an attestation from a platform incorporating a well-designed TPM from a known manufacturer should be considered more convincing than an attestation from a platform incorporating a TPM from an unknown or disreputable supplier.

Given the above discussion, it is reasonable to assume that, before long, TPMs will be compromised and all credentials and keys extracted. These could then be used to emulate a TPM in software in a way that is indistinguishable from the true hardware TPM. The process by which a compromised TPM is detected will largely be reliant on that TPM's interactions with P-CAs, DAA Issuers and SKAE CAs. It has been suggested in [13] that TPM compromise could manifest itself through an excessive number of certification requests originating from a single TPM host platform (where "excessive" is to be determined by a risk-management policy). However, such a naive approach to detection introduces a number of challenges:

- CAs may specify different thresholds for determining what is meant by "excessive", potentially leading to a high number of false positives for CAs with low thresholds.

- Once a compromised TPM has been detected, there is a need to globally propagate this information to prevent the compromised TPM host platform from being (mis)used elsewhere. This requires the establishment of a global revocation infrastructure. Such an infrastructure could be implemented using Certificate Revocation Lists (CRLs) or through an On-line Certificate Status Protocol (OCSP). Neither option, however, is ideal. In the case of CRLs, there are concerns regarding CRL discovery and the timely issuance of revocation information. In the case of OCSP, in order to make its deployment economically viable, CAs typically charge for each revocation check. It is unclear who would pay for such a service in a TC-PKI. In the case of an OCSP request for an SKAE certificate, the verifier would need to contact the SKAE CA, which would need to contact the AIK CA, which in turn would need to contact the platform, endorsement and conformance CAs.

- A CA must consider potential legal issues that might result from the wrongful issuance of revocation statements negatively impacting on a platform's ability to interact with other parts of the infrastructure. As a result of such considerations, CAs may become reluctant to announce suspected compromises.

- To alleviate the risk of a malicious P-CA issuing falsified revocation statements, a means by which the credibility of CAs in issuing such statements can be assessed must be provided. It is currently unclear what form such a mechanism might take.

3.3 Attestation Evidence Gathering and Verification

The exact parameters to be considered when performing integrity measurements on platform components have yet to be standardised[1]. At a minimum, the parameters must be chosen so that each software component's integrity measurement can be uniquely identified. These measurements must also remain consistent to allow ease of verification. However, in the absence of standardisation, platform integrity measurements may fail to capture all elements required by the verifier of a platform component. This is especially true when one considers the complications introduced with respect to software which relies on dynamically linked libraries (DLLs). In this case, a proportion of the platform component's code base may not be measured, as it will not be loaded by the application prior to execution.

Moreover, given the extensible nature of modern computing systems, the number of components that might need to be measured by a TPM is rapidly

[1]For example, the authors of [33] have proposed that a platform component's integrity measurement can be calculated from its instruction sequence, initial state (i.e. the executable file) and input.

increasing. This implies that each register will have to store multiple measurements. As the number of a platform's components increases, so does the complexity of third party verification of attestation statements. It also becomes difficult for a challenger to verify a single component running on a platform.

The introduction of isolation technologies, as described in Section 2, enables a platform to be partitioned into isolated execution environments, thereby (potentially) simplifying attestation statement verification. In this case, a challenger of the platform may be satisfied to verify measurements pertaining to rudimentary platform components, such as the boot software, the isolation layer and software components running in an isolated execution environment rather than verify all software running on the platform. This may ease the platform attestation problem in some situations.

However, even assuming the number of platform component integrity measurements that a challenger must verify is limited, problems relating to platform component updates and patching will still arise. Given current software development practices, frequent patching to OS components and applications can be expected to be the norm for the foreseeable future. But even the order in which patches are applied can result in a "combinatorial explosion" of distinct configurations for a single application, each configuration requiring a distinct reference value for attestation purposes. Frequent patching may also lead to problems with respect to sealed data. If an update or patch is applied to a software component to which a key or data is sealed, this key or data must be unsealed and resealed to the updated software component measurements. Failure to reseal to the updated component measurements will result in the key or data being inaccessible after the patch has been applied.

Property Based Attestation [42] has been proposed to address the problem of managing attestation in the presence of a multitude of possible configurations and system updates. This approach introduces an additional layer of indirection into the attestation and sealing processes. Instead of expecting a verifier to determine if a particular set of PCR values represent a trustworthy software state, a platform's state is certified (by a trusted third party) as satisfying certain properties. A platform is then capable of attesting that its current configuration possesses such a property, allowing a verifier to infer whether a platform is trustworthy or not without knowing which particular software is running. Property Based Attestation also allows data or keys to be sealed to properties. As long as the properties of the updated platform configuration match those of the prior configuration, problems related to patching may be reduced. Sadeghi and C. Stüble suggest that this approach could be realised either through a software-based "Trusted Attestation Service" or through modifications to the TPM hardware.

Unfortunately, Property Based Attestation only succeeds in shifting the problems with attestation to an entity other than the verifier, with all of the original problems persisting for the entity that needs to verify a PCR-based attestation. Nevertheless, the number of entities needing to verify such complex attestations could be significantly reduced, and these entities could be given additional resources to enable them to complete their task. Moreover, a software component satisfying a particular property is by no means guaranteed to still satisfy that property after it has been patched, without rerunning the (potentially expensive) evaluation procedure. This evaluation procedure may contribute to the marginalisation of minority platforms by "altering the economics of interoperability" [38]. The cost of establishing that a given platform state matches some desirable property may be so great that only a few well-funded organisations may be able to obtain such a result. Additionally, exactly what properties can be satisfied using such an approach remains an open question. More positively, Property Based Attestation at least shifts the problems to an expert specialising in the particular business of attestation.

A final issue which must be considered with respect to the successful implementation of platform attestation is that of user observable verification. McCune *et al.* [17] describe a scenario in which a user's platform has become infected with malware. Despite the fact that this infection can be detected by an external entity during an attestation process, the external entity has no way of reliably informing the end user that they have failed their attestation. Malware may simply modify the user's display, resulting in the user believing their platform to be in an acceptable state, and, because of this, going on to disclose sensitive information to the malware.

3.4 Backward Compatibility

As a consequence of the piecemeal roll-out of Trusted Computing technologies, current trusted platforms do not come equipped with CRTMs, isolation technologies, processors or chipset extensions. Instead, current trusted platforms include only a TPM meeting the relevant TCG specifications, and, with the exception of Infineon TPMs, do not even include endorsement credentials. To the best of our knowledge, all currently available platforms lack both conformance credentials and platform credentials. This situation has the potential to create an awkward backward compatibility issue as and when fully-deployed TC-PKIs become available. In particular, the absence of these credentials will make it difficult, if not impossible, for a platform to later acquire AIK credentials without operating at reduced assurance levels.

The absence of CRTMs, isolation technologies, processors and chipset extensions from current TPM-enabled platforms makes the use of much of the TPM Trusted Computing functionality described in Section 2 essentially unreliable. Techniques such as sealing and attestation are unworkable if the host platform's state cannot be reliably measured. In order to later enable these features on an already deployed platform, measurement functionality (in the form of a CRTM and modified operating system) would need to be integrated into the platform. This would require the installation of a new OS and the BIOS to be flashed, tasks that would prove difficult for the average user. On the other hand, this may be feasible in a corporate environment with centralised administrative control of platforms. Indeed, in such deployments, legacy hardwares issue may be less of an issue because of more rapid retirement of platforms. Moreover, software-based isolation environments can be provided through the installation of additional software onto already deployed platforms. Nevertheless, hardware-based isolation, enabled through the processor and chipset extensions, cannot be retrofitted to platforms already in the field. As a result, first generation trusted platforms can never be adequately upgraded to provide all the services associated with a trusted platform.

3.5 Usability

Prevailing wisdom suggests that it is prudent to hide the complexities of security technology from end-users [34]. In the past, applications that have relied on a PKI have failed in cases where security functions have been too unwieldy to be usable by non-experts [12]. In one example [48], the PKI experience was considered so painful by some users that they refused to use the technology if it involved handling certificates. The design of suitable user-interfaces that can communicate rich security information whilst remaining usable has historically been very difficult to achieve [61].

By contrast, using a TPM currently requires a detailed understanding of how the underlying technology works. For example, the very act of enabling a TPM prior to its use is a non-trivial task requiring a user to understand and edit BIOS settings. Once enabled, a user is further confronted with setting a TPM owner password, selecting key types fit for purpose, and enrolling certain keys within a PKI. Further problems may arise with respect to password use and management. In addition to setting a password for TPM ownership, unique passwords may also be associated with protected data or keys in a TPM. While the deployment of numerous passwords may be viewed as a sound security decision, management of such passwords so that access is not jeopardised may prove problematic.

These usability issues are a reflection of the general immaturity of Trusted Computing technology and the associated marketplace. Whilst a huge effort has been put into design and specification of technical aspects of Trusted Computing by the TCG, so far less work seems to have been done to address user-centric issues. We may hope for user-friendly configuration and management tools in future, although even these may not be sufficient to make Trusted Computing accessible to the masses.

3.6 Non-Compliance and Inter-operability

Through the provision of a set of open standards, Trusted Computing specifies security interfaces which allow heterogeneous devices to interact. Unfortunately, many of the additional technological building blocks required to instantiate a trusted platformare not standardised, nor does the TCG dictate implementation specifics to its adopters. As a result, a number of currently available TPMs do not comply with the TPM specifications [41]. The current absence of conformance testing facilities implies that the production of non-compliant TPMs may very well continue for the foreseeable future. In turn, discrepancies in implementation between TPM manufacturers may limit future inter-operability of different trusted platforms.

4 Conclusions

Trusted Computing is undoubtedly a powerful technology, with a huge range of possible applications. Nevertheless, there remain a number of significant obstacles to its widespread use, as we have discussed above. Addressing these challenges is therefore a high priority for future research.

Perhaps the most significant of these obstacles is the deployment and management of the PKI necessary to enable general use of the security services supported by Trusted Computing. These issues are in many ways similar to those which prevented the establishment of a global general-purpose PKI. Nevertheless, deploying domain and company-specific PKIs to support Trusted Computing well-defined and limited environments would appear relatively straightforward, since the majority of the problems simply disappear — this again reflects the experience of deploying conventional PKIs, which have been used very successfully in specific domains.

We have also examined problems arising with the use and interpretation of evidence generated using Trusted Computing functionality. This problem arises in particular because of the number of different components (and versions of components). As with the PKI issues, many of the problems are particularly serious when one considers universal use of Trusted Computing — the issues are likely to be much less serious in a closed/managed environment, e.g. as established within a large organisation, notably because the number of components will be significantly less, and there are likely to be more resources available to evaluate the components.

In conclusion, many challenges to the successful large-scale use of Trusted Computing remain. Nevertheless, these challenges are likely to be much less serious for a very important class of users, namely corporate IT. Providing the full benefits of Trusted Computing to the widest possible audience is a major challenge for future research.

References

[1] Internet Key Exchange (IKEv2) Protocol. RFC 4306, 2005.

