Peter Lipp
Ahmad-Reza Sadeghi
Klaus-Michael Koch (Eds.)

LNCS 4968

Trusted Computing – Challenges and Applications

First International Conference on
Trusted Computing and Trust in Information Technologies, TRUST 2008
Villach, Austria, March 2008, Proceedings

Springer

Peter Lipp Ahmad-Reza Sadeghi
Klaus-Michael Koch (Eds.)

Trusted Computing – Challenges and Applications

First International Conference on Trusted Computing
and Trust in Information Technologies, TRUST 2008
Villach, Austria, March 11-12, 2008
Proceedings

Springer

Volume Editors

Peter Lipp
Graz University of Technology
Institute for Applied Information Processing
and Communications
Inffeldgasse 16a, 8010 Graz, Austria
E-mail: peter.lipp@iaik.tugraz.at

Ahmad-Reza Sadeghi
Ruhr-University Bochum
Chair for System Security
Universitätsstr. 150
44780 Bochum, Germany
E-mail: ahmad.sadeghi@trust.rub.de

Klaus-Michael Koch
Technikon Forschungs- und
Planungsgesellschaft mbH
Burgplatz 3a
9500 Villach, Austria
E-mail: koch@technikon.com

Library of Congress Control Number: 2008932592

CR Subject Classification (1998): D.4.6, K.6.5, E.3, C.2, C.3, D.2, H.4, H.5, K.4

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-68978-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-68978-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12279555 06/3180 5 4 3 2 1 0

Preface

This volume contains papers presented at TRUST 2008, the first international conference on Trusted Computing and Trust in Information Technologies, held in March 2008 in Villach, Austria. The aim of the conference was to create a joint scientific and networking platform covering the core issues of trust in IT systems and trusted computing and to bridge the gaps between international research groups and projects in closely related fields.

The organizers received 43 submissions from 17 countries. Each of the submitted papers was reviewed by three reviewers. Based on these reviews 13 papers were selected as suitable for the conference and the authors were asked to present their work. Further, six renowned speakers from academia, industry and the European Commission were invited for keynotes. The accepted papers are published in this volume together with one paper from Paul England, one of the invited speakers at TRUST 2008.

The conference was supported by the European Commission via the Open-TC project (FP6 IST-027635), by the Austrian Research Promotion Agency (FFG) and by the city of Villach.

We are very grateful to all the members of the Program Committee for their effort during the review process, to our invited speakers for their valuable talks, to all supporters, and the local organizers for making this conference the successful start of a hopefully long series of TRUST conferences!

March 2008

Peter Lipp
Ahmad-Reza Sadeghi

Organization

Scientific Chairs

Ahmad-Reza Sadeghi	Ruhr-University Bochum
Peter Lipp	Graz University of Technology

Organizer

Taru Kankkunen	WiTEC Austria (European Association for Women in Science, Engineering and Technology)

Local Host

Klaus-Michael Koch	TECHNIKON Forschungs- und Planungsgesellschaft mbH

Program Committee

Asokan, N.	Nokia Research Center
Aura, Tuomas	Microsoft Research
Brandl, Hans	Infineon Technologies AG
Breu, Ruth	University of Innsbruck
England, Paul	Microsoft Research
Findeisen, Ralf	AMD Saxony LLC & Co. KG
Grawrock, David	Intel
Härtig, Hermann	Dresden University of Technology
Haselsteiner, Ernst	NXP Semiconductors
Herzog, Peter	ISECOM
Katzenbeisser, Stefan	Philips Research
Kiayias, Aggelos	University of Connecticut
Kruegel, Christopher	Vienna University of Technology
Kuhlmann, Dirk	HP Labs Bristol
Lioy, Antonio	Politecnico di Torino
Martin, Andrew	University of Oxford
Mitchell, Chris	Royal Holloway University of London
Mao, Wenbo	EMC Research China
Neumann, Heike	NXP Semiconductors
Paar, Christof	Ruhr-University Bochum
Persiano, Giuseppe	University of Salerno
Preneel, Bart	Katholieke Universiteit Leuven
Puccetti, Armand	Commissariat à l'Energie Atomique-LIST
Quisquater, Jean-Jacques	Universitè Catholique de Louvain

Rannenberg, Kai	Goethe University of Frankfurt
Rijmen, Vincent	Graz University of Technology
Schunter, Matthias	IBM Zurich
Seifert, Jean-Pierre	Samsung Research
Spitz, Stephan	Giesecke & Devrient
Strongin, Geoffrey	AMD
Stüble, Christian	Sirrix AG, security technologies
Thomborson, Clark	University of Auckland
van Doorn, Leendert	AMD
Varadharajan, Vijay	Macquarie University
Zhang, Xinwen	Samsung Research

Keynotes

David Grawrock	Intel
Paul England	Microsoft
Martin Sadler	HP
Ronald Perez	IBM
Bart Preneel	Katholieke Universiteit Leuven
Dirk van Rooy	European Commission

Table of Contents

Practical Techniques for Operating System Attestation

Paul England

Microsoft Corporation
Paul.England@microsoft.com

Abstract. This paper describes three practical techniques for authenticating the code and other execution state of an operating system using the services of the TPM and a hypervisor. The techniques trade off detailed reporting of the OS code and configuration with the manageability and comprehensibility of reported configurations. Such trade-offs are essential because of the complexity and diversity of modern general purpose operating systems makes simple code authentication schemes using code hashes or certificates infeasible.

Keywords: trusted computing,attestation,security distributed systems, security models.

1 Introduction

Trustworthy computing and the technologies embodied in the Trusted Platform Module (TPM) provide services that allow system-software to authenticate itself to remote parties (commonly described as attestation), and to protect itself against offline attack (sealing) [1] [19]. The TPM cannot observe or measure software running on the platform, so compliant platforms are required to report accurate measurements of security-relevant software or state to the TPM by means of a logging operation called *Extend*. The TPM does not impose any restrictions on how external software and state is assessed as long as the policy can be encoded into 20 bytes and provided to the TPM in a call to *Extend*, however most use of the TPM to date implements a version of secure-boot in which simple external measurement algorithms like the hash of code modules are logged.

The measurements reported to the TPM form the basis for local access control and attestation. However, for the measurements to be useful, the remote entity or the local-sealer must be able to predict and/or interpret the security state based on the associated log. This seems feasible (although still hard) for authenticating platform firmware, but we believe that there are three serious problems that make it impractical to authenticate a general purpose operating system in this way without significant restrictions or simplifications.

First we have the problem of code-diversity. To illustrate the scale of this problem, a typical Windows installation loads two hundred or more drivers from a known set of more than 4 million (with more than 4000 new drivers reported each day). [2] At the time of writing Linux has somewhat less driver coverage (and hence diversity) but the trajectory is similar.

Second we have the issue of operating system state such as the data files that are used to configure the kernel, drivers, or system services. Such state is not usually hashed (or otherwise integrity protected) when the OS is not running, but can have a profound effect on the OS security policy. Examples of security-relevant configuration state include the administrator password or password hash, whether a kernel debugger is enabled, and the access control policy for the system (e.g. the ACL that will be set on the page-file). However, as is the case with executable modules, it is not so much the existence of these settings that is problematic, but more the number individual settings, and the large range of acceptable values. To illustrate the scale of this problem, measurements indicate that Windows performs 20,000 to 50,000 registry reads (depending on the driver set) from the start of boot to the point where the logon screen is displayed. Determining which of these values has security relevance and should be used as the basis of the OS identity for attestation or local access control is daunting.

Finally, even if the relying party can interpret the reported configuration, the information is no use if the system is subject to unexpected change. Security-relevant state transitions can occur if an attacker exploits a bug in the trusted computing base, but can also occur through proper authorized actions of the legitimate OS administrator. Both such events can make configuration information recorded at boot time worthless when later assessing the platform security state.

These practical problems are serious obstacles to the application of trustworthy computing in anything other than very specialized settings. If trusted computing is to be more broadly applicable it is clear that we need grouping and simplification mechanisms that allow us to trade off knowledge of the detailed individual machine configuration with ease of comprehension and management, and we need operating systems (or OS configurations) that are sufficiently secure to make attestation worthwhile. This paper reviews and analyzes some software authentication techniques that have been described in the literature, and introduces others that address complementary scenarios or appear to address some shortcomings of published techniques.

The paper is organized as follows. In section 2 we briefly review the mechanisms for software authentication and authorization provided by the TPM. In section 3 we describe the challenges that must be overcome for OS-identity-based access control systems to be feasible and describe previous work that has addressed the problems that we identify. In section 4 we describe three techniques for OS authentication that range from very precise but inflexible configuration reporting through to flexible but less detailed information of the running OS or its environment. Finally, in section 5 we summarize and techniques we have described and compare and contrast their applicability to various security scenarios.

2 Software Authentication Using the TPM

In this section we briefly review how the TPM supports software authentication. Additional details are available in the specifications [1], and review books and articles [19] [18], [9].

The TPM provides a simple but flexible mechanism for representing the software running on a platform. Software identity (or other security relevant state) is represented in a set of registers called *Platform Configuration Registers*, or PCRs. PCRs are hash-sized registers that are updated using a cumulative hash function called *Extend*. Semantically, *Extend* and the underlying PCRs are a compact representation of a secure log. The initial state of PCR registers is architecturally defined (most start at zero). Most PCRs are only reset when the containing platform is itself reset to an architecturally defined starting state, but (if so configured) some PCRs can be reset by privileged programs running on the platform.

Once a configuration has been established through secure-booting and a sequence of calls to *Extend*, the TPM can be used for local access control decisions (*Unseal*), and can provide cryptographic building blocks that can be used as the basis of software authentication in distributed access control systems (*Quote*, *Unbind*, *ActivateIdentity*, etc.)

Of particular importance to this paper is the fact that the TPM attaches no semantic meaning to the values passed into the *Extend* operation. Early in boot TCG specifications dictate that modules (parts of the BIOS, option-ROMS, OS-loaders) should be hashed and the resulting hash values should be recorded. However, later in boot software is free to use more semantically rich policies and module descriptions as long as the policies can be encoded in one or a sequence of calls to the *Extend* function. This flexibility is the basis of the new and existing techniques for managing state diversity that are described in this paper.

3 Challenges in Authenticating an Operating System

Practical systems for authenticating system software (operating systems or hypervisors) must address the following issues:[1]

Code Diversity. A means of authenticating the kernel, drivers, libraries and other system code that is manageable in the face of great diversity and churn (in supported devices, software versions and publishers)

State. Identification, recording and manageable representation of all security-relevant state that is used during boot and execution of the operating system

Robustness. The OS configurations reported must have meaningful security properties and robustness for attestation to be worthwhile. Solutions must also

[1] In this paper we only consider OS-related problems. Several papers in the bibliography discuss other important issues, including hardware robustness, TPM-identification and certification and firmware diversity/attacks. [13].

be robust to (or properly represent) legitimate configuration changes made by the operating system administrator.

We believe that it is not practical to satisfy all of these requirements for a general-purpose operating system in the general case, so we - like others - look for simplifying assumptions and special cases. In the remainder of this section we review some of the solutions that have been described in the literature. In section 4 we describe some new schemes that build on these techniques and appear practical for complex legacy operating systems.

3.1 Grouping Components with Public Key Cryptography

Public key signing is a convenient technique for grouping programs. A publisher (or other authority) signs program hashes, and the public signing key is used to authenticate the programs in a group. Then, instead of recording the hash of each module loaded, a policy evaluator checks that modules are properly signed with an authorized key before the module is loaded [3] [15]. Such polices can be integrated into TPM-based security by extending a PCR-register with the hash of the policy used (in the simplest case, the hash of the public part of the signing key).

Code signing reduces the size of the set of acceptable PCR registers and is a convenient way of distributing descriptions of "dynamic" configurations, but unfortunately does little for the other problems of code diversity. For instance, authorities still need to evaluate and sign all components, and public-key based systems need revocation infrastructures if they are not to be very fragile.

That said we believe that code-signatures will be an important technique for simplifying the deployment and management of the code and data files used in trusted computing (and indeed in other security settings).

3.2 Isolation Micro-kernels and Hypervisors

One way of sidestepping the issue of authenticating a general-purpose operating system is to introduce an authenticated isolation microkernel or hypervisor [6] [5] [8]. Hypervisors are generally much simpler than operating systems, and so the state problem and (to some extent) the driver diversity problems are reduced. Hypervisors are also often designed to high assurance standards which should help with the robustness problem. Things become better still if the late-launch "dynamic root of trust for measurement" (DRTM) facilities embodied in modern processors are utilized [13]. In this case a simple microkernel hypervisor (managing the CPU, memory subsystems and controllers, and the TPM) can be reliably authenticated and initialized. Such a microkernel need contain very few drivers, and its inherent simplicity argues for higher robustness.

The TPM also allows a certain range of peer-guest device services to be used without taking a full security dependency on that guest. For example, given TPM (or hypervisor) provided key storage, a guest may choose to encrypt all network data and disk IO before exporting it to the un-trusted device-service guest.

Isolation micro-kernels and hypervisors clearly offer the hope of a smaller “attestable” TCB, but do so at the expense of an extremely impoverished programming environment for application development. Such simple environments are probably adequate for certain scenarios (transaction authorization, cryptographic service providers, or the attestation services described in the next section) but do not address the problem of meaningful authentication of a mainstream OS running a standard application. In the remainder of this section we describe approaches that have been suggested for attestation of such systems.

3.3 Property Based Attestation

Property-based attestation introduces a trusted attestation service (TAS) which is a privileged program that can inspect the state, execution environment, and IO of a guest OS or application [20] [14] [4]. The TAS can then provide reliable reporting of facts or “properties” of that execution environment. An important duty of the TAS is to provide higher-level abstractions for machine state than that supported by the TPM itself. It is argued (convincingly) that higher-level security properties will be more useful than detailed descriptions of all of the executable modules that comprise a complex software system.

The TAS needs an execution environment that allows it to observe the execution of programs that it is to report upon, but also needs to be protected from observation and modification by un-trusted external entities. The TAS also needs access to the TPM (or other means for storing keys and authenticating itself). An isolation microkernel or VMM such as that described in the previous section appears ideal for this purpose.

Property-based attestation provides a framework for defining “properties” based on any sort of security policy, attribute, or invariant. However, as we have argued in section 1, it is extremely difficult to reliably assess the security properties of running or quiescent operating system. In section 4 we describe some specific properties that we believe convey useful security information in a mainstream operating system.

3.4 Semantic Attestation

Semantic attestation [10] is another technique designed to raise the level of abstraction of the security properties reported by an attestation service. The original paper described a system that allowed high-level security properties of a Java VM to be attested, although the ideas are more broadly applicable.

In the setting of a general-purpose operating system the VM-configuration reporting described in section 4 seem to provide worthwhile and tractable “semantic” information. Additionally, it is possible to build a service that inspects a peer or dependent guest in order to generate reports that an OS is “virus and root kit free" according to a dated virus definition file. Such reporting is much more useful than a report from a root-kit detector running in the same execution domain as the threats it is reporting since a smart zero-day root-kit running alongside a root-kit detector will interfere with future reliable reporting. However, the nature of the reporting is based on heuristics and known-threats, so is a far less precise and detailed description than one based (say) on program hashes or certificates.

3.5 Read-Only Operating System Images

The state and code-diversity problems described in earlier sections can be mitigated by starting an operating system (and perhaps bundled applications) from an integrity-protected read-only image (e.g. a real or virtual DVD) distributed by a reputable publisher. Of course we expect that trusted OS images will be updated periodically to patch bugs and add functionality, but this centralized publishing system seems far more manageable than schemes based on validating individual drivers, OS-state files and applications. This is the approach used in the creation of the "CA in a Box" reported by Franklin et al. [7].

There are many ways that the identity of the OS image can be defined and measured. Perhaps the most practical is to build a Merkle hash tree over the image sectors. The identity of the OS is then derived from the root hash, and on the hash of all sectors read must be checked against the corresponding entries in the hash tree. [17] [11]

A single OS distribution image is problematic for an OS targeting physical hardware (although many OS distributions contain drivers for a wide variety of target machines), but is far more realistic if the operating system is to run in secure virtual machine. This is because virtual machine monitors provide standardized virtual devices for a wide range of physical device hardware. Thus, the read-only virtual DVD OS authentication mechanism seems well suited when built on the secure virtual machine platforms described in earlier sections.

This technique mitigates the code and state diversity problems by encoding all execution data on the rarely-changed distribution medium (the "virtual DVD"). Of course the diversity-reduction benefits are lost if the running OS is allowed to write to the OS image in such a way that the system is different on the next boot, and this implies that the OS can maintain no persistent state. We can weaken the requirement of no persistent state by providing another (virtual or physical) storage device whose contents does not contribute to the OS identity. However, developers must remain vigilant that no security-sensitive data is stored on this device. Cryptographic techniques can be used to protect the contents of the supplemental storage device when the OS is running or offline.

Booting from a read-only image provides useful remediation of the code and data state diversity problems noted in section 1, but does not help in the creation and maintenance of an OS that is robust against wanted or unwanted security configuration changes, and the limitations on persistent data storage are burdensome. Some new solutions to these problems are covered in the next sections.

4 Practical Techniques for OS Authentication

In the previous section we described the challenges faced in authenticating a general purpose operating system, and some of the techniques that have been proposed to overcome these problems. In this section we describe some new techniques that can be applied to general-purpose operating systems.

4.1 Specialized OS Images

The problem with the read-only image distribution scheme described in the previous section is that most OS instances need to be modified before use, but the simple scheme explicitly prohibits any persistent modification of the OS image. In this section we explore allowing specializations of a standard OS.

Operating systems are specialized-for-purpose by the administrator and to a lesser extent by the users. Some important security-relevant specializations are:

1. The accounts with administrative privileges and their associated passwords, public keys, or Kerberos domain controllers
2. The network domain that the machine is to join (or provisioning of other network authentication credentials)
3. The operating system security policy

These specializations can be applied programmatically or with tools after the OS is installed, or can be applied during installation based on a declarative policy. [2] The need for specialization on the client is lessened if the image publisher performs most or all specializations before distribution. In this case the publisher software publisher (or other third party such an enterprise administrator) prepares an OS image with security policy and installed applications appropriate for their deployment and clients can run these images without further modification.

Another option is to specialize a generic OS image *on the client* on each boot by having a startup program read a data file that describes the OS policy and then apply it to the running OS before it is available for other use. See figure 1.

Of course the OS policy file defines important security characteristics, so should become part of the OS identity (for instance by having the booting generic OS "extend" the hash of the policy file into a PCR register).

This technique for specialization achieves the same ends as applying the specialization before distribution, but it is a more accurate representation of the separation of concerns and responsibilities that we see in real life. For example, most corporations use OS distributions prepared by others, but apply their own security policies. In addition, most home users will want to use a single base OS

[2] For example, the Microsoft Windows family of operating systems allows for the presentation of an "answer file" that the OS uses during installation to automate enterprise installations.

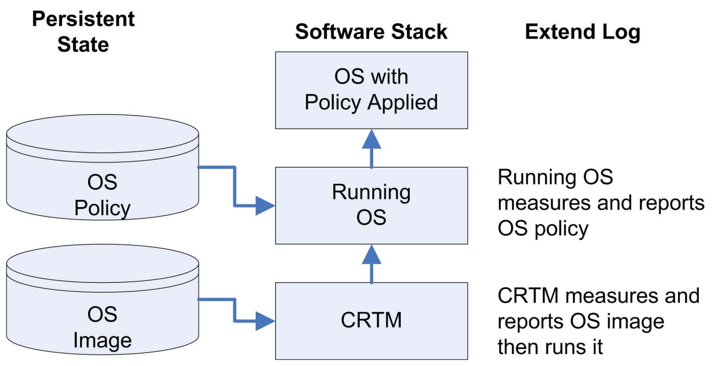


Fig. 1. Illustration of the boot and subsequent specialization of a generic OS image, and how this specialization is reported to the TPM

image and browser, but would like to apply specific specialization appropriate for home-finance, peer-peer, and other activities.

Some policy specializations may contain confidential information. For example, the administrator password or a cryptographic credential that is needed to authenticate the machine to the network. Such data can be kept secret by encrypted distribution of the policy specialization files. The TPM provides convenient key storage and cryptographic primitives that can be used for this purpose (for example a key exchange protocol built on *Quote*).

OS Specialization as a Means of Improving Security. The technique of applying a security policy to the OS before it is available for use can also be used to provide significant mitigation for security vulnerabilities in the operating systems that are distributed. Generally any security or configuration policy that reduces the number or complexity of exposed interfaces makes it harder for an adversity to subvert the trusted computing base by exploiting a bug. Some examples of configuration strategies that should provide significant security benefits are:

1. Limit network connectivity by means of firewall rules, or (better) limiting connectivity to cryptographically authenticated endpoints (SSL or VPN)
2. Limit the programs that the OS is permitted to run
3. Disable administrative access [3]

We believe that a strict security policy can result in an OS identity that is meaningful for a general purpose operating system like Windows or Linux. But a more stringent security policy tends to imply a less flexible and usable machine. Such drawbacks can be mitigated if a virtual machine monitor supports many operating systems with different security specializations.

[3] This means that all administration must be performed at the time of image preparation or specialization, as we have described above.

Authentication of OS Images with Persistent State. The techniques described in the previous sections vastly reduce the diversity in the identity of the booting OS images by denying any permanent modification of the OS image. Next we describe a technique for allowing the persistent state of an OS to evolve without introducing unmanageable state diversity in the configurations reported.

The technique is based upon the observation that shutting down and rebooting the machine should not be a significant security policy event as long as the OS image is protected when the OS is not running. If the OS image *is* protected when offline then it is the OS administrator that controls security-relevant state changes, and a reboot should merely return this system to the same security policy state that the system was in before it was shut down.

This suggests that authenticating the system when it was last booted is less important than authenticating the system when it was initially installed. Of course the TPM "quoting" facility only reports the state of the system on last boot, so we need to maintain some other chain of trust for this to be possible. The TPM provides a rich set of cryptographic primitives, so it should be no surprise that a variety of solutions are possible. An example is described below.

4.2 OS Authentication by Means of a Birth Certificate

In this section we sketch a procedure for securely provisioning a freshly installed OS image with a credential that it can use to authenticate its initial state at a later point in its execution (including after a reboot).[4] We also describe how the OS can use the TPM to protect this credential between reboots. This technique can be used in a virtualized or hardware setting, but for definiteness we describe a VM implementation.

Step 1 - OS Installation and Specialization. A VM is provisioned with an integrity-protected virtual DVD containing the OS to be installed and an integrity protected specialization file. The hashes of these files are recorded in a PCR (or virtual PCR) by the hypervisor. [16] The VM is also provisioned with an empty cryptographically-protected virtual hard disk that it uses as the target of the OS installation (protection of the keys for the crypto-disk is described below). On startup the VM is configured to boot from the installation program on the virtual DVD and the installation program installs the OS on the crypto-disk as normal.

Step 2 - Provision of the Birth Certificate. Before rebooting, the freshly installed OS creates a random software key pair and uses the TPM (or virtual TPM) "*Quote*" facility to mint a certificate for the public key *and* the PCR configuration when the software was first installed. The most important PCRs to report in this step are those that contain the OS installation image and the

[4] We omit detailed descriptions of the cryptographic protocols and necessary behavior of the OS and VMM for brevity.

initial specialization. Since this credential and key is to describe the *initial* OS configuration (rather than the configuration at last boot), we call it a "Birth Certificate." (Use of the birth certificate is described in step 4.)

Step 3 - Protecting the OS Image and Birth Certificate. If the key and birth certificate are to be a reliable indicator of the birth configuration of the operating system, the key must be protected so that it is only accessible to the OS that was originally certified. Protection when running is the responsibility of the VM and the running operating system (i.e. the OS must be configured to be secure). When the OS is not running the birth-certificate private key (and the OS image) must also be protected. All OS data is stored on the crypto-disk, so protecting the disk contents devolves to properly access-protecting the crypto-disk key or keys. We can do this by "sealing" the crypto-disk keys to the hypervisor. Once this step is complete the OS can be shut down at any time.

The authorized hypervisor (or hypervisors) is able to *Unseal* the disk key but must only mount the virtual drive in safe environment. One such safe environment is a freshly started virtual machine.[5]

This procedure does not protect the OS image against rollback. This may be problematic if the administrator installs an important security patch, and an adversary rolls the OS back to the state before the patch was installed and subverts the OS image. This vulnerability can be mitigated by using the TPM hardware monotonic counter.

Step 4: Authenticating the Operating System. The OS (or applications running on the OS) can use the key and associated birth certificate as part of a cryptographic protocol to authenticate itself at any time.

It will often be convenient to use the OS authentication protocol once to get another key that the OS can use to authenticate itself. For example, the birth certificate can be used to prove policy compliance to a service that issues Kerberos machine keys. Once a kerberos machine key has been issued, legacy protocols can be used to prove policy compliance by induction.

The birth-certificate technique provides reporting of the initial OS configuration and the initial security policy (probably including the identity of the initial administrator). After this, the OS (and its administrator) is in control of the OS security state and policy. The birth certificate form of attestation is weaker than boot-time authentication in the sense that the hardware attested configuration is typically further in the past. However, with server and desktop uptimes of months (perhaps with regular hibernations to save power) the differences may not be too great.

The technique also provides less specificity of the attested configuration. The OS may evolve to *any* configuration (or indeed hardware device) approved by the OS administrator. Such reporting will not serve all use cases, but appears to

[5] We also require the OS to be architecturally similar to that at shutdown for the OS to boot properly.

provide an important security improvement for today's managed general purpose OS deployments.

If more up-to-date or detailed knowledge of OS state is required we believe that supplemental property-attestation based on passing a root-kit or virus-check will be most practical.

We believe that the technique of birth credentials is most appropriate for enterprise deployment of trusted computing. For example, using this technique an enterprise IT department could securely and remotely provision a corporate OS image and be assured that they remain in sole administrative control even when the machine is outside their physical control. [6]

4.3 Virtual Machine Policy Attestation

Our final attestation technique *authenticates the characteristics of the virtual machine* that the OS is running in, rather than the OS itself.

An OS running in an environment in which it is subject to a mandatory access control policy is constrained in its actions. Sometimes knowledge of these constraints is sufficient to decide whether to trust the software, *regardless of the OS or applications running in the virtual machine* [12]. This suggests that a useful alternative to attesting the code running in a VM is to attest the security-relevant characteristics of the VM that the OS is running within.

For example, to support media-DRM we might attest a VM environment that does not permit data to be exported apart from to an authorized display device. In this case it does not matter what OS and media player runs within the VM because direct digital content leakage is prohibited by the virtual machine environment. [7] An enterprise may use a similar policy, except they may limit the network services that the work environment can connect to. A third example provides an alternative to human-interactive proofs to guard web sites against automata: In this case the VM might provide evidence that input has been received from an IO device and not a program or automaton.

5 Conclusions

We have described three new practical schemes for meaningful attestation of a general-purpose operating system running existing applications. Each scheme has advantages and disadvantages as summarized in table 1.

We believe all supported scenarios are important and interesting, and note that each of the schemes that we have described has different strengths and weaknesses. Fortunately virtualization technology will allow us to run many such systems simultaneously.

[6] This is subject to the belief that the TPM's EK is embedded in a compliant machine, and subject to the limitations of the hardware against physical tampering.

[7] Note that the VMM must generally provide a secure channel to the guest so that an adversarial guest cannot leak a content encryption key.

Table 1. Summary of the characteristics of the three new OS authentication schemes that we have described, and scenarios for which they seem well suited (the read-only OS image scheme is included for comparison)

Attestation Scheme	Advantages	Disadvantages	Key Scenarios
Read-Only OS Image	Precise knowledge of all software and state. Low image diversity (in VM case)	Preclusion of persistent state limits usefulness of the OS. Specialized OS must be prepared and distributed.	Platform for semantic attestation services. Transaction authorization and cryptographic services
Specialized Read-Only OS Image	Precise knowledge of OS state	Deployment and specialization easier than pure RO-images. OS must be configured to be reasonably secure	As above, but OS image deployment is easier (base images can be used for many purposes)
OS with Birth Certificate	Persistent state modification is allowed	Attestation information is not fresh	Enterprise desktop deployments
VMM Security Policy	No restriction on the software that runs in the VM	Fewer policies can be described at the VM boundary than within the OS	Document and media DRM, High-assurance platforms

References

1. Specifications are available on the TCG web site, `http://www.trustedcomputinggroup.org`
2. Microft Online Crash Analysis data
3. Arbaugh, W., Farber, D., Smith, J.: A secure and reliable bootstrap architecture (1997)
4. Chen, L., Landfermann, R., Lohr, M., Rohe, A.S., Stuble, C.: A protocol for property-based attestation. In: STC 2006: Proceedings of the first ACM workshop on Scalable trusted computing, pp. 7–16. ACM, New York (2006)
5. England, P., Lampson, B., Manferdelli, J., Peinado, M., Willman, B.: A trusted open platform. Computer 36(7), 55–62 (2003)
6. England, P., Peinado, M.: Authenticated operation of open computing devices. In: Batten, L.M., Seberry, J. (eds.) ACISP 2002. LNCS, vol. 2384, pp. 346–361. Springer, Heidelberg (2002)
7. Franklin, M., Mitcham, K., Smith, S.W., Stabiner, J., Wild, O.: Ca-in-a-box. In: EuroPKI: Lecture notes in computer science, pp. 180–190 (2005)
8. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: a virtual machine-based platform for trusted computing. In: SOSP 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 193–206. ACM, New York (2003)
9. Grawrock, D.: The Intel Safer Computing Initiative. Intel Press (2006)
10. Haldar, V., Chandra, D., Franz, M.: Semantic remote attestation: a virtual machine directed approach to trusted computing. In: VM 2004: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium, Berkeley, CA, USA, p. 3. USENIX Association (2004)

11. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: Scalable secure file sharing on untrusted storage. In: FAST 2003: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, pp. 29–42. USENIX Association (2003)
12. Karger, P.A., Zurko, M.E., Bonin, D.W., Mason, A.H., Kahn, C.E.: A retrospective on the vax vmm security kernel. IEEE Trans. Softw. Eng. 17(11), 1147–1165 (1991)
13. Kauer, B.: Oslo: Improving the security of trusted computing. In: Proceedings of the 16th USENIX Security Symposium (2007)
14. Kühn, U., Selhorst, M., Stüble, C.: Realizing property-based attestation and sealing with commonly available hard- and software. In: STC 2007: Proceedings of the 2007 ACM workshop on Scalable trusted computing, pp. 50–57. ACM, New York (2007)
15. Lampson, B., Abadi, M., Burrows, M., Wobber, E.: Authentication in distributed systems: theory and practice. ACM Trans. Comput. Syst. 10(4), 265–310 (1992)
16. Loeser, J., England, P.: Para-virtualized tpm sharing. In: Proceedings of TRUST2008 (these proceedings), London, UK, Springer, Heidelberg (2008)
17. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988)
18. Mitchell, C.: Trusted Computing (Professional Applications of Computing) (Professional Applications of Computing). IEE (2005)
19. Pearson, S.: Trusted Computing Platforms: TCPA Technology in Context (HP Professional Series). Prentice Hall, Englewood Cliffs (2002)
20. Sadeghi, A.-R., Stüble, C.: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: NSPW 2004: Proceedings of the 2004 workshop on New security paradigms, pp. 67–77. ACM, New York (2004)

TOCTOU, Traps, and Trusted Computing[⋆]

Sergey Bratus, Nihal D'Cunha, Evan Sparks, and Sean W. Smith

Dartmouth College, Hanover, New Hampshire

Abstract. The security of the standard TCG architecture depends on whether the values in the PCRs match the actual platform configuration. However, this design admits potential for *time-of-check time-of-use* vulnerabilities: a PCR reflects the state of code and data when it was measured, not when the TPM uses a credential or signs an attestation based on that measurement. We demonstrate how an attacker with sufficient privileges can compromise the integrity of a TPM-protected system by modifying critical loaded code and static data after measurement has taken place. To solve this problem, we explore using the MMU and the TPM in concert to provide a *memory event trapping framework*, in which trap handlers perform TPM operations to enforce a security policy. Our framework proposal includes modifying the MMU to support selective memory immutability and generate higher granularity memory access traps. To substantiate our ideas, we designed and implemented a software prototype system employing the monitoring capabilities of the Xen virtual machine monitor.

1 Introduction

The *Trusted Computing Group* (TCG) [1] works toward developing and advancing open standards for trusted computing across platforms of multiple types. Their main goals are to increase the trust level of a system by allowing it to be remotely verifiable and to aid users in protecting their sensitive information, such as passwords and keys, from compromise. The core component of the proposal is the *Trusted Platform Module* (TPM), commonly a chip mounted on the motherboard of a computer. A TPM provides internal storage space for storing cryptographic keys and other security critical information. It provides cryptographic functions for encryption/decryption, signing/verifying as well as hardware-based random number generation. TPM functionalities can be used to attest to the configuration of the underlying computing platform, as well as to seal and bind data to a specific platform configuration. In the last few years, major vendors of computer systems have been shipping machines that have included TPMs, with associated BIOS support.

⋆ This work was supported in part by the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001, and the Institute for Security Technology Studies, under Grant number 2005-DD-BX-1091 awarded by the Bureau of Justice Assistance. The views and conclusions do not necessarily represent those of the sponsors.

The key to TCG-based security is: *A TPM is used to provide a range of hardware-based security features to programs that know how to use them.* TPMs provide a hardware-based *root of trust* that can be extended to include associated software in a chain of trust. Each link in this chain of trust extends its trust to the subsequent one. It should be noted that the semantics of this extension for each link of the chain are determined by the programmers (including the BIOS programmer). More specifically, the programmer defines the conditions applying to the system's state S_{i+1} that are checkable in the state S_i under which the transition $S_i \rightarrow S_{i+1}$ is deemed to preserve trust. These conditions strongly rely on our understanding of the relationship between the software active in S_i and S_{i+1}. For example, a developer may trust a process in S_{i+1} that is created from an ELF file after verifying in S_i that either the entire file or some of its sections such as code and data hash to a known good value. Implicit in this decision is the assumption that the hash measurement is enough to guarantee the trustworthy behavior of the process.

In this work, we explore an additional set of TPM-based security architecture features that programmers can take advantage of to secure data that they perceive as sensitive and enforce a new class of policies to ensure their software's trustworthiness.

In particular, we note that the current TCG architecture only provides load-time guarantees. Integrity measurements are taken just before the software is loaded into memory, and it is assumed that the loaded in-memory software remains unchanged. However, this is not necessarily true—an adversary can exploit the difference between when software is measured and when it is actually used, to induce run-time vulnerabilities. This is an instance of the *time-of-check time-of-use* (TOCTOU) class of attacks. In its current implementation, the TPM holds only static measurements and so these malicious changes will not be reflected in its state. Code or data that is correct at the time of hashing may be modified by the time of its use in a number of ways, e.g., by malicious input. Change-after-hashing is a considerable threat to securing elements in the TCG architecture.

