Achieving Attestation with Less Effort: An Indirect and Configurable Approach to Integrity Reporting

Jun Ho Huh
Information Trust Institute
University of Illinois at Urbana-Champaign
Urbana, IL, USA
jhhuh@illinois.edu

Hyoungshick Kim
Computer Laboratory
University of Cambridge
Cambridge, UK
hk331@cl.cam.ac.uk

John Lyle
Oxford University Computing Laboratory
Oxford, UK
john.lyle@comlab.ox.ac.uk

Andrew Martin
Oxford University Computing Laboratory
Oxford, UK
andrew.martin@comlab.ox.ac.uk

ABSTRACT

This paper proposes an *indirect attestation* paradigm for verifying the trustworthiness of end user platforms. This approach overcomes several criticisms of attestation by maintaining the user's freedom to choose their own software configurations and minimising the whitelist management overhead for the relying party. Each user platform defines its own acceptable software combination in terms of reference integrity measurements, and reports the local verification results to the relying party through a late-launched, trusted Platform Trust Service. The relying party simply checks this verification result and a *security meta-policy* that has been used to ensure the quality of the security checks performed locally. The Platform Trust Service is also responsible for reporting whether this meta-policy is satisfied. By configuring the meta-policy, the relying party selects an indirect attestation paradigm that best meets their high-level security requirements.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Security

Keywords

indirect attestation, whitelist management, security meta-policy

1. INTRODUCTION

Trusted Computing (TC) provides a means to reliably report the integrity of platform configurations to a remote party – this capability is referred to as 'remote attestation' by the Trusted Computing Group (TCG) [2]. Remote attestation has often been suggested as a mechanism suitable for authenticating user platforms: an obvious application is a web server that verifies the user's platform integrity and grants access to only those running with trustworthy configurations. Such an attestation system can be used together with protected key storage to build a strong two-factor authentication mechanism – the user's private signature key can be strongly protected through a Trusted Platform Module (TPM), which is normally a hardware chip embedded in the motherboard.

Despite the potential security benefits, there are some inherent practicality and usability issues associated with attestation which can be overlooked – these issues often impede further advances in attesting/verifying user platform configurations in systems that manage large number of users. For instance, one challenge of implementing attestation for an online system such as Internet banking is the management of application whitelists [8]. Pre-defining and validating which configurations/combinations are acceptable and keeping the whitelists up to date would be very costly tasks for the service providers. Agility is another potential problem: if the user configurations are patched, the service provider must accept those patched systems immediately, which may be hard to do in practice. With enforcement of these whitelists, there is also the danger of *locking* the user into a particular platform configuration – this would severely affect usability and privacy. Moreover, the process of maintaining and distributing Trusted Computing software (required for attestation) would have to be managed efficiently and be transparent to the end users. Perhaps it is these overwhelming challenges that discourage researchers and service providers from further exploring the idea of attesting user platform configurations. Without further investigation, attestation in large user-base systems may remain as a theoretical concept forever.

It is in this spirit that we propose a configurable approach for selecting an *indirect attestation* paradigm that is suitable for the security needs of the service provider. Our approach would give more freedom and privacy to the users in terms of choosing their own acceptable platform configurations. The costs associated with maintaining the whitelists and performing attestation are spread to the end users, software vendors, and security companies. We also evaluate security and usability of the proposed indirect attestation system, and discuss the implications of using it.

The following section explains the Trusted Computing concepts. Section 3 explains the challenges of making attestation work, and Section 4 proposes our indirect attestation system. We then explain to what extent the whitelist management and platform locking (software freedom) issues can be resolved in Section 5. Our conclusions are at Section 7.

2. TRUSTED COMPUTING CONCEPTS

Trusted Computing provides a means to *measure* the software loaded during the platform's boot process. Measurements are taken by calculating a cryptographic hash of binaries before they are executed. Hashes are stored in Platform Configuration Registers (PCRs) in the TPM. They can only be modified through special TPM ordinals, and the PCRs are never directly written to; rather, measurements can only be *extended* by an entity. This is to ensure that no other entity can just modify or overwrite the measure value. As a result, every executable piece of code in this *authenticated boot* process will be measured and PCRs extended sequentially (*transitive trust*). Any malicious piece of code (e.g. rootkit) executed during the boot process will also be recorded and identified. A 'PCR event log' is created during the boot process and stores all of the measured values (and a description for each) externally to the TPM. These values can be extended in software to validate the contents of the event log. The resulting hash can be compared against the reported, signed PCR values to see if the event log is correct.