[2] R. Anderson. Cryptography and Competition Policy: Issues with 'Trusted Computing'. In *Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing (PODC 2003)*, pages 3–10, Boston, Massachusetts, USA, 2003. ACM Press, New York, USA.

[3] R. Anderson. 'Trusted Computing' Frequently Asked Questions - Version 1.1. `http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html`, August 2003.

[4] B. Arbaugh. Improving the TCPA Specification. *IEEE Computer*, 35(8):77–79, August 2002.

[5] B. Balacheff, L. Chen, S. Pearson, D. Plaquin, and G. Proudler. *Trusted Computing Platforms: TCPA Technology in Context.* Prentice Hall, Upper Saddle River, New Jersey, USA, 2003.

[6] B. Balacheff, L. Chen, S. Pearson, G. Proudler, and D. Chan. Computing Platform Security in Cyberspace. *Information Security Technical Report*, 5(1):54–63, 2000.

[7] S. Balfe, E. Gallery, C. J. Mitchell, and K. G. Paterson. Crimeware and Trusted Computing. In M. Jakobsson and Z. Ramzan, editors, *Crimeware*. Addison-Wesley, 2008.

[8] S. Balfe, A. D. Lakhani, and K. G. Paterson. Securing Peer-to-Peer Networks using Trusted Computing. In C. J. Mitchell, editor, *Trusted Computing*, chapter 10, pages 271–298. The Institute of Electrical Engineers (IEE), London, UK, 2005.

[9] S. Balfe and K. G. Paterson. Augmenting Internet-based Card Not Present Transactions with Trusted Computing: An Analysis. Technical Report RHUL-MA-2006-9, Department of Mathematics, Royal Holloway, University of London, London, UK, 2005. `http://www.rhul.ac.uk/mathematics/techreports`.

[10] S. Balfe and K. G. Paterson. e-EMV: Emulating EMV for Internet Payments using Trusted Computing Technology. Technical Report RHUL-MA-2006-10, Department of Mathematics, Royal Holloway, University of London, London, UK, 2006. `http://www.rhul.ac.uk/mathematics/techreports`.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield. XEN and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pages 164–177, Bolton Landing, New York, USA, 19–22 October 2003. ACM Press, New York, USA.

[12] B. Beckles, V. Welch, and J. Basney. Mechanisms for Increasing the Usability of Grid Security. *International Journal of Man-Machine Studies*, 63(1-2):74–101, 2005.

[13] E. Brickell, J. Camenisch, and L. Chen. Direct Anonymous Attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS 2004)*, pages 132–145, Washington DC, USA, 2004. ACM Press, New York, USA.

[14] L. Chen, S. Pearson, and A. Vamvakas. On Enhancing Biometric Authentication with Data Protection. In R. J. Howlett and L. C. Jain, editors, *Proceedings of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, volume 1, pages 249–252, Brighton, Sussex, UK, 30th August – 1st September 2000. IEEE.

[15] S. Crane. Privacy Preserving Trust Agents. Technical Report HPL-2004-197, HP Labs, Bristol, UK, 11 November 2004.

[16] T. Dierks and C. Allen. The TLS Protocol. RFC 2246, 1999.

[17] J. McCun eand A. Perrig, A. Seshadri, and Leendert van Doorn. Turtles All The Way Down: Research Challenges in User-Based Attestation. In *Proceedings of 2nd USENIX Workshop on Hot Topics in Security (HotSec 2007)*, August 2007.

[18] E. Gallery and C. J. Mitchell. Trusted Mobile Platforms. In A. Aldini and R. Gorrieri, editors, *Foundations of Security Analysis and Design IV (FOSAD 2007)*, volume 4677 of *Lecture Notes in Computer Science*, pages 282–323. Springer-Verlag, Berlin, Germany, September 2007.

[19] E. Gallery and A. Tomlinson. Secure Delivery of Conditional Access Applications to Mobile Receivers. In C. J. Mitchell, editor, *Trusted Computing*, IEE Professional Applications of Computing Series 6, chapter 7, pages 195–238. The Institute of Electrical Engineers (IEE), London, UK, 2005.

[20] T. Garfinkel, M. Rosenblum, and D. Boneh. Flexible OS Support and Applications for Trusted Computing. In *Proceedings of the 9th USENIX Workshop on Hot Topics on Operating Systems (HotOS-IX)*, pages 145–150, Kauai, Hawaii, USA, 18–21 May 2003. USENIX, The Advanced Computing Systems Association, Berkeley, California, USA.

[21] P. Gutmann. PKI: It's Not Dead, Just Resting. *Computer*, 35(8):41–49, 2002.

[22] Intel. LaGrande Technology Architectural Overview. Technical Report 252491-001, Intel Corporation, September 2003.

[23] M. Kinateder and S. Pearson. A Privacy-Enhanced Peer-to-Peer Reputation System. In K. Bauknecht, A. Min Tjoa, and G. Quirchmayr, editors, *Proceedings of the 4th International Conference on E-Commerce and Web Technologies*, volume 2738 of *Lecture Notes in Computer Science*, pages 206–216, Prague, Czech Republic, 2–5 September 2003. Springer-Verlag, Berlin-Heidelberg.

[24] D. Kuhlmann, R. Landfermann, H. Ramasamy, M. Schunter, G. Ramunno, and D. Vernizzi. An Open Trusted Computing Architecture — Secure Virtual Machines Enabling User-Defined Policy Enforcement. www.opentc.net, June 2006.

[25] H. Löhr, H. V. Ramasamy, A-R. Sadeghi, S. Schulz, M. Schunter, and C. Stüble. Enhancing Grid Security Using Trusted Virtualization. In *Proceedings of the 4th International Conference on Autonomic and Trusted Computing (ATC 2007)*, volume 4610 of *Lecture Notes in Computer Science (LNCS)*, pages 372–384, Hong Kong, China, 11–13 July 2007. Springer-Verlag, Berlin-Heidelberg.

[26] W. Mao, F. Yan, and C. Chen. Daonity: Grid Security with Behaviour Conformity from Trusted Computing. In *Proceedings of the 1st ACM*

workshop on Scalable Trusted Computing (STC 2006), pages 43–46, Alexandria, Virginia, USA, 3 November 2006.

[27] M. C. Mont, S. Pearson, and P. Bramhall. Towards Accountable Management of Identity and Privacy: Sticky Policies and Enforceable Tracing Services. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA 2003)*, pages 377–382, Prague, Czech Republic, 1–5 September 2003. IEEE Computer Society.

[28] M. C. Mont, S. Pearson, and P. Bramhall. Towards Accountable Management of Privacy and Identity Information. In E. Snekkenes and D. Gollmann, editors, *Proceedings of the 8th European Symposium on Research in Computer Security (ESORICS 2003)*, volume 2808 of *Lecture Notes in Computer Science*, pages 146–161, Gjøvik, Norway, 13-15 October 2003. Springer-Verlag, Berlin.

[29] A. Pashalidis and C. J. Mitchell. Single Sign-on using Trusted Platforms. In C. Boyd and W. Mao, editors, *Proceedings of the 6th International Conference on Information Security (ISC 2003)*, volume 2851 of *Lecture Notes in Computer Science*, pages 54–68, Bristol, UK, 1–3 October 2003. Springer-Verlag, Berlin-Heidelberg.

[30] S. Pearson. Trusted Agents that Enhance User Privacy by Self-Profiling. Technical Report HPL-2002-196, HP Labs, Bristol, UK, 15 July 2002.

[31] S. Pearson. How Trusted Computers can Enhance for Privacy Preserving Mobile Applications. In *Proceedings of the 1st International IEEE WoWMoM Workshop on Trust, Security and Privacy for Ubiquitous Computing (WOWMOM 2005)*, pages 609–613, Taormina, Sicily, Italy, 13–16 June 2005. IEEE Computer Society, Washington, DC, USA.

[32] M. Peinado, Y. Chen, P. England, and J. Manferdelli. NGSCB: A Trusted Open System. In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *Proceedings of 9th Australasian Conference on Information Security and Privacy (ACISP 2004)*, volume 3108 of *Lecture Notes in Computer Science (LNCS)*, pages 86–97, Sydney, Austrailia, 13–15 July 2004. Springer-Verlag, Berlin-Heidelberg, Germany.

[33] M. Peinado, P. England, and Y. Chen. An Overview of NGSCB. In C. J. Mitchell, editor, *Trusted Computing*, IEE Professional Applications of Computing Series 6, chapter 7, pages 115–141. The Institute of Electrical Engineers (IEE), London, UK, April 2005.

[34] G. Price. PKI—An Insider's View (Extended Abstract). Technical Report RHUL-MA-2005-8, Department of Mathematics, Royal Holloway, University of London, Surrey, England, UK, June 2005.

[35] A. Pridgen and C. Julien. A Secure Modular Mobile Agent System. In *Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2006)*, pages 67–74, Shanghai, China, 22–23 May 2006. ACM Press, New York, USA.

[36] G. J. Proudler. Concepts of Trusted Computing. In C. J. Mitchell, editor, *Trusted Computing*, IEE Professional Applications of Computing Series 6, chapter 2, pages 11–27. The Institute of Electrical Engineers (IEE), London, UK, April 2005.

[37] J. Reid, J. M. Gonzalez Nieto, and E. Dawson. Privacy and Trusted Computing. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA 2003)*, pages 383–388, Prague, Czech Republic, 1–5 September 2003. IEEE Computer Society.

[38] Electronic Frontier Foundation S. Schoen. Comments on LT Policy on Owner/User Choice and Control 0.8. `http://www.eff.org/Infrastructure/trusted_computing/eff_comments_lt_policy.pdf`, December 2003.

[39] Electronic Frontier Foundation S. Schoen. Give TCPA an Owner Override. `http://www.linuxjournal.com/article/7055`, December 2003.

[40] Electronic Frontier Foundation S. Schoen. Comments on TCG Design, Implementation and Usage Principles 0.95. `http://www.eff.org/Infrastructure/trusted_computing/20041004\_eff\_comments\_tcg\_principles.pdf`, October 2004.

[41] A-R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, and M. Winandy. TCG inside?: a note on TPM specification compliance. In *Proceedings of the 1st ACM workshop on Scalable trusted computing (STC 2006)*, pages 47–56, Alexandria, Virginia, USA, 2006. ACM, New York, NY, USA.

[42] A-R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In C.F. Hempelmann, editor, *Proceedings of the 2004 workshop on New security paradigms (NSPW 2004)*, pages 67–77, Nova Scotia, Canada, 2004. ACM, New York, NY, USA.

[43] R. Sandhu and X. Zhang. Peer-to-peer access control architecture using trusted computing technology. In E. Ferrari and G-J. Ahn, editors, *Proceedings of the 10th ACM symposium on Access control models and technologies (SACMAT 2005)*, pages 147–158, 1–3 June 2005.

[44] L. F. G. Sarmenta, M. van Dijk, C. W. O'Donnell, J. Rhodes, and S. Devadas. Virtual monotonic counters and count-limited objects using a TPM without a trusted OS. In *Proceedings of the 1st ACM workshop on Scalable trusted computing (STC 2006)*, pages 47–56, Alexandria, Virginia, USA, 2006. ACM, New York, NY, USA.