This paper explores this TOCTOU problem. Section 2 places it in the context of the TCG architecture, Section 3 demonstrates this vulnerability. Section 4 explores a solution space: making the TPM aware when memory that has been measured at load-time is being changed in malicious ways at run-time. Section 4 then speculates on a hardware-based solution, but also presents a software proof-of-concept demonstration using Xen's memory access trapping capabilities. Section 5 evaluates the security and performance of our solution. Section 6 explores related work. Section 7 concludes with some avenues for future work.

2 TOCTOU Issues in the TCG Architecture Perspective

Generally, the focus of current trusted platform development efforts has been on mechanisms for establishing the chain of trust, platform state measurement, protection of secrets and data, and remote attestation of platform state. Clearly, without cheap and ubiquitous implementations of these basic mechanisms, commodity trusted platforms would not happen.

However, as Proudler remarks in [2], "Next-generation trusted platforms should be able to enforce policies". With the introduction of policies, the trusted system engineer's focus must necessarily extend from the above mechanisms to *events* that are classified and controlled by the policy.

Accordingly, it becomes natural to formulate the concept of what constitutes the measured state at the upper levels of the trust chain in terms of events subject to the policy: a sequence of policy-allowed events starting from a measured "good" state should only lead to another "good" state.

In fact, the concept of the underlying system of controlled events is central to the policy: whereas *policy goals* are defined in terms of the system's states, events determine the design of the underlying OS mechanisms and the policy language. For instance, in case of SELinux MAC policies, events are privileged operations realized as system calls hooked by the Linux Security Modules (LSM) framework.

One can argue (see, e.g., [3]) that LSM's implicit definition of the class of controlled events has substantially influenced both the scope and language of SELinux policies, making certain useful security goals such as, e.g., "trusted path," hard to express, and leading to a variety of alternative more manageable methods being adopted by practitioners. SELinux's example shows that defining a manageable set of controlled events is crucial to engineering the policy.

Another necessity faced by a policy designer is a workable definition of measured state. Quoting Proudler [2] again, "We know of no practical way for a machine to distinguish arbitrary software other than to measure it (create a digest of the software using a hash algorithm) and hence we associate secrets and private data with software measurements."

However, the semantics of what constitutes measured, trusted and private data in each case is inevitably left to the program's developers. Thus the particulars of what [2] refers to as *soft policy* are left to the developer and ultimately relies on his classification and annotation of different kinds of code constituting the program and data handled by the program with respect to their importance for policy goals. We refer to such annotation that allows the programmer to distinguish different kinds of data (such as public vs. private vs. secret, in the above-mentioned paper's classification) as "secure programming primitives" and note that introduction of new kinds of such primitives (e.g., read-only vs. read-write data, daemon privilege separation, trusted execution path patches, etc.) usually lead to improving the overall state of application security. In each case, it was up to the programmer to design the software to take advantage of the new security features; as we remarked above, TPMs are no different.

Recent alternative attestation schemes (e.g., [4]) move from measurement of a binary to an attestation that the measured binary is trustworthy, but still are based on static measurement. However, trustworthiness does not depend merely on what the binary looks like when it is loaded, but also on what it does (and what happens to it) when it executes.

It is from these angles that we approach the current TCG specification and propose to leverage it to control a set of memory-related events in which an application programmer can express a meaningful security policy.

We note that our proposal does not change the "passive" character of the TCG architecture: its scope is still restricted to providing security features to programs that are written to take advantage of them. Our contribution lies in introducing a framework of events and their respective handlers that invoke the TPM functionality. To base a policy of this framework, the programmer, as before, needs to separate program code and data into a number of classes based on their implications for the trustworthiness of the program as a whole, and specify the trust semantics for each class.

In this paper, we report our initial exploration, focusing on policies that enforce selective immutability of code and selected data in a program. The policy can be specified as immutability requirements for specific sections of the program's executable file and bundled with it, e.g., included as a special section in the ELF format.

Why extend security policy to memory events? Many architectures include OS support for trapping certain kinds of memory events. For instance, the ELF format supports annotation of program segments to be loaded into memory as writable or read-only, as well as executable or not intended for execution. Most of these annotations are produced automatically by the compiler–linker–loader chain that also takes care of aligning the differently annotated segments on page boundaries (since the x86 hardware supports these annotations by translating them to the appropriate PDE and PTE protection bits, which apply at page level).

We note that programmer's choice in this annotation has been traditionally kept to a minimum and not always exactly matched the programmer's intentions: e.g., constant data would be placed in the `.rodata` section, which would then be mapped, together with the `.text` section and other code sections to the loadable segment designated as non-writable *and* executable.

We further note that these automatic mappings of code and data objects[1] to different loadable segments are typically meant to apply at load time and persist throughout the lifetime of the process. More precisely, once an object has been assigned to one of these sections based on the programmer's guessed intentions, the programmer can no longer easily change its memory protections without reallocating it entirely.

We also note that "service" sections of process image such as `.got`, `.dtors`, etc., involved in such extremely security-sensitive operations as dynamic linking (and thus specifically targeted by many exploitation techniques[2]) do not, as a rule, change their protections even after all the relevant operations are completed, and

[1] We speak of "code objects" to distinguish between, e.g., the `.text`, `.init/.fini`, and `.plt` sections that all contain code dedicated to different purposes, similar to the more obvious distinction between the `.data`, `.ctors/.dtors`, and `.got` data objects sections.

[2] E.g., `http://www.phrack.com/issues.html?issue=59&id=9`, `http://www.security-express.com/archives/bugtraq/2000-12/0146.html`, etc.

write access to them is no longer necessary (but can still be exploited by both unauthorized code and authorized code called in an unanticipated manner).

Yet programmers may well conceive of changing roles of public, private or secret data throughout the different phases or changing circumstances of their programs' execution, and may want, *as a matter of security policy goals*, to change the respective protections on these objects in memory. Indeed, who else other than programmer would better understand these transitions in data semantics?

We contrast this situation with that in UNIX daemon programming before the wide adoption of privilege drop and privilege separation techniques. Giving the programmers tools to modify access privileges of a process according to the different phases of its execution resulted in a significant improvement of daemons' trustworthiness, eliminating whole classes of attacks. We argue that a similar approach applied to the sensitive code and data objects would likewise benefit the programmers who take advantage of it, and formulate their security goals in terms of memory events.

Secure programming primitives controlling memory events would provide assurance that the program could be trusted to trap on those events that the programmer knows to be unacceptable in any given phase, resulting in more secure programs. For example, programmers using a custom linking mechanism (such as that used by Firefox extensions) will be able to ensure that their program be relinked only under well-defined circumstances, defeating shellcode/rootkit attempts to insert hooks using the same interfaces.

Thus we envision a programming framework that provides a higher granularity of MMU traps caused by memory access events, and explicit access policies expressed in terms of such events. A modified MMU accommodating such a policy would provide additional settable bits that could be set and cleared to cause a trap on writes, providing a way to "seal" memory objects after they have been fully constructed, and to trap into an appropriate handler on events that could violate the "seal". The handler would then analyze the event and enforce the programmer's intentions for the data objects, now explicitly expressed as a policy (just as privilege drops expressed an implicitly assumed correct daemon behavior).

We discuss an interesting practical approach to achieving higher memory trapping granularity in Section 6—an example from a different research domain where the same need to understand and profile programs' memory access behaviors posed the same granularity problem as we and other security researches face.

In the following discussion we assume feasibility of higher granularity memory event trapping framework as described above, and explore the opportunities that such MMU modifications and respective trap handlers would provide for TOCTOU-preventing security policies when used in combination with the TCG architecture.

3 Vulnerability Demonstration

We'll begin by considering *attestation* of a platform's *software*. The TPM's PCRs measure the software; the TPM signs this configuration (using a protected key).

Then the user passes this configuration on to a third party which presumably uses this configuration information to decide whether or not to allow the measured platform to run a piece of software or join a network session.

Notice that a key component of this system is that if measures a binary at *load* time. If a binary changes after it has been loaded, the configuration of the system will not match the attested configuration. If a program can change this segment of RAM *after* it has been loaded, than its behavior can be modified, even though the measurement in the TPM shows that the program is in its original state. Such a change in the program's behavior can be accomplished in different ways, such as by an exploit supplied by an adversary in crafted input to the process itself, or though manipulation of the process's address space from another process through a lapse of kernel security. We note that in this paper we do not consider TOCTOU on hardware elements or carried out via hardware elements.

We must consider the potential of an attacker achieving malicious changes to the code or data of the running trusted process created from a measured executable at the end of the TPM-based chain of trust. We note that these scenarios are no less relevant for TCG compliant trusted systems, since "trusted" does not mean "trustworthy." Commodity operating systems have a long history of not being trustworthy, even if users choose to trust them. (Indeed, the term *trusted computing base* arose not because one *should* trust it, but rather because one had no choice *but* to trust it.)

We take the worst case: a kernel vulnerability that allows the attacker limited manipulation of x86 Page Tables (PT).[3] In Linux (as in other operating systems), these structures also contain information on the permissions needed by a process to read or modify the particular memory segment, interpreted by the MMU on every memory access while converting virtual addresses to physical ones.

To demonstrate[4] TOCTOU, we wrote a kernel module that replaced data at a given virtual address in a process image with the given process ID. In order to show that simply monitoring a process's page tables was not a sufficient defense against this attack, our module did not modify the target process's data structures at all, but rather changed other process' pages containing one `.text` segment or PTs. Section 5 provides the details of several such attacks.

We have constructed a small example login program to demonstrate this attack. After we have loaded and measured it, we used our module to overwrite the opcode 0x74 of the critical `je` instruction in the password check routine of the running process with 0x75 (`jne`), remapping PT entry to point to the target page, and resulting in login with any wrong password. When the module is unloaded, process image is restored to its original state, so that looks pristine to future TPM-based or other measurements.

[3] Several Linux kernel vulnerabilities allowed attackers to bypass memory mapping and IPC restrictions are summarized, e.g., in [5].

[4] For our initial demonstration, we considered an IBM NetVista PC equipped with a Atmel TPM v1.1b running the Trusted GRUB boot-loader and the 2.6.15.6 Linux kernel that included the statically compiled TPM driver. We later repeated the attack on PCs equipped with STMicro v1.2 TPMs.

Thus applications that use the TPM's ability to *seal* a credential against specific PCR values can be subverted in what was measured in the PCRs changes; applications that measure data and configuration files, as well as software, can also be subverted. Consider, for example, the open source LiveCD *Certification Authority* package [6] that uses a TPM to hold the CA's private key and to add assurance that the key would only be used when the system was correctly configured as the CA—by wrapping the private key to specified values in a specified subset of the PCRs. The TPM decrypts and uses that key only when the PCRs have those values. If a user has means to modify arbitrary regions of memory, they can render the measurements of the TPM useless, unless the TPM somehow keeps continuous measurements of the loaded program's memory. Because of TOCTOU, any hole in the OS breaks the trust.

4 Solution and Prototype

To address this problem, we need the TPM to "notice" when measured memory changes. As our proof-of-concept showed, the TPM needs to worry not just about writes to the virtual address and address space in question, but also about writes to any address that might end up changing the memory mapped to the measured region. Thus, we need to connect the TPM with memory management, so that the MMU would trap on memory operations that can affect TPM-measured memory regions, and inform the TPM of them via the corresponding trap handler.

To evaluate the feasibility and effectiveness of this idea, we need to actually try it—but experimental modifications to modern CPUs can be a large task. So instead, we build a software proof-of-concept demonstration.

4.1 Components

Since we needed a lightweight way to experiment with virtual changes to machines, we decided to start with the *Xen* virtual machine monitor [7], which allows for the simultaneous execution of multiple guest operating systems on the same physical hardware.

Xen. Xen is being used in this project not for its virtualization features, but as a layer that runs directly below the operating system—similar to the placement of the hardware layer in a non-virtualized environment. Its placement helps us study possible hardware features. In a Xen based system, all memory updates trap into the thin hypervisor layer—making it easy to monitor and keep tabs on changing memory. Redesigning the MMU hardware is tricky, so we do not want to attempt that until we were certain that the end goal was useful. A potentially better alternative to using Xen would have been to use an open-source x86 emulator (such as Bochs [8] or QEMU [9]). However, as of their current implementation, none of these emulators have support for emulating a TPM. Also, the only currently existing software-based TPM emulator [10] does not integrate with any of these. Integrating them would be a major task in itself.

Virtual TPMs. For our prototype we will be using the unprivileged Domain-1 as our test system. Unprivileged VMs cannot access the system's hardware TPM, and so, to provide Domain-1 with TPM access, we need to make use of virtual TPMs (vTPM) [11].

4.2 Design Choices

We considered two ways to use XEN in our implementation.

– First, the strategic placement of the thin Xen hypervisor layer between the machine's hardware and the operating system could be seen as a way to prototype changes that could be made in hardware (i.e. in the MMU). With this approach, the purpose of Xen would be to solely demonstrate a proposed hardware change, and would not be intended to be integrated into the *TCG Software Stack* [5] (TSS). Xen's role would be that of a "transparent" layer, manifesting features that would ideally be present in hardware. *Effectively, we use Xen to emulate a hardware trap framework for intercepting memory events of interest to our policy.*
– Alternatively, Xen could be used with the purpose of incorporating it into the TSS. The trusted boot sequence would now include the measurement of the Xen hypervisor executable, the Domain-0 Kernel and applications running in Domain-0, subsequent to the system being booted by a trusted boot-loader. In this model, our *Trusted Computing Base* (TCB) will be extended all the way up to the hosting virtual machine environment. The TCG trust management architecture is currently defined only up to the bootstrap loader; in this alternative approach, we would need to extend the chain of trust up to applications running in Domain-0.

Several recent projects (see Section 6) are exploring using hypervisors for integrity protection. However, as the hypervisor layer is not currently part of the TCG trust management architecture, incorporating it into the TSS will necessitate a revision of the TCG specification. Consequently, we went with the first approach.

We also considered two ways to hook memory updates to the TPM. In the **dynamic TPM** approach, we would update the TPM's state every time the measured memory of an application is changed. At all times then, the TPM's state will reflect the current memory configuration of a particular application, and of the system as a whole. This would allow a remote verifier to be aware of the current state of the application in memory, and to make trust judgments based on these presently stored PCR values.

When the hypervisor detects a write to the monitored area of an application's memory, it would invoke a re-measurement of the application in memory. The re-measurement would involve calculating a SHA1 hash of the critical area of the binary in memory (as opposed to the initial measurement stored in the PCR, which was of the binary image on disk). This re-measured value would

[5] The TCG Software Stack is the software supporting the platform's TPM.

be extended to the TPM. In this case, monitoring of memory writes would be enabled for the entire lifetime of an application, as the TPM state would need to be updated each time the application's measured memory changed.

In the **tamper-indicating TPM** approach, we would update the TPM's state only the first time that the measured memory of an application is changed. This would allow a remote verifier to easily recognize that the state of the application in memory has changed, and hence detect tampering. When the hypervisor detects the first write to a critical area of an application's memory, it would *not* invoke a re-measurement of the application; instead, would merely extend the TPM with a random value. In this case, monitoring of memory writes could be turned off after the first update to the TPM, as that update would be sufficient to indicate tampering. Monitoring subsequent writes (tampering) will not provide any further benefit. This strategy will not have as much of a negative impact on performance as the first approach.

For performance reasons, we chose the second approach.

4.3 Implementation

Our prototype implementation consists of three primary components: the instrumented Linux Kernel for reporting, the modified Xen hypervisor for monitoring, and the invalidation in the TPM.

Reporting. We instrumented the paravirtualized Kernel of the Domain under test (in our prototype – Domain-1) to allow it to report to the hypervisor the PTEs, and physical frames that these PTEs map to, of the memory to be monitored, as shown in Figure 1 (a).

To enable this feature, we added two new hypercalls. `HYPERVISOR_report_ptes` reports to the hypervisor a list of PTEs that map the memory that needs to be monitored. The PTEs are essentially the entries that map the `.text` section of the binary into memory. `HYPERVISOR_report_frames` reports to the hypervisor a

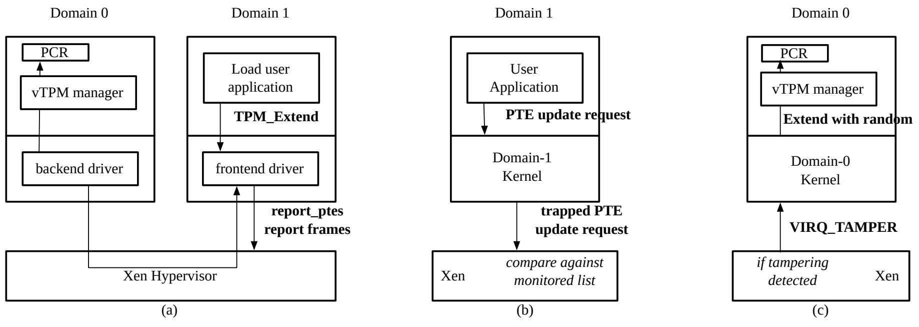


Fig. 1. (a) reporting to the hypervisor the PTEs and frames to be monitored; (b) monitoring the reported PTEs and frames for update; (c) updating PCR in vTPM of Domain-1 on tamper detection

list of physical memory addresses that need to be monitored. The addresses are the physical base addresses of each frame that contain memory that needs to be monitored.

These hypercalls make use of a new function that we have added to the kernel, `virt_to_phys()`, which walks a process's page tables in software to translate virtual addresses to physical addresses. We pass to this function the start and end virtual addresses of the `.text` section of the binary to be monitored. Using the fact that there are 4096 bytes of data on each page[6], it calculates the number of virtual pages spanned by the address range passed to it. It then accesses an address on each page of the range, so as to have it mapped into memory. This step is required to overcome potential problems due to *demand loading.*[7] At this point, the whole of the `.text` section of the binary is mapped into memory. This step however, has performance implications in that it slows down application start-up; unfortunately, dealing with this requires modification of the existing dynamic linker-loader to support deferred loading of trusted libraries. Although complex, it appears to be a promising direction of future research.

The function then walks the page tables of the process to translate the virtual addresses to physical addresses (physical base address) of each frame in the range. A data structure containing a list of these addresses is returned to the calling function.

Also, on program exit (normal or abnormal), we need to have the monitored PTES and frame addresses removed from the monitored list. To do this, we instrumented the Kernel's `do_exit` function to invoke a new hypercall.

`HYPERVISOR_report_exit` reports to the hypervisor when an application that is being monitored exits. The hypervisor's monitoring code then deletes the relevant entries from its monitored lists.

Monitoring. Once the required PTEs and frame addresses are passed down to Xen, it will monitor them to detect any modifications made to them, as shown in Figure 1.

Writes to these physical memory addresses, or updates to these PTEs to make them map to a different subset of memory pages or make them into writable mappings, will be treated as tampering. The reason for this is that since we are monitoring the read-only code section of an application, neither of the above updates are legitimately required.

The most convenient and reliable method of detecting these types of updates is to 'hook' into Xen's page table updating code. As mentioned earlier, all page table updates in a Xen system go through the hypervisor. This enables us to put in code that can track specific addresses and PTEs.

The default mode of page table updates on our experimental setup is the Writable Page Table mode. In this mode, writes to page table pages are trapped and emulated by the hypervisor, using the `ptwr_emulated_update()` function. Amongst other parameters, this function receives the address of the PTE that

[6] Our experimental system has a 4Kb page size.

[7] Demand loading is a lazy loading technique, where only accessed pages are loaded into memory.

needs to be updated and the new value to be written into it. After doing a few sanity checks, it invokes Xen's `update_l1e()` function to do the actual update.

We instrumented `update_l1e()` to detect tampering. Amongst other parameters, this function receives the old PTE value and the new PTE value that it needs to be updated to. To detect tampering, we perform the following checks:

– **For PTEs:** we check to see if the *old* PTE value passed in is part of our monitored list. If it is, it means that a 'trusted PTE' is being updated to either point to a different set of frames, or to make it writable. The alternate set of frames are considered as potentially malicious frames, and the updated writable permission leaves the corresponding trusted memory open for overwriting with malicious code.
– **For frames:** We first check to see if the *new* PTE value passed in has its writable bit set. If it does, we calculate the physical address of the frame it points to. We then inspect if this physical address is part of our monitored list. If it is, it means that a 'trusted frame' is being mapped writable by this new PTE. The writable mapping, created by this new PTE is interpreted as a means to overwrite the 'trusted frame' with potentially malicious code.

Once the tampering is detected in the hypervisor layer, we need to be able to indicate this fact to Domain-0. We do this by creating a new virtual interrupt, `VIRQ_TAMPER`, that a guest OS may receive from Xen. `VIRQ_TAMPER`, is a global virtual *Interrupt Request* (IRQ), that can be allocated once per guest, and is used in our prototype to indicate tampering with trusted memory.

Invalidating. Once tampering of trusted memory is detected in the hypervisor layer, the Domain under test needs to have its integrity measurements updated. This is done by way of updating the Domain's platform configuration in its virtual TPM, as shown in Figure 1.

Our intention is to have the hardware (MMU) cause this update by generating an trap, which would invoke an appropriate trap handler. The latter, having verified that the memory event is indeed relevant to our policy goals, will in turn perform the TPM operation.

Considering that, in our prototype, the hypervisor together with Domain-0 are playing the role of the hardware, we need to have either of them perform the update action. However, as there are no device drivers present in the hypervisor layer, the hypervisor is unable to interface with the virtual TPM of Domain-1, and so this task is redirected to the privileged Domain-0.

The hypervisor will indicate tampering to Domain-0 by sending a specific virtual interrupt (`VIRQ_TAMPER`) to it. A Linux Kernel Module in Domain-0 will receive this interrupt, and will proceed to extend the concerned PCR in the virtual TPM of Domain-1 with a random value.

We have to make use of the virtual TPM Manager (`vtpm_managerd`) to talk to the virtual TPM of Domain-1. In its current implementation, the virtual TPM manager only delivers TPM commands from unprivileged Domains to the software TPM. Domain-0 is not allowed[8] to directly interface with the software TPM. However, for our prototype, we need Domain-0 to have this ability, and so we have to mislead the virtual TPM Manager into thinking that the TPM commands from Domain-0 are actually originating from Domain-1.

In Domain-0, we construct the required TPM I/O buffers and command sequences required for a `TPM_Extend` to a PCR of Domain-1. As described earlier, there is a unique instance number associated with each vTPM. To enable Domain-0 to access the vTPM instance of Domain-1, we prepend the above TPM command packets with the instance number associated with Domain-1. This effectively help us forge packets from Domain-1.

5 Evaluation

Our prototype on x86, runs on a Xen 3.0.3 virtual machine-based system. Xen's privileged and unprivileged domains run Linux Kernel 2.6.16.29. Our evaluation hardware consists of a 2 GHz Pentium processor with 1.5 GB of RAM. Virtual machines were allocated 128 MB of RAM in this environment. Our machine has an Atmel TPM 1.2.

We implemented three attack scenarios subverting measured memory by exploiting the previously mentioned TOCTOU vulnerability. These attacks, seek to change the `.text` section of a loaded binary. The `.text` section is mapped read-only into memory, and so, is conventionally considered safe from tampering.

Scenario 1. The attacker overwrites the trusted code of a victim process by creating writable page mappings to the victim process's trusted frames from another process, as shown in Figure 2 (a).

We carried out this attack by modifying[9] a PTE in our malicious process to map to a physical frame in RAM that the victim process's trusted code was currently mapped to. We modified the PTE to hold the frame address of the victim process page that we wanted to overwrite. The PTE that we chose to update already had its writable bit set, so we did not need to update the permission bits. Using this illegitimate mapping we were able to overwrite a part of the trusted frame with arbitrary data.

It is interesting to note that this attack was possible without having to tamper with any of the victim process's data structures.

Our prototype detects that a writable mapping is being created to a subset of the physical frames that it is monitoring, and randomizes the relevant PCR to indicate tampering.

Scenario 2. The attacker modifies the trusted code of a victim process by updating the mappings of its `.text` section to point to rogue frames in RAM, as shown in Figure 2 (b).

[8] Domain-0 is only allowed to access the actual hardware TPM or the software TPM, but not the vTPM instances of other unprivileged domains.

[9] The attack could also be carried out by creating a new PTE that maps to the victim process's frames.

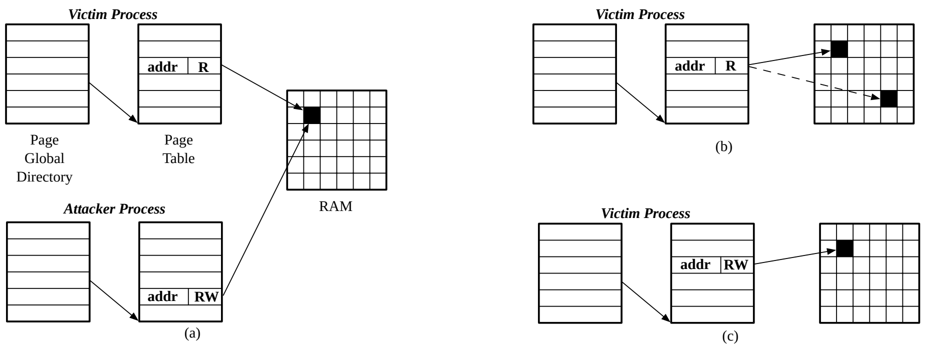


Fig. 2. (a) attacker manipulates PTE(s) of his process to map to trusted frames of victim process, and overwrites memory in RAM; (b) attacker manipulates PTE (address portion) of victim process to map to rogue frames in RAM; (c) attacker manipulates PTE (permission bits) of victim process to make frames writable, and overwrites memory in RAM.

We carried out this attack by using our malicious process to update the address portion of a PTE in the victim process that was mapping its code section. The updated address in the PTE mapped to rogue physical frames in RAM that were part of our malicious process. Due to these updated mappings, the victim process's trusted code was now substituted with the content of our rogue frame.

Our prototype detects that a subset of its monitored PTEs are being updated to point to different portions of RAM, and randomizes the relevant PCR to indicate tampering.

Scenario 3. The attacker overwrites the trusted code of a victim process by updating the permission bits of its `.text` section to make them writable, as shown in Figure 2 (c).

We carried out this attack by using our malicious process to update the permission bits of a PTE in the victim process that was mapping its code section. We updated the permission bits to set the writable bit making the corresponding mapped frame writable. We used this writable mapping to modify the trusted code in the victim process with arbitrary data.

Our prototype detects that a subset of its monitored PTEs are being updated to make them writable, and randomizes the relevant PCR to indicate tampering.

Limitations. It should be noted that the system of events we intend to capture and, therefore, the properties that we can enforce with it, deals at this point exclusively with memory accesses to code and data objects that can be distinguished by the linker and loader based on annotation in the executable's binary file format. As usual, the choice of events that we can trap to interpose our policy checks, limits our model of the system's trust-related states and transitions between those states, in particular, of transitions that bring the system into an untrusted state. In other words, the choice of a trappable event system essentially determines the kinds of exploitation scenarios that the policy based on it can and cannot stop.

As the analysis above demonstrates, our choice defeats a range of classic scenarios. It does not, however, prevent other types of attacks that do not depend on modifying protected code and data objects.

For example, our event system does not equip us for dealing with exploits that modify the control flow of an unchanged binary by providing non-executable crafted input data[10], for the essential reason that "bad" transitions in the state of software associated with these exploits are hard to express in terms of these events. The same goes for cross-layer and cross-interface input-scrubbing application vulnerabilities, such as various forms of SQL or shell command injection, where malicious commands are passed to a more privileged and trusted (and therefore less constrained) back-end. Obviously, such vulnerabilities should be mitigated by a different complementary set of secure programming primitives.

One case where a secure programming primitive based on our event system may help is that of protecting writable data known to the programmer to persist unchanged after a certain well-known phase of normal program execution. In particular, the programmer can choose to place such data in a special ELF loadable segment[11] that can be sealed after the completion of that execution phase. This, in fact, is a direct analogue to UNIX daemon privilege separation, which provides the programmer with the means to effectively disallow privileged operations that he knows to be no longer needed.

As described, our system of events also does not prevent return-to-library[12] or return-to-PLT[13] attacks in which no executable code is introduced (but, rather, existing code is used with crafted function activation frames placed on the stack to effect a series of standard library calls to do the attacker's work).

However, in this case our event system is by design more closely aligned with the actual events and transitions of interest: library calls and PLT stub invocations can be easily distinguished from other programmatic events based on the information contained in the binary format (such as that in the `.dynamic` section tree and symbol table entries). Accordingly, they can be trapped as cross-segment memory accesses, with minimal additional MMU support. This, in turn, enables policy enforcement based on intercepting such events and disallowing all but those permitted by the policy.

Although a detailed description of such an event system and policy mechanism and its comparison with other proposed return-to-library and return-to-special-

[10] E.g., `http://phrack.org/issues.html?issue=60&id=10` for exploitation of integer overflows, `http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-iss-sourceaudit.ppt` for a range of other exploitable conditions.

[11] In particular, the GNU tool chain offers a compiler extension to map variables to ELF sections. See, e.g., `http://www.ddj.com/cpp/184401956`.

[12] E.g., `http://www.milw0rm.com/papers/31`, `http://phrack.org/issues.html?issue=56&id=5`

[13] E.g., `http://phrack.org/show.php?p=58&a=4`, `http://www.phrack.org/issues.html?issue=59&id=9`. Note, in particular, the use of the dynamic linker to defeat load address randomization.

ELF-section countermeasures is beyond the scope of this paper, we intend to continue the study of the underlying trapping framework in another publication.

Performance. Although the primary goal of our software prototype is to investigate the future hardware features needed to provide the proposed secure programming primitives, we also measured varios performance overheads of the prototype itself, to quantify some of the design choice trade-offs discussed in Section 4.2. Our results, found in [12], show that the actual overhead of our prototype system is almost negligible, making it a very usable and deployable.

6 Related Work

Attacks. Kursawe, et. al [13] look at a number of passive attacks against TPMs by monitoring signals across the bus that the TPM resides on in their test machine and hint at more active attacks. Sadeghi, et. al [14] also discuss testing TPMs for specification compliance.

Bernhard Kauer [15] demonstrated how to fool a TPM into thinking the whole system has been rebooted. Sparks also documents this and provides a YouTube video [16]. Rumors exist that other attacks are coming [17]. Sparks also presents evidence [18] that the CRT-based RSA operations in TPM may be susceptible to Boneh-Brumley timing attacks [19].

Attestation. IBM designed and implemented a TPM-based *Integrity Measurement Architecture* (IMA) to measure the integrity of a Linux system. Their implementation [20] was able to extend the TCG trust measurement architecture from the BIOS all the way up into the application layer. Integrity measurements are taken as soon as executable content is loaded into the system, but before it is executed. An ordered list of measurements is maintained within the kernel, and the TPM is used to protect the integrity of this list. Remote parties can verify what software stack is loaded by viewing the list, and using the TPM state to ensure that the list has not been tampered with.

The *Bear/Enforcer* [21,22] project from Dartmouth College developed a *Linux Security Module* (LSM) to help improve integrity of a Linux system. This LSM calculates the hash of each protected file as it is opened, and compares it to a previously stored value. If a file is found to be modified, Enforcer does some combination of the following: denies access to the file, writes an entry in the system log, panics the system or locks the TCG hardware.

Sadeghi et al. (e.g., [4]) developed a *property-based attestation* extension to the TCG architecture that allows binary measurements to be mapped to properties the relying party cares about, and these properties to be reported instead. Haldar et al. [23] propose an approach based on programming language semantics.

Copilot [24] is a run-time kernel integrity monitor that uses a separate bus-mastering PCI add-in card to make checks on system memory. The Copilot monitor routinely recomputes hashes of the kernel's text, modules, and other critical data structures, and compares them against known good values to detect for any corruption.

BIND [25] is a service that performs fine-grained attestation for establishing a trusted environment for distributed systems. Rather than attesting to the entire contents of memory, BIND attests only to a critical piece of code that is about to execute. It narrows the gap between time-of-attestation and time-of-use, by measuring code immediately before it is executed, and protects the execution of the attested code by using a sand-boxing mechanism. It also binds the code attestation with the data that it produces. It requires programmer annotations, and runs within a Secure Kernel that is available in the new *LaGrande Technology* (LT)-style CPUs.

Additionally, as discussed back in Section 4.2, several recent projects use a secure hypervisor to extend trust to systems' runtime. *Overshadow* [26] and *SecVisor* [27] use the hypervisor's extra level of memory indirection/virtualization for protecting the runtime integrity of code and data. We were also made aware of the HP Labs effort [28] that extends the software chain of trust wih a trusted VMM hypervisor for fine-grained immutability protection of both OS kernel and applications.

7 Conclusions and Future Work

We show that current assumptions about the run-time state of measured memory do not properly account for possible changes after the initial measurement. Specifically, previously measured memory can be modified at run-time, in a way that is undetectable by the TPM. We argue that these assumptions of the OS and application trustworthiness with respect to memory operations can and should be backed up by a combination of a memory event trapping framework and associated TPM operations performed by its trap handlers, to ensure that the programmer's expectations of access patterns to the program's sensitive memory objects are indeed fulfilled and enforced.

We demonstrated several software-based TOCTOU attacks on measured memory, considered ways to detect such attacks—by monitoring the relevant PTEs and physical frames of RAM—and presented a Xen-based proof-of-concept.

One avenue of future work is to explore how to have the TPM reflect other avenues of change to measured memory. Currently, we only protect against physical memory accesses that are resolved by traversing the page tables maintained by the MMU; one future path is protecting against *Direct Memory Access* (DMA) as well. In our initial prototype, we have not implemented any functionality related to paging to disk, and leave that to future work.

We would also like to explore more carefully monitoring *data objects* as well as software, elevating it to a subset of the program's policy goals. In particular, we suggest that the programmer should be provided with the secure programming primitives to reflect the changing access requirements of sensitive data objects. In particular, besides critical code, it would be beneficial to monitor important data structures, such as those involved in linking: various function pointer tables (*Global Offset Table* (GOT), *Procedure Linkage Table* (PLT) and similar application-specific structures. We argue that the programmer should be given tools to express the security semantics of such objects.

In their current specification and implementation, the sealing/wrapping and signing facilities do not bind secrets and data to their 'owner' process. (By 'owner' we refer to the application that either encrypted/signed a piece of data or generated a key.) This lack of binding could have security implications. Any running application on the system could potentially unseal the data or unwrap the key, and use it for unauthorized purposes. The TCG specification does have a provision to guard against this – specifying a password [14] that will be checked against at the time of unsealing or unwrapping, in conjunction with checking the value in the PCRs. However, if no password is specified, or if it is easily guessable, it leaves open the possibility for unauthorized use.