2.1 Remote attestation

Remote attestation involves the TPM reporting PCR value(s) that are digitally signed with TPM-generated 'Attestation Identity Keys' (AIKs), and allowing others to validate the signature and the PCR contents [6]. The relying party validates the signature by using the public part of the AIK and validates the AIK with the AIK credential. The PCR log entries are then compared against a list of 'known-good' *reference values* to check if the reported PCRs represent an acceptable configuration. This list is often referred to as an 'application whitelist'. Attestation can be used on a platform that supports authenticated boot to verify that only known pieces of software are running on it.

2.2 Late launch

CPUs from both Intel and AMD now allow an arbitrary piece of code to be 'late-launched' [6] on an untrusted platform. Starting from a dynamic root of trust (which can be initiated anytime after platform boot), this code is executed in a protected environment and measured without any interference from the software currently running. On an Intel platform, PCRs 17-20 are reset and used to store the measurements of the secure initiliasation (SINIT) module and the trustworthy code to be late-launched. The SINIT module is responsible for measuring and launching the trustworthy code. This enables attestation of the SINIT module and trustworthy code. Attestation of the two components typically involves sending a TPM Quote of PCRs 17 and 18 for an Intel platform or PCR 17 for an AMD platform. A remote verifier checks that these PCRs were reset properly and the hashes of the SINIT module and trustworthy code. In Section 4.2 we discuss how a late-launched Platform Trust Service can be used to report local security checks reliably to the service provider.

3. WHY IS ATTESTATION DIFFICULT IN PRACTICE?

3.1 Whitelist management issues

The reality is that people have not been able to make remote attestation work properly: modern operating systems and software are too complex and change too much. Although Microsoft Windows is still the dominant operating system, other operating systems like Linux and Mac OS X are also widely used. Furthermore, there are hundreds and thousands of different proprietary security software available (e.g. anti-virus and firewall) and a service provider might want to verify that the user is running some combination of these. We argue that it is infeasible for the service provider (such as the bank) to know about all different acceptable user platform configurations (operating systems, versions, and security software) and maintain the whitelist efficiently.

To make matters worse, this service provider would have to additionally manage old versions of operating systems and applications, since not all users update their systems in a timely manner. On top of this, the service provider might also want to verify the integrity of important configuration files. It is not just about the size of whitelists, but also the huge effort that needs to go into discovering new acceptable configurations and combinations, and keeping them up to date.

> This is the first problem: no service provider would pay the cost of managing such activities and maintaining huge whitelists for enhancing security.

3.2 Platform locking issues

At first glance, mandating certain security configurations might seem like a practical solution to the whitelist management problems: the service provider would select a number of configurations that they consider trustworthy and force the users to use them to access their service. However, mandating the use of proprietary security software can impose severe usability and compatibility penalties, and must be treated with caution. There is the danger of *locking* people into a particular platform, depriving them of the option of using an online service via their choice of browser, operating system, and security software. A good example of this is the South Korean Internet banking system [7]: in order to use Internet banking, the users are obliged to install some number of proprietary security software such as anti-virus, firewall, and keystroke encryption software that are provided by the bank. These are typically installed on the user's machine in the form of ActiveX plugins. As a result, the users can only use Microsoft Internet Explorer to do Internet banking and have no option but to use Windows as their main operating system.

Likewise, mandating certain security configurations to facilitate meaningful attestation will have similar implications.

To most users, this concept of being obliged to use certain platform configurations, on their own PCs, is likely to be unacceptable. Due to poor usability and compatibility, some users may even move away to other banks that provide more compatible and open Internet banking services. Hence, no bank – unless required by some state law – would take this kind of risk just to improve security.

> This is the second problem: most users will not tolerate the inconvenience of being obliged to use a certain configuration, and the service providers would be reluctant to frustrate their users to the extent that they might decide to use another service.

Moreover, insisting on certain platform configurations means that attackers know exactly what to look for vulnerabilities in, potentially making them less secure than they would otherwise be. With these issues in mind, the next section proposes a compromise solution for attestation that allows the users to select their own acceptable security software/configuration to use.

4. INDIRECT ATTESTATION PARADIGM SELECTION

We propose an indirect attestation system which allows users to set their own *security policy* and attest their conformance to relying service providers. Service providers are also able to configure and check certain high-level *meta-policies* about the user's security policy, without finding out exactly what it contains. By configuring the meta-policies, service providers select an indirect attestation paradigm that best meets their security needs as well as their clients' usability expectations.