[45] S. E. Schechter, R. A. Greenstadt, and M. D. Smith. Trusted Computing, Peer-to-Peer Distribution, and the Economics of Pirated Entertainment. In *Proceedings of the 2nd Annual Workshop on Economics and Information Security*, 2003.

[46] S. Schoen. Trusted Computing: Promise and Risk. Whitepaper, Electonic Frontier Foundation, October 2003.

[47] SETCo. SET Secure Electronic Transaction 1.0 specification — the formal protocol definition. `http://www.setco.org/set_specifications.html`, May 1997.

[48] R. O. Sinnott. Development of Usable Grid Services for the Biomedical Community. In *Useability in e-Science Workshop: An International Workshop on Interrogating Usability Issues in New scientific Practice, within the Lab and within Society (NeSC 2006)*, Edinburgh, Scotland, UK, 26–27 January 2006.

[49] A. Spalka, A. B. Cremers, and H. Langweg. Protecting the Creation of Digital Signatures with Trusted Computing Platform Technology against Attacks by Trojan Horse Programs. In M. Dupuy and P. Paradinas, editors, *Proceedings of the 16th Annual Working Conference on Information Security (IFIP/Sec'01) of Trusted Information: The New Decade Challenge*, volume 193 of *IFIP Conference Proceedings*, pages 403–419, Paris, France, 11–13 June 2001. Kluwer Academic Publishers, Boston, Massachusetts, USA.

[50] E. Sparks. A Security Assessment of Trusted Platform Modules. Technical Report TR-2007-597, Department of Computer Science, Dartmouth, Hanover, New Hampsire, USA, June 2007.

[51] R. Stallman. *Free Software, Free Society: Selected Essays of Richard M. Stallman*, chapter 17 – Can You Trust Your Computer?, pages 115–119. GNU Press, Boston, Massachusetts, USA, 2002.

[52] TCG. Interoperability Specification for Backup and Migration Services. TCG specification version 1.0 revision 1.0, The Trusted Computing Group (TCG), Portland, Oregon, USA, June 2005.

[53] TCG. Subject Key Attestation Evidence Extension. TCG specification version 1.0 revision 7, The Trusted Computing Group (TCG), Portland, Oregon, USA, June 2005.

[54] TCG. TCG Infrastructure Working Group Reference Architecture for Interoperability (Part I). TCG specification version 1.0 revision 1, The Trusted Computing Group (TCG), Portland, Oregon, USA, June 2005.

[55] TCG. TCG PC Client Specific Implementation Specification For Conventional BIOS. TCG specification version 1.20 final, The Trusted Computing Group (TCG), Portland, Oregon, USA, June 2005.

[56] TCG. TPM Main, Part 1: Design Principles. TCG Specification Version 1.2 Revision 94, The Trusted Computing Group (TCG), Portland, Oregon, USA, March 2006.

[57] TCG. TPM Main, Part 2: TPM Data Structures. TCG Specification Version 1.2 Revision 94, The Trusted Computing Group (TCG), Portland, Oregon, USA, March 2006.

[58] TCG. TPM Main, Part 3: Commands. TCG Specification Version 1.2 Revision 94, The Trusted Computing Group (TCG), Portland, Oregon, USA, March 2006.

[59] TCG. TCG Specification Architecture Overview. TCG specification revision 1.4, The Trusted Computing Group (TCG), Portland, Oregon, USA, August 2007.

[60] F. von Lohmann. Meditations on trusted computing. Electronic Frontier Foundation Article, 2003.

[61] A. Whitten and J. D. Tygar. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium (SSYM 1999)*, pages 14–14, Washington, District of Columbia, USA, 1999. USENIX Association, Berkeley, California, USA.

[62] Z. Yan and Z. Cofta. A Method for Trust Sustainability Among Trusted Computing Platforms. In S. Katsikas, J. Lopez, and G. Pernul, editors, *Proceedings of the 1st International Conference on Trust and Privacy in Digital Business (TrustBus 2004)*, volume 3184 of *Lecture Notes in Computer Science (LNCS)*, pages 11–19, Zaragoza, Spain, 30 August–1 September 2004. Springer-Verlag, Berlin-Heidelberg, Germany.

[63] M. Yung. Trusted Computing Platforms: The Good, the Bad, and the Ugly. In R. N. Wright, editor, *Proceedings of the 7th International Conference of Financial Cryptography (FC 2003)*, volume 2742 of *Lecture Notes in Computer Science (LNCS)*, pages 250–254, Guadeloupe, Frence West Indies, 27–30 January 2003.

Advances on PrivacyCAs

Martin Pirker, Ronald Toegl, Daniel Hein, Peter Danner

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria
{mpirker,rtoegl,dhein,pdanner}@iaik.tugraz.at

ETISS 2008 Workshop on Challenges for Trusted Computing

Abstract. Millions of Trusted Platform Modules (TPMs) have been shipped as part of common personal computers until today. They introduce novel mechanisms which improve the security of computer systems on the Internet. Their Attestation concept support a remote verifiers decision on the trustworthiness of a host. An additional required trusted third party, a PrivacyCA, certifies that the used Attestation Identity Key (AIK) is actually a tamper-protected TPM hosted key. Consequently, a PrivacyCA is required to follow a privacy policy, which a client has to trust in. Unfortunately, this required TC support infrastructure is lacking general availability and does not scale to the number of TPMs available.

In this paper we present a versatile, efficient and platform-independent PrivacyCA service. We demonstrate how such a trusted service can be seamlessly integrated in virtualized environments by creating a minimum Trusted Computing Base (TCB) for it. Furthermore, we improve the trustworthiness on different layers. The protocol interface is generated from a formal specification, the source code is open to inspection and the PrivacyCA is able to attest its runtime state and policy to clients. Finally, we outline how these small, self-contained service compartments can be integrated in future trusted datacenters, to allow massive scalability.

1 Introduction

The current design of common computer systems and networks no longer fulfills the security needs of modern applications in a worldwide connected infrastructure. The recent rise of the concept of *Trusted Computing* introduces a dedicated hardware trust anchor in ordinary personal computers. The Trusted Platform Module (TPM) is able to protect security critical cryptographic operations from software-based attacks. However, the use of protected cryptographic mechanisms alone is not sufficient to convince remote machines or human users that a service can actually be *trusted*. To enable judgement of results and associated trust levels, keys need to be vouched for by suitable entities. Thus, a Public Key Infrastructure (PKI) is required to support Trusted Computing on an Internet scale deployment. So far hundreds of millions of TPMs have been shipped[14]. Consequently, to support such an amount of potential clients a highly scalable architecture is needed.

A particular incarnation of the PKI concept is the *PrivacyCA*. It serves to protect the privacy of the user in a trust-enabled networked environment. In this scenario *Remote Attestation*, the process of supplying sufficient proof that a host is in a well defined state, is a central notion. To achieve this, an entity is required to certify that the reported state is indeed witnessed by a real TPM. Providing the unique identity of the TPM with every attestation would allow to trace the activities of the user. Thus, in order to anonymize the client, it becomes necessary to provide an anonymized certificate that attests that the client uses hardware hosted TPM keys. A PrivacyCA is a PKI that performs this service. It confirms that keys are within the protection of specification-compliant TPM implementations and thus may be trusted under certain conditions – but without revealing the specific TPM host(s).

Outline of the Paper Section 2 gives a short introduction to Trusted Computing mechanisms and related cryptographic keys and certificates. In addition, we argue the need for a Trusted Computing supporting PrivacyCA infrastructure service. In Section 3 we apply different approaches to show how the trustworthiness of such a service can be increased in practice. In Section 4 we discuss how to apply a highly flexible architecture to solve the problem of TC PKI scalability. Putting theory to practice, Section 5 presents a general process to minimize the size of compartments for specialized Trusted-Computing enabled Java applications. Employing it we demonstrate the integration of the PrivacyCA service in distributed and virtualized environments by creating a special purpose compartment. The paper considers related work in Section 6 and concludes in Section 7.

2 Background

2.1 Trusted Computing

Software enforced security can be broken by software based attacks. To overcome this dilemma, the Trusted Computing Group (TCG) [26] has defined a set of specifications, which the *Trusted Platform Module* (TPM) is the integral part of. A TPM is a distinct component on the motherboard of the platform, supplying support for cryptography, random number generation, and secure storage. This chip provides passive services for the system software running on the platform. The TC concept does not claim to enforce perfect security under all conditions and tasks but defines a trustworthy system as *a system that behaves in the expected manner for the intended purpose.*

A system configuration and therefore the system's behavior is identified by the running software stack of which measurements are recorded. A *measurement* is defined as the hash of the binary executable. Each software component is measured before it is given control to, starting with the *Core Root-Of-Trust for Measurement* (CRTM). In a PC platform, the BIOS typically takes over the role of the CRTM. This process leads to a chain-of-trust, following the so-called *transitive trust* model.

To prevent tampering with the platform measurements the TPM utilizes *Platform Configuration Registers* (PCRs). The TPM receives measurements x from system software and hashes the input to the PCR with index i and content PCR_i^t using the *extend* operation

$$\mathrm{PCR}_i^{t+1} = \text{SHA-1}(\mathrm{PCR}_i^t || x).$$

The basic TCG model envisions a *static* chain-of-trust from hardware reboot onwards. Newer developments in the areas of CPUs and chipsets provide the option of a *dynamic* switch to a trusted system state. A special CPU instruction allows to switch the system into a defined security state and then runs a measured piece of software which has full system control. Close hard-wired cooperation of main CPU, the chipset and the TPM guarantees that the result is a system in a thoroughly measured state. Additionally, such TC-enabled hardware platforms [8] also support hardware enforced virtualization, thus providing the concept of an isolated and measured compartment.

In the proposed OpenTC architecture [15], Trusted Computing utilizing the TPM is applied to virtualized environments. Based on a choice of the Xen [3] virtual machine monitor or the Fiasco/L4 [12] μ-kernel, it allows the creation, execution and hibernation of isolated compartments[1], each executing an unmodified guest OS.

We consider the sum of all layers of software which influence the integrity and behavior of an software application to form its *Trusted Computing Base* (TCB). This is reflected in the chain-of-trust which is unique for each specific service.

Thus, a report on the PCR state reflects the exact state of a system. Presented with such a report an external stakeholder can form an informed opinion about a system's trustworthiness.

[1] Note that such hardware-emulating compartments are often called "Virtual Machines". In this paper we use the term exclusively for the language-based Java Virtual Machine.

This central concept of a TC-enabled platform is known as *Remote Attestation*. The authenticity and integrity of this report must be preserved even if the TCB is compromised or the network channel is insecure. To this end, the TPM acts as *Core Root-Of-Trust for Reporting* (CRTR). Upon request, it signs the PCR state with a special key of which the private part is guaranteed to never have left the protection of the TPM. These keys are referred to as *Attestation Identity Keys* (AIKs). To verify the authenticity of the report's signature and thus the secure status of the key, credentials backed by a PKI are needed.