One way of resolving this problem would be to have a dedicated resettable PCR on the TPM that would be extended with the hash of the binary of the currently executing process on the system. This PCR would be included in the set of PCRs that are used for sealing/wrapping against. As a consequence, the 'owner' process would be required to be currently active on the processor for unsealing/unwrapping to be successful. Every time there is a context-switch (indicated by a change of value in the `CR3` register), the above-mentioned PCR would first be reset, and then be extended with the relevant hash value. This mechanism would prevent a process that is not the 'owner' of a particular sensitive artifact from accessing it.

Implementing the dynamic TPM, as described in Section 4, would be a radical step forward in the way TPMs currently operate. It would enable the TPM to hold the run-time memory configuration of a process, and hence allow for more accurate trust judgments.

Essentially, the TPM itself is a "secure programming primitive", a new tool for software developers to secure critical data and enforce policy. Our proposed TOCTOU countermeasures—and future research built on them—are an extension of this tool for developers to take advantage of these fundamental new primitives to secure data and enforce policy. We note that extending the TCG architecture with additional primitives for ensuring trustworthy memory behaviors appears to be a natural direction towards policy-enforcing next generation trusted systems as per vision outlined in [2].

We also argue that allowing developers to express the intended properties of their memory and code objects as a matter of policy, and providing a TCG architecture-based mechanism for enforcing such policies will help developers to manage previously neglected aspects of their program's trustworthiness. We draw historical parallels with other successful secure programming primitives that provided developers with capabilities to express the intended privilege and access behaviors in enforceable ways, and that have undoubtedly resulted in improving programs' trustworthiness.

[14] The password is stored in a wrappedKey data structure associated with the corresponding asymmetric key.

Acknowledgments

We would like to thank our colleages Dr. Douglas McIlroy, Dr. Andrew Campbell, Ph.D. candidates John Baek and Alex Iliev, and our other helpful colleagues at ISTS and Dartmouth PKI/Trust Laboratory for useful discussions. Preliminary versions of these results have appeared in the second author's thesis.

References

1. Trusted Computing Group: Homepage, `http://www.trustedcomputinggroup.org`
2. Proudler, G.: Concepts of Trusted Computing. In: Mitchell, C. (ed.) Trusted Computing, IET, pp. 11–27 (2005)
3. Bratus, S., Ferguson, A., McIlroy, D., Smith, S.: Pastures: Towards Usable Security Policy Engineering. In: ARES 2007: Proceedings of the The Second International Conference on Availability, Reliability and Security, Washington, DC, USA, pp. 1052–1059. IEEE Computer Society, Los Alamitos (2007)
4. Sadeghi, A.R., Stüble, C.: Property-Based Attestation for Computing Platforms: Caring about Properties, not Mechanisms. In: New Security Paradigms Workshop (2004)
5. Arce, I.: The Kernel Craze. IEEE Security and Privacy 2(3), 79–81 (2004)
6. Franklin, M., Mitcham, K., Smith, S.W., Stabiner, J., Wild, O.: CA-in-a-Box. In: Chadwick, D., Zhao, G. (eds.) EuroPKI 2005. LNCS, vol. 3545, pp. 180–190. Springer, Heidelberg (2005)
7. Xen: Virtual Machine Monitor, `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/`
8. Bochs: IA-32 Emulator Project, `http://bochs.sourceforge.net/`
9. QEMU: Open Source Processor Emulator, `http://www.qemu.com/`
10. Strasser, M.: Software-based TPM Emulator for Linux. Department of Computer Science. Swiss Federal Institute of Technology Zurich (2004)
11. Berger, S., Caceres, R., Goldman, K., Perez, R., Sailer, R., van Doorn, L.: vTPM – Virtualizing the Trusted Platform Module. In: 15th Usenix Security Symposium, pp. 305–320 (2006)
12. D'Cunha, N.: Exploring the Integration of Memory Management and Trusted Computing. Technical Report TR2007-594, Dartmouth College, Computer Science, Hanover, NH (May 2007)
13. Kursawe, K., Schellekens, D., Preneel, B.: Analyzing trusted platform communication (2005), `http://www.esat.kuleuven.be/cosic/`
14. Sadeghi, A.R., Selhorst, M., Stüble, C., Wachsmann, C., Winandy, M.: TCG Inside - A Note on TPM Specification Compliance.
15. Kauer, B.: OSLO: Improving the security of Trusted Computing. Technical report, Technische Universitat Dresden, Department of Computer Science (A later version appeared at USENIX Security 2007) (2007)
16. Sparks, E.: TPM Reset Attack, `http://www.cs.dartmouth.edu/~pkilab/sparks/`
17. Greene, T.: Integrity of hardware-based computer security is challenged. NetworkWorld (June 2007)
18. Sparks, E.: A Security Assessment of Trusted Platform Modules. Technical Report TR2007-597, Dartmouth College, Computer Science, Hanover, NH (June 2007)
19. Boneh, D., Brumley, D.: Remote Timing Attacks are Practical. In: Proceedings of the 12th USENIX Security Symposium (2003)

20. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: USENIX Security Symposium, pp. 223–238 (2004)
21. Marchesini, J., Smith, S.W., Wild, O., Stabiner, J., Barsamian, A.: Open-Source Applications of TCPA Hardware. In: Yew, P.-C., Xue, J. (eds.) ACSAC 2004. LNCS, vol. 3189, pp. 294–303. Springer, Heidelberg (2004)
22. Marchesini, J., Smith, S.W., Wild, O., MacDonald, R.: Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love The Bear. Technical Report TR2003-476, Dartmouth College, Computer Science, Hanover, NH (December 2003)
23. Haldar, V., Chandra, D., Franz, M.: Semantic Remote Attestation: A Virtual Machine Directed Approach to Trusted Computing. In: USENIX Virtual Machine Research and Technology Symposium (2004)
24. Petrom Jr., N.L., Fraser, T., Molina, J., Arbaugh, W.A.: Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor. In: 13th USENIX Security Symposium, pp. 179–194 (2004)
25. Shi, E., Perrig, A., van Doorn, L.: BIND: A Fine-Grained Attestation Service for Secure Distributed Systems. In: IEEE Symposium on Security and Privacy, pp. 154–168 (2005)
26. Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J., Ports, D.R.: Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems, pp. 2–13. ACM, New York (2008)
27. Seshadri, A., Luk, M., Qu, N., Perrig, A.: SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In: SOSP 2007: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, pp. 335–350. ACM, New York (2007)
28. Cabuk, S., Plaquin, D., Dalton, C.I.: A Dynamic Trust Management Solution for Platform Security Using Integrity Measurements. Technical report, Hewlett-Packard Laboratories (April 2007)

A Software-Based Trusted Platform Module Emulator

Mario Strasser[1] and Heiko Stamer[2]

[1] ETH Zurich, Switzerland
strasser@tik.ee.ethz.ch
[2] Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
stamer@theory.informatik.uni-kassel.de

Abstract. When developing and researching new trusted computing technologies, appropriate tools to investigate their behavior and to evaluate their performance are of paramount importance. In this paper, we present an efficient and portable TPM emulator for Unix. Our emulator enables not only the implementation of flexible and low-cost test-beds and simulators but, in addition, provides programmers of trusted systems with a powerful testing and debugging tool that can also be used for educational purposes. Thanks to its portability and interoperability, the TPM emulator runs on a variety of platforms and is compatible with the most relevant software packages and interfaces.

1 Introduction

The Trusted Computing Group (TCG) has developed various specifications for trusted computing building blocks and software interfaces, among which the Trusted Platform Module (TPM) [12] is likely to be the most popular one. These specifications are of growing interest and the increasing availability of TPMs will soon make it a standard component in today's personal and mobile computers.

When developing and researching new technologies, appropriate tools to investigate their behavior and to evaluate their performance are of paramount importance. This is particularly true for complex large-scale distributed systems comprising hundreds of nodes and users. In the networking community, test-beds and software simulators are the methods of choice in order to conduct network performance analysis and protocol evaluations. We argue that these methods would also be valuable tools for evaluating and understanding the behavior of trusted computing systems. However, the security properties of hardware TPMs permit only one TPM per platform and limit the degree to which the effects of an executed command can be undone as well as to what extent the internal state of a TPM can be (re)stored or preset. Moreover, certain actions cannot be revoked at all or require the physical presence of an operator. Also, TPMs provide almost no feedback in the case of internal or usage errors. Due to these limitations, building a test-bed for the execution and evaluation of TPM-based distributed trusted computing systems is extremely cumbersome and costly. Moreover, the realistic (hardware-supported) simulation of such systems is often infeasible.

In this paper, we advocate the usage of software-based TPMs to solve this hardware dependency, and present an efficient and portable open-source TPM emulator for Unix [9]. The possibility to run more than one TPM emulator instance per platform and to restore previously stored or artificially created states allows for the implementation of flexible and low-cost test-beds and simulators. Running entirely in software, the TPM emulator can further be used to enhance virtual machines, thus enabling the execution of TPM-based software in a trustworthy virtualisation environment [2].

The TPM emulator also facilitates the evaluation of TPM extensions and firmware enhancements. In particular, it can be used to simulate new TPM commands (e.g., TPM_ExecuteHashTree [8]) and vendor extensions before including them in a hardware specification or even before the real development process starts. Due to the fast simulation in software, extensive preliminary tests can be performed much more efficiently. Researchers from Princeton University, NEC Labs, and Texas Instruments R&D, for instance, used our emulator to evaluate the impact on the TPM's energy and execution time overhead, if the RSA algorithm is replaced with elliptic curve cryptography (ECC) [1].

Finally, the emulator provides programmers of trusted computing systems with a powerful testing and debugging tool that can also be used for training and educational purposes. Thanks to its portability and interoperability, the TPM emulator runs on a variety of different computing platforms (e.g., Intel, AMD, StrongARM, and PowerPC) and is compatible with the most relevant software stacks (e.g., TrouSerS [14] and jTSS [13]).

The remainder of this paper is organized as follows: In Section 2 we summarize the main features and capabilities of a TPM. We outline the design and implementation of our TPM emulator in Section 3, report on the current state of the implementation in Section 4, and conclude the paper in Section 5.

2 Trusted Computing and Trusted Platform Module

In this section, we summarize the basic definitions regarding trusted computing and provide a brief overview of the structure and functionality of the TPM. For a comprehensive description, the reader is referred to the TCG documents [10,12] and recently published books [6,5].

Trusted computing is about embedding a trusted computing base (TCB) [3] in a computing platform that allows a third party to determine the trustworthiness of the platform, that is, whether or not the platform is a *trusted platform*. The TCG [10] has specified a TCB for trusted computing by three so-called roots of trust: the *Root of Trust for Storage* (RTS), the *Root of Trust for Reporting* (RTR), and the *Root of Trust for Measurement* (RTM). The *Trusted Platform Module* (TPM) as specified by the TCG is a hardware component that acts as a root of trust for storage and reporting. More precisely, it provides four major classes of functions: (1) secure storage and reporting of platform configurations, (2) protected key and data storage, (3) cryptographic functions, and (4) initialization and management functions.

2.1 Root of Trust for Measurement

When a computer is booted, control passes between different subsystems. First the BIOS is given control of the computer, followed by the boot loader, the operating system loader, and finally the operating system. In an *authenticated boot*, there is a bootstrapping process in which, starting with the BIOS, one subsystem measures the next in the boot sequence and records the obtained value before executing its successor. More precisely, the BIOS measures (i.e., cryptographically hashes) the boot loader prior to handing over control. The boot loader, in turn, measures the operating system loader, and the operating system loader measures the operating system. These measurements reflect what software stack is in control of the computer at the end of the boot sequence; in other words, they reflect the platform configuration. A *Platform Configuration Register* (PCR) is a TPM register where such measurements are stored and which is initialized at startup and extended at every step of the boot sequence. In order to allow the storage of an unlimited number of measurements in a specific PCR register, the new value depends on both, the old value and the new value to be added. Consequently, updates to PCRs are noncommutative and cannot be undone until the platform is rebooted.

2.2 Root of Trust for Reporting

Each TPM has an unique *Endorsement Key* (EK) whose public key is certified by a trusted third party, such as the TPM manufacturer. For privacy reasons, the EK is only used to obtain credentials from a certification authority (Privacy CA) for an *Attestation Identity Key* (AIK), which the TPM generates by itself. These are signing keys whose private key is only used for signing data that has originated from the TPM. For example, a remote party can query the TPM for PCR values. The query contains the set of registers to look up and a nonce, in order to check for replay attacks. The TPM replies with the requested PCR values and a signature on these values and the given nonce by one of its AIKs. Consequently, the TPM *attests* to, or *reports* on, the platform configuration.

2.3 Root of Trust for Storage

The protected storage feature of a TPM allows for the secure storage of sensitive objects such as TPM keys and confidential data. However, storage and cost constraints require that only the necessary (i.e., currently used) objects can reside inside a TPM; the remaining objects must be stored outside in unprotected memory and only loaded into the TPM on demand. To achieve the desired protection, externally stored objects are encrypted (or *wrapped* in TCG terminology) with an asymmetric *storage key*, which is referred to as the parent key of the object. A parent key can again be stored outside the TPM and (possibly along with other keys) be protected by another storage key. The thereby induced storage tree is rooted at the so called *Storage Root Key* (SRK), which is created upon initialization of the TPM and cannot be unloaded. Each key is either marked as being *migratable* or *non-migratable*. If the key is migratable, it might be replicated and moved to other TPM-protected platforms. Otherwise, it is bound to an individual TPM and is never duplicated. Furthermore, the TPM specification distinguishes between *binding* and *sealing*: Binding is the operation of encrypting an object with the public key of a so-called *binding key*. If the binding key is non-migratable, only the TPM that created the key can use the corresponding private key; hence, the encrypted object is effectively bound to a particular TPM. Sealing takes binding a step further: the object is not only bound to a particular TPM, but in addition it can only be decrypted, if the current platform configuration matches the PCR values associated with the protected object at the time of creation.

2.4 Cryptographic Functions

A TPM must support a minimum set of cryptographic algorithms and operations. However, a TPM is not supposed to be a cryptographic accelerator, i.e. there are no specified throughput requirements for any of the cryptographic functions. The mandatory cryptographic functions are: a) Either a true hardware-based or a algorithmic pseudo random-number generator; b) a message digest function SHA-1 and corresponding message authentication code (HMAC) engine; and c) a RSA signing, encryption and key-generation unit.

2.5 Auxiliary Functions

TPMs which conform to the TPM specification version 1.2 additionally provide the following auxiliary functions: *Monotonic counters* implement an ever-increasing incremental value which is designed to not wear out in the first 7 years of operation (with an increment once every 5 seconds). *Non-volatile storage* provides the manufacturers and owners of a TPM with a protected and shielded non-volatile storage area for storing sensitive or confidential information. *Auditing* gives the TPM owner the ability to determine that certain operations on the TPM have been executed. *Authorization delegation* allows for the delegation of individual TPM owner privileges (i.e., the right to use individual owner authorized TPM commands) to individual entities.

2.6 Startup

A TPM has three different startup modes: *clear*, *state*, and *deactivated*. In mode clear, the platform starts in a cleared state where most data is reset to its default value. In mode state, which is the default mode, a previously saved state is restored and the TPM continues its operation from the restored state. In mode deactivated, the TPM deactivates itself and rejects any further operations.

2.7 TPM Command Structure

TPM commands (responses) are byte arrays which consist of three main parts: a request (response) header, some command-specific input (output) parameters, and an optional authorization trailer (see Figure 1). Presence or absence of the authorization trailer depends on the used command (response) tag field. Integer values which are larger than one byte are encoded in network byte-order and might have to be converted into the local byte-order of the host platform as part of the unmarshaling process. Since version 1.2 of the TPM specification, commands can also be sent from and to the TPM in an encrypted form using the transport encryption commands.

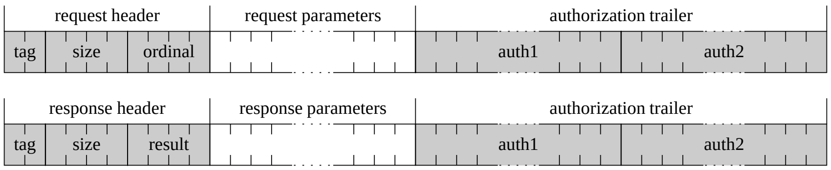


Fig. 1. TPM command structure

Command Authorization. The purpose of several authorization protocols (e.g., OIAP and OSAP) is to prove that the initiator of a command has permission to execute and to access the involved data objects. For instance, they are used to establish platform ownership, restrict key usage, and to apply access control to (protected) objects. The proof comes from the knowledge of a shared secret, the so called authorization data. The TPM treats knowledge of the corresponding authorization data as a complete proof, i.e. no other checks are necessary.

3 Design and Implementation

The TPM emulator package comprises three main parts (see Figure 2): a user-space daemon (tpmd) that implements the actual TPM emulator, a TPM device driver library (tddl) as the regular interface to access the emulator, and a kernel module (tpmd_dev) that provides the character device `/dev/tpm` for low-level compatibility with TPM device drivers.

3.1 TPM Device Driver Library and Kernel Module

The most convenient way to access a TPM is by means of the so-called TPM Device Driver Library (TDDL) as specified by the TCG. The library provides a simple and standardized interface for connecting to a TPM, querying its state and capabilities as well as for sending TPM commands and receiving the corresponding responses (for a comprehensive description of its functionality we refer to the TSS specification [11]). Applications that use the TDDL interface can be

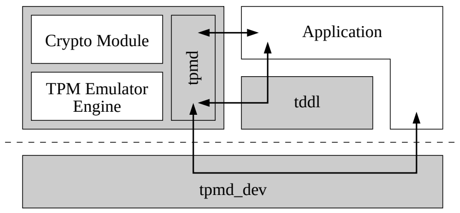


Fig. 2. Overview of the TPM emulator package

forced to use the TPM emulator instead of a real TPM by simply exchanging this library. However, despite the existence of this well defined and easy to use interface, several applications, tools, and libraries rather directly access the TPM by its device driver interface, i.e. by the device file `/dev/tpm`. Therefore, and in order to be compatible with hardware TPMs at the lowest possible level, the TPM emulator package includes a Linux kernel module called `tpmd_dev`. This module simulates a hardware TPM driver interface by providing the device `/dev/tpm` and by forwarding all commands to the TPM emulator daemon. This approach is completely transparent for an application and thus nearly indistinguishable from the interaction with a real TPM.

3.2 TPM Emulator Daemon

The TPM emulator daemon implements the actual TPM emulator and consists of the daemon application, the TPM emulator engine, and the cryptographic module. After initializing the TPM emulator engine, the daemon application opens a Unix domain socket and listens on it for incoming TPM commands. Upon the reception of a valid command, the request is processed by the TPM emulator engine and the corresponding response returned to the sender of the command. Per default the socket name `/var/run/tpm/tpmd_socket:0` is used but, if required (e.g., if more than one TPM emulator instance has to be started), an alternative name can be specified as a command-line argument.

TPM Emulator Engine. From a functional point of view, the TPM emulator engine is further divided in the parameter (un)marshaling and (de)coding entity; the command execution engine; the cryptographic engine; and the key, data, and state storage entity. Each part interacts with each other over a small set of well-defined functions.

The public API of the TPM emulator engine consists of only three main functions. The first function to be called is `tpm_emulator_init()` which initializes the emulator. Afterwards the function `tpm_handle_command()` can be used to execute an arbitrary number of TPM commands, before a call of the function `tpm_emulator_shutdown()` shuts down the emulator.

Initialization and Self-Test. The initialization of the TPM emulator proceeds in three steps: First, all data elements are set to their default values. Next, an internal self-test of all cryptographic functions is performed. If one of the tests fails, the emulator goes into a fail-stop mode and returns a corresponding error message on each command call. Finally, if all tests succeeded, the emulator starts in the specified startup mode and the remaining internal data is initialized accordingly. In the particular case of startup mode *save* this includes the restoration of all persistent data from the persistent data storage file (`/var/lib/tpm/tpm_emulator-1.2.x.y` as default). Whenever the restoration of a previously stored state fails, the emulator also switches to fail-stop mode.

Shutdown and Data Storage. On a regular shutdown, that is, if the emulator is not in fail-stop mode, all persistent data is written into the persistent data storage file. Otherwise, the internal state of the TPM emulator is discarded in order to avoid that temporary malfunctions lead to a permanent failure of the emulator. Finally, all allocated resources such as memory and file handles are released.

Command Execution. The execution of a TPM command is initiated by calling the function `tpm_handle_command()` with the marshaled TPM command (i.e., a byte array) as input parameter. The command is then processed in three steps: In the first step, the command is decoded into its three main components: the request header, the command parameters, and the (optional) authorization trailer (see Figure 1). After verifying whether the command is allowed to be executed in the current state of the TPM, the input parameter digest is computed and the parameters are further unmarshaled according to the specific TPM command. The second step performs the actual execution of the command and sets the command response parameters. If required, the authorization trailer is verified in order to ensure command authorization and integrity of the input parameters. In the last step, the response authorization is computed and combined with the output parameters and the response header. Finally, the response is marshaled into its byte representation and returned to the caller.

Portability and System Independence. In order to facilitate the porting of the TPM emulator, the TPM emulator engine contains no operating system or platform dependent functions (e.g., memory allocation and file handling), but accesses them by a set of predefined functions. The actual implementation of these functions is in the responsibility of the OS-dependent application. The therefore required API as well as all system specific settings (e.g., whether the system uses little- or big-endian byte ordering etc.) are specified in the header file `tpm_emulator_config.h`.

Similar considerations lead to the separation and encapsulation of all cryptographic functions. The TPM emulator accesses the required functionality, i.e. the SHA-1, HMAC, random number generator, RSA, RC4, and multiple precision integer arithmetic (bignum) methods by means of well defined interfaces, but does not make any further assumptions regarding their implementation. This allows for an easy replacement of individual function blocks, e.g. in the case of (partly) available hardware support.

3.3 Implementation Issues

Some technical issues that occurred during implementation are worth to be noted here: First of all, initially the TPM emulator was designed as a sole kernel module. Thus, implementation and debugging of most functions (in particular those needed for the RSA support) was not straightforward due to the limited kernel stack size (4 KByte or 8 KByte on x86) and the non-availability of third party libraries. Moreover, the persistent storage needed for saving the internal state of the TPM was obtained by directly accessing the local file system from the kernel module. This direct access can cause unintended side effects if the user context changes and should thus be avoided. All these issues have been solved in our new design by introducing a user-space daemon. Finally, the typographical errors and ambiguities in early revisions of the TPM specification [12] often turned out to be additional stumbling blocks during development.

4 Performance Evaluation and Compliance Tests

The two main criteria for evaluating the presented software-based TPM emulator are (i) its performance compared to existing hardware-based TPMs and (ii) its compliance to the current TPM specification. For the performance evaluation, the experiments were conducted on a IBM ThinkPad T43p (Pentium M @ 2 GHz and 1 GByte RAM) using Linux kernel 2.6.23 and a hardware TPM v1.1b from the National Semiconductors Corporation.

Figure 3 shows the average execution time for a set of light-weight and computation-intensive TPM commands. One can observe that for computation-intensive commands the emulator is on average about ten times faster than the hardware-based TPM. In the case of the light-weight command TPM_Extend, the performance advantage is even three orders of magnitude (i.e., factor thousand). This difference shows that – other than for our TPM emulator – the communication with a hardware TPM constitutes a non negligible overhead with respect to the overall command execution time.

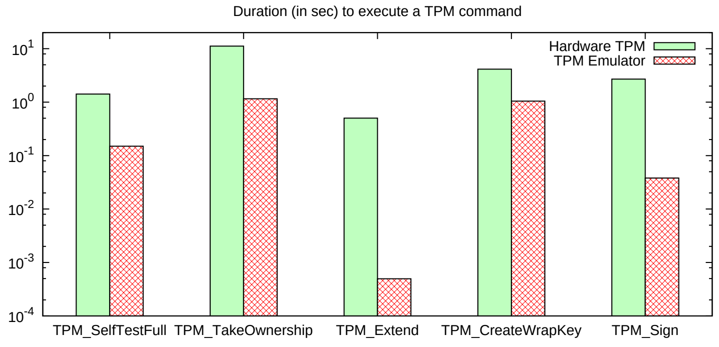


Fig. 3. Performance comparison of the emulator with a hardware TPM v1.1b

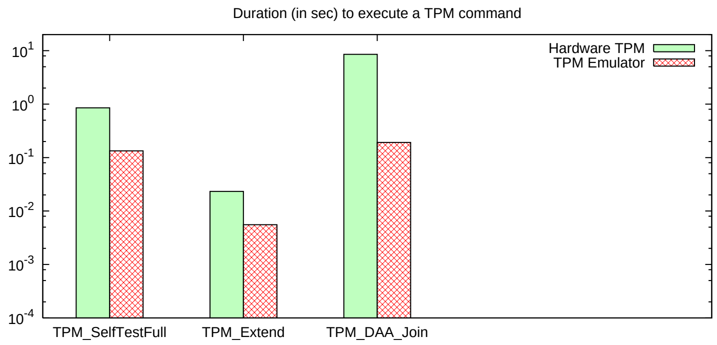


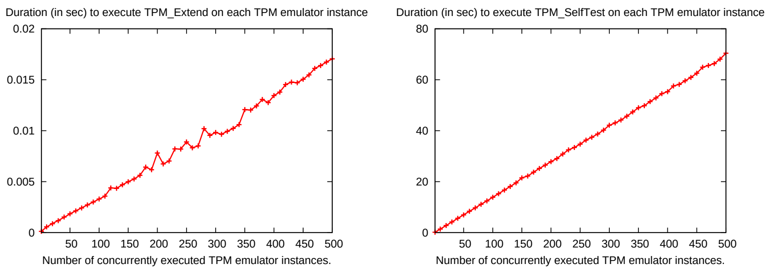
Fig. 4. Performance comparison of the emulator with a hardware TPM v1.2

The above results have been confirmed on a Lenovo ThinkPad X61s (Core 2 Duo @ 1.6 GHz and 2 GByte RAM) using Linux kernel 2.6.24 and a hardware TPM v1.2 from Atmel Corporation (see Figure 4). Even the computation-intensive commands like TPM_DAA_Join are up to hundred times faster. This is not really surprising because the TPM emulator can use sophisticated algorithms from third party libraries (e.g., the GNU Multiple Precision Arithmetic Library [4]) and a considerably faster general-purpose CPU.

In order to investigate the scalability of our TPM emulator in relation to the number of concurrently executed emulator instances on the same platform, a number of TPM emulator daemons was started and a series of TPM command executions was initiated on each instance in parallel. The measured duration between sending the commands and receiving all corresponding responses is shown in Figure 5. One can see that for light-weight (Figure 5(a)) as well as for computation-intensive commands (Figure 5(b)) the required time increases linearly with the number of concurrently running emulator instances. This shows that except for the unavoidable slowdown due to the availability of only one single CPU, no significant overhead is introduced by concurrently executing more than one emulator instance. This is of particular importance if one needs to simulate a whole distributed system on a single host.

The results of several TPM compliance and functionality tests, in particular those provided by TrouSerS [14] (package testsuite and package tpm-tools), jTSS [13] (package jTSS and package jTpmTools), IBM DAA Test Suite [15] and RUB/Sirrix [7], are summarized in Figure 6. Note that, although the emulator still lacks some functionality, the most important and frequently used commands are already supported and are compliant with the most recent TPM specification [12]. A detailed list of the TPM commands can be found in Appendix A.

Currently the following function blocks are still missing: some capability areas or commands as well as all migration and delegation functions.



(a) TPM_Extend (b) TPM_SelfTest

Fig. 5. Duration for the execution of two commands on each TPM emulator instance

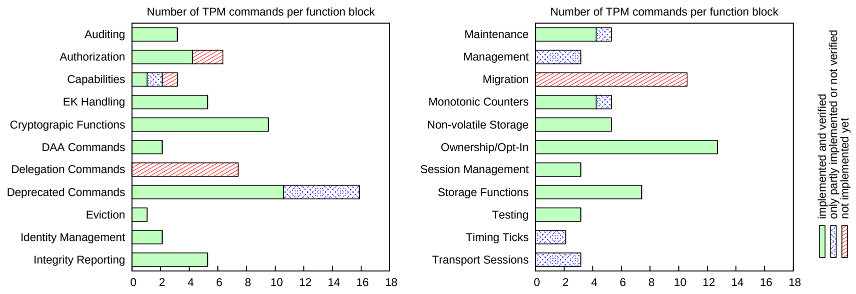


Fig. 6. Summarized results of several TPM compliance and functionality tests

5 Conclusions and Future Work

In this paper, we presented a software-based TPM emulator for Unix. Even though the emulator still lacks some functionality, it is compatible with the most relevant software stacks and works very well with almost all available TPM-enabled Unix applications. In fact, the most important and frequently used commands are already supported. We showed that hundreds of (concurrently running) emulator instances can be efficiently simulated on a single platform, allowing for the implementation of low-cost test-beds and simulators. In addition, the emulator provides programmers of trusted computing systems with a powerful testing and debugging tool that can also be used for training and educational purposes. The emulator is still a work in progress and our future attention will focus on the completion of the missing commands. A further step is the portage of the TPM emulator package to Windows and Mac OS X operating systems.

References

1. Aaraj, N., Raghunathan, A., Ravi, S., Jha, N.K.: Energy and Execution Time Analysis of a Software-Based Trusted Platform Module. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2007), pp. 1128–1133 (2007)

2. Anderson, M.J., Moffie, M., Dalton, C.I.: Towards Trustworthy Virtualisation Environments: Xen Library OS Security Service Infrastructure. Technical Report HPL-2007-69, HP Laboratories Bristol (April 2007)
3. Matt Bishop. Computer Security: Art and Science. Addison Wesley, Reading (2003)
4. Granlund, T., et al.: GNU Multiple Precision Arithmetic Library.
5. Challener, D., et al.: A Practical Guide to Trusted Computing. IBM Press (2007)
6. Mitchell, C., et al.: Trusted Computing. IET (2005)
7. Sadeghi, A.-R., Selhorst, M., Stüble, C., Wachsmann, C., Winandy, M.: TCG inside? A Note on TPM Specification Compliance. In: Proceedings of the 1st ACM Workshop on Scalable Trusted Computing (STC 2006), pp. 47–56 (2006)
8. Sarmenta, L.F.G., van Dijk, M., O'Donnell, C.W., Rhodes, J., Devadas, S.: 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), pp. 27–42 (2006)
9. Strasser, M., et al.: Software-based TPM Emulator, `http://tpm-emulator.berlios.de/`
10. Trusted Computing Group. Architecture Overview
11. Trusted Computing Group. TPM Software Stack (TSS) Specification, Version 1.2, `https://www.trustedcomputinggroup.org/specs/TSS/`
12. Trusted Computing Group. TPM Specification, Version 1.2, Revision 103, `https://www.trustedcomputinggroup.org/specs/TPM`.
13. TU Graz, IAIK. jTSS – Java TCG Software Stack, `http://trustedjava.sourceforge.net/`
14. Yoder, K., et al.: TrouSerS – Open-source TCG Software Stack, `http://trousers.sourceforge.net/`
15. Zimmermann, R.: IBM Direct Anonymous Attestation Tools – TPM Test Suite, `http://www.zurich.ibm.com/security/daa/`

A Appendix: Supported TPM Commands

The following table shows the current state of implementation for each command from the most recent TPM specification [12]. The background color of a row corresponds to the state, i.e., *completed* , *partially implemented* , or *missing* .

Each row contains several information about the command: the name; the version of the TPM specification, where the command was defined at first; the assigned ordinal; a letter indicating, whether the command is mandatory (M), optional (O), deprecated (D), or removed (X); the source file, where the command is implemented; and some space for additional notes. These notes include the result of the performed compliance tests and the percentage of work to do, if the command is not yet or only partially implemented.