4.1 User security policies

User security policies take the form of *program execution rules* and govern the identity of software trusted to run on the user's platform. For example, one form of policy would be a whitelist of programs trusted by the user. Another policy would be to specify which *vendors* of software are trusted, and then allow the platform to run any program which has a Reference Integrity Measurement (RIM) signed by them. A combination of these would allow the user to create a policy approving only a subset of the software signed by a certain vendor.

More complicated policies are also possible. A whitelist could be specified only up to a certain stage in the boot process – for example, until after the operating system is loaded – and then any software may be used. Alternatively, all applications running with super-user privilege might need to be whitelisted after the kernel, but all normal software may be unrestricted. There are numerous possibilities which could be explored as future work.

Perhaps a more important issue is how and by whom the security policies are created. The system we propose allows for the user to have complete control over their security policy. However, some users may prefer delegating the management to a trusted authority, such as an anti-virus company or ISP. In this case we imagine the trusted authorities regularly providing a signed whitelist of trusted software to be used as the security policy. Another alternative might be the provision of security policy templates, which allow the user to select the specific programs they trust.

4.2 Basic Attestation System

An overview of the proposed attestation system is shown in Figure 1. The key principle behind this system is that the relying service provider does not check the vast majority of the user's platform integrity measurements or necessarily insist on a specific operating system and software. Instead, the service provider checks that a known 'Platform Trust Service' (PTS) [1] component has been *late-launched* (see Section 2.2). The PTS is given the responsibility for checking the platform's other PCR values against the user-selected RIMs contained in the security policy. The service provider checks that the PTS has been run, that it has reported a 'PCR values OK' type of response, and that it is using an acceptable security meta-policy (see Section 4.3 for more details). The following protocol shows how the user and service provider interact. The AIK certificate is labelled as $\{\texttt{cert}(AIK)\}_{PCA}$, $nonce_{sp}$ is a nonce generated by the service provider (SP), PCR_{17-20} is the attestation of the value of PCRs 17 to 20, $\texttt{PCR Log}_{17-20}$ is the integrity measurement log for PCRs 17-20, and `meta-policy` is the meta-policy description. The basic attestation protocol is taken from Sailer et al. [10] and we assume that all communication takes place using transport-layer security:

1. $User \longrightarrow SP$: Login Request , $\{\texttt{cert}(AIK)\}_{PCA}$
2. $SP \longrightarrow User$: $nonce_{sp}$
3. $User \longrightarrow SP$: $\{PCR_{17-20}, nonce_{sp}\}_{AIK}$, $\texttt{PCRLog}_{17-20}$, `meta-policy`

At this stage, the service provider does all the usual checks including verifying the signatures on the AIK certificate, nonce and PCR values. Then the integrity measurement log for PCRs 17-20 is checked to make sure that a trusted PTS was properly late-launched – this log is checked against a fairly static and small list of known-good measurements for the PTS and SINIT module (see Section 2.2) that are managed by the service provider. The integrity of the PTS indicates the correctness of the reported security check/result. To minimise the chances of being compromised at runtime, the PTS should be designed as a small and simple piece software that only manges the following functions. First, it will check the user platform's PCRs against the user's security policy. Then it will *extend* into the PCRs the hash of any assertions made in the meta-policy, such as 'policy updated in the last month' or 'policy signed by authority X'[1] and check that the security policy satisfies the meta policy. It will then extend the result of this check: either a 'PCR values OK' or a 'PCR invalid' string. When the service provider receives the verified PCR log, it needs to check that it just consists of a 'PCR values OK' result and an *acceptable* meta-policy. This check would ensure the integrity of the user platform configuration to the extent specified in the security policy as well as the conformance with the meta-policy. Following a successful platform authentication, the service provider can go on to request authentication of the user as per normal.

4.3 Platform Trust Service and security meta-policies

As mentioned in the previous section, the Platform Trust Service is late-launched in this scenario to check the platform's PCRs against a user-defined security policy. It can

[1]The precise vocabulary of the security policies and meta-policies is left as future work.