2.2 Trusted Computing PKI Components

In the following sections we outline the components present in a public key infrastructure supporting Trusted Computing and their intended role. Security credentials may be instantiated in different formats, however the credential standard document of the TCG [28] describes credentials in the concrete instantiation of X.509 certificates [13] or attribute certificates [10].

Endorsement Key / EK Certificate Every TPM is designed to host a unique Endorsement Key (EK) pair. The private part is stored in a non volatile memory inside the TPM and cannot be retrieved, once inserted. A corresponding TPM Endorsement certificate hosts the public part of the key pair. This certificate represents an assertion that the specific TPM conforms with the required TCG specifications and that the private Endorsement Key is guarded by a TPM. It is suggested that the TPM manufacturer, the entity which inserted the EK, creates and signs the Endorsement certificate.

As the Endorsement Key uniquely identifies a TPM and hence a specific platform, the privacy of the platform user(s) might be at risk if the EK would be utilized for frequently used operations. As a consequence, the EK was defined to be used only with a strictly limited set of operations, enforced by the TPM.

Platform Certificate A system manufacturer vouches for all parts of a platform except the TPM with a Platform Endorsement (PE) credential. It represents an assertion that the specific platform incorporates a properly certified TPM (thus a reference to the EK certificate is included) and the necessary support components to enable an architecture conforming to TCG specifications.

Attestation Identity Key / AIK Certificate As an alternative to the unique and privacy sensitive EK, the TCG introduced Attestation Identity Keys (AIKs) and associated AIK certificates, which do not contain direct evidence of their specific hosting TPM. A trusted third party, a so called privacy certification authority service (PrivacyCA, PCA), issues these AIK certificates and assures that identity keys are TPM hosted. Thus, in order to create an AIK certificate the following protocol steps take place between a client system with a TPM and the PCA:

– A client application, running on a machine containing a TPM, initiates the TPM functions to a) create a new attestation identity RSA key pair (identity key) and b) a certification request structure intended for the PrivacyCA. The request is encrypted with the public key of the PrivacyCA, which must be known to the client.
– The request is transported to the PrivacyCA.
– The PrivacyCA decrypts the request and validates the its content. Included are, among other information, the EK and PE certificates of the platform. On successful validation the PCA issues an AIK certificate, encrypted with the public key of the EK of the TPM. This assures that the result package can only be decrypted by the intented recipient TPM.
– The PrivacyCA result is transported back to the client system.
– The client asks the systems TPM to decrypt the received data package using the private EK. On successful completion of this protocol the plaintext AIK certificate is obtained.

To summarize, an activated AIK comprises an identity TPM key pair and an associated certificate issued by a third party PCA service, attesting that the key pair is bound to a TPM. An AIK certificate contains information fragments from both, EK and PE certificate, but does not contain a link back to the EK certificate. Additionally, an AIK certificate hosts a client chosen random *label* string which allows for later recognition in a set of AIK certificates and may be considered a pseudonymous identity. This label is a specific TCG certificate extension and must not be confused with standard certificate naming fields.

The TCG standardized the steps, data structures and the PrivacaCA entity outlined above, but no specific protocol is stipulated for exchange of the request and response packets.

3 Our Architecture

3.1 Guidelines for Trusted Services

In order to create a service that justifies being considered a trusted third party, a trade-off between trustworthiness and practicality of implementation needs to be made. Thus, based on our experiences we propose a set of guidelines for building trusted services.

Where possible, a service should be *secure by design.* To formally proof the security properties of all layers in a Trusted Computing design – including BIOS, Operating System, libraries and application code – is currently not feasible. However, critical elements and especially interfaces can be designed and analyzed in a formal way. Likewise, even informal arguing of security is difficult with today's bloated platforms. Thus, it is desirable to keep the *Trusted Computing Base as small as possible.* This includes the removal and deactivation of unnecessary features. In addition, the use of *mature components* is a pragmatic way to gain confidence in the trustworthiness of a host. It is also essential that all components are maintained. The *choice of the runtime environment* should be guided by the needs of a service and help to prevent implementation errors. *Trusted Computing* should be considered in the design of trusted services. A service should be built in a way that it can attest its state and mode of operation to its clients. Finally, iterative refinement should eliminate questionable components, at design time as well as later in the lifetime of the service.

3.2 A Trustworthy PrivacyCA in a Box

We now apply these guidelines to the creation of a PrivacyCA and describe the decisions taken on the different layers.

Network Interface Network entity interaction requires a common protocol understood by all participants. Multiple protocols are available in the area of PKI and credential management. For Trusted Computing a protocol should be able to support common PKI services as well as TC specific attributes, queries and data blobs.

The TCG considered this infrastructure problem in [27]. The two candidates mentioned were the CMC protocol [17] for X.509 certificates and the XKMS protocol [18] for XML-based credentials. The XKMS option appears attractive because it is able to wrap legacy CA services designed for X.509 certificates and express certificate management in XML, as well as providing an attractive upgrade option. However, the significant implementation overhead and complexity introduced by XML weighs against its deployment. A PrivacyCA implementation which uses XKMS was published at [19] and has been operational since 2007.

The exchange of data in the AIK cycle (see Section 2.2) consists of transferring opaque Binary Large OBject data ("BLOBS") and is one of the primary functions of a PrivacyCA. This voids one advantage of XML, which is to keep structured data human-legible. Thus, we believe a simple wrapping representing just the command description and the payload blob(s) is sufficient. Instead of using a verbose XML based protocol, we implement a simple and more compact protocol that can be (semi)-autogenerated from formal specifications. Thus, protocol design can be performed by way of state machine design, which in addition provides an easy, human-understandable level of abstraction.

The notion of a trusted-third-party (TTP) implies that a client connecting to it assumes a secured end-to-end communication channel. We propose the client to use a key derived from the a priori known public PrivacyCA key to establish an encrypted TLS-session.

Trusted Java For the creation of a security aware network service, the Java platform provides a compelling choice. As outlined in [25], Java not only provides a mature platform and several language features which aid programmers in creating robust implementations, but also several libraries for using Trusted Computing features [19] are readily available.

Special Purpose Compartment For functionality and security assessment the code base should be as small as possible. One should aim to include only components which are absolutely necessary for operation. This includes not only the selection of packages which provide just enough functionality, but also custom stripping of selected packages of functionality are deemed unnecessary. It is tempting to intuitively argue that less Lines-of-Code generally equal less defects. However, the vulnerability density is not always linear to size. Overall, the reduction of code base complexity aids the goal of detecting and understanding security issues.

We eschew the creation of a special purpose platform, instead we use a well maintained off-the-shelf operating system and mature library components. This is a practical oriented approach which cannot compete with specifically designed solutions, built and verified from scratch. It offers, however, a good balance of prototyping speed, maturity, features, invested effort and security, thus providing practical trustworthiness for all but the most critical intended service purposes.

Remote Attestation of the Service A PrivacyCA may provide different certification and data retention policies, with the privacy options in the range from *remember nothing* to *remember everything*. In the first case the PCA checks the credentials presented by the client and, if all are found valid, it issues an AIK certificate for the presented public key. Immediately afterwards all information is wiped out from memory. The PCA relies solely on its own certificate signature for certificate validation at a later stage. In the latter case the PCA keeps (more or less) detailed records of credentials received and issued. This offers the opportunity for e.g. mass revocation of certificates linked to a compromised TPM series, but also establishes the risk of information leaks of EK – AIK certificate associations.

Thus, it improves the trustworthiness if a client is able to get a proof of an enforced policy by the means of remote attestation. The Java runtime environment with TC support libraries allow the easy implementation of such a service [9]. Building upon our assumption that we restrict ourselves to a self-contained compartment, its image can easily be measured (hashed) at startup and be part in the chain-of-trust. It can be attested which PrivacyCA software image is running. Additionally, using open source software only, this image is completely open to any method of security and privacy assessment.

4 A Scalable Scenario

In an future trust-enabled Internet a PrivacyCA potentially has to serve millions of customers. Following the thought of previous sections we outline a scenario how a scalable Trusted PrivacyCA (TPCA) could be implemented. Figure 1 illustrates the structure of our TPCA proposal and illustrates its major players.

A datacenter is required to provide the infrastructure that is capable of handling the expected workload. The security of any CA depends on keeping its signing key(s) secret. To ensure this, the scenario presented herein necessitates a Trusted Virtual Datacenter (TVDc) [5]. A TVDc is a datacenter that uses hardware isolation mechanisms to separate the virtual machines that handle

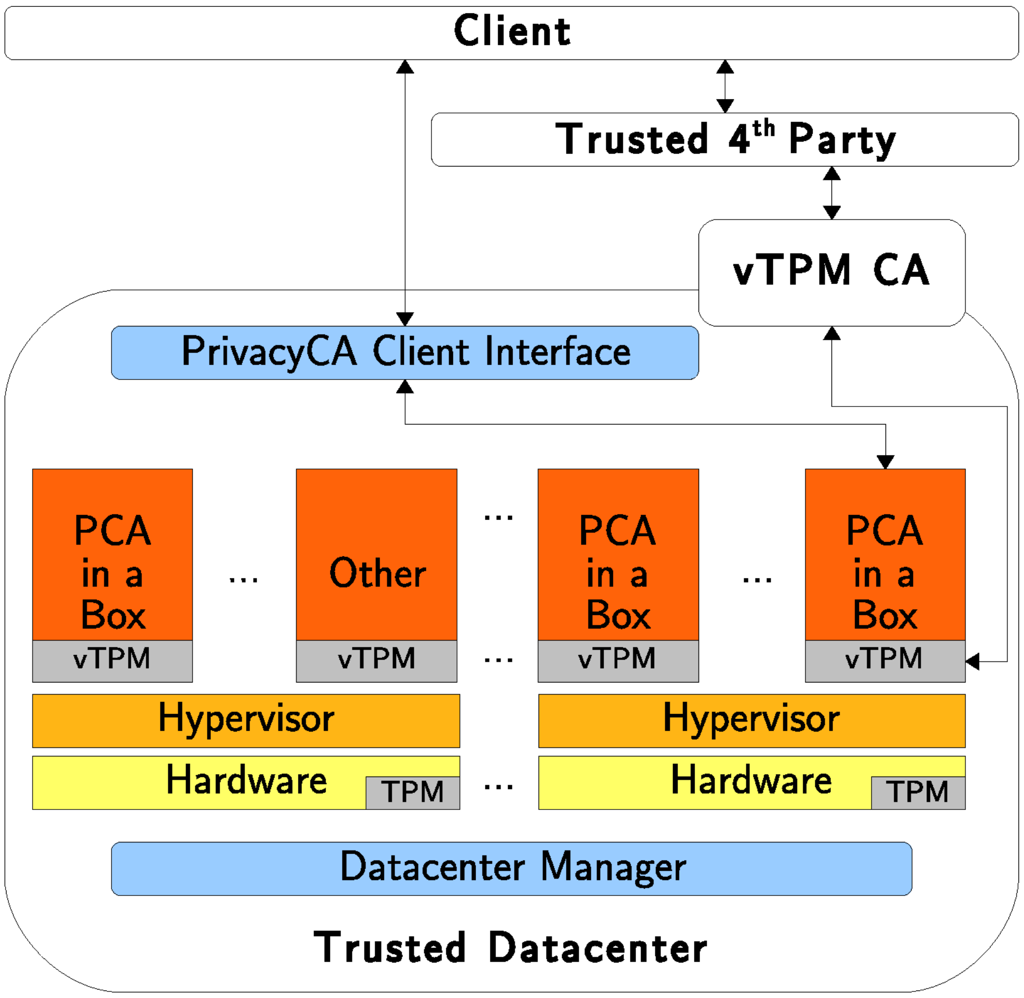

Fig. 1. Structure of a scalable Trusted PrivacyCA (TPCA)

its workload. We also assume that the operator handles physical and operational security in a way appropriate for the services he offers.