1. Admin Startup and State (pp. 5–10)

TPM_Init	1.1	151	M	`tpm_startup.c`	
TPM_Startup	1.1	153	M	`tpm_startup.c`	
TPM_SaveState	1.1	152	M	`tpm_startup.c`	

2. Admin Testing (pp. 11–14)

TPM_SelfTestFull	1.1	80	M	tpm_testing.c	passed [14,13]
TPM_ContinueSelfTest	1.1	83	M	tpm_testing.c	
TPM_GetTestResult	1.1	84	M	tpm_testing.c	passed [14,13]

3. Admin Opt-in (pp. 15–22)

TPM_SetOwnerInstall	1.1	113	M	tpm_owner.c	
TPM_OwnerSetDisable	1.1	110	M	tpm_owner.c	
TPM_PhysicalEnable	1.1	111	M	tpm_owner.c	
TPM_PhysicalDisable	1.1	112	M	tpm_owner.c	
TPM_PhysicalSetDeactivated	1.1	114	M	tpm_owner.c	
TPM_SetTempDeactivated	1.1	115	M	tpm_owner.c	
TPM_SetOperatorAuth	1.2	116	M	tpm_owner.c	

4. Admin Ownership (pp. 23–36)

TPM_TakeOwnership	1.1	13	M	tpm_owner.c	passed [14,13]
TPM_OwnerClear	1.1	91	M	tpm_owner.c	
TPM_ForceClear	1.1	93	M	tpm_owner.c	
TPM_DisableOwnerClear	1.1	92	M	tpm_owner.c	
TPM_DisableForceClear	1.1	94	M	tpm_owner.c	
TSC_PhysicalPresence	1.1		M	tpm_owner.c	
TSC_ResetEstablishmentBit	1.2		M	tpm_owner.c	

5. Capability Commands (pp. 37–42)

TPM_GetCapability	1.1	101	M	tpm_capability.c	TODO: 45 %
TPM_SetCapability	1.2	63	M	tpm_capability.c	
TPM_GetCapabilityOwner	1.1	102	M	tpm_capability.c	

6. Auditing (pp. 43–51)

TPM_GetAuditDigest	1.2	133	O	tpm_audit.c	failed [14]
TPM_GetAuditDigestSigned	1.2	134	O	tpm_audit.c	
TPM_SetOrdinalAuditStatus	1.1	141	O	tpm_audit.c	

7. Administrative Functions – Management (pp. 52–57)

TPM_FieldUpgrade	1.1	170	O	tpm_management.c	
TPM_SetRedirection	1.1	154	O	tpm_management.c	
TPM_ResetLockValue	1.2	64	M	tpm_management.c	

8. Storage Functions (pp. 58–81)

TPM_Seal	1.1	23	M	tpm_storage.c	passed [14,13,7]
TPM_Unseal	1.1	24	M	tpm_storage.c	passed [13]
TPM_UnBind	1.1	30	M	tpm_storage.c	passed [14,13,7]
TPM_CreateWrapKey	1.1	31	M	tpm_storage.c	passed [14,13,7]
TPM_LoadKey2	1.2	65	M	tpm_storage.c	emulated by 32
TPM_GetPubKey	1.1	33	M	tpm_storage.c	passed [14]
TPM_Sealx	1.2	61	O	tpm_storage.c	passed [14]

9. Migration (pp. 82–108)

TPM_CreateMigrationBlob	1.1	40	M	tpm_migration.c	
TPM_ConvertMigrationBlob	1.1	42	M	tpm_migration.c	
TPM_AuthorizeMigrationKey	1.1	43	M	tpm_migration.c	
TPM_MigrateKey	1.2	37	M	tpm_migration.c	
TPM_CMK_SetRestrictions	1.2	28	M	tpm_migration.c	

TPM_CMK_ApproveMA	1.2	29	M	`tpm_migration.c`	
TPM_CMK_CreateKey	1.2	19	M	`tpm_migration.c`	
TPM_CMK_CreateTicket	1.2	18	M	`tpm_migration.c`	
TPM_CMK_CreateBlob	1.2	27	M	`tpm_migration.c`	
TPM_CMK_ConvertMigration	1.2	36	M	`tpm_migration.c`	

10. Maintenance Functions (optional, pp. 109–118)

TPM_CreateMaintenanceArchive	1.1	44	O	`tpm_maintenance.c`	
TPM_LoadMaintenanceArchive	1.1	45	O	`tpm_maintenance.c`	
TPM_KillMaintenanceFeature	1.1	46	O	`tpm_maintenance.c`	
TPM_LoadManuMaintPub	1.1	47	O	`tpm_maintenance.c`	
TPM_ReadManuMaintPub	1.1	48	O	`tpm_maintenance.c`	

11. Cryptographic Functions (pp. 119–136)

TPM_SHA1Start	1.1	160	M	`tpm_crypto.c`	passed [14,13]
TPM_SHA1Update	1.1	161	M	`tpm_crypto.c`	passed [14,13]
TPM_SHA1Complete	1.1	162	M	`tpm_crypto.c`	passed [14,13]
TPM_SHA1CompleteExtend	1.1	163	M	`tpm_crypto.c`	passed [14]
TPM_Sign	1.1	60	M	`tpm_crypto.c`	passed [14,13,7]
TPM_GetRandom	1.1	70	M	`tpm_crypto.c`	passed [14,13,7]
TPM_StirRandom	1.1	71	M	`tpm_crypto.c`	passed [14,13]
TPM_CertifyKey	1.1	50	M	`tpm_crypto.c`	passed [14,13]
TPM_CertifyKey2	1.2	51	M	`tpm_crypto.c`	

12. Endorsement Key Handling (pp. 137–145)

TPM_CreateEndorsementKeyPair	1.1	120	M	`tpm_credentials.c`	disabled
TPM_CreateRevocableEK	1.2	127	O	`tpm_credentials.c`	
TPM_RevokeTrust	1.2	128	O	`tpm_credentials.c`	
TPM_ReadPubek	1.1	124	M	`tpm_credentials.c`	passed [14,13]
TPM_OwnerReadInternalPub	1.2	129	M	`tpm_credentials.c`	

13. Identity Creation and Activation (pp. 146–152)

TPM_MakeIdentity	1.1	121	M	`tpm_identity.c`	passed [14,13]
TPM_ActivateIdentity	1.1	122	M	`tpm_identity.c`	passed [14,13]

14. Integrity Collection and Reporting (pp. 153–163)

TPM_Extend	1.1	20	M	`tpm_integrity.c`	passed [14,13]
TPM_PCRRead	1.1	21	M	`tpm_integrity.c`	passed [14,13,7]
TPM_Quote	1.1	22	M	`tpm_integrity.c`	passed [14,13]
TPM_PCR_Reset	1.2	200	M	`tpm_integrity.c`	not supported [13]
TPM_Quote2	1.2	62	M	`tpm_integrity.c`	passed [14]

15. Changing AuthData (pp. 164–168)

TPM_ChangeAuth	1.1	12	M	`tpm_authorization.c`	passed [14]
TPM_ChangeAuthOwner	1.1	16	M	`tpm_authorization.c`	

16. Authorization Sessions (pp. 169–181)

TPM_OIAP	1.1	10	M	`tpm_authorization.c`	not random [7]
TPM_OSAP	1.1	11	M	`tpm_authorization.c`	not random [7]
TPM_DSAP	1.2	17	M	`tpm_authorization.c`	
TPM_SetOwnerPointer	1.2	117	M	`tpm_authorization.c`	disabled

17. Delegation Commands (pp. 182–201)

TPM_Delegate_Manage	1.2	210	M	tpm_delegation.c	
TPM_Delegate_CreateKeyDelegation	1.2	212	M	tpm_delegation.c	
TPM_Delegate_CreateOwnerDelegation	1.2	213	M	tpm_delegation.c	
TPM_Delegate_LoadOwnerDelegation	1.2	216	M	tpm_delegation.c	
TPM_Delegate_ReadTable	1.2	219	M	tpm_delegation.c	
TPM_Delegate_UpdateVerification	1.2	209	M	tpm_delegation.c	
TPM_Delegate_VerifyDelegation	1.2	214	M	tpm_delegation.c	

18. Non-volatile Storage (pp. 202–215)

TPM_NV_DefineSpace	1.2	204	M	tpm_nv_storage.c	
TPM_NV_WriteValue	1.2	205	M	tpm_nv_storage.c	
TPM_NV_WriteValueAuth	1.2	206	M	tpm_nv_storage.c	
TPM_NV_ReadValue	1.2	207	M	tpm_nv_storage.c	
TPM_NV_ReadValueAuth	1.2	208	M	tpm_nv_storage.c	

19. Session Management (pp. 216–223)

TPM_KeyControlOwner	1.2	35	M	tpm_context.c	failed [14]
TPM_SaveContext	1.2	184	M	tpm_context.c	
TPM_LoadContext	1.2	185	M	tpm_context.c	

20. Eviction (pp. 224–226)

TPM_FlushSpecific	1.2	186	M	tpm_eviction.c	TODO: 75 %

21. Timing Ticks (pp. 227–230)

TPM_GetTicks	1.2	241	M	tpm_ticks.c	failed [14,13]
TPM_TickStampBlob	1.2	242	M	tpm_ticks.c	failed [14,13]

22. Transport Sessions (pp. 231–244)

TPM_EstablishTransport	1.2	230	M	tpm_transport.c	failed [14]
TPM_ExecuteTransport	1.2	231	M	tpm_transport.c	failed [14]
TPM_ReleaseTransportSigned	1.2	232	M	tpm_transport.c	failed [14]

23. Monotonic Counter (pp. 245–254)

TPM_CreateCounter	1.2	220	M	tpm_counter.c	passed [7]
TPM_IncrementCounter	1.2	221	M	tpm_counter.c	passed [7]
TPM_ReadCounter	1.2	222	M	tpm_counter.c	passed [7], failed [14]
TPM_ReleaseCounter	1.2	223	M	tpm_counter.c	passed [7]
TPM_ReleaseCounterOwner	1.2	224	M	tpm_counter.c	

24. DAA Commands (pp. 255–282)

TPM_Join	1.2	41	M	tpm_daa.c	passed [15]
TPM_Sign	1.2	49	M	tpm_daa.c	passed [15]

25. Deprecated Commands (pp. 283–309)

TPM_EvictKey	1.1	34	D	tpm_deprecated.c	
TPM_Terminate_Handle	1.1	150	D	tpm_deprecated.c	
TPM_SaveKeyContext	1.1	180	D	tpm_deprecated.c	
TPM_LoadKeyContext	1.1	181	D	tpm_deprecated.c	
TPM_SaveAuthContext	1.1	182	D	tpm_deprecated.c	
TPM_LoadAuthContext	1.1	183	D	tpm_deprecated.c	
TPM_DirWriteAuth	1.1	25	D	tpm_deprecated.c	passed [14,13]
TPM_DirRead	1.1	26	D	tpm_deprecated.c	passed [14,13]
TPM_ChangeAuthAsymStart	1.1	14	D	tpm_deprecated.c	

TPM_ChangeAuthAsymFinish	1.1	15	D	`tpm_deprecated.c`	
TPM_Reset	1.1	90	D	`tpm_deprecated.c`	
TPM_OwnerReadPubek	1.1	125	D	`tpm_deprecated.c`	
TPM_DisablePubekRead	1.1	126	?	`tpm_credentials.c`	spec error?
TPM_LoadKey	1.1b	32	D	`tpm_storage.c`	passed [14,13]

26. Deleted Commands (pp. 310–314)

TPM_GetCapabilitySigned	1.1	100	X		
TPM_GetOrdinalAuditStatus	1.1	140	X		
TPM_CertifySelfTest	1.1	82	X	`tpm_deprecated.c`	bad ordinal [14]
TPM_GetAuditEventSigned	1.1	130	X		
TPM_GetAuditEventSigned	1.1	131	X		

Towards Trust Services for Language-Based Virtual Machines for Grid Computing

Tobias Vejda, Ronald Toegl, Martin Pirker, and Thomas Winkler

Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria
{tvejda,rtoegl,mpirker}@iaik.tugraz.at
{thomas.winkler}@uni-klu.ac.at

Abstract. The concept of Trusted Computing (TC) promises a new approach to improve the security of computer systems. The core functionality, based on a hardware component known as Trusted Platform Module (TPM), is integrated into commonly available hardware. Still, only limited software support exists, especially in the context of grid computing. This paper discusses why platform independent virtual machines (VM) with their inherent security features are an ideal environment for trusted applications and services. Based on different TC architectures building a chain-of-trust, a VM can be executed in a secure way. This chain-of-trust can be extended at run-time by considering the identity of the application code and by deriving attestable properties from the VMs configuration. An interface to provide applications with TC services like sealing or remote attestation regardless of the underlying host architecture is discussed.

1 Introduction

Grid computing promises to provide massive computational power by distributing the workload over a large pool of independent systems. Virtual Machines (VM) allow one to overcome many of the complexities and security issues involved and are a well-suited basis. Still, open security issues exist. We use this context as example to show how these can be mitigated with Trusted Computing (TC).

Grid computing has emerged as new field in the area of distributed computing. "The Grid" provides flexible, secure and coordinated access to shared resources among dynamically changing virtual organizations [12]. Much like the power grid, it aims to make computational power at the level of supercomputing an ubiquitous commodity resource which is available at low costs. Different projects use grid computing to tackle complex problems like climate prediction[1] or the search for collisions in the SHA-1 hash function[2]. Not only large organizations provide resources, but also individual users may donate unused CPU cycles, bandwidth or memory.

[1] http://www.climateprediction.net/
[2] http://www.iaik.tugraz.at/research/krypto/collision/index.php

Such a heterogenous environment is ideally suited for virtual machine environments [13] with middleware based on Java commercially available. Java is an object-oriented, type-safe software environment with built-in security mechanisms such as access control, signed executable code and support for cryptography. Together with the security mechanisms, the intermediate byte-code representation of Java programs provides a “natural” isolation to other applications. It is portable per-se and also allows easy deployment. Building on the original notion of a sandbox, code can be executed with a defined set of access rights to system resources with flexible *stack inspection* mechanisms. Just-In-Time compilation creates native code for computational intensive program hot spots at runtime, thus mitigating the performance disadvantages of bytecode interpretation.

Grid Computing as a Use Case for Trust Services. The general operation model we consider as possible use case in this paper is as follows. A grid user identifies a problem which requires a large computational effort to solve. With the help of standard conformant middleware packages and libraries, she creates a software package. It is then automatically deployed to the participants on the grid. Work data is partitioned and distributed via secure channels to the remote systems, which are composed of a diverse selection of architectures. Work is processed, the results returned and assembled.

Of course, security in such a distributed system, where application code and data is sent to remote systems that are under control of another administrative domain, is critical.

In a current Grid system, the user, respectively the middleware she chooses to use, is required to trust the participants. The following security risks, as identified in [15], arise from this.

- I Participants may deliberately report back wrong answers.
- II Participants may not enforce data integrity or not compute accurately.
- III Data protection requirements of sensitive data cannot be enforced on remote systems.
- IV Theft of intellectual property on the distributed code by the participants is possible.
- V Theft of confidential data by the participants is possible.

1.1 Related Work

A general study on the applicability of TC for grid computing, including use cases, is given in [15]. In [22], integration of VMs with grid computing is categorized and discussed how the policy and sandboxing features of an assumed-trustworthy Java Virtual Machine (JVM) can be used. TC is only suggested as an approach to integrate legacy systems in future grids. In the so-called Trusted Grid Architecture [20] participants integrate grid functionality in a hardware-supported virtualization scheme. Native code grid jobs are run in compartments[3] separated from a legacy OS. A protocol is designed for the attestation of multilateral security. Likewise, in the Daonity [23] project, native virtualization and TC are applied to Grid middleware. Platform independence or the issues that arise when realizing trust and measurements within virtual machines are not considered in these works.

Apart from work in grid computing, several approaches in the area of attestation are relevant to this work. The concept of *Property-Based Attestation* (PBA) [5] provides an alternative to the attestation mechanisms specified by the TCG henceforth called *binary attestation*. A Trusted Third Party (TTP) translates the actual system configuration into a set of properties and issues certificates for those properties. As the certification is done externally, that approach is called *delegation*. Chen et al. [19] have proposed a protocol for property-based attestation, also following the delegation approach. They identify two additional categories to determine properties which are *code control* and *code analysis*. Code control infers regulations on the target machine, e.g. using SELinux as a reference monitor [30]. The attestation includes SELinux and its security policy. An example for code analysis is *Semantic Remote Attestation* (SRA) [6]. SRA utilizes language-based techniques to attest high level properties of an application running in a JVM. The approach is hybrid as it uses binary attestation to attest to the *Trusted Computing Base* (TCB) below the JVM. However, detailed knowledge of the application and eventual protocols is needed to extract high-level properties. A generalized approach based on a language-based VM has not been proposed yet.

1.2 Contribution

With the contributions of this paper we present novel Trust Services for language-based Virtual machines. We show how the afore-mentioned risks in the context of Grid computing can be mitigated as an exemplary use-case. We present a JVM with integrated services for TC based on a Trusted Computing Platform (TCP). Those services allow to extend the chain-of-trust into a language-based VM environment and their transparent usage in a remote attestation scenario. To complete the integration of TC into Java, we present the outline of an API allowing applications to use functionality such as key-management and provide security for application data. We believe this separation of concerns, low-level security services and a high-level API for applications, provides the necessary flexibility to tackle security problems arising in complex architectures, such as found in grid-computing. As our approach is based on the Java programming language, we maintain platform independence.

The JVM services presented in this work include support for property-based attestation. We see sandboxing as a tool to achieve security guarantees for grid

[3] Note that such hardware-emulating compartments are often referred to as Virtual Machine (VM). In this paper we use the term for intermediate code interpreters like the Java VM exclusively.

computing and propose to extract properties from the security policy of the JVM. This allows us to provide a flexible basis for a framework for generalized attestation and sealing. We further outline a straightforward way to integrate those properties into existing proposals of protocols for property-based attestation.

1.3 Outline of the Paper

The remainder of this paper is organized as follows. Approaches to provide a trustworthy execution environment for the Java bytecode interpreter are discussed in Section 2. In Section 3 we present the extension of the so created *chain-of-trust* into the JVM. Section 4 explains how hardware supported TC functionalities can be provided to platform independent applications. The paper concludes in Section 5.

2 Trustworthy Execution of Virtual Machines

Security enforced by software can be manipulated by software based attacks. To overcome this dilemma, the Trusted Computing Group (TCG) [10] has defined the TPM, a separate chip affixed to the mainboard of the computing platform. Similar to a smartcard, the device provides cryptographic primitives such as a random number generator, asymmetric key functionality and hash functions. A non-volatile and tamper-proof storage area keeps data over reboots. Access to the TPM is controlled through authorization of the commands and input data using a secret known to the *owner* of the TPM. The chip itself is a passive device providing services to software running on the main CPU, but does not actively take control or measurements of the host system.

Instead, the TPM allows to record measurements of the platform state in distinct Platform Configuration Registers (PCR). It 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

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

At a later time the TPM can provide a compelling proof by *attesting* the received measurements in the form of *quoting* a set of PCR contents, cryptographically signed with a private Attestation Identity Key (AIK) key bound to the TPM. This integrity report allows a remote party to form a qualified trust decision of the platforms state. Data may also be bound to PCRs, in a process called *sealing*, in which decryption is only possible in a single valid system state.

At current level of technology, it is impossible to provide an full security assessment for open architectures like todays personal computers. Thus, the TC concept does not claim to enforce perfect security under all conditions and tasks. The TCG defines a trustworthy system as *a system that behaves in the expected manner for the intended purpose.*

A system may provide a proof report of its state by applying the *transitive trust* model. Starting from booting the hardware with the BIOS acting as the root of trust each successive software block is measured into the TPM before it is executed, leading to a *chain-of-trust*. There are variations of this concept. The basic TCG model envisions a *static* chain-of-trust from hardware reboot onwards. Newer developments in CPUs and chipsets provide the option of a *dynamic* switch to a trusted system state: A special CPU instruction switches the system into a defined security state and then runs a measured piece of software which assumes full system control. Close hard-wired cooperation of CPU, chipset and TPM guarantees that the result is accurate. Using a hypervisor with support for hardware enforced virtualization, for instance Xen[4], allows execution of trusted and untrusted code in dedicated isolated compartments.

An intuitive assessment of a systems security is its size: The larger a system is the more likely it contains security relevant problems. Additionally, measuring of code takes time, especially when using the TPM itself with its low-performance communication link and hash implementation. Thus it is desireable to minimize the Trusted Computing Base, the critical parts of a system where every small bug can compromise global system security. Using a JVM as trust basis, the question of possible size reduction of the OS layer below arises. This issue has been addressed by Anderson et al. [31] who created a library OS for the Xen hypervisor.

With a choice of the mechanisms above, depending on the actual architecture, the JVM can be executed in a trustworthy way, providing a platform-independent TCB.

3 Trusted Computing Services for the JVM

We now address issues related to the JVM itself, starting with the basic security services provided by a language-based VM environment.

Java provides security mechanisms based on a security policy that enforces access control through stack inspection [24,21]. That is, the decision whether a method is allowed to access a resource is based on the entire call stack. The algorithm searches each stack frame starting from top to bottom (hence starting from the most recently called method) and tests whether the method has the appropriate *permissions* to access the resource. Each stack frame is mapped to a *protection domain* through the class the method belongs to. Each protection domain has assigned a set of permissions[5] which is checked through a *security manager* interface.

The security manager is held by the `System` class. A program fragment that needs access to a security relevant resource has to query the *System* class, check whether the security manager is actually instantiated (i.e. whether the return value is not `null`), and then query the security manager interface for access rights on the resource. The access control model is configured in so-called *policy files* at the granularity of code-bases (i.e. locations on the class path).

[4] `http://www.cl.cam.ac.uk/research/srg/netos/xen/`

[5] The full set of permissions available in the OpenJDK is documented in http://java.sun.com/javase/6/docs/technotes/guides/security/permissions.html

3.1 Code Identity

Code identity, based on binary measurement, is the central metric for remote authentication as defined by the TCG. The identity of the code is obtained through applying the *extend* command to binary executables. For a Java environment this would be class files, and as a special case, JAR files. A JAR file is the typical unit of deployment for an application or a library and contains a collection of classes and associated meta-data.

At run-time, the JVM searches for classes on the so-called *class path* and dynamically loads them into the environment. As Java aims at a networked environment, class path entries may include remote locations as well. Hence, the code that is running on a JVM is generally not known on beforehand. We implemented a measurement architecture for the JVM which performs measurement of class files (resp. JAR files) and places the hashes in a PCR. To this end, we integrated measurement hooks into the class loading process. This allows us to extend the chain-of-trust into the JVM and gives the possibility to integrate further TCG concepts into the Java environment. In the context of Grid computing, this allows the user to verify that the code executed by the participant is actually identical to the distributed package, thus dealing with security risks I and II.

The JVM allows application designers to swap out certain operations to native code, i.e. to use native libraries from Java programs. This feature is intended for low-level access to the hardware and for computationally intensive algorithms. An example is the `System.arraycopy` operation from the runtime library which performs better than any Java equivalent in terms of performance. As such a library directly accesses VM memory, it is a potential security problem for the measurement architecture. We deal with this problem through measuring the code of the native library.

3.2 Trust Properties from the VM Configuration

The binary attestation approach foresees that configuration files of applications are measured and, to allow the verifier determine the security of the configuration, are included in the Stored Measurement Log (SML). As discussed before, the configuration of the access control mechanism is done in a *java policy* file. On a remote verifier side, this policy file has to be evaluated and checked whether it conforms to a given security policy, i.e. whether it matches the verifier's needs. As the policy files are not of a precisely defined format, identical configurations may hash to different PCR values. Hence, there needs to be a non-ambiguous mean to determine the *properties* of a specific configuration.

A work by Sheehy et al. [17] defines delegation as a requirement to ensure the trustworthiness of a generalized attestation mechanism. We address this issue in the following way: Determining the security of the configuration of the JVM can be delegated to a TTP through binary attestation. However, to obtain properties of that configuration, the TTP needs to translate the information as well. The proposed approach can also be used at a TTP to translate the configuration into a set of properties.

Table 1. Trust properties for a JVM security policy for use in grid computing

Trust Property	Semantics	Java Permissions
Runtime	Status of run-time security features	`createSecurityManager`, `setSecurityManager`, `modifyThread`, `modifyThreadGroup`, `stopThread`, `createClassLoader`, `getClassLoader`, `setContextClassLoader`, `enableContextClassLoaderOverride`, `suppressAccessChecks`, any security permission
Native code	Ability to use native libraries	`loadLibrary`
File access	Ability to access the file system	any file permission
Network access	Ability to access the network	any network/socket permission

The Java security model permissions related to the specific scenario of Grid computing demand evaluation. To this end, we group certain permissions of the security policy to obtain a set of *trust properties*. The list of properties we propose is shown in Table 1. Note that this approach is an "all-or-nothing" approach. For instance, the property *file access* tells an attestor whether any code is able to access the file system. Further restrictions apply for the other properties. We have introduced this abstraction of the configuration to reduce the complexity of the access control model of Java as a first step towards determining properties.

We map permissions related to the general run-time security features to a single trust property *runtime*. An attestor has the possibility to determine whether the security manager is activated, and whether an application has the ability to change its implementation, the installed security policy, and class loaders. Furthermore, the settings of security permissions are reflected.

The full set of properties allows us to deal with several issues:

– Sensitive Data: If a specific JVM instance has the property of *file access* set to `false`, data cannot be written to disk, and hence, not be gathered by other, malicious entities. We, of course, assume that the network channel itself is confidential which is a property as well.
– Native Code: in special cases hardware optimization of the computing algorithm might be necessary. In addition to the bytecode, binaries can be distributed for invocation via JNI. As we are able to measure these files as well before executing them, this does not reduce the overall security. Grid applications using native code need the respective property to be `true`.

The proposed properties of the security policy are examples which match the needs of grid computing and allow us to enforce a behavioral policy to handle risk III. In [18], a property P is defined as a bit-string s of length l. As we chose to map permissions to properties that can only be `true` or `false`, l of such properties can be integrated into a single bit-string of respective length. This concept can be extended by using several bits for a single property to allow for more flexibility in defining s. Thus, the complex Java policy file can be reduced to s which can easily be extended to a PCR and thus integrated in attestation protocols, such as a binary attestation protocol or a property-based attestation protocol as proposed in [19].

3.3 Trusted Class Sealing

The distribution of software packages, as on the grid, might endanger intellectual property of the user, for instance, when the computational algorithm itself is of value. Current middleware [14] relies on bytecode obfuscation which makes recovery of the original source code only more difficult but not impossible. An other naïve approach, to encrypt class files and decrypt in a special class loader is easy to attack [26] if the current system integrity is not considered. Thus, to handle security risk IV we propose the concept of *trusted class sealing.*

This should not be confused with the following legacy concepts: Firstly, package sealing, which is a mechanism to control object inheritance [28]. Secondly, object sealing [29], allows one to encrypt all serializable objects, but without hardware protection of the keys and not transparently to the application. Instead, the user seals the sensitive classes to a trusted state of the target environement. Unsealing a class can be done through decoding the byte-stream in the class loading process. For performance reasons, only individual, security sensitive, classes should be sealed.

4 Application Interfaces to TC Mechanisms

The trustworthiness of a system does not end at the mechanics internal to the JVM, but extends into the application layer as well, especially to the maintenance and control middleware of a Grid. To receive sealed data and decrypt it, a participant requires both, high-level networking capabilities as provided by Java and access to TC services. The same holds true for remote attestation, where a networked protocol needs to maintain and transmit both the high-level SML and the PCR states from the TPM. It is a small step to allow also general purpose Java applications the access to TC services based on the TPM hardware device by means of defining an open Application Programming Interface (API).

There are two distinct approaches to integrate such an interface into Java. Firstly by *wrapping* an existing service of the native OS. A second approach is to create a complete *pure Java* TPM access software.

As the TPM is a device with limited resources a singleton software component has to manage it. Besides the VM, as detailed in Section 2, other software such as the OS require TPM services as well, possible already at boot time. Combining the need for an interface that handles both, resource management and concurrent access, makes it clear that this service cannot, in general, be implemented on top of the stack of running software alone, i.e. within a virtual machine.

Yet in case of a fully virtualized environment [25], where an exclusive environment for Java is provided in a compartment of its own and no concurrent accesses occur, the pure Java approach, as detailed in [16] or used by [2], will allow one to reduce the overall size and complexity. Such implementations may also be used if the resources of the TPM alone are virtualized as, for instance, with the TPM Base Services (TBS) [3] in Windows Vista. Still, for grid computing wrapping has the major advantage that it allows deployment on legacy operating systems as found today, which do not provide TPM virtualization. In

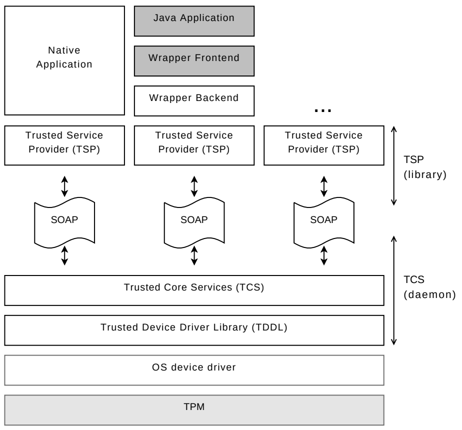

Native Application
Java Application
Wrapper Frontend
Wrapper Backend
...
Trusted Service Provider (TSP)
Trusted Service Provider (TSP)
Trusted Service Provider (TSP)
TSP (library)
SOAP
SOAP
SOAP
Trusted Core Services (TCS)
TCS (daemon)
Trusted Device Driver Library (TDDL)
OS device driver
TPM

Fig. 1. Overview of jTSS Wrapper layered architecture, allowing both integration in Java and concurrent access. Legend: Hardware is light gray, native code is white and Java code is dark grey.

the remainder of this section, we present the wrapper approach. An overview of the architecture is given in Figure 1.

The TCG Software Stack. The TCG not solely specifies TPM hardware, but also defines an accompanying layered software infrastructure called the TCG Software Stack (TSS) [8]. An interface in the C programming language allows applications to access TC functionality in a standardized way.

Generic TPM 1.2 drivers are integrated in recent OS releases, like Linux or Windows Vista. The lowest layer of the TSS, the *Trusted Device Driver Library (TDDL)*, abstracts the drivers to a platform independent interface that takes commands and returns responses as bytestreams. System tools provide basic functionality like resetting or taking ownership.

Since the hardware resources of the TPM are limited, the loading, eviction and swapping of keys and authorization sessions need to be managed. This is implemented in the *Trusted Core Services (TCS)*, which run as a singleton system service for a TPM. Additional functionalities are persistent storage of keys, TPM command generation and communication mechanisms. The TCS event manager handles the SML where PCR extend operations are tracked. To upper layers, a platform-independent Simple Object Access Protocol (SOAP) interface is provided. It is designed to handle multiple requests by ensuring proper synchronization.

System applications can access Trusted Computing functionality by using the *Trusted Service Provider (TSP)* interface. It provides a flat C-style interface. A `Context` object serves as entry point to all other functionality such as TPM specific commands, policy and key handling, data hashing, encryption and PCR composition. In addition, command authorization and validation is provided. Each application has their own instance of the TSP library, which is communicating via SOAP to the underlying TCS.

Java Bindings for the TSS. The jTSS Wrapper software package integrates the TSP service into Java. The Java Native Interface (JNI) is used to call the functions of the TSP. A thin *C Backend* integrates the TSP system library, in our case that of the open source TrouSerS implementation [4]. From it, the JNI link is in large parts autogenerated using the SWIG[6] tool, which creates a set of Java methods out of the C functions and structures. Building on this, the *Java Frontend* abstracts several aspects of the underlying library, such as memory management, the conversion of error codes to exceptions and data types. On top of it all an API is provided to developers. Being a slight abstraction of the original C interface it permits to stay close to the original logic and provides the complete feature set of the underlying TSS implementation.

With features like data sealing and unsealing available to the distributed software, risks like V can be handled and protocols for risks I-III implemented.

5 Conclusions and Outlook

In this paper we outline approaches to increase the security of JVM based computing using Trusted Computing concepts. A chain of trust can be built from boot-up to the execution of the JVM. We propose how property- based attestation can be realized scenarios by deriving abstract security properties from the security policy configuration of the JVM. We show how to create a foundation for remote attestation of grid participants to the user.

With the concept of TPM based sealing of classes we propose a solution of how TC can be applied to protect intellectual property of distributed code. To actually allow middleware as well as general purpose applications access to TPM features, such as data sealing, a software interface is presented. It integrates in todays environments by wrapping existing TC services of the host. This allows a Grid user to establish trust in the participants of a grid.

Our proof-of-concept implementation is based on Sun's OpenJDK and openly avaiable at [7]. It currently encompasses an implementation of binary measurement as outlined in Section 3.1. Our experiments with the measurement architecture show that measurement of single class files can significantly affect the performance of class loading if the number of class files of the application grows large. If JAR-files are measured on the other hand, this overhead can be reduced to a minimum. As JAR-files are a usual way to distribute Java applications, this approach is the most practical one.

Currently, we do not consider the Java notion of signed code, a feature orthogonal to TC. Furthermore, to reduce the complexity, we do not differentiate trustworthiness of code-bases, i.e. different locations of the classpath. We leave these issues open for future work.

The original TSS design strives to provide access to the bare functions of a TPM and introduces only a few high-level abstractions. It is specified in and following the conventions of C. As our Java API focusses on providing feature completeness, its behavior is nearly identical. Repeated passing of numerous

[6] `http://www.swig.org/`

parameters and constants in hardly varying call sequences, instead of objects with intuitive behavior are witnessed. It would be more beneficial, if an easier to use, well-abstracted and object-oriented API were available for Java. We are currently designing and standardizing such a future high-level API [27].

Acknowledgements. The authors thank anonymous reviewers for giving comments on an earlier version of this paper.

The efforts at IAIK to integrate TC technology into the Java programming language are part of the OpenTC project funded by the EU as part of FP-6, contract no. 027635. The project aims at providing an open source TC framework with a special focus on the Linux operating system platform. Started as an open source project the results can be inspected by everybody, thus adding towards the trustworthiness of Trusted Computing solutions.