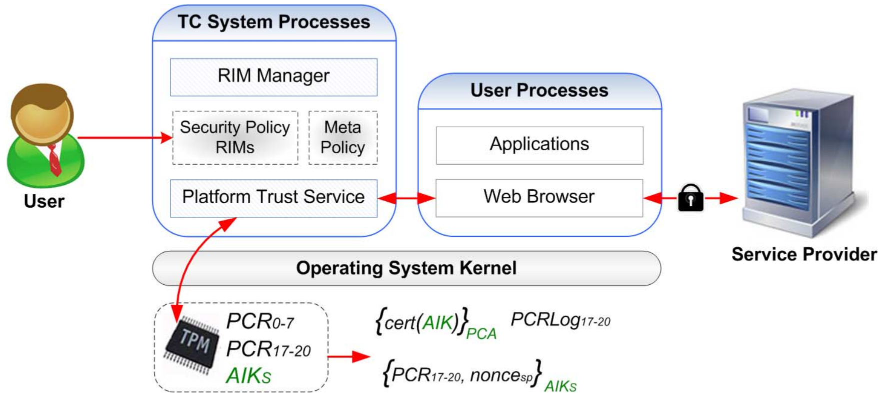

Figure 1: Indirect attestation overview. The late-launched Platform Trust Service is responsible for performing local configuration verification and reliably reporting the results to the service provider.

also check that the user's policy conforms with certain security meta-policies. These are designed to give the relying service provider more information about the user's platform without requiring them to use only software trusted by the service provider or compromising the user's privacy. We suggest the following meta-policies (listed in order of specificity) which provide useful security information.

P1. Non-trivial whitelisting

A whitelist is being checked by the PTS and this whitelist is non-trivially satisfied – i.e. the whitelist does not allow *all* software to be run and is of a reasonable size; this policy may also indicate that everything up to the OS kernel must be whitelisted.

P2. Whitelist signed by trusted authorities

A whitelist is being used by the platform and either this whitelist is signed by a trusted authority (the meta policy could state the authority) or each reference measurement in the whitelist is signed by a number of trusted authorities (again, stated in the meta policy). The service provider might be interested in knowing who the authority is, or whether it is one of the authorities on their list of trusted authorities.

P3. Recently updated security policy

The user has a security policy which has been updated after a given date. This could be implemented in a number of ways, but the easiest would be if the policy was signed by a trusted authority who also includes a timestamp. User-defined policies can also be timestamped (see 'policy provenance' below).

P4. Policy provenance

The meta-policy could also refer to the provenance of the user's security policy. This is required when the user does not use a whitelist signed by a trusted authority, but chooses his or her own selection of trusted software. How can the service provider know that the *user* created the whitelist, rather than a piece of malware running on the user's platform?

Clearly the solution is not to check the policy itself: the user must remain free to choose any security policy, even one that some might consider to be insecure. Instead, we propose that the policy is demonstrated to have been updated only through trusted software, the *RIM Manager* (see Figure 1). This can be late-launched like the Platform Trust Service and provide an interface for users to change the list of parties who are trusted to sign RIMs.

The RIM Manager would finish by creating a TPM Quote containing the platform's PCR values and the hash of the created policy. If a timestamp is required, this could also be included by contacting a trusted external time server and using the TPM's tick counter.

P5. Trusted kernel and pre-kernel boot sequence

The meta-policy contains the identity of all the software which is used in the boot sequence up to and including the kernel. Following this, a whitelist is used but the content of it remains confidential. This works if the service provider believes that the operating system and boot sequence contains the full TCB of the platform. As a further elaboration, the meta-policy could state that all applications run with super-user privilege are also on a disclosed whitelist.

P6. Enforcing certain type of applications

This meta-policy states the *type* of security-critical applications that needs to be installed and loaded during the initial boot. For instance, a service provider might require that an anti-virus software and firewall are loaded on the user platform, without specifying the exact software. The integrity measurement log can be used to capture the type of software that has been loaded/measured – this information can be used by the PTS to check whether all of the required type of software has been loaded properly.

P7. Enforcing certain security configurations

Certain security configurations may also be stated as part of the meta-policy. For instance, a service provider might want to check that the automatic update feature is enabled on the user's OS and anti-virus software. This would be useful for specifying a more fine-grained set of security requirements. To satisfy this policy, the PTS would have to be equipped with a mechanism to scrutinize certain parts of the configuration files (or registries in Windows).

We imagine that some combination of these policies would be used to build up and enforce different levels of security requirements. For instance, policies like P5 and P6 can be used together to ensure that a trusted kernel as well as anti-virus software and firewall are loaded on the user platform.