For the remainder of the text a virtual machine isolated thus, will be called a secure compartment or just compartment. These compartments are equipped with a virtual TPM (vTPM) [4] [23]. The trustworthiness of the vTPM is extended from a hardware TPM. A trusted compartment is run on a thin hypervisor layer that abstracts the hardware and handles the actual isolation.

The "PrivacyCA in a Box" (PBox) design introduced in Section 3 is the integral component of the TPCA and is run in a secure compartment. It is a minimalistic PrivacyCA implementation that is intentionally kept as simple as possible (of which a prototype will be detailed in Section 5). In this scenario the image of PBox is publicly available for review.

Only two mutable data items that are not part of this image are needed for operation. The first is the signing key that is used to create the AIK certificates issued to a client and the second is a certificate serial number range. This secret key has to remain under the control of the datacenter operator at all times. Yet, to allow seamless scalability of the number of PBoxes running in parallel, it must be distributed to each instance of the PBox. A sealing-based solution is an adapted [24] secret distribution scheme [21]. The serial number range assignment enables the datacenter to run several instances of the same PBox concurrently. It also implicitly defines the maximum lifetime of a PBox. To minimize maintenance and maximize scalability a PBox is build as a fire and forget mechanism. After its alloted serial numbers are depleted the compartment is destroyed and depending on the current workload optionally replaced by one or several new instances.

The datacenter manager is the entity with the authorization to manage the compartments. It is also responsible for deciding when to destroy an exhausted PBox and when to create new PBoxes. It inter-operates with the PrivacyCA client interface, basically a load balancing layer which allows transparent client-PBox interaction, to determine the necessary number of concurrent PrivacyCA compartments.

The virtual TPMs enable PrivacyCA-to-client attestation. In extension binary attestation of the PBox image allows to determine the level of privacy (LOP), which a specific PBox-compartment supplies. It is probably safe to assume that operating a PrivacyCA for the workload projected in this scenario is expensive. A business case might be build on providing different levels of privacy at varying costs.

A paying customer might very well demand proofs that his requests receive the appropriate LOP. What better way to ascertain that than by using the same tools that necessitate the PrivacyCA in the first place?

Keeping track of binary attestation values and mapping them to a property [20] like the associated level of privacy is subject to ongoing research [16] [7] [30]. We assume that the simplicity of the PBox implementation will facilitate this process. Nevertheless its complexity could seriously impede the acceptance of TPCA services.

It is obvious that a PBox-setup issuing the AIK certificates for attesting its own configuration would create a "singularity of trust". To prevent this and to hide the complexity of achieving property based attestation we introduce a trusted fourth party (TFP). Its function is that of an independent observer, or to that end a commercial competitor, who provides additional certification that the customer receives the appropriate privacy level, also considering an analysis of the PBox images used.

The TFP determines the LOP provided by a specific PBox by analyzing the state of a compartment based on a measurement report created and signed using the compartments vTPM. The signature on the report needs to be backed by an AIK-certificate. The AIK authority is derived from an endorsement certificate for the vTPM. The vTPM CA provides these certificates. The exact procedure for this is subject to further research. The same is true for the question whether the vTPM CA should be part of the TVDc, the TFP or if it should be an independent entity.

4.1 Client requesting an AIK

To facilitate the understanding of the TPCA operation, we detail the additional steps necessary for a client to receive an AIK certificate in a trust enabled manner.

1. The client requests an AIK certificate. In addition to the parameters prerequisite to perform the default AIK certificate request cycle, the client also submits an identifier. This is necessary in order to grant the client a specific privacy level.
2. The PrivacyCA client interface forwards the request to one of the PBox compartments appropriate for the expected LOP. The PBox supplements its response with the compartments attestation information and sends it back to the client.
3. The client sends the signed measurement log of the PBox to the fourth party. This entity performs the mapping of the binary measurement values to a privacy level. Also, it verifies the authenticity of the signature on the attestation information depending on the vTPM CA to provide a certificate for the virtual TPM. If the TFP is satisfied, it sends an appropriate response to the client, which is then able to complete the transaction.

5 Prototype Implementation

We implemented parts of the architecture and scenario described in the previous sections. We implemented a basic, yet usable PrivacyCA service and at the same time assembled a minimum-sized compartment that provides an environment suitable to run it.

The supported PrivacyCA operations and thus the required runtime components are very limited. The target to minimize the environment severely restricts the available components, but it becomes easier to estimate what adding a new feature entails on the runtime side. The following sections summarize our efforts and choices made.

5.1 PrivacyCA Features

Our first prototype of a PrivacyCA supports the following command set:

ekcert_create This command creates a TPM endorsement certificate for a supplied public key. To our knowledge only one vendor, Infineon, supplies its TPM with an Endorsement certificate. For all other TPMs a trusted entity must create an Endorsement certificate.
ekcert_validate This command validates a TPM endorsement certificate. Our PrivacyCA recognizes Infineon certificates and certificates issued by the previous command.
aik_create Implements the AIK certificate creation cycle as specified by the TCG.
aik_validate Provides a function to validate the AIK certificates issued by our PrivacyCA.
aik_locate Offers a search function for retrieval of a specific AIK certificate, with the AIK label used as the search key.
aik_revoke After revocation the PrivacyCA no longer successfully validates an AIK certificate.

For a compact and robust communication protocol we devised a simple ASCII based solution. Basically, for most commands it is sufficient to prefix a command identifier, followed by the data. Binary data payload is transmitted as Base64 encoded strings. The following example illustrates the basic command request and response of the aik_create command:

```
Request:                          Response:
"CREATE_AIK_REQUEST" "\n"         "CREATE_AIK_RESPONSE" "\n"
"Blob: " base64 "\n"              "Blob1: " base64 "\n"
".\n"                             "Blob2: " base64 "\n"
                                  ".\n"
```

The command emitter code is reduced to a straightforward line by line output of a command and thus not very error prone. The server side parsing is implemented using the Ragel state machine compiler[2] resulting in mostly automatically generated parsing code. Ragel generates executable finite state machines from a regular-expression based input language. It allows to generate code for multiple target languages, not only for Java and thus we hope this encourages development of clients in alternative languages. The operations implemented so far can be mapped to an one request requires one response approach. Looking ahead, however, we expect interaction to become more complex and thus the application of a state machine not only to raw protocol parsing but also for modeling complex series of commands promises a solid foundation.

5.2 Software Image Reduction

The interacting components can be assigned to the following layers: PrivacyCA application, JVM/JRE, OS runtime support, OS kernel and TPM.

At the top level is the actual PrivacyCA application. In order for the Java bytecode to execute, a Java Runtime Environment (JRE) with a Java Virtual Machine (JVM) at its core is required. It consists of a runtime library providing a set of common basic Java classes and a bytecode interpreter (or just-in-time compiler). The virtual machine makes use of the capabilities offered by the native environment. The native environment is composed of a set of high-level application libraries (e.g. graphics, printing, and sound), the C/C++ standard libraries and operating system functions. The operating system kernel implements the low-level services.

[2] A. Thurston, Ragel State Machine Compiler, `http://research.cs.queensu.ca/~thurston/ragel/`,

This full software stack is able to run in a designated virtualized compartment, unaware of the underlying virtualization layer. Hardware virtualization ensures protection from interference of compartments running in parallel on the same machine.

We tailored an environment for the execution of our specific PrivacyCA application program by starting with a running configuration and then reducing the included functionality in the respective layers to the absolute minimum. The process of minimizing the software components by identifying and removing features which are not relevant to the execution of the PrivacyCA, is as follows:

Java Environment To provide the best possible Java compatibility and allow reuse of existing code we choose IcedTea[3], which is based on Sun's official OpenJDK[4]. IcedTea supplies a new build process for OpenJDK which employs only software components available as free-licensed open source software, thus aiding maximum modifiability.

The runtime of a current Java platform well exceeds 100MB in its default configuration. The actual part of it which is needed to run a PrivacyCA is much smaller. First, we can reduce the runtime environment with a dynamic analysis of the classes being loaded while the PrivacyCA is executed. This is accomplished by enabling the class loading profiling feature of the Java VM. Subsequently these classes are copied from the original full-sized runtime and accumulated into a new custom environment. Of course, *Exception* and *Error* classes which are required in case the application's behavior changes due to external events are included. In addition, the set of dynamically loaded *.so* libraries is monitored by using the debug features of the system dynamic linker/loader *ld.so*. This enables us to carry over only those native libraries needed. Additionally, we manually prune miscellaneous unnecessary data files, such as dispensable localization data, images and unused configuration files. This approach reduces the Java runtime for a specific application to a more manageable size in the range of 10 to 20MB. Note that this approach requires manual intervention and reasonable completeness is only achievable for small applications.

A Small Kernel After selection of a target JVM we require an OS which is small in size, yet powerful enough to support the IcedTea Java Virtual Machine. As a further trait, we prefer a candidate which is known for maturity and availability of maintenance.

Of the open source operating systems, GNU/Linux is widely used and actively maintained by a large global community. Virtualization of Linux compartments is possible using the Xen hypervisor[5]. Also, GNU/Linux easily hosts the most recent Java environments, including running and building IcedTea. The Linux kernel features hardware drivers for TPMs of all vendors. It also provides fine-grained configuration options on which parts to include in the resulting kernel image or built as kernel modules. This allows selection of only those capabilities required by our application. This configuration consists of essential kernel functionality and a small set of drivers to enable running standalone or in a virtualized compartment.

A Minimal Runtime The standard `glibc` system library interface to the Linux kernel uses about 20 to 25MB disk space on a typical installation, including several support tools and data files. Additional system tools for simple file manipulation, kernel module loading utilities and the standard `bash` system shell required for the boot process accumulate to over 3MB. The size of an off-the-shelf Linux distribution violates our requirement of a small and manageable system image.

For support of the boot process of the system, we replace the commonly used GNU core utilities along with kernel module tools and the default `bash` shell with the much smaller *Busybox*[6] toolkit. It provides a minimal userland program environment. We further minimize the boot time

[3] `http://icedtea.classpath.org/`

[4] `http://openjdk.java.net/`

[5] Operating environments based on the L4 microkernel can likewise run a paravirtualized version of the Linux kernel.