References

1. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: Proceedings of the 13^{th} USENIX Security Symposium, pp. 223–238. USENIX Association (2004)
2. Sarmenta, L., Rhodes, J., Müller, T.: TPM/J Java-based API for the Trusted Platform Module (2007), `http://projects.csail.mit.edu/tc/tpmj/`
3. Microsoft Developer Network. TPM Base Services (2007), `http://msdn2.microsoft.com/en-us/library/aa446796.aspx`
4. TrouSerS - An Open-Source TCG Software Stack Implementation (2007), `http://trousers.sourceforge.net/`
5. Sadeghi, A.-R., Stüble, C.: Property-based Attestation for Computing Platforms: Caring about Policies, not Mechanisms. In: Proceedings of the New Security Paradigm Workshop (NSPW), pp. 67–77. ACM, New York (2004)
6. Haldar, V., Chandra, D., Franz, M.: Semantic Remote Attestation - Virtual Machine Directed Approach to Trusted Computing. In: Proceedings of the 3rd Virtual Machine Research and Technology Symposium, pp. 29–41. USENIX Association (2004)
7. Pirker, M., Winkler, T., Toegl, R., Vejda, T.: Trusted Computing for the Java^{TM}Platform (2007), `http://trustedjava.sourceforge.net/`
8. Trusted Computing Group. TCG Software Stack Specification, Version 1.2 Errata A (2007), `https://www.trustedcomputinggroup.org/specs/TSS/`
9. Trusted Computing Group. TCG Infrastructure Specifications (2007), `https://www.trustedcomputinggroup.org/specs/IWG`
10. Trusted Computing Group (2007), `https://www.trustedcomputinggroup.org`
11. Trusted Computing Group. TCG Specification Architecture Overview, Revision 1.4 (2007), `https://www.trustedcomputinggroup.org/groups/TCG_1_4_Architecture _Overview.pdf`
12. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. High Perform. Comput. Appl. 15(3), 200–222 (2001)
13. Getov, V., von Laszewski, G., Philippsen, M., Foster, I.: Multiparadigm communications in Java for grid computing. Communincations of the ACM 44(10), 118–125 (2001)
14. Parabon Computation, Inc. Frontier: The Premier Internet Computing Platform Whitepaper (2004), `http://www.parabon.com/users/internetComputingWhitePaper.pdf`

15. Mao, W., Jin, H., Martin, A.: Innovations for Grid Security from Trusted Computing (2005), `http://forge.gridforum.org/sf/go/doc8087`
16. Dietrich, K., Pirker, M., Vejda, T., Toegl, R., Winkler, T., Lipp, P.: A Practical Approach for Establishing Trust Relationships between Remote Platforms using Trusted Computing. In: Proceedings of the 2007 Symposium on Trustworthy Global Computing (in print, 2007)
17. Sheehy, J., Coker, G., Guttman, J., Loscocco, P., Herzog, A., Millen, J., Monk, L., Ramsdell, J., Sniffen, B.: Attestation: Evidence and Trust. Technical report 07 0186, MITRE Corporation (2007)
18. Kühn, U., Selhorst, M., Stüble, C.: Realizing Property-Based Attestation and Sealing with Commonly Available Hard- and Software. In: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing, pp. 50–57. ACM, New York (2007)
19. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.-R.: A Protocol for Property-Based Attestation. In: STC 2006: Proceedings of the first ACM workshop on Scalable trusted computing, pp. 7–16. ACM, New York (2006)
20. Loehr, H., Ramasamy, H., Sadeghi, A.-R., Schulz, S., Schunter, M., Stueble, C.: Enhancing Grid Security Using Trusted Virtualization. In: Xiao, B., Yang, L.T., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) ATC 2007. LNCS, vol. 4610, pp. 372–384. Springer, Heidelberg (2007)
21. Wallach, D., Felten, E.: Understanding Java Stack Inspection. In: Proceedings of the 1998 IEEE Symposium on Security and Privacy, pp. 52–63. IEEE, Los Alamitos (1998)
22. Smith, M., Friese, T., Engel, M., Freisleben, B.: Countering security threats in service-oriented on-demand grid computing using sandboxing and trusted computing techniques. J. Parallel Distrib. Comput. 66(9), 1189–1204 (2006)
23. Mao, W., Yan, F., Chen, C.: Daonity: grid security with behaviour conformity from trusted computing. In: Proceedings of the First ACM Workshop on Scalable Trusted Computing (STC 2006), pp. 43–46. ACM, New York (2006)
24. Gong, L., Mueller, M., Prafullchandra, H., Schemers, R.: Going beyond the sandbox: an overview of the new security architecture in the javaTM development Kit 1.2. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems, pp. 103–112. USENIX Association (1997)
25. Berger, S., Cáceres, R., Goldman, K., Perez, R., Sailer, R., van Doorn, L.: vTPM: Virtualizing the Trusted Platform Module. IBM Research Report, RC23879 (W0602-126) (2006)
26. Roubtsov, V. Cracking Java byte-code encryption, JavaWorld (2003), `http://www.javaworld.com/javaqa/2003-05/01-qa-0509-jcrypt_p.html`
27. Toegl, R., et al.: Trusted Computing API for Java, Java Specification Request 321, Java Community Process (2008), `http://www.jcp.org/en/jsr/detail?id=321`
28. Biberstein, M., Gil, J., Porat, S.: Sealing, Encapsulation, and Mutability. In: Proceedings of the 15th European Conference on Object-Oriented Programming, pp. 28–52. Springer, Heidelberg (2001)
29. Gong, L., Schemers, R.: Signing, Sealing, and Guarding Java Objects. In: Mobile Agents and Security, pp. 206–216. Springer, Heidelberg (1998)
30. Jaeger, T., Sailer, R., Shankar, U.: PRIMA: policy-reduced integrity measurement architecture. In: Proceedings of the eleventh ACM symposium on Access control models and technologies (SACMAT 2006), pp. 19–28. ACM, New York (2006)
31. Anderson, M.J., Moffie, M., Dalton, C.I.: Towards Trustworthy Virtualisation Environments: Xen Library OS Security Service Infrastructure. HP Research Report, HPL-2007-69 (2007)

Embedded Trusted Computing with Authenticated Non-volatile Memory*

Dries Schellekens[1], Pim Tuyls[1,2], and Bart Preneel[1]

[1] Katholieke Universiteit Leuven, ESAT-SCD/COSIC, Belgium
{dries.schellekens,bart.preneel}@esat.kuleuven.be
[2] Philips Research Laboratories, Eindhoven, The Netherlands
pim.tuyls@philips.com

Abstract. Trusted computing is an emerging technology to improve the trustworthiness of computing platforms. The Trusted Computing Group has proposed specifications for a Trusted Platform Module and a Mobile Trusted Module. One of the key problems when integrating these trusted modules into an embedded system-on-chip design, is the lack of on-chip multiple-time-programmable non-volatile memory. In this paper, we describe a solution to protect the trusted module's persistent state in external memory against non-invasive attacks. We introduce a minimal cryptographic protocol to achieve an authenticated channel between the trusted module and the external non-volatile memory. A MAC algorithm has to be added to the external memory to ensure authenticity. As a case study, we discuss trusted computing on reconfigurable hardware. In order to make our solution applicable to the low-end FPGA series which has no security measures on board, we present a solution that only relies on the reverse engineering complexity of the undocumented bitstream encoding and uses a physically unclonable function for one-time-programmable key storage. Clearly, this solution is also applicable to high-end series with special security measures on board. Our solution also supports field updates of the trusted module.

1 Introduction

Trusted computing is a promising technology to increase the security and trustworthiness of today's computing platforms. The main initiative for a new generation of computing platforms is taken by the Trusted Computing Group (TCG), a consortium of most major IT companies. The TCG defines the addition of a small hardware security module and a core initial software component, which act as *roots of trust* in the platform.

* The work described in this document has been partly financially supported by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the IBBT (Interdisciplinary institute for BroadBand Technology) of the Flemish Government, by the FWO project BBC G.0300.07, and in part by the European Commission through the IST Programme under Contract IST-027635 OPEN_TC.

The initial focus of the TCG was on the open personal computer platform, resulting in the specification of a Trusted Platform Module (TPM)[1]. Recently, the TCG Mobile Phone Work Group (MPWG) published the specification for a Mobile Trusted Module (MTM) and proposed a reference architecture. The specification distinguishes between local and remote owner trusted modules, defines a subset of the TPM commands that have to be implemented, and describes mobile specific commands, e.g., to implement secure boot in a standardized way [1].

Currently, TPMs are typically implemented as a *discrete* cryptographic chip physically bound to the rest of platform (i.e., soldered to the mainboard), but chipsets with *integrated* TPMs are forthcoming. For the MTM specification various implementation options exist, especially because a mobile phone will contain multiple trusted modules for different stakeholders (e.g., device manufacturer and cellular operator). The MTM can be implemented in hardware as a separate dedicated chip or integrated into existing chips [2], or as software running on the main processor, possibly in a higher privileged mode [3,4]. If the platform has to support multiple execution engines, software/virtual trusted modules can run in isolated domains provided by a microkernel or hypervisor [5,6,7].

The security of the discrete chip solution is compromised if the communication interface between the trusted module and the rest of the platform is vulnerable to attacks [8]. Additionally, the cost of an extra chip is considerable for low-end embedded devices. Hence, the integration of the trusted module into another chip is preferable. One of the main problems when embedding a security module into an existing chip or running this functionality on the application processor is the lack of on-chip *Multiple-Time-Programmable* (MTP) *Non-Volatile Memory* (NVM)[2]. It is needed by the trusted module to securely store its *persistent state*, which includes cryptographic keys, authorization data and *monotonic counters*. Although external non-volatile memory (e.g., Flash or EEPROM) is commonly available in embedded devices, it can not store the persistent state securely. An adversary can access the communication interface to the external memory with a non-invasive hardware attack and overwrite the state with an older version, even if the state is stored in an encrypted form.

Eisenbarth et al. [10] propose the first implementation of the trusted module on reconfigurable hardware that facilitates *field upgrades* of the trusted module. Upgrades might be required to fix non-compliance firmware bugs [11] or to replace broken/depreciated cryptographic algorithms. They propose furthermore a new FPGA architecture with functionalities similar to trusted computing. Their construction includes a minimal root of trust, called Bitstream Trust Engine (BTE). This trust component is used to include the hardware configuration in the TCG chain of trust, by measuring the configuration bitstream before it is loaded on the FPGA. On top of this, it limits access to the trusted module's persistent state based on the loaded bitstream. Moreover their solution

[1] All TCG specifications referred to in this document are available at `https://www.trustedcomputinggroup.org/specs/`

[2] MTP NVM solutions in standard CMOS logic, requiring no additional masks or processing steps, are becoming commercially available (e.g., [9]).

requires partial configuration, bitstream encryption and authentication, features typically only available on high-end FPGAs[3]. The BTE needs NVM inside the FPGA package to store the complete persistent state together with access control information that defines which bitstream has access to the state. In order to implement this solution, various changes have to be made to the FPGA hardware. History has shown that extensive changes to a hardware architecture are unlikely to occur and that solutions which do not need these changes are often preferred since they are more economically viable.

In this paper, we study embedded trusted computing on devices, which only have integrated *One-Time-Programmable* (OTP) NVM. We propose to store the trusted module's state in an external NVM chip that is extended with a security mechanism. In this way a mutual authenticated and fresh communication channel is realized between the trusted module and the external NVM. Our solution requires the addition of a MAC algorithm and some access control logic to the external NVM. We deem these changes as reasonable because some semiconductor companies already provide a similar security-enhanced NVM [12].

As a case study we look at volatile FPGAs. We propose to implement the required OTP key storage as an intrinsic *Physical Unclonable Function* (PUF). Intrinsic PUFs can be extracted from memory structures on the FPGA or laid down in logic [13]. Our solution only relies on the complexity of full *bitstream reversal* and can hence be realized with current low-end SRAM FPGA technology which does not have any additional built-in security mechanisms. Bitstream reversal is defined as the reverse engineering process that transforms an encoded bitstream into the original logical design, and it is generally believed to be a hard task that takes a lot of effort [14]. This difficulty is not exactly measurable and does not achieve nowadays cryptographic standards. The security of our proposal can be strengthened by bitstream protection techniques such as bitstream encryption.

This paper is organized as follows. Section 2 gives some background on classical bitstream protection techniques and PUFs. In Section 3 we describe the implementation of an integrated trusted module, propose a mechanism to protect the trusted module's persistent state and introduce an authenticated NVM scheme. Section 4 describes our solution for trusted computing on reconfigurable hardware that relies on an intrinsic PUF for key storage and allows field updates. The paper ends with a summary of our work and conclusions.

2 Volatile FPGA Security

Our main example for reconfigurable platforms will be SRAM based FPGAs. SRAM based FPGAs offer very large flexibility and form the bulk of the market. In this section we describe several protection technologies for SRAM based FPGAs.

[3] Xilinx bitstream encryption disables partial configuration, reconfiguration and readback to increase security. So the combination of partial configuration and bitstream encryption is currently unsupported.

2.1 Classical Bitstream Protection Techniques

There are two main classical mechanisms to protect FPGA bitstreams against cloning: bitstream encryption and node locking.

Bitstream encryption provides confidentiality of the bitstream and binds it to the platform. As the name suggests, the bitstream is encrypted with a key which has to be stored in non-volatile memory on the FPGA. When the bitstream is loaded onto the FPGA, it is first fed to a decryption module. The decrypted bitstream is then used to configure the FPGA. In order to implement this technology, non-volatile memory is required on the FPGA for storage of the long-term key. This is currently only provided for the high-end FPGAs and comes at a cost since it requires different memory hardware or a battery [14].

Node locking is a technology that binds the bitstream to a specific platform. The basic idea is that the bitstream is bound to the platform by an identifier that can not be modified by an attacker and that is stored either in an external chip (e.g., Dallas Secure EEPROM [15]) or internally as a device DNA[4] or a PUF. By checking the presence of the correct identifier, the bitstream will determine whether it runs on the expected platform or not. The security of these solutions depends amongst other things on the level of obfuscation provided by the bitstream encoding (see Section 2.3).

2.2 Physical Unclonable Function

We briefly introduce the notion of a Physical Unclonable Function and present some examples of PUFs that are present on ICs due to process variations. Their main advantage is that they are intrinsically present on an IC, i.e., not requiring changes to the hardware. For details we refer to [13,16].

Definition 1. *A PUF is a physical structure that consists of many random, uncontrollable components, that is therefore very hard to clone. The structure satisfies the following properties:*

(i) It is easy to challenge the PUF and measure the response of the PUF according to the challenge. The combination of a challenge C and a response R is called a challenge-response pair.
(ii) A PUF has many challenge-response pairs.
(iii) Given a challenge C, the response R of the PUF is unpredictable.
(iv) The PUF can not be reproduced by the manufacturer.
(v) The PUF should be tamper evident.
(vi) Preferably the PUF is inseparably bound to the object that it is protecting.

The main examples of PUFs that are intrinsically present in ICs in general and SRAM based FPGAs in particular, are SRAM PUFs and Silicon PUFs. SRAM PUFs were introduced in [13] and exploit the random behavior of uninitialized SRAM memory cells during startup. This behavior originates from doping variations in the transistors that make up those cells. Silicon PUFs on the other hand

[4] `http://www.xilinx.com/products/design_resources/security/devicedna.htm`

were introduced in [16] and are based on the delay variations in the circuits laid down in an IC.

In order to use a PUF for key storage, the following procedure has to be followed based on a fuzzy extractor or helper data algorithm [17,18]. First, during an enrollment phase the PUF responses $R_1, \ldots, R_n$ for the challenges $C_1, \ldots, C_n$ are measured. Then for a response R_i one generates a key K_i and helper data W_i using the function Gen of the fuzzy extractor. Later during the reconstruction phase, the noisy PUF response R'_i is measured again and by using the appropriate helper data W_i, the key K_i is reconstructed using the procedure Rep as defined in [18].

It was shown in [13] that PUFs can be used to provide secure key storage on SRAM FPGAs without additional cost or hardware. This implies that PUFs offer a low cost solution to the IP counterfeiting problem.

2.3 Bitstream Reversal

Drimer defines bitstream reversal as the reverse engineering process to transform an encoded bitstream file into a logical functional description (i.e., netlist or HDL) [14]. The encoding of bitstream formats is largely undocumented and obscure. This makes full bitstream reversal practically impossible. Partial bitstream reversal, which decodes static data from bitstreams, is far easier and a tool to do so exists[5].

Hiding cryptographic keys in look-up tables and RAMs should be avoided, because they can be easily extracted. Keys derived from a PUF response are better protected against partial bitstream reversal. In order to capture the secret key, an adversary needs to recover the exact design of the PUF (i.e., type, exact location, etc.) and produce a modified bitstream with the same PUF that outputs the secret key. It is reasonable to assume that currently this is very difficult because the adversary would have to perform a full bitstream reversal, and we will take this as the main assumption in this paper.

3 Embedded Trusted Computing

To transform an existing platform, be it open or closed, into a TCG enabled platform, two changes are typically required.

(i) A trusted module (i.e., TPM or MTM), that acts as the root of trust for storage and reporting, needs to be added to the platform.
(ii) The first code executed on the central processor of the platform has to be immutable in order to serve as root of trust for measurement (CRTM) and/or verification.

In this section we will focus on the integration of a trusted module into an embedded system-on-chip (SoC) design. We will propose a scheme to store the

[5] The Ulogic project (`http://www.uclogic.org`) aims at full netlist recovery from closed FPGA bitstream formats, but currently it is only able to decode static data.

trusted module's persistent state in external memory. Our solution relies on embedded one-time-programmable key storage and authenticated non-volatile memory.

3.1 Trusted Module

The TCG has specifications for two trusted modules: the Trusted Platform Module is primarily designed for the PC platform and the Mobile Trusted Module is more tailored for advanced mobile devices, like smartphones or PDAs. Both modules can be implemented with similar hardware, namely a microcontroller, a cryptographic coprocessor (RNG, RSA, SHA-1, HMAC), read-only memory for firmware and certificates, volatile memory and non-volatile memory. The main difference is the command set of the trusted modules, which in essence translates in a different firmware image. It is conceivable that for some embedded devices a more minimal trusted module is sufficient, but this would not fundamentally change the design of the trusted module.

The trusted module communicates with the central microprocessor of the platform over an I/O bus. The Low Pin Count bus is the standardized interface for PCs to communicate with a TPM, but for embedded systems one is free to use any system-on-chip bus (e.g., ARM's AMBA bus).

The trusted module needs volatile memory for temporary data. This includes key slots to load keys stored outside the trusted module, a number of Platform Configuration Registers that store measurements (i.e., hash values) made during startup of the platform, and information (e.g., nonces) about the concurrent authorization sessions.

3.2 Persistent State

Some information stored in the trusted module has to be stored persistently. We denote the persistent state of a trusted module with $\mathcal{T}$. For the TPM, it includes the Endorsement Key (EK) that uniquely identifies each TPM, the Storage Root Key (SRK) that encrypts other keys maintained by the TPM, the owner's authorization data (i.e., password), and the content of monotonic counters. The persistent state of a MTM contains similar data.

As a dictionary attack mitigation technique, the trusted module keeps track of the number of failed authorization attempts. This information should also be stored persistently. The TPM_SaveState command can be used to temporally save volatile state information (e.g., content of PCRs) in $\mathcal{T}$.

In order to sufficiently protect the persistent state $\mathcal{T}$, the following requirements have to be considered:

(i) **State confidentiality:** It should be infeasible to read the content of $\mathcal{T}$. Disclosure of $\mathcal{T}$ will for instance reveal the private SRK.
(ii) **State integrity:** Unauthorized modification of $\mathcal{T}$ should be infeasible, otherwise an adversary could for instance change the owner's password.

(iii) **State uniqueness:** Cloning of the trusted module must be infeasible. Hence, copying or extraction of the EK has to be prevented.
(iv) **State freshness:** Replay of an old state must be infeasible. This is mainly necessary to protect the monotonicity of counters.

The TCG specifications differ regarding the last requirement. The TPM has to store its general purpose monotonic counters in physically shielded locations, i.e., tamper-resistant or tamper-evident hardware. The mobile specific monotonic counters should only be shielded from software executing outside the context of the MTM[6]. This implies that for MTMs state freshness must not be guaranteed in case of hardware attacks.

3.3 Our State Protection Scheme

When building embedded trusted computing platforms, it is desirable to integrate the trusted module into the application processor. This is more cost effective than a discrete chip. In addition, the overall security is improved because the communication interface between the main microprocessor and trusted module is only accessible with an invasive attack. Most of the trusted module's components are rather straightforward to integrate in a system-on-chip design. However, special attention is required to realize secure storage of $\mathcal{T}$, because the application processor typically lacks reprogrammable non-volatile memory.

We propose a protection scheme that stores the persistent state $\mathcal{T}$ that does not require MTP NVM inside the trusted module. Figure 1 gives a schematic

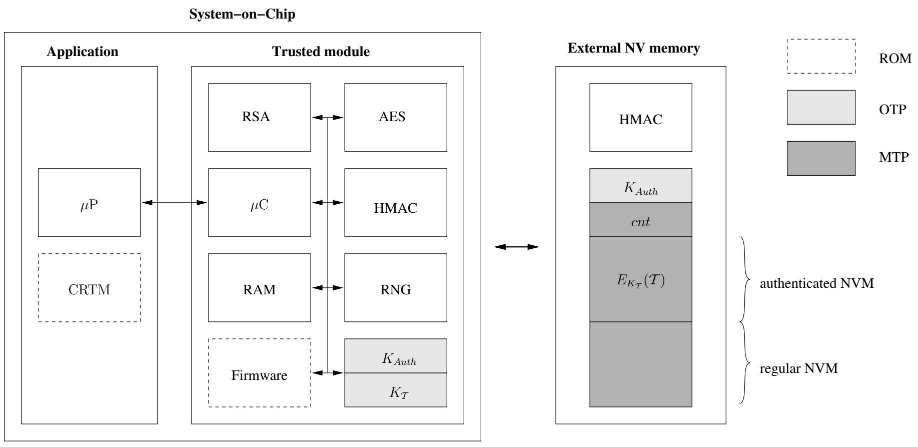


Fig. 1. Protection of the trusted module's persistent state $\mathcal{T}$ with authenticated non-volatile memory

[6] The TCG MPWG intends to tighten the security requirements to the level of the TPM specifications "immediately when it becomes feasible to implement such counters in mobile phones".

overview of our proposal. For efficiency reasons, our solution requires a symmetric key algorithm, more specifically AES, even though this is not part of the TCG specifications. This algorithm could also be used to protect the trusted module's key hierarchy, by making the SRK a symmetric key[7].

The state $\mathcal{T}$ is stored externally, but encrypted: $E_{K_\mathcal{T}}(\mathcal{T})$. The state encryption key $K_\mathcal{T}$ is device specific and has to be stored in one-time-programmable NVM embedded in the trusted module. It is best to use an optimized memory encryption scheme like [19].

To protect the integrity of $\mathcal{T}$ and achieve state freshness, we propose to store the encrypted state in authenticated non-volatile memory. The communication to this security-enhanced external memory is protected by a cryptographic challenge-response protocol that uses a shared secret key K_{Auth}. The trusted module uses random numbers as source of freshness in the protocol, while the external memory makes use of a monotonic counter c. Alternatively, the external NVM can use a random number generator to generate fresh nonces.

The external memory will store the key K_{Auth} and the counter c in its non-volatile memory. The OTP memory that is needed to store the keys $K_\mathcal{T}$ and K_{Auth} inside the trusted module, can be implemented with fuses or PUFs. In Section 4.1 we will describe the realization of secure key storage with an intrinsic PUF in detail.

State Initialization. First, the keys K_{Auth} and $K_\mathcal{T}$ have to be generated randomly and programmed in the trusted module and the external memory. In Section 3.4 we will explain how to program K_{Auth} in the external NVM and in Section 4.1 we will describe the process when a PUF is used for key storage inside the trusted module.

Next, the internal state of trusted module will be initialized. This includes the generation of the EK (in case of TPM and local owner MTM) and the SRK (in case of remote owner MTM). The device manufacturer can also create an endorsement certificate on the public EK or program the necessary information for secure boot.

3.4 Authenticated Non-volatile Memory

The encrypted state $E_{K_\mathcal{T}}(\mathcal{T})$ is stored in security-enhanced external NVM. This external memory exposes a regular memory interface and an authenticated one. Legacy read and write operations work on the regular memory range, but they will be ignored by the external NVM on the authenticated memory range. When accessing in an authenticated memory address, a cryptographic challenge-response protocol provides data origin authentication and assures a fresh communication (see Figure 2). The protocol uses a MAC algorithm keyed with a shared secret K_{Auth} and nonces. The key K_{Auth} is programmed in the trusted module and the external NVM during a pairing phase.

In order to implement the above protocol, the external memory contains the following components: a MAC algorithm, some control logic and internal NVM to

[7] This might not be allowed because of export restrictions.

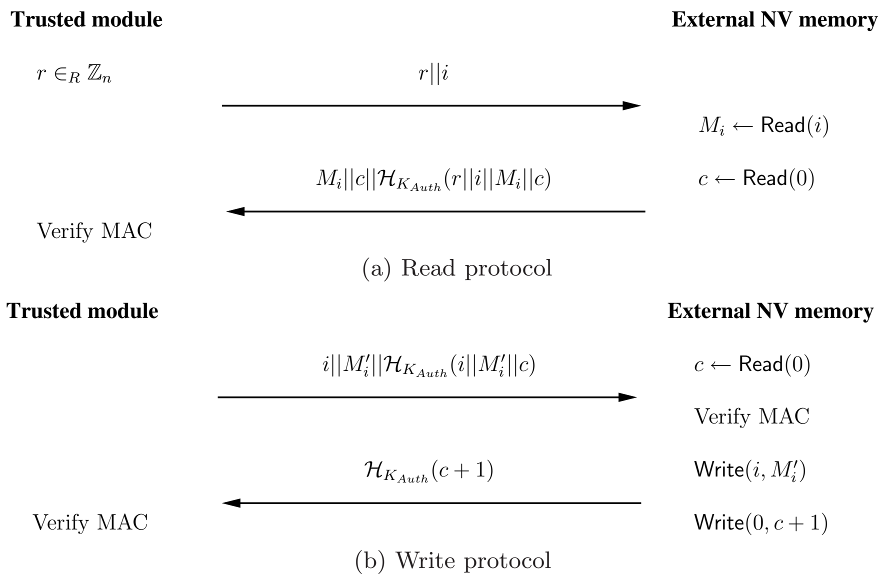

Trusted module
External NV memory
$r \in_R \mathbb{Z}_n$
$r||i$
$M_i \leftarrow \mathsf{Read}(i)$
$M_i||c||\mathcal{H}_{K_{Auth}}(r||i||M_i||c)$
$c \leftarrow \mathsf{Read}(0)$
Verify MAC
(a) Read protocol
Trusted module
External NV memory
$i||M_i'||\mathcal{H}_{K_{Auth}}(i||M_i'||c)$
$c \leftarrow \mathsf{Read}(0)$
Verify MAC
$\mathcal{H}_{K_{Auth}}(c+1)$
$\mathsf{Write}(i, M_i')$
Verify MAC
$\mathsf{Write}(0, c+1)$
(b) Write protocol

Fig. 2. Cryptographic protocols to access the authenticated memory at address i

store the key K_{Auth} and the monotonic counter c. We propose to use the HMAC algorithm since this is already present in a TCG compliant trusted module. Another option is a MAC algorithm based on a block cipher, because we already added a AES coprocessor to the trusted module.

We assume that the authentication key K_{Auth} is the same for the whole authenticated memory range. If more flexibility is needed, each memory address i could have its own access key $K_{Auth,i}$. For instance, Intel's authenticated Flash memory allows to specify multiple authenticated memory ranges and associates a different key with each range [12].

Read Protocol. In order to read securely from the external memory, the read command consists of the following steps (see Figure 2(a)).

1. The trusted module generates a nonce r.
2. The trusted module sends the address i together with r.
3. The external memory reads the memory at address i: M_i.
4. The external memory reads the counter c from the internal address 0.
5. The external memory returns M_i accompanied with c and a MAC on r, i, M_i and c.
6. The trusted module verifies the MAC. If the MAC is ok, it accepts the message M_i and performs the consequent operations. Otherwise the trusted module stops all operations.
7. The trusted module store c in volatile memory, because this acts as a challenge for a subsequent write operation.

Write Protocol. In order to write data securely to the external memory, the write command consists of the following steps (see Figure 2(b)).

1. The trusted module retrieves c from internal RAM (it got this value during the previous operation).
2. The trusted module sends the data M_i' that it wants to write at address i, accompanied with a MAC on i, M_i', and c.
3. The external memory reads the monotonic counter c from the internal address 0.
4. The external memory checks the MAC. If the MAC is ok, the trusted module used the same value for c.
5. The external memory stores the data M_i' address i and increments the monotonic counter by writing $c + 1$ to internal address 0.
6. The external memory returns a MAC on the new counter value (i.e., $c + 1$).
7. The trusted module verifies the MAC and determines whether the write operation was successful by verifying that the monotonic counter was incremented. If these checks fail, the trusted module stops all operations.
8. The trusted module increments the monotonic counter in volatile memory.

If c gets lost, the trusted module can retrieve the current value by performing the read operation.

Pairing Phase. A trusted entity generates the authentication key K_{Auth} and programs it in the trusted module and the external memory. When the different parts (trusted module, external memory) are produced by different parties, secure key distribution protocols have to be used to distribute the key safely to those parties. Typically, these protocols rely on symmetric cryptography and a key distribution center (KDC) or on asymmetric cryptography and a public key infrastructure (PKI). However, this is not the topic of this paper.

The external memory has to offer an interface that guarantees one-time-programmability of K_{Auth}. Various options to implement OTP Flash memory are described in [20].

3.5 Security Analysis

Conceptually our state protection scheme satisfy all the security requirements that are listed in Section 3.2. State confidentiality is provided by the symmetric encryption scheme, uniqueness by the device specific $K_{\mathcal{T}}$, and state integrity and freshness by the cryptographic protocol.

However, an adversary can also perform a denial-of-service attack, e.g., by blocking the communication to the external NVM. An attacker can also try to learn the secret keys with side channel analysis (SCA) or perform an invasive attack. When designing the cryptographic coprocessor of the trusted module and the external NVM, SCA countermeasures have to implemented.

To better protect against invasive attacks, it is advised to realize the key storage in both devices with a PUF and use an authenticated encryption scheme.

4 Reconfigurable Trusted Computing

As a case study, we look firstly at volatile reconfigurable hardware. More specifically we investigate how the trusted module's secret keys are stored securely. Since volatile FPGAs do not have non-volatile memory on board, we propose to extract the secret keys from an intrinsic PUF.

The fact that the system-on-chip design is stored externally as a configuration bitstream, enables field upgrades of the trusted module. This comes however with a second security challenge to allow for secure updates. We will also describe how to protect the design against partial bitstream reversal that exposes static data. Clearly a bitstream encryption mechanism offers strengthened security.

Table 1 summarizes all data that has to be stored in the external non-volatile memory to implement our trusted module.

Table 1. Content of external non-volatile memory

name	type	description
B_{SoC}	regular	FPGA bitstream containing SoC design
$W_{\mathcal{T}}$	regular	helper data to derive state encryption key $K_{\mathcal{T}}$
W_{Auth}	regular	helper data to derive link authentication key K_{Auth}
$E_{K_{\mathcal{T}}}(\mathcal{T})$	authenticated	encrypted persistent state $\mathcal{T}$
M_{ROM}	authenticated	trusted module firmware ROM and CRTM
PK_{ROM}	authenticated	public key to verify signed ROM updates

4.1 Realization of Secure Key Storage

The trusted module needs two cryptographic keys (i.e., $K_{\mathcal{T}}$ and K_{Auth}) to access $\mathcal{T}$. These keys have to remain confidential and need to be device specific. Some low-end FPGAs, like Xilinx Spartan-3A series, have a unique device identifier, but this identifier can be read by any bitstream. A first option is to derive the keys $K_{\mathcal{T}}$ and K_{Auth} from the identifier with a secret algorithm. We note however that device DNA is currently not yet present on all FPGA types, thus we will not pursue this option further.

Therefore, we propose to derive the keys $K_{\mathcal{T}}$ and K_{Auth} from an intrinsic PUF. As explained in Section 2.2, a delay PUF can be implemented with the CLBs of the FPGA. Alternatively to implement an SRAM PUF, the startup values of an uninitialized block RAM can be used. The keys will be extracted from two PUF responses $R_{\mathcal{T}}$ and R_{Auth} at the point in time when needed, using the Rep function of the fuzzy extractor and corresponding helper data $W_{\mathcal{T}}$ and W_{Auth} respectively. These helper data are stored in the external memory. The corresponding challenges, $C_{\mathcal{T}}$ and C_{Auth}, are embedded in the FPGA's bitstream or stored in regular external NVM memory. The PUF guarantees that given the challenges its responses are still unpredictable.

Enrollment Phase. In order to use a PUF to generate keys, an enrollment phase has to be carried out. During this phase the PUF is challenged for the first time with the challenges $C_{\mathcal{T}}$ and C_{Auth} and the responses $R_{\mathcal{T}}$ and R_{Auth} are measured. The Gen function of the fuzzy extractor is used to generate the keys $K_{\mathcal{T}}$ and K_{Auth} for the first time together with their accompanying helper data $W_{\mathcal{T}}$ and W_{Auth}. The helper data are then stored in the external memory. We note that this phase can be carried out during the pairing phase discussed in Section 3.4.

The system designer can choose to create a separate enrollment bitstream B_{PUF} that contains the same PUF as the bitstream B_{SoC} that will be deployed afterwards.

4.2 Protection of the Bitstream

The bitstream B_{SoC} contains the system-on-chip design and is stored in regular external NVM. The following security requirements should be considered:

(i) **Design integrity:** Unauthorized modification of the system-on-chip design must be impossible. More specifically the integrity of a number of components should be guaranteed: the trusted module, especially its firmware, the main processor, the CRTM code, and the communication interface between the trusted module and the CPU.
(ii) **Design confidentiality:** The design can contain cores whose intellectual property has to be protected. Additionally, the cryptographic keys that are embedded in the trusted module, must remain secret.
(iii) **Design freshness:** Reconfigurable hardware allows field upgrades of the design. It must not be possible to replay an older and insecure version of the design.

Bitstream Obfuscation. On low-end reconfigurable devices we can only rely on the reverse engineering complexity of the bitstream encoding for security purposes. According to the above mentioned literature, this gives a decent level of assurance that IP cores can not easily be reverse engineered and that directed modification of the logic is difficult.

An adversary can extract the challenge $C_{\mathcal{T}}$ and C_{Auth} from the bitstream, but due to the unpredictability of the PUF responses, this knowledge is insufficient to learn any information about the keys $K_{\mathcal{T}}$ and K_{auth}. In order to be successful, he must perform a full bitstream reversal and create a malicious design with exactly the same PUF as B_{SoC} that outputs the secret keys. According to the state of the art [14], this is infeasible at this point in time.

Embedded ROMs. If the CRTM code and the trusted module's firmware are embedded inside the bitstream B_{SoC}, partial bitstream reversal will possibly reveal the contents of these embedded ROMs and perhaps enable an adversary to create a bitstream with modified code.

The easiest way to overcome this problem is by storing the code, denoted with M_{ROM}, in authenticated non-volatile memory. The system-on-chip design needs to be extended with some extra logic that performs the cryptographic protocol to access the authenticated NVM. This logic has to have access K_{Auth}, which is stored with the intrinsic PUF. As a positive side effect, the trusted module does not have to use FPGA block RAMs as ROM.

Bitstream Encryption. On high-end FPGAs bitstream encryption can be used to obtain better design confidentiality. Additionally, it is difficult to make meaningful modifications to the design if the bitstream is encrypted. Typically the plain bitstream contains linear checks (i.e., CRC), so bit flips get detected. Ideally, the bitstream should be cryptographically authenticated as well, for instance with an additional MAC algorithm or by using an authenticated encryption scheme, but no commercial FPGA hardware supports this [14].

4.3 Field Updates

The TCG specifications define the TPM_FieldUpgrade command, but the implementation is free. We distinguish two type of field updates: firmware updates and bitstream updates. The trusted module's firmware is stored in authenticated NVM as M_{ROM} and the trusted module's hardware is stored separately in regular NVM as part of bitstream B_{SoC}.

Firmware Updates. Firmware updates are fairly straightforward as we can use the authenticated write operations of the external memory. The system designer has to store a public key PK_{ROM} inside the trusted module. We propose to store this key in authenticated NVM to protect its integrity. The trusted module has to offer an extra command that loads a signed firmware image $sign_{SK_{ROM}}(M'_{ROM})$. The authenticated memory scheme assures that only the trusted module can overwrite M_{ROM}.

In order to protect against rollback[8], a version number v_{ROM} needs to be included in the firmware ROM: $v_{ROM} \subset M_{ROM}$. The trusted module has to check whether the version of the new firmware is higher than its own version: $v'_{ROM} > v_{ROM}$.

Bitstream Updates. In some situations (e.g., to replace a cryptographic coprocessor), it might be necessary to update the FPGA bitstream B_{SoC}. It is crucial that the new bitstream includes exactly the same PUF. Otherwise, the secret keys K_{Auth} and $K_{\mathcal{T}}$ become inaccessible and consequently the trusted module can no longer write to the authenticated NVM.