5. OBSERVATIONS

5.1 How the costs are spread

The usefulness of the proposed attestation system lies in the benefit each party gains from using it versus the associated costs. The user gains the ability to have a third party assess their system and report on whether their security policies are being implemented. This is possible, but difficult for users to do for themselves without a trusted external hardware token [5]. The costs to the user involve having a platform capable of late launch, and the overhead of managing a security policy. This may be negligible if the policy is provided by a trusted authority, but might be onerous otherwise. The service provider gains additional information about the user's platform state, and provides extra value (or a better service) for the user who may be willing to pay for the benefit. Moreover, the service provider can also check the security policy for conformance with meta-policies of varying levels of granularity. The cost to the service provider is minimal: they require knowledge of a small amount of trusted software, plus any trusted parties involved in the process. They may also need to provide support to the user if their platform fails the attestation verification.

5.2 What indirect attestation achieves

In Section 3 we identified whitelist management and platform locking as the two key problems of attestation. Based on the meta-policies suggested in Section 4.3, Table 1 evaluates to what extent these problems can be resolved using the proposed attestation system and what security properties can be ensured.

5.3 Security evaluation

As Table 1 shows, enforcing different level of meta-policies will have different security implications. A service provider might only enforce P1, indicating that everything up to the OS kernel level as well as all applications running with super-user privilege must be measured and verified. Conforming to this meta-policy, the attestation result will guarantee that the user's choice of platform configuration, up to the OS kernel level, is integrity protected. Moreover, it will show that all privileged applications are integrity protected. Such a result, however, does not give much indication as to how secure and up to date the platform configuration is. For instance, a user might decide to miss a few security-critical updates on the OS. As long as the integrity of this old version of OS is verified, the PTS will report a 'PCR values OK' response to the service provider. In consequence, the user is never locked into a particular platform configuration.

Meta-policies like P6 and P7 can be enforced in addition to P1 to provide more security assurance of the user platform. For instance, P6 may indicate that an anti-virus software and firewall must be loaded during boot time, and P7 might state that the automatic-update feature must be enabled on both the OS and anti-virus software. These extra policies would provide much more security assurance of the user platform than merely using P1, but may require the user to install new pieces of software and change settings. Alternatively, enforcing P2 would allow the service provider to rely on the trusted third parties to correctly verify the user's security policy and essential security properties that must be met.

Some service providers, that operate in a more controlled user environment, may feel the need to enforce P5, publishing a list of trusted OS measurements that the user may choose from. The user would end up installing an OS version that has been listed by the service provider, but would still be able to use their choice of applications from the OS level up. Combining P5 with policies like P6 and P7 can give chain of trust as far as to the browser, ignoring all other software running. Such a combined meta-policy would ensure a highly secure environment on the user platform, but only with some level of platform locking as discussed in Table 1.

5.4 Integration with mobile reference architecture

Our system and the mobile reference architecture [3] share some similarities in distributing the signed RIMs through trusted authorities. What is missing in the mobile architecture is a trusted reporting component, the late-launched PTS in our model, that would report the local secure boot results to relying parties in a trustworthy manner. Hence, we believe the ideas for late launching the PTS – perhaps after going through some generalisation – could be applied to the mobile architecture, enabling high assurance mobile services without placing any heavy burden on the service providers.

6. RELATED WORK

Sadeghi and Christian Stüble [9] have proposed Property-Based Attestation (PBA), which provides a level of indirection between integrity reporting and the evaluation of a platform's trustworthiness so that *properties* are attested rather than binary hashes. There are a range of implementation options, one of which is to have a trusted third party provide property certificates linking reference integrity measurements to properties [9]. PBA can be implemented in our system by having meta-policies signed by a trusted party which specify certain property to integrity measurement mappings, and then having the PTS locally check for this property. PBA, however, cannot support many of our meta-policies, including P1 and P4, and does not allow the user to specify their own security-policy (i.e. the whitelist). Furthermore, our approach integrates the PTS which can look at policies referring to fine-grained platform properties, as described in P6 and P7.

Local verification of trusted computing platforms has been proposed by Ali et al. [4]. Their local verification mechanism records system calls to identify with a high probability whether a runtime attack has occurred. However, this also requires users to agree in advance on their software configurations and so the problems that our approach attempts to solve are quite different. Seshadri et al. [11] have proposed SecVisor, which is a small hypervisor that is late-launched to protect the code integrity of kernels and ensure that only approved code can execute in kernel mode. The notion of relying on a late-launched component to perform a set of trusted operations locally runs commonly through SecVisor and our solution; but the set of operations that are managed by our PTS and SecVisor are, again, quite different and designed based on different security requirements.