[6] `http://www.busybox.net/`

complexity by using a set of base and init shell files provided by the *sys-apps/baselayout-lite* package made available by the *Embedded Gentoo* project[7]. In contrast to the variants usually employed in desktop-oriented distributions this package features a minimal set of configuration files needed for starting and running a GNU/Linux system.

Considering the requirement on size we chose the uClibc[8] as an alternative C library with a drastically reduced footprint. It only needs about half the size of the library files compared to `glibc` for a complete installation. uClibc implements parts of the GNU `libc` API, but omits some specialized and nowadays rarely used functions. However, this provides enough functionality to startup and run Java.

Results for a minimal PBox Compartment The size of the components in a snapshot of our current prototype can be approximated as follows, with the total compartment reaching a size of approximately 16 Megabytes.

OS Kernel: 900 kB Linux kernel; *OS runtime*: 451 kB BusyBox, 65 kB Baselayout-lite; *C libraries*: 3545 kB libstdc++, 1096 kB uClibc, 45 kB GCC runtime; *Java Core*: 8100 kB Stripped Icedtea JRE; *Java Application*: 102 kB PrivacyCA server core, 789 kB lib IAIK JCE, 140 kB lib JTSS TSP, 48 kB lib TCcert.

6 Related Work

To our knowledge the first experimental public PrivacyCA responder[9] service was put into operation at IAIK as part of the OpenTC[10] project and served as basis for the advanced results reported in this paper.

A seemingly private effort to offer a PrivacyCA was announced[11], but this responder only supports creation of AIK certificates. Others works [31] demonstrate interaction with a PrivacyCA service, however have not released any prototype so far.

Java is still evolving and thus older results may not reflect newer research developments or productive services. In a recent proposal, Anderson et al. [2] created a small sized Xen library OS running exclusively on top of the Xen hypervisor. Due to the lack of basic features it is not able to run a modern Java Runtime Environment such as OpenJDK. There also exists a significant body of research on Java virtual machines and using Java as an operating system or a component thereof. However, most previous results [11], [29] or projects (SanOS[12]) are currently unmaintained. Also, mobile Java platforms such Sun's KVM[13] may be smaller, however their feature set is massively restricted and not compatible with full Java applications or libraries. More recent efforts like JNode[14] or [1] do not consider that binary size is a goal, thus resulting in a TCB is too large for our purposes. The ongoing *Java Kernel*[15] project features a method of dividing JRE libraries into separate bundles, which are later fetched at runtime as required. The dynamic loading of classes from remote services and the reliance on a full-featured Windows environment introduces additional overhead and possible extra points of attack.

As alternative to the trusted third party concept of PrivacyCA, [6] proposes Direct Anonymous Attestation. TPM implementations are available, yet the required software infrastructure has not been provided for. It remains a theoretical concept so far.

[7] `http://www.gentoo.org/proj/en/base/embedded/`
[8] `http://www.uclibc.org/`
[9] `http://opentc.iaik.tugraz.at/`
[10] `http://www.opentc.net/`
[11] `http://sourceforge.net/mailarchive/forum.php?thread_name=da7b3ce30801131643j74be4064l52daa8c0e90efa83%40mail.gmail.com&forum_name=trousers-users`
[12] `http://www.jbox.dk/sanos/`
[13] `http://java.sun.com/products/cldc/wp/KVMwp.pdf`
[14] `http://www.jnode.org/`
[15] `http://weblogs.java.net/blog/enicholas/archive/2007/05/java_kernel_unm.html`

7 Conclusions and Future Work

In this paper we describe the creation of an advanced PrivacyCA service. We incorporate the results of operating a experimental public prototype setup for more than a year. We also demonstrate how a trusted service may be built using technologies such as protocol generation or virtualization. We demonstrate a way to reduce the TCB of a Java-based service significantly and provide a self-contained, trusted PrivacyCA-service compartment image that requires less than 16 MB. We also propose to apply such images in parallel in the context of trusted virtual data centers, thus handling the problem of PKI scalability.

Future work will consider use cases in the context of distributed computing and grids [30] and advanced certification mechanisms for virtual TPMs. An integration of the newest TCG credential type, an *unified credential*[28], may stimulate new workflows. We also desire to work on closing the gap between automatic generation of an implementation and the formal security analysis of network protocols and to apply this to future extended PrivacyCA interfaces. We identify the need for an automated process for the identification of required runtime components, using e.g. employing static source analysis.

Singaravelu et al. [22] suggest to extract security critical modules out of the legacy application. Each of these modules is transferred into a separate, trusted compartment called *AppCore* which features a small TCB. We believe that our reduced Java environment is ideally suited to implement similar modifications for Java applications and hope to demonstrate this in the future.

References

1. G. Ammons, J. Appavoo, M. Butrico, D. D. Silva, D. Grove, K. Kawachiya, O. Krieger, B. Rosenburg, E. V. Hensbergen, and R. W. Wisniewski. Libra: a library operating system for a jvm in a virtualized execution environment. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 44–54, New York, NY, USA, 2007. ACM.
2. M. J. Anderson, M. Moffie, and C. I. Dalton. Towards trustworthy virtualisation environments: Xen library os security service infrastructure. Technical Report HPL-2007-69, HP Research, 2007.
3. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.
4. S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vTPM: virtualizing the trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 305–320, 2006.
5. S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan. TVDc: managing security in the trusted virtual datacenter. *SIGOPS Oper. Syst. Rev.*, 42(1):40–47, 2008.
6. E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145, New York, NY, USA, 2004. ACM.
7. L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, and C. Stüble. A protocol for property-based attestation. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, 2006.
8. David Grawrock. *The Intel Safer Computing Initiative.* Number ISBN 0-9764832-6-2. Intel Press, 2006.
9. K. Dietrich, M. Pirker, T. Vejda, R. Toegl, T. Winkler, and P. Lipp. A practical approach for establishing trust relationships between remote platforms using trusted computing. In G. Barthe and C. Fournet, editors, *Trustworthy Global Computing*, volume 4912 of *LNCS*, pages 156–168. Springer Verlag, 2008.
10. S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization. `http://www.ietf.org/rfc/rfc3281.txt`, Apr. 2002.
11. M. Golm, M. Felser, C. Wawersich, and J. Kleinöder. A Java operating system as the foundation of a secure network operating system. Technical report tr-i4-02-05, Univ. of. Erlangen, Dept. of Comp. Science, Lehrstuhl 4, 2002.
12. M. Hohmuth. The Fiasco kernel: Requirements definition. Technical Report ISSN 1430-211X, Dresden University of Technology, 1998.

13. R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate and CRL Profile. `http://www.ietf.org/rfc/rfc3280.txt`, Apr. 2002.
14. IDC. The Future of Trusted Computing. `https://www.trustedcomputinggroup.org/news/Industry_Data/IDC_Presentation.pdf`.
15. D. Kuhlmann, R. Landfermann, H. V. Ramasamy, M. Schunter, G. Ramunno, and D. Vernizzi. An open trusted computing architecture — secure virtual machines enabling user-defined policy enforcement. Research Report RZ 3655, IBM Research, 2006.
16. U. Kühn, M. Selhorst, and C. Stüble. Realizing property-based attestation and sealing with commonly available hard- and software. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, 2007.
17. M. Myers, X. Liu, J. Schaad, and J. Weinstein. Certificate Management Messages over CMS. `http://www.ietf.org/rfc/rfc2797.txt`, Apr. 2000.
18. S. H. Mysore and P. Hallam-Baker. XML key management specification (XKMS 2.0). W3C recommendation, W3C, June 2005. `http://www.w3.org/TR/2005/REC-xkms2-20050628/`.
19. M. Pirker, T. Winkler, R. Toegl, and T. Vejda. Trusted computing for the Java^{TM}platform. `http://trustedjava.sourceforge.net/`, 2008.
20. A.-R. Sadeghi and C. Stüble. Property-based attestation for computing platforms: caring about properties, not mechanisms. In C. Hempelmann and V. Raskin, editors, *NSPW*, pages 67–77. ACM, 2004.
21. P. E. Sevinç, M. Strasser, and D. Basin. Securing the distribution and storage of secrets with trusted platform modules. In *WISTP 2007*, pages 53–66, 2007.
22. L. Singaravelu, C. Pu, H. Härtig, and C. Helmuth. Reducing TCB complexity for security-sensitive applications: three case studies. In *EuroSys '06: Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 161–174, New York, NY, USA, 2006. ACM.
23. F. Stumpf, M. Benz, M. Hermanowski, and C. Eckert. An approach to a trustworthy system architecture using virtualization, 2007.
24. R. Toegl, A. Leung, G. Hofferek, K. Greimel, R. Phan, and R. Bloem. Formal analysis of a TPM-based secrets distribution and storage scheme. In *Proceedings of TrustCom 2008*, 2008. Accepted for publication.
25. R. Toegl and M. Pirker. An ongoing game of tetris: Integrating trusted computing in Java, block-by-block. In *Proceedings of Future of Trust in Computing*. Vieweg + Teubner, 2008. in print.
26. Trusted Computing Group. `https://www.trustedcomputinggroup.org/`.
27. Trusted Computing Group. TCG Reference Architecture for Interoperability (Version 1.0). `https://www.trustedcomputinggroup.org/specs/IWG`, June 2005.
28. Trusted Computing Group. TCG Credential Profiles Specifications (Version 1.1, rev 1.014). `https://www.trustedcomputinggroup.org/specs/IWG`, May 2007.
29. L. van Doorn. A secure Java virtual machine. In *Proceedings of the 9th UESENIX Security Symposium*. USENIX Association, 2000.
30. T. Vejda, R. Toegl, M. Pirker, and T. Winkler. Towards Trust Services for Language-Based Virtual Machines for Grid Computing. In *Proceedings of TRUST 2008*, LNCS. Springer Verlag, 2008. in print.
31. J. Zic and S. Nepal. Implementing a portable trusted environment. In *Proceedings of Future of Trust in Computing*. Vieweg + Teubner, 2008. in print.

Attacking the BitLocker Boot Process

Sven Türpe, Andreas Poller, Jan Steffan, Jan-Peter Stotz
and Jan Trukenmüller
Fraunhofer-Institute for Secure Information Technology SIT
Rheinstrasse 75,
64295 Darmstadt, Germany
{tuerpe,poller,steffan,stotz,truken}@sit.fraunhofer.de

We discuss attack scenarios against the TPM-based boot process of BitLocker. BitLocker is a disk volume encryption feature included in some recent versions of Microsoft Windows. BitLocker is capable of using the TPM to manage all or a portion of its secret encryption keys. Specifically it uses the sealing feature to ensure keys are released only if the platform is in a predefined, trusted state. We present six ways in which an attacker may gain access to secret key material by manipulating the boot process in ways not prevented by the trusted computing technology. We also discuss their causes and contributing factors.