Because B_{SoC} is stored in regular external NVM, it can always be overwritten by an older version. A possible solution to prevent this is to lock the bitstream B_{SoC} to the firmware M_{ROM}. Every bitstream update will then be accompanied by a firmware update. The binding mechanism has to assure that new firmware M'_{ROM} does not function correctly with the old bitstream B_{SoC}. For instance, some extra logic in B_{SoC} checks whether an expected identifier is present in M_{ROM} and halts the design if this is not the case. Like the node locking schemes described in Section 2.1, this solution relies on the difficulty of bitstream reversal.

[8] Version rollback could be desirable because updates can cause issues.

Trust Model. In our approach the system designer, creating the FPGA bitstreams and issuing firmware updates, has to be trusted.

5 Conclusion

In this paper we studied the integration of a trusted module into a system-on-chip design that lacks embedded reprogrammable non-volatile memory. We proposed a scheme to store the trusted module's persistent state externally in authenticated memory. Access to this external memory is protected by a minimal cryptographic challenge-response protocol that guarantees state integrity and freshness. Our solution can be made robust against most invasive attacks. This will be addressed in forthcoming work.

We also considered the implementation of trusted computing on reconfigurable hardware. In order to maximize the applicability of our solution we only relied on the complexity and obscurity of the bitstream format and used intrinsic physical unclonable functions for key storage. We also took into account field updates on the implementation.

Our main focus was on embedded trusted computing, but the components that we propose can be used in other security applications where state information has to be protected.

References

1. Ekberg, J.E., Kylänpää, M.: Mobile Trusted Module (MTM) - an introduction (November 2007), `http://research.nokia.com/files/NRCTR2007015.pdf`
2. Dietrich, K.: An Integrated Architecture for Trusted Computing for Java enabled Embedded Devices. In: 2nd ACM workshop on Scalable Trusted Computing – STC 2007, pp. 2–6. ACM, New York (2007)
3. Wilson, P., Frey, A., Mihm, T., Kershaw, D., Alves, T.: Implementing Embedded Security on Dual-Virtual-CPU Systems. IEEE Design and Test of Computers 24(6), 582–591 (2007)
4. Khan, M.H., Seifert, J.P., Wheeler, D.M., Brizek, J.P.: A Platform-level Trust-Architecture for Hand-held Devices. In: ECRYPT Workshop, CRASH – CRyptographic Advances in Secure Hardware, Leuven, Belgium, p. 16 (2005)
5. Berger, S., Cáceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: Virtualizing the Trusted Platform Module. In: Proceedings of the 15th USENIX Security Symposium, Berkeley, CA, USA, p. 21. USENIX Association (2006)
6. Zhang, X., Acıiçmez, O., Seifert, J.P.: A Trusted Mobile Phone Reference Architecture via Secure Kernel. In: 2nd ACM workshop on Scalable Trusted Computing – STC 2007, pp. 7–14. ACM, New York (2007)
7. Kasper, M.: Virtualisation of a SIM-Card using Trusted Computing. Master's thesis, Private Fernfachhochschule Darmstadt (2007)
8. Kursawe, K., Schellekens, D., Preneel, B.: Analyzing trusted platform communication. In: ECRYPT Workshop, CRASH – CRyptographic Advances in Secure Hardware, Leuven, Belgium, p. 8 (2005)
9. De Vries, A., Ma, Y.: A logical approach to NVM integration in SOC design. EDN Magazine (2) (January 2007), `http://www.impinj.com/pdf/EDN_NVMinSoC.pdf`

10. Eisenbarth, T., Güneysu, T., Paar, C., Sadeghi, A.R., Schellekens, D., Wolf, M.: Reconfigurable Trusted Computing in Hardware. In: 2nd ACM workshop on Scalable Trusted Computing – STC 2007, pp. 15–20. ACM, New York (2007)
11. Sadeghi, A.R., Selhorst, M., Stüble, C., Wachsmann, C., Winandy, M.: TCG inside? A Note on TPM Specification Compliance. In: 1st ACM workshop on Scalable Trusted Computing – STC 2006, pp. 47–56. ACM, New York (2006)
12. Alves, T., Rudelic, J.: ARM Security Solutions and Intel Authenticated Flash (2007), http://www.arm.com/pdfs/Intel_ARM_Security_WhitePaper.pdf
13. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA Intrinsic PUFs and Their Use for IP Protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
14. Drimer, S.: Volatile FPGA design security – a survey (December 2007), http://www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf
15. Baetoniu, C., Sheth, S.: FPGA IFF Copy Protection Using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs (August 2005), http://www.xilinx.com/support/documentation/application_notes/xapp780.pdf
16. Gassend, B., Clarke, D.E., van Dijk, M., Devadas, S.: Silicon Physical Unknown Functions. In: Atluri, V. (ed.) ACM Conference on Computer and Communications Security – CCS 2002, pp. 148–160. ACM, New York (2002)
17. Linnartz, J.P.M.G., Tuyls, P.: New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates. In: Kittler, J., Nixon, M.S. (eds.) AVBPA 2003. LNCS, vol. 2688, pp. 393–402. Springer, Heidelberg (2003)
18. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 523–540. Springer, Heidelberg (2004)
19. Suh, G.E., Clarke, D.E., Gassend, B., van Dijk, M., Devadas, S.: Efficient Memory Integrity Verification and Encryption for Secure Processors. In: 36th Annual International Symposium on Microarchitecture, pp. 339–350. ACM/IEEE (2003)
20. Handschuh, H., Trichina, E.: Securing Flash Technology. In: Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J.P. (eds.) 4th International Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC 2007, pp. 3–17. IEEE Computer Society, Los Alamitos (2007)

The Zurich Trusted Information Channel – An Efficient Defence Against Man-in-the-Middle and Malicious Software Attacks

Thomas Weigold, Thorsten Kramp, Reto Hermann, Frank Höring, Peter Buhler, and Michael Baentsch

IBM Zurich Research Laboratory
{twe,thk,rhe,fhr,bup,mib}@zurich.ibm.com

Abstract. This paper introduces the Zurich Trusted Information Channel (ZTIC, for short), a cost-efficient and easy-to-use approach to defend online services from man-in-the-middle and malicious software attacks. A small, cheap to manufacture and zero-installation USB device with a display runs a highly efficient security software stack providing the communications endpoint between server and customer. The insecure user PC is used solely to relay IP packets and display non-critical transaction information. All critical information is parsed out of the mutually-authenticated SSL/TLS connections that the ZTIC establishes to the server and shown on the display for explicit user approval.

Keywords: Authentication, Malicious Software, Man-in-the-middle, Secure Token, Secure Internet Banking.

1 Introduction

The Internet is an integral part of our daily lives, and the proportion of people who expect to be able to access sensitive personal information anywhere anytime is constantly growing. Here, Internet banking services can be considered one of the prime application domains. Information about financial institutions, their customers, and their transactions is extremely sensitive and, thus, providing such services over a public network subject to malicious intrusions introduces a number of challenges for security and trustworthiness.

Any Internet banking system must solve the issues of authentication, confidentiality, and integrity. For the latter two, Secure Socket Layer/Transport Layer Security (SSL/TLS) is the de facto standard. For authentication, a variety of schemes ranging from static passwords, to one-time passwords, to various challenge-response protocols involving symmetric or asymmetric cryptography and possibly hardware tokens are in use. These schemes are confronted with different types of attacks, most prominently phishing, malicious software (MSW, for short), and man-in-the-middle (MITM, for short) attacks. An overview of state-of-the-art schemes and attacks along with a discussion of their effectiveness is given in [1].

With phishing, an attacker tries to trick the user via spoofed emails and mock-up web pages to reveal her secret credentials. Recent improvements in Internet banking solutions, though, such as the transition from static/one-time password schemes to short-time passwords generated by hardware tokens on demand, made online services more resistant against phishing. Meantime, attackers moved on to more advanced attacks such as MITM, MSW, or a combination of both. In fact, research shows that such advanced attacks have notably increased in 2007 while “classic” phishing attacks have decreased substantially [2] and that virus scanners are losing [12].

MITM normally is a so-called “network attack”: Here, the attacker unnoticeably intercepts messages between the client and the server, masquerading as the server to the client and vice versa. MSW, in contrast, aims at manipulating a user’s client PC by means of a virus or Trojan horse. Sometimes, attacks are built upon a combination of MITM and MSW if, for instance, MSW modifies the client PC in a way that enables or facilitates a subsequent MITM attack. Frequently, MSW attacks also redirect the client to fake servers or manipulate data displayed in the web browser. The latter is referred to as the “man-in-the-browser” attack.

With advanced attacks increasing in prevalence, in essence, no information either displayed or keyed in on the user’s computer can be trusted any longer. Particularly, any SSL/TLS channel which is not mutually authenticated and/or terminates in a potentially unsafe PC must be considered insecure. Consequently, the processes of establishing trust when, for example, authorizing a banking transaction must be completely moved off the PC. Various schemes to do this have been proposed over the past few years (cf. Section 2 for an overview), yet have not been deployed on a broader basis as being either too complex to install on the server and/or client side, too difficult or tedious to use, too expensive, not really protective, or all of the above.

In this paper we introduce a novel approach to defend online services such as Internet banking against MITM and MSW attacks. For this, we have developed a small and cost efficient USB device comprising a display, buttons, a smart card reader, and a highly efficient security software stack. The ZTIC represents a trusted and tamper-resistant platform used at the client to secure communication with the server as well as interaction with the user. This way it increases the security properties of contemporary Internet banking solutions while requiring no software installation at the client side and none to little changes to server side software.

The paper is organized as follows: After a review and discussion of the state of the art, the design of the ZTIC and how it conceptually defends against MITM/MSW attacks is explained. This is followed by the presentation of performance measurements used for system optimization, the current ZTIC hardware, as well as its initial performance. The paper closes with a discussion of future work and conclusions.

2 State of the Art

The majority of Internet banking systems in large-scale use today is vulnerable against MITM/MSW attacks mainly for two reasons. Firstly, the SSL/TLS connection used to secure the communication between client and server ends on the potentially unsafe client PC. Secondly, the display and keyboard of the client PC is used for all interaction with the user. Consequently, there is no truly reliable way for the user to verify that she does connect to a genuine server nor can she decide whether the information displayed on her screen is genuine or not. For example, once the client PC is compromised, MSW can fake all information displayed, modify or replace the SSL/TLS implementation, manipulate digital certificates, silently modify transaction data sent to the server, and so forth. Most of today’s two-factor authentication schemes such as one-time code lists, short-time code generating hardware tokens, and smart card based solutions therefore do not withstand such attacks [3, 4, 5].

One approach to increase security is to make use of an additional communication channel as done in case of the so-called mTAN (mobile transaction number) system [6]. Here, a server-generated one-time password is sent to the user’s mobile phone via SMS (short message service) along with the details of a given bank transaction. The user can then verify the transaction details and approve it by copying the password to the web browser. The mTAN system considers the mobile phone as a trusted platform with respect to the Internet banking service since it is unlikely that an attacker has gained control on both, the client PC and the mobile phone. Under this assumption the system is resistant against MITM/MSW attacks as long as the user carefully examines the transaction details. However, today’s mobile phones increasingly become open multi-application platforms connected to the Internet and, as such, will eventually face similar attack scenarios as PCs. Additionally, the mTAN system introduces a substantial privacy and confidentiality issue since short messages sent to the phone can be traced, at least by the mobile network operator(s) involved. Also, most users do not enable PIN protection to restrict physical access to their mobile phones while they are switched on. Finally, the mTAN system requires significant changes to the server side software, the SMS communication generates additional costs per transaction, SMS delivery is by definition best effort and not reliable, and copying a password for each transaction is rather inconvenient.

The mTAN approach already indicates that for securing any e-commerce system against MITM/MSW attacks it is inevitable to provide some trusted device to the user. The FINREAD (FINancial Transactional IC Card READer) project aims at establishing a standard for such a trusted device [7]. FINREAD is a set of open technical specifications that define the properties of a secure smart card reader device, which has tamper resistant properties and is accompanied by a secure display and a secure keypad. FINREAD allows building highly secure Internet banking solutions as demonstrated in [8]. However, due to a lack of mobility and, thus, user convenience combined with high costs, it has not made its way into mass market.

Providing the user with a trusted device where she can securely verify and approve transactions is clearly a sensible first step. However, it is also critical to secure the communication between the bank’s server and that device. An innovative approach has been taken with the AXS-Card solution [9]. The AXS-Card is a stand-alone device equipped with a display, keypad or fingerprint reader, and an optical interface device. As with the mTAN, the server generates transaction information to be eventually shown on the trusted client device. Initially, though, this information is encrypted and displayed on the screen of the client’s PC as a flickering code read by the optical interface of the AXS-Card. Only then, assuming that the cryptographic protocol used is secure, the user’s AXS-Card can decrypt and display the information successfully. The advantage of this approach is its mobility and the fact that no additional software needs to be installed on the client PC. However, the secure communication takes place on application level rather than on SSL/TLS protocol level and, therefore, a classical MITM can be established for eavesdropping without the help of MSW. Also, the secure communication is strictly one way, server side software needs to be adapted, and the user-device interaction is rather unusual.

Another innovative approach that claims to secure online services is the so-called Internet Smart Card (ISC) [10]. It consists of a smart card with an integrated USB interface which enables the card to be attached directly to the PC with no smart-card reader or driver software required. The ISC then acts as a full member in the local area network and even includes a web server; therefore, the user can connect to it with any standard web browser. Communication between the ISC and the remote server can be secured via a mutually authenticated SSL/TLS connection whereas the cryptographic keys used remain securely stored on the card. This way the classic MITM attack, where the attacker sits between the card and the server, becomes impossible. Unfortunately, this approach does not withstand MSW attacks because the ISC does not comprise a display. A display, however, is inevitable to ensure that the transaction information sent to the server can be verified. Furthermore, the ISC cannot be PIN protected because it does not provide any means for secure data input such as a keypad.

The mIDentity mobile banking solution represents another smart-card based approach [11]. It consists of a USB stick hosting some flash memory as well as a smart card. All the software required, even the web browser used, is included on the stick for mobility and plug-and-play convenience. Its usage scenario, though, is rather a classical smart-card approach. The web browser is running on the PC and uses the smart card to secure a SSL/TLS connection between the web browser and the server. There is no secure display and the secure connection still ends on the PC. Consequently, it does not offer protection against MITM/MSW attacks. Additionally, the need to transfer a full web browser from the USB stick to the PC for each banking session may well introduce user acceptance problems. Finally, promptly updating the web browser on the USB stick to keep up with released security fixes to protect against known vulnerabilities of the browser remains an administrative challenge.

Obviously, most state-of-the-art solutions for securing Internet banking or similar sensitive online services against MITM/MSW attacks exhibit significant short-comings. Of all the ones discussed above, the AXS-Card and the mTAN solutions appear technically most promising, though none combines an effective security concept with cost efficiency, convenience, and mobility without requiring significant server side software changes. The latter, however, we consider a crucial prerequisite for large scale adoption. In the following Section we therefore present a novel approach, the ZTIC, which aggregates all these features into one device.

3 The Zurich Trusted Information Channel

Conceptually, the Zurich Trusted Information Channel (ZTIC) adds a trusted and tamper-resistant secure communication endpoint, prototypically embodied as a USB stick, to an otherwise untrustworthy client PC. Through this endpoint a user then can securely communicate with sensitive online services such as a banking server.

As illustrated in Figure 1, all communication between the user and the server is passed through and processed by the ZTIC which in turn is hooked into the communication path by a networking proxy running on the PC. When in use, the ZTIC continuously scans the data exchanged between client and server for sensitive operations such as bank transfers. For each sensitive operation, it intercepts the communication flow, extracts crucial information for display and verification, and proceeds only after the user has explicitly confirmed the operation by pressing an OK button on the ZTIC. Non-sensitive operations are just passed along without user interaction. Additionally, the ZTIC may serve as a holder for sensitive personal information, such as a private key used in SSL/TLS client authentication. If non-repudiation is a strong design goal of an authentication solution, it is also possible to utilize a smart card within the ZTIC to protect private data from extraction and duplication.

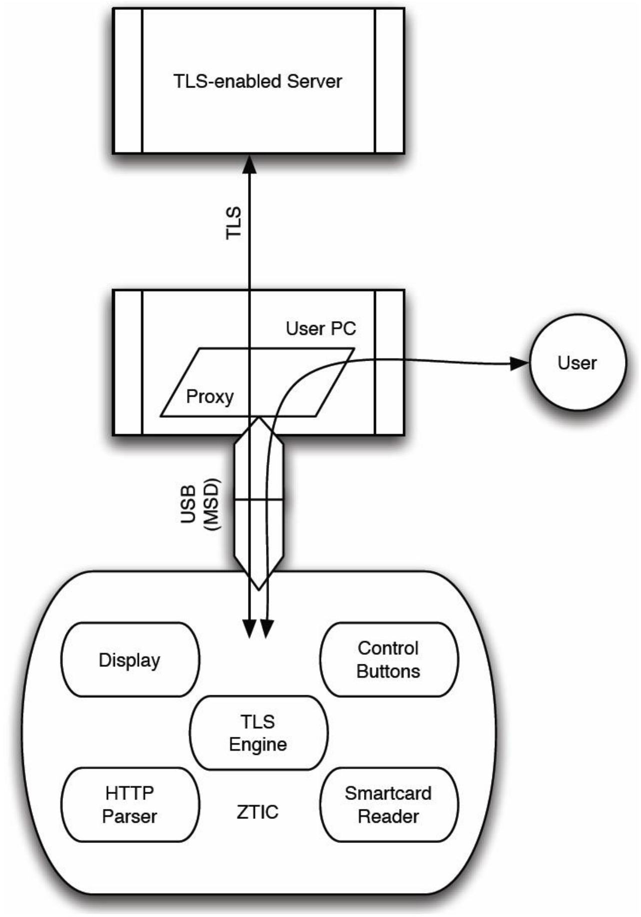


Fig. 1. ZTIC Configuration

For all this, conceptually, the ZTIC hardware consists at minimum of a processing unit, volatile and persistent memory, a small display, at least two control buttons (OK, Cancel), and optionally, a smart-card reader. Its software is minimally configured with a complete TLS engine including all cryptographic algorithms required by today's SSL/TLS servers, an HTTP parser for analyzing the data exchanged between client and server, plus a custom system software implementing the USB mass storage device (MSD) profile.

Furthermore, the ZTIC is pre-loaded with the aforementioned networking proxy. This proxy hooks the ZTIC into the client/server communication path, yet, since running on the potentially insecure client PC, itself does not contribute to the ZTIC's security. The proxy merely sends data to and receives responses back from the ZTIC by reading and writing a dedicated, virtual file on the ZTIC's drive, respectively. This approach makes the ZTIC dependent only on standard USB MSD drivers readily available on all operating systems.

Finally, in addition to this "standard" software stack, each ZTIC is configured with user credentials, one or more X.509 client and server certificates enabling TLS with client authentication, and a server-specific HTTP parsing profile that configures the ZTIC's generic HTTP parser to selectively identify sensitive operations such as a money transfer from the communication stream.

Referring to Figure 1, a typical client/server interaction involving the ZTIC thus follows these steps:

1. The user plugs the ZTIC into her (potentially insecure) client PC which mounts the ZTIC as a USB mass storage device and auto-starts the proxy also residing on the ZTIC. The proxy automatically launches the user's default web browser and connects to the pre-set server of choice.
2. The proxy in response triggers the ZTIC to initiate and run a TLS session handshake including mutual authentication. Hereby, the proxy "blindly" relays all the communication exchanged between ZTIC and server, that is, without being able to interpret the communication itself. Upon successful completion of the handshake, both client and server have been authenticated and a TLS session has been established between the ZTIC and the server.
3. From then on, all interactions between the client and the server, still "blindly" relayed by the proxy, are scanned by the ZTIC for sensitive operations. If found, the ZTIC sends the user's operation request and transaction data only after explicit user consent has been given as described before. For instance, considering again a bank transfer, the ZTIC extracts the transaction's crucial data such as the recipient's bank account information and the amount of money to transfer, displays this information on its display, and awaits the user's explicit confirmation via a press on its OK button before proceeding. If the user detects any manipulations in the transaction's crucial data, she can abort the transaction by pressing the Cancel button.
4. Likewise, confirmation data from the server can also be dynamically extracted and shown on the ZTIC's display for user verification, if needed.

So, how does this scheme defend against MITM/MSW attacks? Fundamentally, it is impossible to prevent MITM/MSW attacks on insecure client PCs. Even with an HTTPS connection an attacker can still get hold of and modify the plain data (e.g., by tampering with the web browser). The ZTIC therefore does not try to prevent MITM/MSW attacks to begin with—in fact, it cannot—but instead focuses on exposing MITM/MSW attacks to the user who then can take appropriate action. Being a MITM itself, the ZTIC makes explicit what really is communicated between client and server, and while a MITM/MSW may still passively read the transaction data exchanged, any modification will be inevitably disclosed by the ZTIC. Only sloppy verification of the transaction data shown on the ZTIC's display by the user will allow modified transaction data still to be sent to the server.

Furthermore, provided the user credentials and the X.509 certificates for TLS mutual authentication are maintained securely on the ZTIC (e.g., on an embedded smart card), no MITM/MSW can impersonate the ZTIC while the ZTIC in turn can verify that it connects to a genuine server. Since the proxy only implements a relay service, any attack on the proxy may, at worst, just cease the service to work with no other harm done. Note that while the proxy may communicate with the ZTIC in plain, all the (user-confirmed) communication between the ZTIC and the server is protected by the TLS session ending in the ZTIC, thus keeping MITM/MSW locked out from modifying transaction data unnoticed.

It should be noted, though, that for the security of this approach it is crucial to keep the HTTP parsers evaluating all transaction data operating exactly the same on the ZTIC as on the server. Otherwise, the ZTIC could be tricked to show transaction information different to the one processed by the server later on.

Deployment Considerations

From a user convenience and deployment management perspective, the ZTIC requires no manual software installation by the user, no manual server certificate checks, and is small enough to qualify for mobile usage, for instance, as a keychain "fob." Furthermore, the user is still free in the choice of her web browser and, thus can promptly take advantage of new features and security patches made available by browser developers. A single ZTIC can also be configured with multiple HTTP parsing profiles, user credentials, and X.509 certificates to allow secure communication with multiple servers from different service providers.

From the point of view of cost, both the device itself and the server software have to be considered. The manufacturing costs for a ZTIC in the prototype configuration mentioned above would certainly qualify for broad deployment to bank customers, for instance. Most importantly, though, for a wide acceptance by service providers, minimal server side changes are required. Only TLS client authentication, a standard feature of today's web servers, must be enabled and accompanied by an appropriate public-key infrastructure (PKI). In particular, no changes to the user authentication protocols in place are necessary and, depending on the scheme used, may even be completely handled by the ZTIC, potentially supported by the embedded smart card and without any user interaction for added convenience. Then, of course, the ZTIC would have to be PIN protected and manually "opened" by the user prior to connecting to the desired server.

In the original system design, the networking proxy shuttled plaintext data back and forth between ZTIC and web browser. In the deployment version, the networking proxy provides an SSL-protected communication path to the web browser. This way, users are no longer required to “unlearn” that secure connections are indicated by a “key lock” symbol and the *https://* address indicator. In addition, and more importantly, the web browser is discouraged from caching the banking web pages as data received via SSL/TLS is typically subject to more stringent caching rules than data received via plain HTTP.

4 Performance Measurements

In addition to its effectiveness against MITM/MSW attacks, it is also important to understand how the ZTIC affects the communication throughput and, perhaps even more important, the performance as perceived by the user when interacting with her service of choice. This is a challenge because, on the one hand, there are serious cost constraints associated with a device such as the ZTIC dictating the use of an inexpensive and thus relatively slow processing unit, limited memory, and full-speed USB (at a nominal maximum of 12 MBit/s) only. On the other hand, TLS imposes complex cryptographic operations and users are accustomed to a broadband connection experience nowadays.

In order to navigate these partially contradicting design goals, we have established a simple baseline measurement setup to determine where overall system optimizations are most effective. This setup consists of two components, namely an MSD throughput optimizer and an SSL/TLS pass-through measurement proxy.

MSD Throughput Optimizer

As all data transferred between web browser and server needs to be transferred between proxy and ZTIC twice, a test suite to determine the best choice of file transfer block sizes and interaction protocol between proxy and ZTIC was necessary. In essence, on the ZTIC itself, it consists of a simple USB routine storing incoming data in RAM locations from which it can be retrieved again by an external software component. This external software component is geared towards sending and receiving chunks of data to and from this on-ZTIC test code. A PC-side test software is instrumented to vary block sizes to determine the maximum throughput and to inform design trade-offs dictated by the (small) size of the ZTIC’s RAM hardware buffers. All timing measurements have been done using the Windows® hardware high-precision timer on a dedicated standard 2.13 GHz benchmark PC running Windows XP®.

According to the results of these measurements as depicted in Figure 2, larger packets clearly create higher overall throughput rates. At about 15 kB, however, the effect of larger packets tapers off noticeably. Initially, we were surprised by the difference of these measurements and the nominal 12 MBit/s maximum data rate

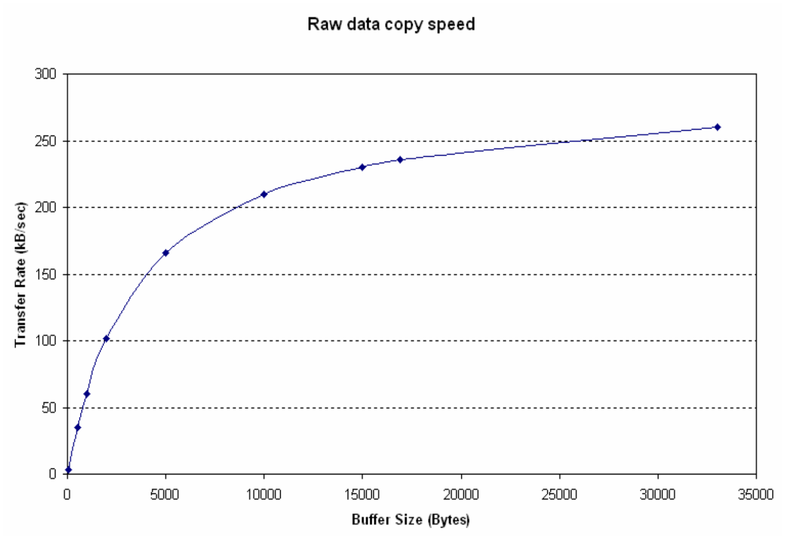


Fig. 2. USB MSD Two-way throughput

promised by USB 2.0's Full Speed capabilities. By using a USB line analyzer, though, we could establish that the host operating system in conjunction with the 64 bytes USB packets supported by the chosen chip already severely limit the "raw" USB throughput. At hardware level, measuring the real time elapsed between start block and acknowledgement command, the maximum theoretical throughput (if no PC-side processing were performed) is already only 680 kB/s for 16 kB data packets transmitted. Taking PC-side processing into account, which typically is 1-2 msec per USB MSD packet, only a maximum, one-way throughput rate of 500 kB/sec (at 16 kB block size) is explicable for Full Speed USB running the MSD protocol. Therefore, 250 kB/sec for two-way/back-and-forth data transfer as measured is logical.

SSL/TLS Pass-through Measurement Proxy

This proxy runs on a PC and its sole purpose is to measure the actual amounts of data transferred during a typical banking session between web browser and server. Also of importance are the numbers of SSL/TLS session establishments, as each one requires an interaction with the ZTIC, and the distribution of data packets and their relative sizes as explained above.

The proxy therefore contains all cryptographic functionality to operate as an SSL/TLS client to different web servers. It also contains functionality to re-write the data received on the fly so that the web browser always directs requests for the web server under test to the proxy first. For example, absolute links in the HTML code (pointing to the server directly) are re-written such that the browser sends subsequent requests to the proxy.

Again, timing was performed using the Windows® high-precision timer on our benchmark PC. Prior to each test run, the browser cache has been emptied to ensure comparability between test runs and between servers. Furthermore, network access was via an average broadband (DSL) connection, likely the most typical configuration in which the ZTIC will eventually be used.

As test servers, we have chosen different international bank's SSL/TLS-protected web servers. For each server, the following steps have been performed:

- Retrieval of (demo) e-banking entry page;
- Authentication procedure using demo account;
- Initiation of a payment transaction.

For each operation we waited until all (sometimes slightly delayed) requests from the browser have been executed. The measurements have been taken at different times of day, both during office and non-office hours, as well as on holidays and work days to ensure the typical differences in Internet load and delay are captured. The servers chosen offer publicly accessible SSL/TLS-protected demo accounts (supporting SSL/TLS server authentication only). We have chosen two from our home country, Switzerland, two from Austria and Germany each, as well as one from Australia to determine the effects distance plays both in terms of network hops as well as geography. The Australian server's usage in our tests is special insofar as it did allow us only to perform the last of the three steps listed above because it does not have a publicly accessible, SSL/TLS protected fully functional demo account: All other non-European banks seem to use HTTP only for their demo accounts.

Table 1. USB MSD Two-way throughput

	Bank A (CH)	*Bank B (CH)*	*Bank C(DE)*	*Bank D (DE)*	*Bank E (AT)*	*Bank F (AT)*	*Bank G (AUS)*
Bytes IN	507997	588707	157230	198851	147800	127888	29680
Avg. IN Throughput [kB/s]	23.2	58.7	10.7	10.2	24.4	4.5	2.6
Hiscore IN [kB/s]	98.8	267.3	122.5	217.0	159.4	27.4	8.8
Bytes OUT	62753	29043	26517	35046	12491	30333	9204
Avg. OUT Throughput [kB/s]	2.9	2.9	1.8	1.8	2.7	1.1	0.8
Time to completion	22.1 s	10.2 s	15.4 s	19.7 s	4.7 s	28.6 s	11.0 s
SSL/TLS packets	518	343	357	161	104	418	49
RSA key size[Bits]	2048	1024	1024	1024	1024	2048	1024
Cipher	AES-128	AES-128	AES-128	AES-128	RC4-128	RC4-128	RC4-128

The results of our measurements shown in Table 1 are the median values of 10 measurements for each server. The connection between test proxy and web browser has been SSL protected using a “localhost” certificate we created for this purpose. This way, the test is as much as possible in line with the secure deployment configuration of the ZTIC.

As expected, the down-stream data volume (i.e., from web server to browser) is much larger than the up-stream data volume from browser to server. Typically, the difference seems to be an order of magnitude.

One of several interesting observations is that the server of Bank A has a significantly lower nominal throughput rate than the otherwise similar server of Bank B, even though it uses the same stream cipher (AES-128) and transfers roughly the same amount of data. In our judgment, the reason for this is the more frequent SSL/TLS session establishment and the larger size of the private key.

Equally interesting is the observation that the maximum data transfer rate (for the largest packets—dubbed “Hiscore IN” in Table 1) is at about the throughput rates we can theoretically achieve with maximum-length USB MSD packets transferred through the ZTIC (cf. Figure 2). This result justifies our choice of a Full Speed USB device over a more expensive High Speed USB 2.0 device as such a device could not be run even close to saturation.

5 The First Prototype

As mentioned before, the minimum requirements of the ZTIC hardware are: a processing unit, volatile and persistent memory, a small display, at least two control buttons (OK, Cancel) and, optionally, a smart-card reader. The current prototype uses an inexpensive system-on-chip unit (a 32-bit ARM CPU, 64 KB of RAM, 256 KB of Flash memory, and a simple USB Full Speed controller), a 128x32 pixel monochrome OLED display, two control buttons, and a smart-card reader for smart cards in the SIM form factor (i.e., ISO 7810 ID0). Figures 3/4 show the prototype unit.

The embedded software running on the ZTIC is predominantly written in C. The ZTIC does not run a traditional operating system, but consists of a series of independently operating state machines. This “lean-ness” in design has two advantages: Firstly, the on-ZTIC code is rather small. The cryptographic engine requires about 60 kB of object code while the rest of the system (USB/MSD, display and I/O handler plus SSL/TLS operations logic including HTML parsing) takes approximately 45 kB. This way, a small and inexpensive chip could be chosen. Secondly, for future security

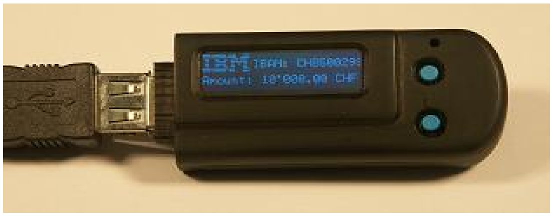


Fig. 3. The ZTIC prototype unit (casing)

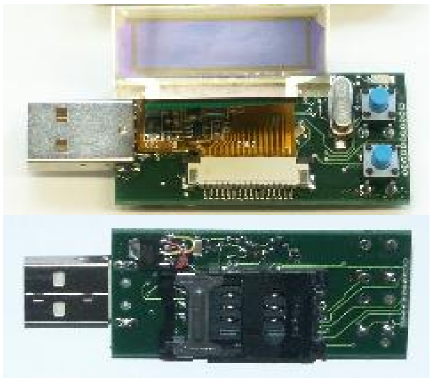


Fig. 4. The ZTIC prototype unit (front, back)

certifications, no undue effort in certifying large chunks of relatively unimportant code needs to be expended.

The observations made using the MSD throughput optimizer led us to a system design with a communications buffer providing 2 x 16,5 kB RAM (for data-in and data-out, respectively). For data streaming in one direction only, typically down-stream, both can be linked together to provide a single 33 kB buffer for maximum efficiency.

The data store on the ZTIC has been implemented as a FAT12 file system such as to seamlessly work in all present-day operating systems. Also, this choice led to a reduction in size for the file system allocation tables and the minimum block size chosen: 512 bytes have proven during experimentation to be a most suitable balance between bulk (encryption/decryption) and short-term (SSL/TLS session establish-ment) operations required for the ZTIC's optimal operation.

The networking proxy shuttling data back and forth between web browser, ZTIC, and Internet has been implemented in C and initially runs on Windows XP®, Linux and Apple® Mac OS X. A port to further platforms is trivial due to the simple interfaces used: Sockets for communication with web browser and Internet, and files for interaction with the ZTIC. As the design goal was complete system integration on the ZTIC, this component also has been optimized for size. The smallest version running on Windows® has a size of 40 kB and, therefore, fits comfortably together with other file system data (e.g., *autorun* information, certificates, log files) along side the ZTIC embedded security software into the chip's 256 kB of Flash memory with over 100 kB to spare for enhancements.

Due to these optimizations, the time elapsing between insertion of the ZTIC and the corresponding popup for user approval of automatically running the proxy off the ZTIC is less than 2 seconds on our benchmark PC running Windows XP®. Depending on the choice of web browser, already running web sessions, and the size of the server SSL/TLS certificate and key size, the web site preset by the ZTIC is normally up and visible after another 2-3 seconds.