Policy	Security properties	Whitelist management	Platform locking
P1	It may indicate that the user is only running software from a whitelist of a size below certain number.	The user manages one security policy that includes a software whitelist; the service provider does not manage any whitelist.	No platform locking issue.
P2	Ensures that the user's security policy that represents the user's platform configuration is verified and signed by one or more trusted authorities.	The user manages one software whitelist; the service provider does not manage any whitelist; each trusted authority manages some portion of the RIMs.	No platform locking issue.
P3	Ensures that the user's RIMs are up to date.	The user manages one software whitelist; the service provider does not manage any whitelist.	No platform locking issue.
P4	Ensures that the user-defined whitelist has not been tampered with.	The user manages one software whitelist; the service provider does not manage any whitelist.	No platform locking issue.
P5	Trusted kernel can be loaded.	The user manages one software whitelist excluding the OS kernel measurements; the service provider manages the trusted OS kernel measurements.	The user may be locked into certain OS trusted by the service provider.
P6	The use of certain types of security-critical applications (e.g. anti-virus software) can be ensured.	The user manages one software whitelist; the service provider does not manage any whitelist.	Certain types of applications can be mandated.
P7	Important security settings/configurations can be ensured, e.g. enabling automatic-update on the OS.	The user manages one software whitelist; the service provider does not manage any whitelist.	Certain software configurations can be mandated.

Table 1: Evaluation of whitelist management and platform locking problems

7. CONCLUSIONS

Attestation does not work well in large user-base systems due to the associated whitelist management costs and the danger of locking users into certain platform configurations. We proposed an indirect attestation system that spreads the whitelist management overheads – which would otherwise all fall on the service provider – among the users, software vendors and security companies (trusted third parties). Each user defines their own acceptable platform configuration in terms of the reference integrity measurements. Upon attestation, a late-launched Platform Trust Service checks whether the current platform configuration matches these reference values, and reports this local verification result to the service provider. Additionally, the Platform Trust Service checks that the user's local whitelist conforms with high-level meta-policies defined by the service provider. Configuring the meta-policies allows the service provider to select a suitable indirection attestation paradigm to be used. The service provider merely examines the reported verification result and conformance with its meta-policies in order to authenticate the user platform. Instead of obliging the user to configure their platform in one particular way and checking whether they have, our system verifies whether the integrity of the user's *choice of configuration* has been maintained.

The proposed attestation system explores one suitable compromise between the whitelist management overheads, usability and security. We hope that our work would provide some initiatives for further discussion and investigation of similar ideas for making attestation more manageable in large user-base systems.

8. REFERENCES

[1] TCG Infrastructure Working Group Platform Trust Services Interface Specification (IF-PTS). Specification version 1.0, November 2006.

[2] Trusted computing group backgrounder. `https://www.trustedcomputinggroup.org/about/`, October 2006.

[3] TCG Mobile Reference Architecture. Specification version 1.0, June 2007.

[4] T. Ali, M. Nauman, and X. Zhang. On leveraging stochastic models for remote attestation. In *INTRUST 2010: Proceedings of the 2nd International Conference on Trusted Systems*, 2010.

[5] E. Bangerter, M. Djackov, and A.-R. Sadeghi. A Demonstrative Ad Hoc Attestation System. In V. R. Tzong-Chen Wu, Chin-Laung Lei and D.-T. Lee, editors, *ISC '08: Proceedings of the 11th International Conference on Information Security*, volume 5222 of *Lecture Notes in Computer Science*, pages 17–30, Taipei, Taiwan, September 2008. Springer.

[6] D. Grawrock. *Dynamics of a Trusted Platform*. Intel Press, February 2009.

[7] H. Kim, J. H. Huh, and R. Anderson. On the Security of Internet Banking in South Korea. Technical Report RR-10-01, OUCL, March 2010.

[8] A. Lee-Thorp. Attestation in Trusted Computing: Challenges and Potential Solutions. Technical report, Royal Holloway,2010.

[9] A.-R. Sadeghi and C. Stüble. Property-based Attestation for Computing Platforms: Caring About Properties, Not Mechanisms. In *NSPW '04: Proceedings of the 2004 Workshop on New Security Paradigms*, pages 67–77, New York, NY, USA, 2004. ACM.

[10] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *USENIX Security Symposium*, volume 13, pages 223–238. USENIX Association, 2004.

[11] A. Seshadri, M. Luk, N. Qu, and A. Perrig. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 335–350, New York, NY, USA, 2007. ACM.