1 Introduction

The trusted computing platform as specified by the Trusted Computing Group does not support secure booting [1, 13]. Not in a strict sense, that is: there are indeed features in the platform that allow the TPM to keep track of the boot process and notice tampering with most (but not all [5]) components involved. Other functions of the TPM, such as remote attestation and sealing, are designed to use the result of it.

This leads to the question what exactly the capabilities and limitations of the existing functions are. In this paper we explore this question along one particular software design and implementation: BitLocker. Included with some editions of Microsoft Windows Vista and Windows Server, BitLocker encrypts volumes on disk and uses the sealing function of a v1.2 TPM for some aspects of its key management. We discuss several attack scenarios against the BitLocker boot process. So far, our work is limited to theoretical considerations and to analysis. We do not (yet) discuss practical implementation of the attacks we describe, and we try to fully understand the problem before devising solutions. Our analysis is based on the version of BitLocker included with pre-SP1 versions of Windows Vista Ultimate. We expect our points to remain valid for the SP1 version but haven't verified this yet.

The remainder of this paper is organized as follows. Section 2 briefly describes the design of BitLocker, focusing on its key management and how it is using the TPM. Our adversary model is outlined in section 3. Section 4 describes attack scenarios that seem feasible and either yield secret key or data or achieve some important steps towards a successful attack. Causes and contributing factors are discussed in section 5, followed by the conclusions in section 6. Related literature is referenced where appropriate but not specifically discussed.

A Disclaimer Note that there are two distinct attack strategies against which BitLocker should ideally protect. *Opportunistic* attacks use only what is easily obtained under common real-world conditions. An example is recovering data on a disk or computer that has been bought in used condition from somebody else, or stolen somewhere. A *targeted* attack is different in that the attacker attempts to get access to data on a specific, predetermined disk or machine, usually within some time and resource constraints. According to Microsoft, BitLocker is designed to withstand at least opportunistic attacks. Considering targeted attacks as we are doing here may be beyond its specification. However, disk encryption along with TPM-based key management might be expected and perceived to be more powerful than what the manufacturer is willing to promise, and we deem it useful to explore the actual security properties and limitations regardless of claims and cautionary notes.

2 An Overview of BitLocker

2.1 Security and Attack Objectives

The primary security objective is confidentiality of any data stored on the encrypted volume. As a corollary, secret keys that could be used, along with the contents of the disk, to obtain the cleartext are to be kept confidential as well. An attack against BitLocker can be considered successful if through the attack:

- the attacker obtains the cleartext of all data on the encrypted volume, or a considerable portion thereof, or
- the attacker obtains such supposedly secret key material that obtaining the cleartext from ciphertext becomes trivial, and obtaining the ciphertext is not more difficult than it would be to obtain cleartext from an unencrypted volume, all side conditions being the same.

Manipulating either the BitLocker instance itself or its execution environment would be an obvious and straightforward way of obtaining both cleartext data and secret key material. Integrity of the BitLocker software and the platform executing it is therefore an important secondary security objective. A tertiary objective is availability; there are few usage scenarios where it is acceptable to risk losing all data just to keep them confidential under all circumstances.

2.2 Integrity Model and Design Constraints

BitLocker works, at boot time, as a component of the boot loader and later as a driver of the operating system kernel. Its design assumes that the kernel boots from a BitLocker-protected volume, that BitLocker sufficiently protects the integrity of data on this volume, and that anything that happens after initiating the OS boot process is sufficiently controlled by other security mechanisms. We do not challenge these assumptions here; see [6, 8] for two known attacks against the running system.

According to these assumptions, BitLocker has to protect the integrity of the boot loader and its execution environment up to the point where the kernel can be read from the locked volume. This code is read from an unencrypted part of the disk and needs to be supplied with a secret key for the AES algorithm. This is where the TPM is being used in. BitLocker uses the *sealing* function to store all or part of its key material in such a way that it becomes accessible only if the platform configuration as represented by the PCR values is in line with the reference configuration. The reference configuration is determined by the administrator accepting the current system configuration at some point in time. This adoption of a reference configuration is initially done during BitLocker activation but can be repeated at any time from the running Windows system.

2.3 Key Management and Recovery Mechanisms

Apart from special cases—BitLocker can also be operated without a TPM or with all key material being managed by the TPM—key material is divided. One part is managed by the TPM and released only if the platform is in the trusted state, the other is supplied by the user as a password and/or key file on a USB memory stick.

If the TPM works as desired, there is no way according to the design to gain access to all required key material if the platform state measured is different from the reference state. This is intended if the platform state is modified by an attack, it is not, if state is modified for a legitimate reason and the change can not be reverted easily, e.g. after component failure and repair. BitLocker therefore offers two recovery mechanisms, the recovery password and the recovery key. Both are designed to circumvent the TPM and supply BitLocker with its secret key independent of the current platform state. The recovery mechanisms don't correct the problem, though. This is left to the administrator who, after the recovery boot, may set a new reference state from the running system.

The actual encryption key does not change during the recovery process.

2.4 User Experience

The user experience hides most of the details. When switching on their PC, users will experience a text-mode prompt for their PIN and/or USB stick if the platform is in reference state. Otherwise they will be prompted for their recovery key or password. Depending on how the computer is being used, users may experience a recovery prompt from time to time, e.g. after accidentally leaving a bootable CD or DVD in the drive or a bootable USB stick plugged into their computer.

3 Adversary Model

As any disk encryption scheme, BitLocker is supposed to protect the confidentiality of data against an adversary who gets physical access to the computer or even brings the computer into his possession. We therefore assume the attacker is interested in the data stored on the BitLocker-protected volume. Furthermore, we limit our discussion to such attacks that do actually exploit physical access in some way.

- Copy the encrypted volume and any other data on disk but return the computer
- Modify the encrypted volume and any other data on disk and return the computer
- Modify the hardware and return the computer
- Take away the computer, returning it later or not
- Replace the computer with an identical-looking copy
- If modified software gets executed on the machine, use any peripheral component of the system

We will not discuss the effort required in each case but assume that each of these attack building blocks is considerably cheaper than brute-forcing AES. We further assume that each of these physical attack building blocks may be executed an arbitrary number of times in combination with any other, provided a single rule is observed: if the attack becomes obvious, the user and administrator will be cautious in all subsequent steps. In other words, the user and administrator will not knowingly support an attack.

4 Attack Scenarios

4.1 Replace and Relay

This is a hardware-level phishing attack. The attacker replaces the entire target machine with another computer prepared for the attack. The replacement, when turned on, produces all the messages and prompts that the original machine would have produced. Up to the point where BitLocker would start, it takes all user inputs (via keyboard or USB) and relays them to the attacker, e.g. using radio. The attacker, being in possession of the unmodified original system, uses this information to start up the stolen computer.

Requirements This attack requires that:

- the attacker is capable of replacing the BitLocker-protected machine altogether with an identically-looking copy, and
- the machine is plausibly turned off or in suspend-to-disk mode when the legitimate user returns, and
- the replacement device is capable of relaying user input to the attacker.

The attacker will have to remain—or leave some device—in proximity to the target until the next boot is initiated by the victim. The attacker will also need some prior knowledge of non-secret facts, specifically everything that might be needed to perfectly reproduce the user experience.

Result As a result of this attack, the attacker receives the user-controlled secrets. Depending on the mode in which BitLocker is deployed on the target system, the result is either key material or authentication credentials or both. Either one can be used in conjunction with the unmodified system to start up the operating system. Security mechanisms of the operating system remain intact; another attack will be required to actually access any encrypted data. Such attacks exist [6, 8]. The attack will likely be noticed right after the victim provided credentials or keys to the spoofed machine. This attack may be combined with any attack that yields the TPM-managed portion of the key material.

Extensions and Variants A more sophisticated version of this attack involves two-way communications, turning the replacement into a terminal of the stolen target machine. This would probably require quite some additional effort but might extend the time span between success and detection of the attack. All variants of this attack may also be attempted against recovery mechanisms, which yields sufficient key material to decrypt disk contents immediately.

4.2 Plausible Recovery

The attacker modifies the BitLocker code on disk, adding a backdoor. Such a backdoor could be as simple as saving a clear key in some location on disk or elsewhere in the system from where it can be retrieved later. This modification will of course be detected the next time the system is started by a legitimate user. However, the attacker hopes that the user applies one of the TPM-independent recovery mechanisms to overcome the problem. The attacker later visits the system again to collect the key. Encrypted volume data could be copied during each visit to the target system as the actual encryption key does not usually change.

Requirements This attack requires:

- that recovery mechanisms are used at all, and
- that the attacker can physically access the target machine at just the right time without taking it away permanently, and
- that the reported platform validation error seems plausible for the victim.

One obvious implementation of this attack would be to wait for a situation that plausibly changes the state of the platform, such as a repair. It may also be possible to provoke such a situation. The attacker will then have to sneak into the process somewhere before the user accepts the seemingly legitimate modification. This would mask the malicious change with the legitimate one.

Result If the attack succeeds, the attacker has successfully planted a backdoor into the system in such a way that *all* software-based security features could be circumvented. The attack is unlikely to be noticed by the victim. In order to get both the encrypted data and the secret key the attacker will have to visit the target system at least twice. However, the backdoor may also use other channels to leak cleartext data, possible increasing the risk of detection.

4.3 Spoofed Prompt

Similar to the plausible recovery attack, the attacker modifies BitLocker on the target system and lives with the fact that the TPM will detect this modification. The attacker adds code that spoofs the user interface of BitLocker up to the point where the user has given up his secrets. The malicious code may spoof either the normal-operations UI or the prompt for a recovery key.

Requirements This attack requires that the attacker can physically access the target system. It is not necessary that the attacker takes the system away permanently.

Result The attack is easily detected as soon as secrets have been provided to the spoofed prompt. After detection it is generally possible to prevent the attacker from interacting with the compromised system again. Also, the TPM will refuse to unseal its part of the key material while the platform is in this modified state. If a recovery prompt is successfully spoofed and operated by the user, the attack will yield sufficient key material for decryption of a volume.

Extensions Although it may work under some circumstances, this attack does not appear very critical. However, the next subsection describes a more critical extension.

4.4 Tamper and Revert

The tamper and revert attack extends the spoofed prompt attack. Instead of simply accepting that platform modifications can be detected, the attacker attempts to exploit tampering yet hiding it. This becomes surprisingly easy if one additional boot cycle is possible. The attacker could make a temporary modification to TPM-verified code. If we stick to the spoofed prompt example, this means to add a cleanup function to the malicious code, whose purpose it is to restore the former platform state. After a reboot—which might be initiated by the malicious code after showing a bogus error message—the platform state as measured will be compliant with the reference PCR values again.

Requirements Requirements are similar to those of the *spoofed prompt* attack. In addition the attacker needs to get away with a boot cycle after platform integrity failure without disturbing the victim so much as to spoil further steps of the attack. Depending on how the credentials or keys obtained are transmitted to the attacker, a further visit to the system may or may not be required.

Results This attack yields copies of keys controlled by the user. In a simple implementation these keys will end up in clear somewhere on the target system itself but more sophisticated approaches can be imagined, for instance sending key somewhere using a built-in WLAN interface. Additional effort is required on the attacker's part to gain access to TPM-managed key material.