System Performance

Utilizing the timing instrumentation in the ZTIC proxy, we re-created the same measurements as with the baseline SSL/TLS measurement proxy. The results are summarized in Table 2.

Table 2. SSL/TLS session data running over the ZTIC

	Bank A (CH)	*Bank B (CH)*	*Bank C(DE)*	*Bank D (DE)*	*Bank E (AT)*	*Bank F (AT)*	*Bank G (AUS)*
Bytes IN	507920	588710	157230	198777	147800	127921	29680
Avg. IN Throughput [kB/s]	18.1	35.4	8.2	9.2	24.4	4.6	2.6
Hiscore IN [kB/s]	61.4	111.9	72.4	103.7	105.2	23.1	8.0
Bytes OUT	62565	29046	26517	34983	12491	30333	9204
Avg. OUT Throughput [kB/s]	2.2	1.8	1.4	1.6	2.1	1.1	0.8
Time to completion	28.3 s	16.8 s	20.3 s	21.8 s	6.1 s	28.3 s	11.6 s
Delay	28%	65%	31%	11%	31%	0%	5%

When comparing Tables 1 and 2, a drop in user-level response time of between 0% and 65% is evident. These large differences warranted some further investigations. Most performance differences can be explained by the differences in network distance between our measurement client and the servers: Obviously, the Australian server responds slowest overall and the Swiss servers fastest. Finally, while the two Swiss servers are transmitting approximately the same amount of data, they are behaving rather differently in terms of end-user visible performance. We attributed this to the difference in the number of SSL/TLS packets transmitted by the two servers: Bank B's server transmits fewer but larger packets while the Bank A site uses more but smaller ones, most probably creating a bigger overall overhead.

Nominally, we can claim that the overall ZTIC system comprising data transfer and cryptographic operation on the off-PC hardware introduces an overall measured performance degradation of an average of 30% when compared to our baseline SSL/TLS pass-through proxy server. Given the increase in security that this low-cost ZTIC provides, we consider this a very promising result. Additionally, it should be noted that the ZTIC's performance very much depends on the concrete usage pattern that a web server forces onto its clients by the way its web pages are structured.

When judging the overall system responsiveness, more subjective impressions come into play, such as the web site interaction speed determined to a large degree by the speed with which the page builds. For the servers we have used for our experimentation, initial user feedback was very positive. The main pages build very fast and it is only the speed with which large background design patterns or many smaller navigation icon images appear that is held back by the on-ZTIC processing of SSL/TLS data.

6 Discussion and Future Work

Acceptance

So far, only IT-literate people have been subjected to the ZTIC and its impact to secure authentication transactions. We are now beginning to devise acceptance testing by end-users. We will also explore ways to replace the proxy by a VPN-like driver to avoid the dynamic rewrite of web pages.

Of equal importance will be handling issues, such as whether or not the ZTIC needs to come with a cable to be attached to inaccessible USB ports, or whether the current, simple version for immediate attachment and a display that can be toggled in its orientation, will prove sufficient.

Certification

An appropriate security evaluation of the ZTIC software stack itself may be done to minimize the risk—or indeed even prove the absence—of common software vulnerabilities such as buffer overflow errors. By offloading the most security-sensitive, long-lived data and operations to an (already certified) smart card, this effort can be kept to a minimum.

Performance

The most time-consuming cryptographic operations within the ZTIC are symmetric operations, typically AES for encrypting/decrypting the data stream and asymmetric sign/verify operations during SSL/TLS session establishment. We have already changed the core RSA engine into an assembly code version, achieving a performance speed-up of a factor 4 in raw signature verification. Similar results could be obtained by changing the SHA1 and AES core routines into assembly code as well. Thus, we currently do not foresee the need to use a hardware accelerator, both from the perspective of system-wide throughput limitations by USB MSD as described above as well as from a hardware cost perspective. Further justifying this statement are tests using RC4 as a symmetric cipher suite. Server B accepted the use of RC4 when the ZTIC was configured to not offer using AES as a cipher suite and the resulting performance gain was astonishing: On the same benchmark setup, a peak throughput of 170 kB/sec could be observed, that is, more than 50% faster in end-to-end performance when compared to AES-usage. Of course, computing RC4 is about 4 times as fast as performing an equivalent AES operation. By using faster implementations of the key algorithms, we therefore expect to push the boundary closer to the maximum defined by Full Speed USB and end-to-end server throughput as determined above.

The publicly available servers used for testing do not support TLS with client authentication. Therefore, we plan setting up such a server to measure the performance impact. Client authentication additionally requires the ZTIC to generate a digital signature during the initial TLS handshake. As the ZTIC prototype is able to generate a RSA signature (1024 Bits) in 400ms, we expect a rather minor overall performance penalty.

Another alternative to speed up the decryption of data received from the servers is the concept of performing this within the proxy. For this to work, the proxy and ZTIC must exchange the downstream TLS session state. A large numbers of session re-negotiations might limit the positive effect, though.

Functionality

With the basic ZTIC system fully operational now, various enhancements of the concept are conceivable. The ZTIC, for instance, could provide its security services to a VPN gateway instead of (or in addition to) the web browser authentication proxy we have described so far, thus extending the use of the concept widely beyond securing transactional data. A version of the ZTIC with a somewhat larger form factor has been created that allows the insertion of ID1 (credit card size) smart cards in order to preserve and leverage existing investments in smart card deployments (e.g., into EMV CAP cards). Experiments with user acceptance will need to be conducted.

PIN entry so far needs to be performed via the embedded buttons in the same way a two-button digital watch is adjusted. Alternatively, and arguably more conveniently, a secure PIN-entry process using, for instance, a random challenge displayed by the ZTIC and entered by the user in a PC-based user interface—factoring in the user PIN, of course—can be devised. With an ID1 card reader, the addition of a keypad to more conveniently enter the PIN is also clearly a viable technical alternative. User acceptance testing can help to settle these convenience-vs.-cost questions.

Finally, considering maintenance, the ZTIC itself might be updated by a server with new HTTP profiles as well as renewed user credentials and X.509 certificates once a TLS connection has been established. Regular credential updates or changes to a service provider's web presence then can be easily propagated to the ZTIC population.

7 Conclusions

This paper introduced the ZTIC, a cost-effective and user-friendly security system preventing both malicious software and man-in-the-middle attacks requiring minimal to no changes in both server- and client-side software typically used for sensitive transactions. We have justified the system design choices with real-world performance tests pinpointing the end-to-end performance bottlenecks. A first prototype implementation on low-cost hardware has already shown favourable overall system performance levels. Indicators for SSL/TLS servers particularly well suited for use with the ZTIC have been found. Various possible improvements have been discussed aiming to enhance the ZTIC into a universally acceptable, cost-effective security device countering increasingly sophisticated attacks in the Internet.

Table 3 summarizes our design goals and briefly reiterates how the ZTIC meets these.

Table 3. Design goals and ZTIC solutions

Design goal	*ZTIC solution*
Authentication, integrity, confidentiality	Operation of SSL/TLS
Protection from Malicious PC Software	Display of critical data on trusted device (ZTIC); no persistent caching of any data on the PC
Protection from Man-in-the-Middle Attacks	Mutually authenticated end-to-end session establishment (via SSL/TLS)
Protection from Phishing attacks	Automatic session establishment with securely stored credentials
Non-Repudiation	Use of smart card slot and PIN entry
User acceptance/ convenience	Small, well-known (key fob) form factor; no software/driver installation; works on all operating systems; fast user interaction
Service provider acceptance	None to little server side changes required; no new customer support requirements due to ease-of-installation
Cost-efficiency	Low-footprint software on cheap chip; option to retain smart card investments

Acknowledgments

We would like to express our thanks to the whole team of Engineering Services at the Zurich Research Lab that has put together the ZTIC hardware; we are particularly grateful for the work of Hansruedi Steinauer who provided us with quick design turnarounds and always working prototypes.

References

1. Weigold, T., Kramp, T., Baentsch, M.: Remote Client Authentication. IEEE Security & Privacy journal (accepted, 2008) (to be published)
2. Federal Office of Police, Swiss Reporting and Analysis Centre for Information Assurance MELANI. Semi-annual report 2007/1, http://www.melani.admin.ch/
3. RSA SecurityID Token, RSA Security (2007), http://www.rsa.com/node.aspx?id=1156
4. Schneier, B.: Two-Factor Authentication: Too Little, Too Late. Comm. ACM 48(4), 136 (2005)
5. Schneier, B.: Fighting Fraudulent Transactions (November 27, 2006), http://www.schneier.com/blog/archives/2006/11/fighting_fraudu.html
6. Federal Office for Information Security. The IT Security Situation in Germany (2007), http://www.bsi.de/english/publications/securitysituation/Lagebericht_2007_englisch.pdf
7. The FINREAD (FINancial Transactional IC Card READer) project, http://www.finread.com

8. Hiltgen, A., Kramp, T., Weigold, T.: Secure Internet Banking Authentication. IEEE Security & Privacy 4(2), 21–29 (2006)
9. AXSionics AG. The Internet Passport, http://www.axsionics.ch
10. Giesecke & Devrient GmbH. Internet Smart Card Technologie, http://www.gi-de.com/portal/page?_pageid=36,53930&_dad=portal&_schema=PORTAL
11. Kobil Systems. mIDentity Mobile Banking, http://www.kobil.de/fileadmin/download/products/ONLNEFLYER-MIDENTITY-MOBILEBANKING_1V00_20060420_DE.PDF
12. Hines, M.: Malware flood driving new AV: InfoWorld (December 14, 2007), http://www.infoworld.com/article/07/12/14/Malware-flood-driving-new-AV_1.html

A Model for New Zealand's Identity Verification Service

Clark Thomborson

Department of Computer Science
The University of Auckland, New Zealand
cthombor@cs.auckland.ac.nz

Abstract. We develop a simple model of the processes by which identity and anonymity are managed by complex systems. We explain New Zealand's recently-proposed Identity Verification Service in terms of our model. We also indicate how our model might be used to guide the architecture of a next generation of trustworthy computing, and how it might be used to define a precise taxonomy of authentication.

Keywords: Identification, authentication, trust, security modelling.

1 Introduction

Our primary goal in this work is to develop the simplest possible model which can represent the full range of identity-management systems, at a level of detail which will support a security analysis.

We have chosen the acronym CEO for our model to indicate a possible application, in which a Chief Information Security Officer is describing identity management and security management technologies at just enough detail to allow their Chief Executive Officer (CEO) to make strategic decisions about the identification systems within their organisation.

Because our model has a simple structure, it can be used to clarify and extend the definitions in existing taxonomies of concepts and systems for identification and authentication.

Our CEO model was inspired in part by the ECO extension of Secure Tropos [5]. The entities in the ECO model have entitlements, capabilities and objectives. Arcs in the ECO model indicate how these entities interact to acheive their objectives, by exercising their capabilities and by making service requests (within the bounds of their entitlements) from more capable entities.

Our CEO model is complementary to the ECO model. Our entities have fixed entitlements, capabilities, and objectives. The activities of entities in our model are focussed on the consumption and delivery of identification and anonymity services.

The fundamental constructs in our model are Credentials, Entities, and Observations. Credentials are disclosed by entities, revealing the value of identifying attributes of itself or of other entities. In addition to disclosing and receiving credentials, entities can make observations of other entities. An observation is is an independent evaluation of an entity's observable attributes, and a credential consists of information from prior credentials and observations.

Our model can represent authentications in addition to identifications. Authentications are usually taxonomised as "what you know", "who you are", and "what you have". Using our model, sharp distinctions can be made between these taxonomic categories.

We find it necessary and sufficient, at least for the purpose of describing the Integrity Verification Service of New Zealand [1], to have three types of relationships between entities. Entities may be in a superior-inferior relationship, forming a hierarchy. Entities may be in an anonymous peerage. Furthermore, entities may have alter egos or aliases, allowing members of a peerage or hierarchy to participate in other hierarchies and peerages.

The three types of relationships (aliased, hierarchical, and peering) in our CEO model must, we believe, be incorporated in the design of a general-purpose trusted computer system. If, for example, a trusted computer system has support only for hierarchical relationships with other trusted computer systems, then it will be unable to form a trustworthy relationship with any computer system unless one of these two systems is the hierarch. Some may believe that there will be only one hierarch for a single trusted computer system that spans the world, but this outcome seems unlikely to us in light of the wide variety of legal, social, and political environments in which our computers operate.

A system which is trustworthy for use in international commerce would not, we believe, be in a hierarchical relationship with the computer system of any nation's security agency. Nor do we think it likely that any national security agency would accept a subservient relationship with a computer system that is used for international commerce. Neither a commmercial system, nor any security agency's computer system, should be in a hierarchical relationship with a trusted computer system which is being used by citizens in some jurisdiction which allows them to communicate with each other in a highly anonymous fashion. Computer systems which preserve anonymity in a trustworthy fashion could, we believe, be implemented as an extension of the anonymity-preserving peerages in our CEO model.

National security agencies are hierarchical, and for this reason we believe their computer systems could be implemented by extending the operations and powers available to the supervisors and inferiors in our CEO model. Cooperating commercial enterprises could be well-modelled as individual hierarchies which are able to communicate with each other using aliased relationships (if suitably extended) in our CEO model. The security attributes of the aliased relationships on each connection between two commercial enterprises should, we believe, be defined by the contractual, economic, and legal arrangements between the cooperating entities. Such a structure – if built – would be broadly in line with the requirements on a Collaboration Oriented Architecture announced recently by The Jericho Forum [2,4]. Clearly, our proposed design for a worldwide trusted computer system is very incomplete and highly speculative.

We devote the majority of this paper to describing the CEO model. At the risk of confusing the reader, in our exposition we indicate many possible extensions to our model, in order to indicate how it might be extended to handle the more complex relationships arising in a trustworthy computer system with a wider range of functionality than identity verification. We close the paper by arguing that our CEO model is sufficient to predict the identity disclosures, and anonymity preservations, among the entities in New Zealand's proposed Identity Verification Service (IVS). The IVS is a complex system, providing semi-anonymous identification credentials for use by different governmental agencies. Typical identification management systems are much less complex, and could be represented without using the anonymity-preserving peerages of the CEO model. We challenge the reader to develop a model of the IVS with fewer types of entities and fewer types of relationships than our CEO. We are unable to provide a proof of minimality, but we strongly suspect that the CEO model is optimal with respect to the Occam's Razor criterion: a model should be no more complex than is required to adequately represent the features of primary interest in the real world. In this case, the features of interest are the identity disclosures and anonymity preservations in an identity management system.

2 The CEO Model

Each entity has a private storage area for recording their observations and for retaining the values of secret identifying attributes. This stored information is subject to confidentiality, integrity, and availability constraints in our base model. If these constraints were relaxed in a refinement of the model, we could explore the properties of a system in which some entities were able to tamper with, or to inspect, another entity's private store. Almost all of our model assumptions have the property that, if they were relaxed, they would reflect a system with a security vulnerability. Thus a list of our model assumptions would be useful to security analysts who are looking for all possible vulnerabilities in an identification system. In future work we intend to present our model in a formal set of axioms, but in this paper we explain our model only in a narrative style.

In our base model, there is a fixed population of n entities $E = e_1, e_2, ..., e_n$. Refinements of our model will have a birth-and-death process with a maximum population size of n.

There are two types of entities in our base model: active and passive. Active entities are able to observe the identifying attributes of other entities, and they are able to disclose information in credentials. A passive entity can neither observe nor disclose, but it does have identifying attributes, two of which can be observed through its Superior port, and one which can be observed through its Alias port.

An entity has four ports, through which it performs four different sets of observations and disclosures with its superiors, inferiors, peers, and aliases. Passive entities cannot have any inferiors. We draw entities with circles, using larger shaded circles for active entities. We draw connections between ports with lines and arrows. See Figure 1.

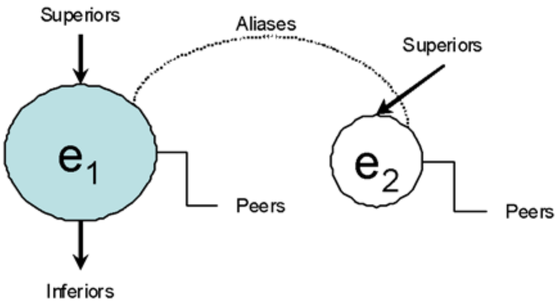


Fig. 1. Active and passive entities have separate ports for connections to their superiors, peers, and aliases. Active entities may have inferiors. The active and passive entities in this figure are aliased.

In the first phase of each timestep, each active entity discloses different credentials through its four ports. In the second phase of each timestep, each active entity makes an observation through its Alias and Inferior ports, and it also receives the credentials transmitted by other entities (including itself) through a connected port.

Through its Alias port, an entity can observe just one attribute of its aliases (including itself). We call this attribute the "observable identity" or ObId of an entity. Through its Inferior port, an entity makes a somewhat more intrusive observation which reveals the value of both the ObId and the PeerId, or peer identity, of every inferior.

The ObId value for each entity is unique. The PeerId value is not unique: all peers have the same value for this attribute.

Entities have a third identity attribute, its "self identity" or SelfId. The SelfId is not observable by any other entity. An entity uses its SelfId to sign and encrypt the credentials it discloses through its Superior and Alias ports. A superior broadcasts its SelfId through its Inferior port in an otherwise-uninformative credential.

Credentials disclosed through the Peer ports are signed and encrypted with the PeerId. Peer-disclosed credentials are semi-anonymous in the following respects. There is no way for any entity to determine, merely by inspecting a peer-signed credential, which of the peered entities signed or sent it. However it may be possible for an entity to determine the source of a peer-disclosed credential by indirect means. Entities are able to discover the size of their peerage by counting the number of credentials they receive through their Peer port in a timestep. An entity which is in a peerage of size two will thus be able to recognise all credentials disclosed by its other peer, by a process of elimination.

We assume that all active entities are able to create a signature, such as a keyed cryptographic hash, which can be used to validate the content of a credential. We also assume that active entities can encrypt a credential, so that it is readable only by the addressed recipient(s). Suitable signing, encrypting, validation, and decrypting operations are available in any implementation of asymmetric cryptography. In refinements of our model, we could introduce a new class of active entities which are incapable of signing and encrypting credentials by cryptographic means. These cryptographically-incapable active entities would allow us to analyse the vulnerabilities introduced into a system when the human entities in the system are dependent on their computerised agents (aliases) to perform cryptographic operations.

The SelfId and PeerId attributes in our model are a few thousand bits in length, so that they are long enough to carry the public portions of both the signing and encrypting keys. The private portions of these keys are never disclosed by entities in our base model. We assume that the signing and encrypting operations are immune to adversarial attack by entities who don't have the private portions of these keys.

2.1 Restrictions on Connectivity

In our base model, we greatly restrict the connectivity of entities.

Supervisions. Connections between Inferior and Superior ports are called supervisions, or supervisory channels. There are no Peer or Alias ports in any supervision. Entities with Inferior ports in a supervision are called superiors. Entities with Superior ports in a supervision are called inferiors. If there are no inferiors in a supervision, it is called a trivial supervision. All passive entities have trivial supervisions.

We draw the Inferior port at the bottom of the circle denoting an entity in our figures. Superior entities use this port to collect information from their inferiors in our base model, directly through observations of ObId and PeerId values, and indirectly by receiving the credentials disclosed by their inferiors. In refinements of our model, the Inferior port would be used by superiors to distribute commands, rewards, and punishments to their active inferiors, and to modify the attributes of their passive inferiors. In our base model, the credentials disclosed through the Inferior port merely inform inferiors of the SelfId of their superiors.

In each timestep of our base model, active entities disclose a portion of their current map of the system (describing known identities, ports, and channels) in a credential through their Superior port. Inferiors are thus entirely honest, and partially disclosing, when disclosing credentials to their superiors in our base model. Inferiors reveal everything they have learned from their own inferiors, and everything they can observe themeselves, but they do not reveal the private portions of their cryptographic keys SelfId and PeerId, and they do not reveal the credentials they receive from their aliases.

We have made these assumptions about honest and partially-disclosing inferiors in our base model to reflect the situation obtaining in a hierarchy which demands and receives allegiance from its members, but which does not compel its members to act as spies within other organisations. A refinement to our model would allow us to represent inferiors who do not honour their (apparent) allegiance to at least one hierarchy, or who betray the membership understandings (e.g. for anonymity) of at least one of their peerages.

After the first timestep in our model, all active entities have learned the SelfId of their supervisors, due to the downward broadcast of SelfId through the Inferior channel by all entities. In later timesteps, active entities use these SelfIds to encrypt the credentials they subsequently disclose through their Superior port. All active entities learn the SelfId and PeerId of the other inferiors on this supervisory channel during the first timestep. Thus inferior entities on a supervisory channel learn the SelfId of all entities on their supervisory channel; they also learn the PeerIds of the other inferiors; but they do not obtain any other information about their hierarchy through this channel.

Peerages. Peer ports are on a distinct set of channels which we call peerages. Direct observations through a Peer port are uninformative. Peer-disclosed credentials are signed and encrypted with the PeerId identity key, and are devoid of other information.

As noted above, superior entities are able to observe the PeerId and ObId attributes of their inferiors, and active inferiors disclose their PeerIds and SelfIds to their superiors. An entity with inferiors, who is a member of the same peerage as one or more of its inferiors, will thus be able to pierce the veil of anonymity on that peerage: some or all of the SelfId and ObId attributes of other peers will be known to this superior entity. However the superior will not be able to correlate the peer-disclosed credentials accurately with these identities, except in the extreme case where the peerage is of size two, in which case a process of elimination will quickly distinguish the peer-disclosed credentials of the other peer from their own peer-disclosed ones.

The anonymity situation for a system containing a hierarchy and a peerage is only slightly more difficult to analyse when an inferior has an aliased membership in the same peerage as their supervisor or one of the supervisor's aliases. The inferior does not disclose the SelfIds of its aliases to its superior, for SelfIds are unobservable and the inferior does not forward alias-disclosed credentials onto its Superior channel. However the inferior does disclose the ObId of its aliases to its superior, and the superior does receive copies of all credentials received by its own aliases. So a superior entity may know some ObId information about some or all the peerages in which it has an aliased or direct membership.

Our base model is somewhat unbalanced in its treatment of hierarchies and peerages. Hierarchies will undermine the anonymity properties of peerages, but the uninformative peer-disclosed credentials will not undermine the command-and-control properties of a hierarchy. Our base-model hierarchies are essentially confidentiality systems with the star property of the Bell-La Padula model, in which the inferior entities are unable to learn the confidential information (in our case the system map) reported to the superior by other inferiors. If our CEO model is extended to a general system of trusted computing, we would have to make its peerages more disclosing. In particular, peers in a general-purpose system must be able to conduct anonymous meetings by disclosing credentials containing their proposals, counter-proposals, questions, answers, points of order, and votes. We would introduce a new category of information, and some integrity-focussed objective for our entities in addition to the identity-discovery processes in our base model, to reflect the operations of a hierarchy in the Biba model of security. The maps circulating in our base model provide a system with a measurement of its own state, and in a general-purpose refinement we would expect all hierarchies, peerages, and aliases to employ a Clark-Wilson model of security when regulating the allowable changes to this state. The result of all these elaborations would be a model of much greater complexity than the CEO base model, but with a greatly increased range of applicability. There are many simplifying assumptions in our CEO base model which are appropriate only in identity management systems, resulting from its implicit adoption of the Bell-La Padula model by all its hierarchies, and by the strictly non-disclosing nature of its peerages.

Later in this paper, we will show that the very simple peerages, aliases, and hierarchies of our base model are sufficient to generate the partially-anonymised but validated identity credentials in New Zealand's Identity Verification Service.

Aliases. Alias ports are on a third set of channels which we call aliases. These are distinct from the supervisions and the peerages. In each timestep, active entities disclose their current map of the system through their Alias port. Active entities observe the ObId attribute of all their aliased entities, and they receive all credentials that were disclosed by their active aliases during the first phase of each timestep.

We use aliased entities to model the identification devices and roles possessed by a single person. In our base model, we distinguish aliases only on the basis of activity. If there are two or more active entities in an aliased relationship, these may represent distinct roles which can be played by a single person in their private and professional capacities. An active alias may also represent an agent who is representing (acting on behalf of) its alias. This category includes smart cards, as well as persons holding a power of attorney or other authorising credential for an agency relationship.

We use aliased passive entities to represent the information stored by one entity about another entity. This category includes tokens that are physically in possession of the aliased entity, as well as database entries that are physically removed (but logically linked) to their alias. In our base model, an entity will be able to observe the ObId of its passive aliases, but not their PeerId attribute. The superior of a passive entity can observe both its ObId and its PeerId. Thus a database record, or a token, that is aliased to an entity, may contain identifying information (its PeerId) that is inaccessible to its aliases.

In a refinement of our model, we might distinguish between various forms of agency, roles, tokens, and database records using different types of aliasing. We note that most real-world aliases are not as fully disclosing, nor as reliably honest, as the active entities in our base model. In particular, a computer account on an untrustworthy computer system will not disclose all of its activities to all of its aliased entities. Few, if any, humans would be capable of analysing such traces of activity, even if these were disclosed. We might hope to have trustworthy computerised agents (active aliases) who would scrutinise such reports on our behalf, and of course these reports must only be summaries except when a deep investigation is launched. The development of feasible audit systems for trusted computing or even for the subcase of an identity management system is, clearly, a monumental undertaking. Our CEO model merely hints at its necessity and how it might be structured.

2.2 Taxonomy of Authentications in Our Model

It is customary to distinguish three types of authentications: "what you are", "what you know", and "what you have". These distinctions can be made in our model, but we rename the categories to reflect the full breadth of their coverage.

- Observed. When an entity observes an identifying attribute of an entity, confirming a presumed value for this attribute which was obtained by some independent means, we call this an observed authentication.
- Credentialed. When an entity obtains a credential from an entity, confirming a presumed value for an identifying attribute of this entity which was obtained by some independent and prior means, we call this a credentialed authentication.
- Indirect. When an entity confirms a presumed value for an identifying attribute of an entity, using independent evidence obtained from a third entity, we call this an indirect authentication.

Due to space limitations, we do not explore the definitional nuances, implications, and characteristic vulnerabilities of these taxonomic categories in this paper. We note, in passing, two important subcases of credentialed authentication. The credential may contain a secret, such as a PIN for a credit card, or evidence of the knowledge of a secret, such as a hash of a password. Alternatively the credential may contain additional but non-secret identifying information which corroborates the presumed identity.

We have identified two important subcases of indirect authentication. The indirection may be a credential containing a report of an observation of an aliased passive entity, such as an identification token. Alternatively, it may be a credentialed observation of an active entity who has an inferior (aliased) role in an organisation as one of its trusted third parties.

The New Zealand Identity Verification System (IVS) uses a trusted third party for an indirect authentication of the validated identity. The other subcase of indirect authentication is also used in this system, for a token is used in an indirect authentication of a login at the Government Login Service (GLS) whenever an human activates one of their currently-passive aliases at the GLS. The active forms of these aliases are shown as x_6 and x_7 in Figure 2. Our base model does not support the creation, destruction, or modification of the active/passive state of aliased entities, and our focus in this paper is on the validation of identity at the IVS, so we do not show the login token in this figure. We do show the supervisory path for the third-party indirect authentication of entity x_1 by entity Ref to entity IVS. We will discuss other aspects of this figure, after completing the definition of our model.

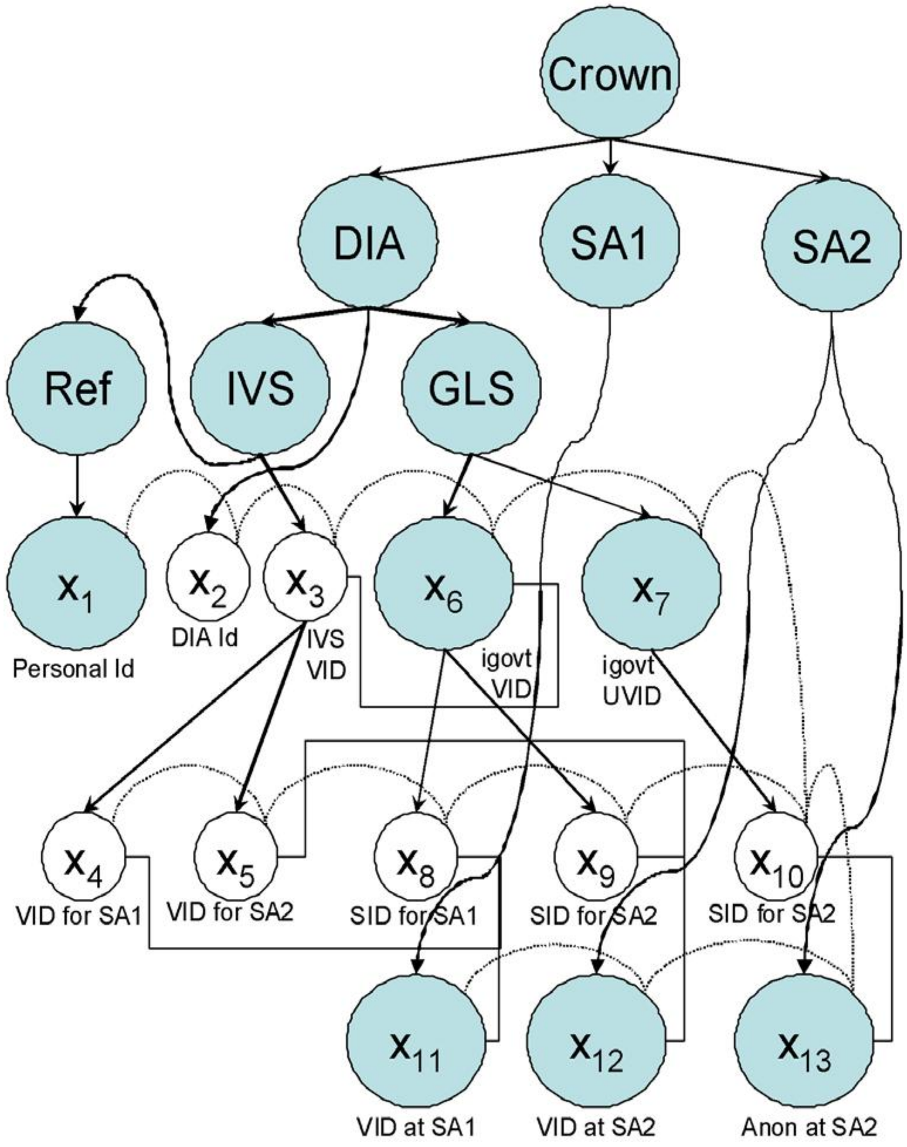


Fig. 2. Thirteen aliased entities in New Zealand's digital identity management system

2.3 Detailed Assumptions in the Base Model

In this section, we briefly outline the detailed, technical assumptions in our CEO model.

The memory and computational capacities of active entities is unbounded: each can store all of its prior observations. New credentials are produced by randomized computations on an entity's stored observations.

The timesteps of each entity are identical. Our systems thus evolve in a lock-step, fully-synchronised manner. In an extended model, the timesteps may be unequal, in which case some entities would be able to produce credentials and to make observations more rapidly than others.

When an active entity publishes a credential on a channel, this credential is delivered to all other active entities with a port on this channel.

When an active entity makes an observation on a channel, the result varies depending on the channel and port. No attributes are observable on Peer channels. The ObId attribute of all connected entities is observable on Alias channels. The ObId and PeerId attributes of inferior entities are observable to any superior entities on a Supervisory channel. Of course, these are not appropriate assumptions for all systems. In refinements, a observation by an alias or a superior may return noisy information about ObId, thereby inducing a non-zero false-positive or false-negative error rate in the identifications and authentications. Aliased entities might be given some power to destroy or detach their tokens, if it is desirable to simulate systems in which inferior entities are able to change (or at least to disrupt) their supervisory relationships.

Active entities are fully honest and revealing through their Alias ports. That is, they always accurately reveal their SelfId value in all credentials they issue to their aliases, and they always accurately reveal everything they have learned (from reading credentials issued by other entities) in each credential they disclose through these ports.

Active entities are fully honest through their Superior ports, disclosing copies of all credentials received through their Inferior port, and all observations made through their Peer and Alias ports.

Active entities disclose only their SelfId through their Inferior port.

Active entities are unrevealing through their Peer port, disclosing only a null credential carrying the PeerId. This disclosure allows peers to discover the size of their peerage.

We impose some structure on observations and credentials, to simplify our descriptions. There is very little loss of generality in this structural imposition, because entities receiving a credential could interpret it by any algorithmic method, using all of its previous observations as contextual clues. The only functional limitation, from an information-theoretic perspective, is that a single credential contains a number of bits which is limited by our entity population-size parameter n.

Since there are up to n entities in our system, a unique identifier for an entity must be at least $\lceil \log_2 n \rceil$ bits long. Ceiling functions ($\lceil ... \rceil$) would be inappropriate in our simple model. In order to limit the proliferation of ceiling functions and rather unimportant constants, we adopt the big-O notation of computational complexity – despite the risk of befuddling our target readership of CEOs and CIOs.

Each field in an observation or credential in our system has $O(\log n)$ bits. A single field in a single observation or credential may unambiguously specify any one of n^c entities, where the value of the constant c is suppressed by our big-O notation.

When our model is used to describe real-world systems, the maximum number of entities, and the length of integers, observations, and credentials will typically be constants. Readers who are uncomfortable with big-O notation should assume that integers in our model are a few thousand bits in length. The maximum number of entities n in such an application of our model is astronomically-large, because $2^{1000} \approx 10^{300}$. One of our integers can hold the public portions of two cryptographic keys, for signing and encrypting an entity's credentials and observations. Other entities, after they learn these public keys, are able to validate signatures from this entity, and to encrypt credentials so that they are readable only by this entity.

In refined applications of our model, credentials and observations may conform to a standard such as X.509 for digital certificates. However X.509 is somewhat too complex for our simple, general-purpose model.

Each credential and observation has a five-field header, $O(n)$ fields in the body, and one field in their terminator. See Table 1. The producer and intended consumer(s) of a credential or observation are named in its ¿From and To fields respectively.

Table 1. Credentials and Observations have five integers in their header, an encrypted list of (Attribute, Value) pairs in their body, and a cryptographic hash Signature. All fields are Integers with $O(\log n)$ bits. Lists have $O(n)$ entries, where n is the maximum number of entities in the system.

Field Name	Type
From	Integer
To	Integer
Subject	Integer
Sequence	Integer
Length	Integer
Body	List of (Attribute, Value) pairs
Signature	Integer

The special value 0 may be used in the To field of a credential, indicating that it is unencrypted: readable by any entity. If any other value appears in the To field, then only entities with the matching SelfId or PeerId value can interpret the Subject, Sequence, Values, and Attributes in the credential.