4.5 Preemptive Modification

This attack is similar to the plausible recovery attack, but at a different point in time. The recovery attack targets systems on which BitLocker has already been activated. Preemptive modification attacks earlier, before BitLocker has been activated at all.

When defining the reference state for future booting, the operator has no choice other than using the current platform state. BitLocker does not provide the user with any means of verifying that this current state has or hasn't any particular property. If an attacker manages to modify critical parts of the platform before BitLocker is activated, this modification therefore goes unnoticed and will be incorporated into the trusted (but not trustworthy) platform state.

Requirements Preemptive modification requires that the attacker gets physical access to the target system *before* BitLocker is activated. Arbitrary modifications are possible at that time that would weaken the security of the BitLocker instance affected forever. Another physical visit may be required later to retrieve a disk image for decryption or leaked cleartext. However, the system may also be modified in such a way that it leaks data at runtime. Everyone who gets physical access to the machine or OS installation media before BitLocker setup is a potential attacker.

Results The attacker potentially gains read and write access to all data handled on the system throughout its lifetime. This attack is hard to detect unless there are additional means of verifying the integrity of executable code against external references.

4.6 TPM Reset

TPM reset attacks have been described in the literature before and are included here for the sake of completeness. The attacker, in temporary or permanent possession of the target system, attaches equipment to the hardware to record the measurements sent to the TPM during a clean boot process. Next the machine is booted with a system of the attacker's choice, e.g. from a CD. While this system is running, the attacker performs the reset attack and replays the clean sequence of measurements to the TPM. As a result, the attacker will be able to unseal the TPM-managed portion of the key material outside the trusted operating system.

Reset attacks have been described and demonstrated against implementations of version 1.1 of the TCG specification. The current version 1.2 contains mitigations that make such attacks more difficult to mount but by no means impossible.

Requirements The attacker needs to be in possession of or co-located with the target computer for some amount of time. This would be the case e.g. after the attacker has stolen the target computer. Mounting the reset attack may require irreversible modifications to the hardware, traces of which could be detected at least by close examination.

Results As a result of this attack, an attacker in possession of the target computer is able to extract key material sealed by the TPM. The only prerequisite is that the attacker must be able to boot the machine into the trusted platform state at least once. In other words, this attack cannot be applied after one that modified the platform in a detectable way unless this modification is reversible.

5 Causes and Contributing Factors

This section identifies factors that make the overall system—a PC with BitLocker and Trusted Computing technology—vulnerable to the attacks described above. Factors include fundamental properties of the security mechanisms involved as well as features in the design and implementation of BitLocker and the Trusted Computing platform.

Passive TPM. Theory suggests that secure booting requires an appropriate action if the measured state deviates from the reference. For instance the boot process might be halted, or it may be possible to fix the issue once it had been detected [13]. The Trusted Computing platform, however, has been designed to work with a *passive* TPM: functions like sealing and attestation depend on prior measurements of platform state, but the TPM does not actively enforce anything. Our attack scenarios confirm the requirement of active enforcement for secure boot. The spoofed prompt and tamper and revert attacks would be much harder to carry out if the boot process would stop immediately after a modification had been detected. However, recovery features may also be harder to implement in this case.

No trusted path to the user. BitLocker uses secrets to authenticate the user: the PIN and key material. The channel between the legitimate user and the system in a trusted state is prone to spoofing and man-in-the-middle attacks (replace and relay; spoofed prompt; and tamper and revert). or specifically, the system lacks context-awareness and the user is unable of authenticating the system. Similar problems exist elsewhere, e.g. ATM skimming. Both directions of authentication can be discussed separately:

- *No context-awareness.* The BitLocker has no means of determining whether the computer is under control of a legitimate user or somebody else. It simply assumes that whoever provides the correct key or credential is a legitimate user. Although requesting a PIN or key may be interpreted as authentication, it is not a very strong one, and adding stronger authentication may be difficult.
- *Lack of system authentication.* While BitLocker is capable of authenticating its user at least in the weak sense described above, the user has no means of verifying authenticity and integrity of the device. Keys and passwords are to be entered into an unauthenticated computer.

History-bounded platform validation. The Trusted Computing platform detects and reports platform modifications only within the scope of the current boot cycle. BitLocker uses this feature through the sealing function of the TPM and does not add anything. The system is therefore unable to detect, and react to, any tampering in the past that has not left permanent traces in the system.

Incomplete diagnostic information. If current and reference state are out of sync, it is difficult or even impossible for the user or administrator to determine the exact cause(s). This leaves the user with a difficult choice: to use recovery mechanisms blindly, or not to use them at all. The lack of diagnostic information contributes to the plausible recovery attack. Note that detailed diagnostic information may not be required where a trusted state can be enforced, e.g. by re-installing software from trusted sources.

Lack of external reference. This is another issue that has already been discussed in the literature. BitLocker is capable only of using any current platform state as a reference for future boot cycles. There are no means of verifying that this reference state is trustworthy, opening the road to preemptive modification attacks.

TPM reset. TPM reset attacks imply that the TPM cannot reliably detect platform modifications if the attacker is in physical possession of the computer for sufficiently long time. This may be critical here since theft and other physical-access attacks are the key component of the adversary model.

Recovery mechanisms that circumvent the TPM altogether. Except for TPM reset and preemptive modification, all attacks described above do or may profit from the recovery mechanisms built into BitLocker. These mechanisms pose a particularly attractive target as they yield a key that is independent from the TPM and thus can be used more flexibly. The plausible recovery attack would not even be possible without recovery mechanisms.

Large amount of unprotected disk space. This is a secondary contributing factor to attacks involving purposeful, detectable modification of the platform (plausible recovery; spoofed prompt; tamper and revert). Large amounts of disk space are available for the attacker to install software or data in. This may be difficult to avoid, though.

Almost arbitrary sequence of partial attacks. Another, possibly application-specific, secondary factor is an effect of BitLocker's function and key management. In order to succeed, the attacker needs to accomplish several intermediate goals: copy ciphertext, and get access to various components of the encryption key. Due to the design of BitLocker, the respective attack operations can be executed in arbitrary order, unless one operation permanently changes the platform, the TPM or the knowledge of the victim in such a way that other operations become impossible.

The barn door property. This term has been coined by Whitten and Tygar [12], describing the fact that security often involves operations that are not easily reversed. There is often no *undo*. This is particularly true for confidentiality: once broken, it cannot be restored. There are many ways for the attacker to gain at least some partial success, but there are few situations where the attacker could lose anything achieved before. This may be different in applications with different security objectives and a different adversary model.

Table 1 shows how these causes and factors contribute to the attacks described before. Each column represents an attack, each line a cause or factor. If a factor contributes to an attack—makes it possible, makes it easier, or makes the result more useful for the attacker—the respective cell is marked with an X. The last two lines contain question marks in all cells: the authors do not fully understand the impact of these factors yet.

6 Conclusion

The caution exercised by Microsoft regarding claims about the security of BitLocker seems justified. While BitLocker may indeed protect against opportunistic theft of a computer that is turned off at the time, there are several plausible scenarios for targeted attacks. The trusted computing platform combined with the specific purpose, design and implementation of BitLocker fails to protect against these attacks. Although this does not necessarily imply grave deficiencies on either part, developers and users alike should be aware of these scenarios and the limitations of trusted computing.

Table 1: Attack scenarios and contributing factors.

	Replace and relay	Plausible Recovery	Spoofed prompt	Tamper and revert	Preemptive modification	TPM reset
Passive TPM			X	X		X
No trusted path to user	X		X	X		
No context awareness	X					
Lack of system authentication			X	X		
History-bounded platform validation				X		X
Incomplete diagnostic information		X				
Lack of external reference		X			X	
TPM reset	X		X	X		X
Recovery mechanisms circumventing TPM	X	X	X	X		
Unprotected disk space		X	X	X		
Arbitrary sequence of partial attacks	?	?	?	?	?	?
The barn door property	?	?	?	?	?	?

Our results yield various questions for further research. How easy or difficult is implementing these attacks in practice? Are there issues that we may have overlooked, or are some of the attacks even easier than they appear? Are there countermeasures that can be implemented easily? Where do countermeasures belong, into the application or the trusted computing technology? Is it possible to overcome the fundamental trade-off and implement secure recovery mechanisms? If so, how?

This paper represents a preliminary theoretical analysis. The authors intend to continue this work in three directions. First, the analysis needs to be refined. The final result shall contain an exact description of what an attacker can or cannot achieve in each scenario and what the side conditions are. It would be interesting to determine exact limits to attack optimization. Estimations of the effort required for and risk involved in each step of an attack will also be part of further analysis.

Second, we plan to implement the attacks described here. The purpose is not to create new hacker tools but to gain a deeper understanding of what works and what doesn't, and a better idea of parameters such as effort, time and side effects.

The final step is to devise specific countermeasures. They shall cover two distinct fields: improvements to the trusted computing technology, and recommendations for software implemented on top of it.

References

[1] Chris J. Mitchell (editor). Research workshop on future TPM functionality: Final report. http://www.softeng.ox.ac.uk/etiss/trusted/research/TPM.pdf.

[2] Niels Fergusson. AES-CBC + Elephant diffuser: A disk encryption algorithm for windows vista. Technical report, Microsoft, 2006.

[3] David Grawrock. *The Intel Safer Computing Initiative: Building Blocks for Trusted Computing*. Intel Press, 2006.

[4] Trusted Computing Group. TCG platform reset attack mitigation specification. https://www.trustedcomputinggroup.org/specs/PCClient/TCG_PlatformResetAttackMitigationSpecification_1.00_0340308-1.pdf.

[5] James Hendricks and Leendert van Doorn. Secure bootstrap is not enough: Shoring up the trusted computing base. In *Proceedings of the Eleventh SIGOPS European Workshop, ACM SIGOPS*. ACM Press, 2004.

[6] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. Technical report, Princeton University, 2008.

[7] J. D. Tygar and Bennet Yee. Dyad: A system for using physically secure coprocessors. Technical report, Proceedings of the Joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment, 1991.

[8] Michael Becher, Maximillian Dornseif, and Christian N. Klein. Firewire: all your memory are belong to us. Slides, http://md.hudora.de/presentations/#firewire-cansecwest.

[9] Saar Drimer and Steven J. Murdoch. Chip & PIN (EMV) relay attacks. http://www.cl.cam.ac.uk/research/security/banking/relay/.

[10] Saar Drimer and Steven J. Murdoch. Keep your enemies close: Distance bounding against smartcard relay attacks. In *USENIX Security 2007*, 2007.

[11] Evan R. Sparks. Security assessment of trusted platform modules. Technical report, Dartmouth College, 2007.

[12] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt. In *Proceedings of the 8th USENIX Security Symposium*, 1999.

[13] William A. Arbaugh, David J. Farbert, and Jonathan M. Smith. A secure and reliable bootstrap architecture. In *In Proceedings 1997 IEEE Symposium on Security and Privacy*, pages 65–71. IEEE Computer Society, 1997.