The special value 0 in the From field of a credential indicates that it is unsigned. Signed credentials have a non-zero value in the From field, and all active entities can validate that the Signature is an accurately-computed keyed hash of the remaining fields in the credential. Only the entities with a SelfId or PeerId value matching the From field of a credential can construct a Signature value which would indicate that the credential is valid.

The Subject field of an observation contains the name of the input port through which this observation was made. In the base model, the Subject field of a credential always contains a special value which we denote "Map", signifying that the credential contains a map of all the entities currently known to the issuing entity. This map also reveals the connectivity of these entities. We construct maps by encapsulating credentials, using a reserved value "Header" in the Attribute field of the body of a credential. The encapsulation protocol adds eight integers to the length of the included credential, so a credential in our model may contain as many as $O(n)$ observations of the SelfId, PeerId, and ObId attributes in our model. Because there are only n entities in our model, and each entity is on at most three non-trivial channels, a single credential can contain a complete map.

The Sequence field contains the timestep of the issuing entity, at the time this credential was produced.

The Length field contains the number of the (Attribute, Value) pairs contained in the body of the credential.

3 New Zealand's Identity Verification Service

Our base model has only a few structures, but these are sufficient to represent the Identity Verification Service (IVS) being developed by the New Zealand government [1]. The IVS is designed to provide strong assurances of citizen identity for the Government Logon Service (the GLS) without violating New Zealand's Privacy Act [3]. The GLS is already in place, and it offers an online, single-signon service, to New Zealand citizens and residents at some service agencies (SA1 and SA2) in Figure 2.

The primary functionality added by the IVS to the GLS is that New Zealand citizens would be allowed to maintain different digital identifiers at different service agencies of the New Zealand government. The unique GLS digital identifier is susceptible to abuse by service agencies, for it would facilitate data matching even in cases where this is not authorised.

In Figure 2, person x_1 has disclosed identity information, including a certificate from a trusted referee (Ref), to the Identity Verification Service (IVS). The IVS has created a verified identity x_3 for x_1 after consulting record x_2 held by the Department of Internal Affairs (DIA). The Government Logon Service (GLS) logon x_6 is verified, and is in a peerage of size two with x_3. Unverified GLS entity x_7 has no IVS peer. Single-signon session IDs x_8, x_9, x_{10} are constructed by the GLS for verified (x_{11}, x_{12}) and anonymous (x_{13}) logons to service agencies SA1 and SA2. Service agencies are assured that the IVS peer (x_4, x_5) of each of their VIDs is a verified alias of a DIA entity x_2, and that this DIA entity has no other aliases with a VID at this service agency.

We note that the PeerIDs of our CEO model give us a natural way to represent the anonymised identifiers provided by the IVS. Aliases x_4 and x_5 are the IVS records of the two validated identities for citizen x_1. These two digital identities were created by the IVS upon the request of the citizen, for use at service agencies SA1 and SA2 respectively. These identities are "validated" in the sense that they are guaranteed to correspond to a live and valid NZ person, and are agency-unique: x_1 cannot create more than one validated identity at any agency. A higher degree of anonymity is possible, if the agency-uniqueness property is not required, for example the unvalidated digital identity x_7 might be used to download forms but not to obtain a government benefit.

We note that anonymity is technically assured by the system of Figure 2, if the entities in the real world behave as predicted by our base model. In our model, service agencies SA1 and SA2 obtain different (and thus unmatchable) PeerIDs for their different views (x_{11} and x_{12}) of citizen x_1. If these PeerIDs were the only information provided by the IVS and the GLS to the service agencies, then interagency data matching would indeed be impossible. However the service agencies are also provided with personally-identifying information (PII) in x_{11} and x_{12}, such as name and birthdate, on a need-to-know basis. Of course it would be quite feasible for agencies to perform approximate data-matching by using the PII, so governmental oversight is still required to assure that unauthorised data-matching is not taking place. The PII data is very "clean", so it would tend to facilitate data matching relative to the "dirtier" identity data that would be available to agencies through a non-computerised system. However the PII provided to each agency is controllable by the IVS. This single locus of control would make it possible for the Crown, or a citizen-controlled peerage, or both, to impose a strict auditing regime on service agencies, the IVS and the GLS.

Our analysis reveals a very intriguing prospect: that the most important civil function of the IVS might be something quite different to its design goal of assuring the privacy and security of online civil interactions with the New Zealand government. The IVS will also provide the technical means for the creation and maintenance of an anonymous peerage on all New Zealand citizens. This peerage, if it is ever established, could become a trustworthy means for civil oversight of governmental operations. Votes in the peerage would be meaningful, because each citizen would be assured of having at most one vote on any resolution put to the peerage. The peerage would, of course, have to defend itself against governmental intrusions, however it is not inconceivable that a democratically-elected government would decide to foster, rather than impede, the development of such a peerage.

If we compare the IVS system to its obvious alternative, of revealing the GLS-generated ObId key for x_6 along with personally-identifying information to all online service agencies, then it is clear that the provision of separate identities (x_{11}, x_{12}) to different service agencies would prevent the data matching that would be trivially accomplished if they were both provided with x_6. Furthermore, if we compare the IVS system to a system in which the service agencies are never provided with accurate identity information from the DIA, then we see many functional and security advantages. For example, the service agencies are assured that no citizen has more than one validated identity in their database, this would greatly limit the scope for benefit fraud. It would also greatly decrease the frequency of confusion, by agencies and citizens, arising whenever an individual has multiple (but non-fraudulent) unlinked identifications at that agency.

4 Discussion and Future Work

Our base model has only two types of entities, three types of channels, and two types of communications (credentials and observations). The actions of these entities are quite simple and deterministic. For this reason, it would be remarkable if our model were able to accurately model all of the information flows in a complex identification system such as New Zealand's Identity Verification System. Indeed, the information flows in our model of the IVS are almost accurate, but not entirely so. In particular, our CEO model would predict that the verification of identity provided by a referee to the IVS would be submitted directly to the IVS. We imagine that our model's predicted route for the referee's declaration was in fact considered by the designers of the IVS. It certainly is a feasible option, however we can see that the clerical work at the IVS is less in the actual scheme: the applicant is required to obtain this document themselves, and to transmit it directly to the IVS with their application. This routing has some advantage in transparency, for the applicant is able to see for themselves what the referee has written. It has some disadvantage in security, for an applicant may impersonate a referee, and an applicant may modify an actual referee's declaration. However if any fraud is suspected, an official of the IVS could contact the referee to confirm the document, and this is precisely the information flow predicted by our CEO model of the IVS. We conclude that our CEO model accurately describes the flow of secure information in the IVS, but not the actual flow in cases where trust (in this case in the veracity of the referee's declaration) has been used to optimise the design.

We have very briefly indicated, in the introduction of this paper, how our CEO model might be extended and then used to guide the architecture of a next generation of trustworthy computing systems: one which would support both hierarchies and peerages, and which would also support (aliased) connections between systems with different security constraints. In the body of this paper, we have indicated how an extension to our model might be used to analyse the security of an identification system. We have also clarified the existing three-category taxonomy of authentication: "what you have", "what you are", and "what you know".

References

1. State Services Commission. Request for Expressions of Interest: All-of-government Authentication Programme - Identity Verification Service - Build Contract. New Zealand (October 2007)
2. Condon, R.: Jericho Forum puts meat on deperimeterisation (March 14, 2008), `http://searchsecurity.techtarget.co.uk/`

3. Edwards, J.: Privacy Impact Assessment of the All of Government Authentication Programme - Identity Verification Service. New Zealand (December 2005)
4. The Jericho Forum. Position paper: Collaboration oriented architectures (April 2008), `http://www.opengroup.org/jericho`
5. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Requirements engineering for trust management: model, methodology, and reasoning. Int. J. Inf. Sec. 5(4), 257–274 (2006)

Pseudonymous Mobile Identity Architecture Based on Government-Supported PKI

Konstantin Hyppönen[1], Marko Hassinen[1], and Elena Trichina[2]

[1] University of Kuopio, Department of Computer Science
POB 1627, FIN-70211, Kuopio, Finland
{Konstantin.Hypponen,Marko.Hassinen}@cs.uku.fi
[2] Spansion International Inc., Willi-Brandt-Allee 4, 81829 Munich, Germany
Elena.Trichina@spansion.com

Abstract. An electronic ID scheme must be usable in a wide range of circumstances, especially in ordinary situations, such as proving your right to a concession ticket on a bus. One of the problems to be addressed is privacy. Indeed, when documents are read by electronic means, a lot of information is not only revealed, but can be copied, stored and processed without our consent. Another issue is ubiquity, reliability and acceptance of the involved technology. In this paper we attempt to address these issues by combining an officially recognised national mobile e-ID infrastructure with identification procedures based on controlled identity revelation. We report a prototype implementation of an identity tool on a mobile phone with a PKI-SIM card.

1 Introduction

A space for an identification card, a driving license, and maybe a passport has been reserved in almost every person's wallet. Considerable amount of information about a person's identity is often revealed to people who actually need only a small part of it. For example, a young-looking customer who buys a can of beer could reveal her full name, social security number, birth place, and other private information to a cashier, only to prove her right to buy the beverage. Such situation, although routinely accepted when dealing with traditional "paper-based" documents, may provide a fertile ground for identity thefts when electronic means are involved. Although people could be concerned about their privacy, various IDs are checked way too often for the privacy to be properly protected.

New types of e-ID documents, such as chip-based identity cards or biometric passports, are mostly targeted at improving the security and usability of former IDs. However, the meaning of "security" here is largely limited to a better resistance to counterfeiting, while the provision of other aspects of security is at the same level or even lower than that in older non-electronic documents. For example, biometric passports could increase the risk of skimming sensitive data, as they can potentially be read from distance [1]. Moreover, some users are reluctant to use electronic IDs, as they are afraid the data read electronically into various systems will be stored there for over-long time.

Another aspect of electronic identity schemes is related to their usability, convenience and universality. An e-ID scheme should enable identification to occur with equal facility in the electronic realm as in the face-to-face situation; in other words, it should have characteristics of a utility [2].

Our contributions. This paper presents an idea of using a mobile phone equipped with a tamper-resistant security element (e.g., a SIM-card) as a "portable privacy proxy". The security element contains as an applet an electronic ID that can (1) work as an officially recognised ID in a number of applications, and (2) provide controlled identity revelation (which is one definition of privacy). The main body of the paper is a description of the architecture and protocols used in the system.

The paper is organised as follows. The next section briefly describes official e-ID schemes based on national e-ID cards, biometric passports and officially accepted PKI-enabled SIM cards. To make a further step in the direction of turning a mobile device into a personal privacy proxy, we state in Sect. 3 security requirements for a mobile identity system with an emphasis on privacy. In Sect. 4 we describe a possible architecture of such a system and related protocol where an identification request is satisfied in a form of a proof which only confirms the identity attributes requested, without revealing any unrelated information. We show that the system can be implemented with today's handset technology and using existing tools and platforms in Sect. 5, which describes in details our proof-of-concept implementation of the pseudonymous mobile identity scheme. A security analysis of the proposed solution is presented in Sect. 6. In conclusion, we emphasise that in this paper we only make the very first step in what may to become an exiting and practical research avenue. We discuss possible extensions of the concept of a mobile phone as a portable privacy proxy.

2 Currently Used Electronic IDs

Governments of many countries all over the world have introduced various types of chip-based identification documents for their citizen. The main objectives for the switch to electronic IDs are their better resistance to forgery, and use in new electronic governmental services [3]. We give a short overview of currently used official electronic IDs here.

Identity Cards. Most of the European countries have introduced electronic identity cards, driven by the demands set by the common European legislation, namely, by the Digital Signature Directive [4]. Also in some Asian countries electronic identity cards are already in use, see `http://www.asiaiccardforum.org`.

As an example of an identity card, we briefly describe here the Finnish Electronic Identity (FINEID) card. The card along with the supporting infrastructure [5] is a system maintained by the Population Register Centre (PRC) of Finland. The card is a usual microprocessor smart card accepted as a valid ID in all EU countries and several others. It contains two Citizen Certificates (public key certificates): the authentication certificate of the card holder and the signature certificate of the card holder. Private keys of both certificates are generated by the card and stored exclusively in its protected memory. Additionally, the card contains the PRC's own Certification Authority (CA) certificate. The card can perform operations involving a private key (e.g., calculate a digital signature) after the user has entered the PIN code corresponding to the key. No biometric information is stored in the chip.

The chip facilities of the FINEID card are mainly used for user authentication in online services. For example, one can request a tax card or check own pension accrual on the Internet, by inserting the card into a card reader and entering the PIN code. It is also possible to use the card for customer authentication in commercial applications. For example, some post, bank, and loyalty applications can be accessed with the card.

Biometric Passports. Biometric passports (also called e-passports or Machine Readable Travel Documents) are now issued by many countries. All data that is included on the information page of the passport is also stored in a chip embedded into the page. In addition to the photo of the passport's owner, also other biometric information is projected to be stored in the chip in the future generations of the passport.

The biometric passport security features are based on the international standards issued by the International Civil Aviation Organization (ICAO) [6]. The standards cover integrity, confidentiality and authenticity of the passport data and the contents of the messages sent between the passport and a passport reader. In the current generation of the passports this is achieved by the mechanism called Basic Access Control (BAC).

In order to communicate with the passport chip, the reader must first optically read the machine readable zone printed in the bottom of the main data page. From this data, the reader constructs the access key which is then used for encrypting the commands sent to the chip and decrypting the answers received from it. However, it was shown in several studies [7,1,8] that part of the contents of the machine readable zone can be easily guessed, making it possible to guess the access key in a relatively short amount of time.

Extended Access Control (EAC), a standard developed by the European Union, can solve the problem of remote skimming, as it requires that the reader must first authenticate itself to the chip. A challenge-response mechanism is used: The reader proves that it possesses the private key for a certificate trusted by the chip. It must be noted, however, that key management is not easy in this scheme. For example, as the chip does not have a reliable source of time, it cannot reliably check whether the certificate presented by the reader is valid. This makes it possible for malicious terminals to use old expired certificates for compromised keys in some cases.

SIM-based Identification and Mobile Identity. The use of the mobile phone as an ID token for user authentication is common in many business applications. As the mobile phone is a highly personal device, subscriber ID can in some cases be used as the customer ID. Relying on the authentication of the (U)SIM card in the phone, mobile network operators can establish trust between the user and a third-party company. The most usual application here is roaming, where the third-party company is a mobile operator in a foreign country. The use of authentication based on SIM card and PIN code is rather limited because it is not fully reliable. The PIN code is asked only on power-on, and if the phone is stolen or lost, it can usually be used by anyone. Therefore, for stronger authentication, SIM Applet Toolkit (SAT) applets with extra PIN codes are used.

The SIM card can work also as an officially recognised ID. In Finland, a mobile subscription customer can request a FINEID-compatible PKI-SIM card from their mobile network operator. Such card can be used for mobile user authentication, and for creating digital signatures legally equal to the handwritten signature.

At the moment, two operators in Finland issue such PKI-SIM cards. The current implementation of FINEID on SIM cards is based on a platform developed by SmartTrust (`http://www.smarttrust.com`). The advantages of the platform include its compatibility with almost any GSM mobile phone, and the availability of all the infrastructure needed (the system is supported by three mobile network operators in Finland). However, there are also certain drawbacks in the current approach. For example, authentication cannot be done without the participation of the mobile network operator, and naturally, operators charge both customers and service providers for authentication. Apparently, this is a major threshold for joining the system: the system has been in place for more than two years, yet only a few service providers and fewer than 200 people use it.

Applets installed on the SIM card or in the phone can be used not only for user authentication, but also as a powerful mobile identity tool. The notion of identity is larger than that of identification. In addition to the identification data, it encompasses other elements that make each human being unique, for example, characteristics that signify membership to a particular group, and establish status within that group [9]. Furthermore, phone-based identity can include payment and personalisation components, and credentials associated with specific information services, enabling pervasive computing with mobile phones [10].

In mobile identity, like in any other electronic identity, issues of security, usability and trust are of major importance. As the mobile phone is constantly connected to a wireless network, it is seen by many users as a potentially insecure device that can reveal sensitive information about one's identity to unauthorised parties. The application looks even more dangerous if it includes a payment component. Therefore, the implementations of any mobile identification or identity must ensure that the user has full control over the data sent by the application. The user must be able to prevent disclosure of any information by the device. Moreover, the systems that need information about the user must receive only a necessary minimum of it, and remove the information as soon as it is not needed. Overall, the "design for privacy" principle should be used [11].

3 Security Requirements

The main goal of this work is a mobile identity system that allows controlled identity revelation. We argue that such system can replace usual IDs in many applications where officially recognised identification is required, at the same time increasing the level of privacy.

The parties participating in the identification or transaction authorisation scenario are Prover (Peggy) and Verifier (Victor). Victor needs a certain proof about Peggy's identity or her rights. There is also a Trusted Third Party (TTP, Trent), which is an official authority that has issued Peggy her electronic identity card. In addition to normal requirements applied to usual travel documents (i.e. authenticity and integrity of stored information, and impossibility of impersonating the holder), we set the following privacy-related requirements:

1. *Minimisation of data collection.* Victor receives only necessary information about Peggy's identity or her rights.
2. *User consent.* Peggy is informed about the set of information requested by Victor. She can either accept of reject the request. Victor receives no information about Peggy if she chooses to reject the request.
3. *Confidentiality.* (a) Nobody can read any information from the mobile identity device, unless Peggy specifically allows this. The property holds true also in the case when the device is lost or stolen. (b) Nobody can learn any information about Peggy's identity by eavesdropping on the communication between Peggy and Victor.
4. *Easiness of revocation.* In case Peggy's mobile identity device is stolen or lost, Peggy can easily place it on a black list of revoked identities, by contacting Trent.

One can see that these privacy requirements are set higher than for usual travel documents. Indeed, it is difficult to restrict data collection from usual identification documents. Moreover, revoking them is also cumbersome: In most cases, a criminal can use a stolen passport or driving license until its expiry date. In what comes to these two problems, we want to provide some significant improvements with our mobile identity tool.

4 Mobile Identity Tool: Architecture and Protocols

The main logical component of the system is the user's public-key certificate which, however, does not include any data about the user except a hash of one selected biometric feature. For the sake of clarity we assume that a digital photograph of the person is used as the biometric.

The certificate is stored on the SIM card, as part of a mobile identity management applet. The applet contains the following information (see also notation in Table 1):

– The user's certificate, signed by the Certification Authority (CA).
– The user's private key corresponding to the public key in the certificate.
– The CA certificate.
– The user's photo, hash of which is included in the certificate.
– A number of fields containing the user's personal information, such as name, surname, date of birth, social security number, nationality, and so on.

Table 1. Protocol Notation

P, V	Parties: Peggy (prover), Victor (verifier)
BIO_X	Biometric characteristic (e.g., photo) of subject X
SKEY_X	The secret RSA key of subject X
PKEY_X	The public RSA key related to SKEY_X
CERT_X	The public key certificate of subject X
$\{m\}_K$	RSA encryption of the message m under the key K
$H(m)$	A cryptographic hash of the message m; We use SHA-1 as the hash function in our protocols
MSG	A message
SIG	A digital signature
TIME_X	A timestamp generated by subject X
$+$	Separator between the contents of message and its signature

In addition, the applet contains a program for operating the information listed above. The basic requirement for the program is its ability to decode requests arriving at it, and provide requested information. The information is provided in form of *proofs*, structure of which is described later on. The proofs are signed with SKEY_P.

Another program, running as part of the phone software, is used for mediating requests and proofs between the SIM card applet and the outside world. The following protocol is used for requesting information about the user.

1. Request composition and transfer. Victor prepares a request for Peggy's personal information. The request contains a number of triples ⟨data field identifier | operator | [value]⟩. The data field identifier denotes the field containing certain type of Peggy's personal information. The operator is one of $=$, $<$, $>$, $\neq$. The comparison value is optional: For instance, a request for the subject's surname is formed as ⟨surname identifier | = | ⟩. In case the operator is other than =, a value must be specified. For example, if Victor wants Peggy to prove that she is over 20 years old, he sends ⟨date of birth identifier | < | 7.1.1988⟩, if the transaction takes place on 6.1.2008. Victor attaches his public-key certificate and a timestamp to the request, and signs the request with his private key. Then he sends the request to Peggy.

$$\mathrm{V} \rightarrow \mathrm{P}: \quad \langle \text{field id} \mid \text{operator} \mid [\text{value}] \rangle \mid \langle \ldots \rangle \ldots \mid \mathrm{CERT}_V \mid \mathrm{TIME}_V \; + \; \mathrm{SIG}$$

2. Consent acquirement. Software at Peggy's phone verifies Victor's certificate and signature. If verification is successful, the phone asks Peggy for a confirmation, showing her Victor's request. If Peggy chooses to comply with the request, she enters her PIN code. If Peggy has entered her PIN code, the request is forwarded to the SIM card. In addition, current time TIME_P is attached to the request.

3. Proof construction. Having received the request, the SIM card decodes it and constructs the proof of the form

$$\mathrm{P} \rightarrow \mathrm{V}: \quad \langle \text{field id} \mid \text{operator} \mid \text{value} \rangle \mid \langle \ldots \rangle \ldots \mid \mathrm{CERT}_P \mid \mathrm{TIME}_P \mid \mathrm{BIO}_P \\ + \; \mathrm{SIG}$$

In the calculation of the signature, Peggy's certificate and biometrics are not included (only field IDs, operators, field values and the timestamp are used). The proof is sent by the SIM card to the mobile phone software, which encrypts the proof with Victor's public key and forwards it to Victor. As an optimisation, it is not necessary to send Peggy's certificate or her full-sized photo from the SIM card every time when a proof is needed. Instead, the certificate and the photo can be retrieved from the card by phone software and stored therein.

4. Proof verification. Having received the proof from Peggy, Victor first checks the validity of the certificate presented by Peggy, and verifies her signature of the proof. Then Victor calculates the hash of the picture and compares it with the subject name field in the certificate. After that, he verifies that Peggy looks the same as in the picture received along with the proof. The picture is shown to Victor on screen.

An example of an identity proof scenario implemented by this protocol is presented in Fig. 1.

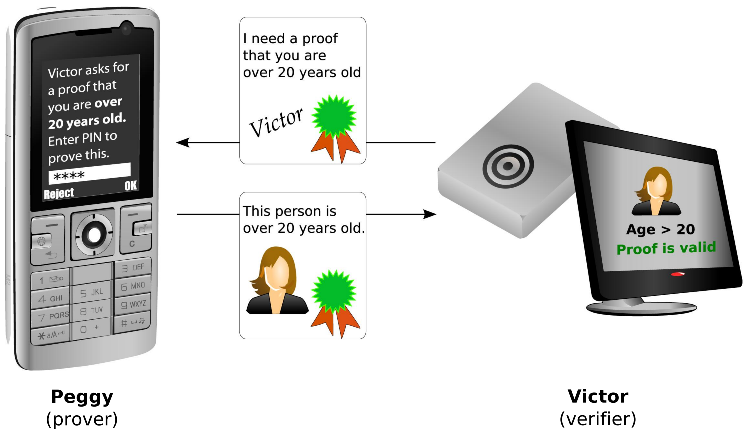

Victor asks for a proof that you are over 20 years old. Enter PIN to prove this.
****
Reject
OK
I need a proof that you are over 20 years old
Victor
This person is over 20 years old.
Age > 20
Proof is valid
Peggy
(prover)
Victor
(verifier)

Fig. 1. An example of an identity proof scenario

5 Implementation

We suggest to use Near Field Communication (NFC) (`www.nfc-forum.org`) as the interconnection technology between Peggy's mobile phone and Victor's terminal. The reasons for this choice (instead of, e.g., Bluetooth) are usability and speed. Bluetooth requires time consuming pairing, while with NFC, pairing is not required. GPRS or other packet data connection technologies are not suitable because we want to minimise the communication costs, allow offline transactions, and improve usability. Currently, only a few models of mobile phones with NFC support are available on the market. However, the penetration of NFC as the communication technology has been predicted to grow rapidly [12].

NFC phones that support Contactless Communications API [13] enable applications to get launched when a certain tag or reader is touched by the phone. This means that in order to provide information about herself Peggy has to put her phone close to the NFC reader attached to Victor's terminal, verify the request shown on the phone screen, and enter the PIN code.

We have developed a proof-of-concept implementation of a mobile identity tool. We used J2ME with Contactless Communications API [13] and Security and Trust Services API (SATSA) [14] for the implementation of the phone software, and Java Card for the implementation of the SIM card software. SATSA is used to facilitate exchange of messages between the J2ME applet and the SIM card. At the moment, none of the mobile phones that support NFC have support for SATSA (and vice versa). Therefore, we tested the application using a mobile phone emulator (Nokia 6131 NFC SDK). For testing the application with real NFC phones, we also implemented a version which does not use SATSA API, but instead stores all user identity information in the J2ME applet. The implementation scheme is outlined in Fig. 2.

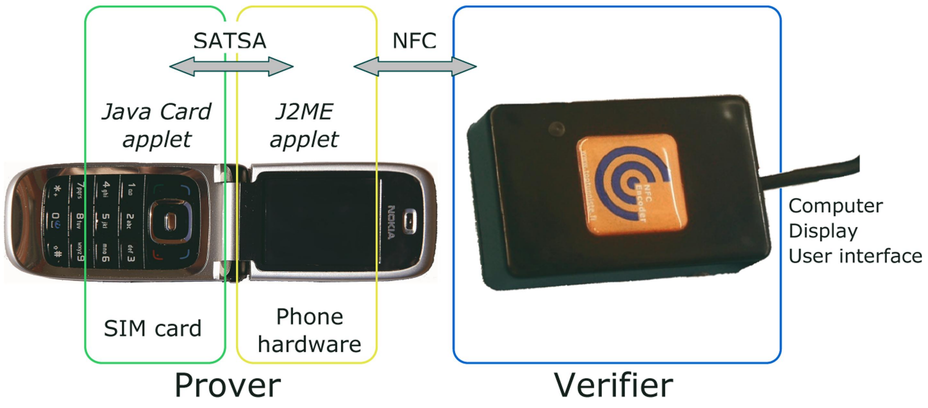


Fig. 2. Implementation: main idea and technologies involved

In the NFC+SATSA version of the application, the Java Card mobile identity applet was implemented using the JCOP Tools SDK [15], and stored on a JCOP31 smart card. The J2ME applet communicates with the SIM card using SATSA. However, in the phone emulator this communication is forwarded to the smart card placed in a smart card reader. We implemented a small mediator program which accepts commands arriving from the phone emulator (they are sent as messages of TLP224 protocol), sends them to the card, and forwards the card's responses back to the emulator. The emulator was running within the Sun Wireless Toolkit. Required certificates were generated using OpenSSL.

Although a certificate with a hash of a photo as the subject name could seem to be difficult to implement and standardise, this is actually not the case. A famous quote by Bob Jueneman on the IETF-PKIX mailing list is "...There is nothing in any of these standards that would prevent me from including a

1 gigabit MPEG movie of me playing with my cat as one of the RDN components of the DN in my certificate." Indeed, generation of such certificates using OpenSSL turned out to be easy.

Because none of the APIs supported by the Nokia 6131 phone provide all required functions for cryptographic processing, we have used an additional package available from the Legion of Bouncy Castle (`http://www.bouncycastle.org`). The package provides classes for encryption and decryption, calculating digests, and handling certificates, among others. However, there are no ready-made tools for certificate signature verification in Bouncy Castle. We implemented these procedures ourselves.

As the identity field codes, we used standard X.509 attribute codes [16]. For cryptographic processing we used a package available from the Legion of Bouncy Castle (`http://www.bouncycastle.org`).

6 Security Evaluation

In the Sect. 3 we provided an outlook on requirements that a mobile identity tool must implement. Here, we evaluate our mobile identity tool with respect to these requirements.

Requirements on minimisation of data collection and on user consent are connected. If Victor requests too much information, Peggy would (1) reject his request and (2) make a complaint to authorities, appealing to Victor's privacy policy. On the other hand, Peggy cannot control what information is stored in Victor's system after a transaction. However, the same drawback is in all other electronic identity tools described in the Introduction section. Ideally, Victor's software and hardware has to be evaluated by an independent laboratory and certified to meet privacy standards, to make sure that only necessary information (if any) is stored therein.

Information about Peggy's identity is securely stored on a tamper-resistant SIM card by Trent and is not modifiable afterwards. Identity proofs are signed using Peggy's private key stored only on the card. Although one might argue that our scheme is dangerously dependent on the tamper-resistance of the card, we strongly believe the situation is not different from normal electronic identity cards. Indeed, if Mallory can find a way to tamper with the card to change identity field values, he can use the same way to alter the private key. The level of protection provided by smart cards is sufficient for identity documents: many countries issue smart card based electronic IDs. Victor can therefore be confident that the proof is authentic and has not been changed in transit. The proof is encrypted with Victor's public key, making it difficult for an eavesdropper to learn any information about Peggy by listening to the communication. To impersonate Peggy, an attacker needs to get access to Peggy's mobile identity device, learn Peggy's PIN code associated with the identity applet, and have a similar appearance as in Peggy's photo.

In case Peggy's mobile identity device is stolen or lost, Peggy can contact Trent and place her identity on a black list of revoked identities. The same type of blacklisting as that for usual credit cards can be used in this case.

We stress the fact that the mobile identity tool presented in this paper is pseudonymous rather than anonymous. Indeed, Victor can cooperate with other identity verifiers to get more information about Peggy, by collecting all information that corresponds to a given hash of the photo. The importance of checking and standardising Victor's equipment and software is accentuated by this observation. Compared with other electronic IDs, identity theft in this model is anyway more difficult.

Additional protection of the mobile identity tool is provided by the NFC technology. Its operational distance is about 2 cm in modern mobile phones, which makes it difficult for an attacker to access the phone covertly. Even if the attacker uses a powerful directional antenna for extending the range of NFC communication, information cannot be read without Peggy's consent.

Yet another attack idea is to install a "virus" to Peggy's mobile phone, to have all information about Peggy's identity transmitted to the attacker (for example, over GPRS). However, this attack is hard to implement because of the protections provided by SATSA. To connect to an applet in the SIM card, the J2ME applet ("virus", in this case) must be digitally signed using the operator domain or TTP domain certificate. In the SIM card there is an access control entry with a hash of this certificate. The implementation of SATSA on the mobile phone compares this hash with that of the certificate used for signing the J2ME applet. The J2ME applet is allowed to exchange messages with the FINEID applet only if the hashes match. This security mechanism is in place to ensure that the J2ME applet has not been tampered with. Therefore, the attacker must have the "virus" signed by the operator or TTP before it can be used for identity theft. Another option would be to find a serious vulnerability in the implementation of SATSA and exploit it.

We summarise the security features of the mobile identity tool, and compare it with other electronic IDs in Table 2.

Table 2. Security features of our mobile identity tool, compared with other e-IDs

Security feature	Electronic ID cards	Biometric passports	SIM-based ID (Smart-Trust)	Mobile Identity
Minimisation of data collection	–	–	–	Yes
User consent	Yes[1]	–	Yes[1]	Yes
Authentication and integrity	Yes	–	Yes	Yes
Confidentiality	No/Yes[2]	–	Yes[3]	Yes
Easiness of revocation	Yes[4]	Yes[4]	Yes	Yes

[1] Only request for *all* identity information can be accepted or rejected.
[2] From stolen cards, information can be read.
[3] Mobile network operator can always read all information.
[4] Not for offline transactions.

7 Conclusions

This paper described how to use a mobile phone as an electronic ID. One of our main achievements is controlled identity revelation that helps protect the user against identity theft. With the use of a governmental PKI this ID can become officially recognised so that it can be used in various official purposes. At the same time, the amount of identity information the user has to surrender can be minimised.

We set strict security requirements for our electronic ID, namely, minimisation of data collection (the verifier gets the minimum necessary amount of personal details), user consent (all actions regarding personal details have to be approved by the user), authenticity and integrity (all parties are strongly authenticated and data integrity is guaranteed), and easiness of revocation (in case of compromised credentials, it is easy to revoke those credentials).

Although we have used the photo of the user as the biometric identifier in our examples and the proof-of-concept implementation, any other biometric (e.g., fingerprints or iris codes) can be easily used in the system. We gave a comparative security analysis of our scheme in contrast to similar schemes available today. It showed that our scheme can provide better security than any of the rivals.

In addition to the security and privacy features implemented by our mobile identity tool, we want the mobile identity to be dynamic. In other words, it should be possible to add new information, rights and certificates to the device. It should be also possible to revoke incorrect information, expired rights and compromised keys from the device. We are currently working on extending the architecture to meet these important requirements and also to include the support for various identity profiles (professional activity profile, entitlements profile, etc.) In the future work, we shall also provide a detailed description of the infrastructure for issuing and updating mobile identity -enabled SIM cards.

References

1. Juels, A., Molnar, D., Wagner, D.: Security and privacy issues in e-passports. In: Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference, pp. 74–88 (2005)
2. McEvoy, N.A.: e-ID as a public utility. Consult Hyperion, Guilford, UK (2007), `http://www.chyp.com`
3. CEN/ISSS Workshop eAuthentication: Towards an electronic ID for the European Citizen, a strategic vision. Brussels (2004) (accessed 10.10.2007), `http://europa.eu.int/idabc/servlets/Doc?id=19132`
4. The European Parliament and the Council of the European Union: Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. Official Journal L 013, pp. 0012–0020 (2000)
5. Population Register Centre of Finland: What is the citizen certificate? Helsinki, Finland (2005) (accessed 10.10.2007), `http://www.fineid.fi/vrk/fineid/home.nsf/en/products`

6. ICAO: PKI for machine readable travel documents offering ICC read-only access, version 1.1. Technical Report (2004)
7. Witteman, M.: Attacks on digital passports. Talk at the What The Hack conference (2005) (accessed 10.10.2007), `http://wiki.whatthehack.org/images/2/28/WTH-slides-Attacks-on-Digital-Passports-Marc-Witteman.pdf`
8. Hoepman, J.H., Hubbers, E., Jacobs, B., Oostdijk, M., Schreur, R.: Crossing borders: Security and privacy issues of the european e-passport. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., Kawamura, S. (eds.) IWSEC 2006. LNCS, vol. 4266, pp. 152–167. Springer, Heidelberg (2006)
9. Roussos, G., Peterson, D., Patel, U.: Mobile identity management: An enacted view. International Journal of Electronic Commerce 8, 81–100 (2003)
10. Roussos, G., Marsh, A., Maglavera, S.: Enabling pervasive computing with smart phones. IEEE Pervasive Computing 4, 20–27 (2005)
11. The Royal Academy of Engineering: Dilemmas of privacy and surveillance: Challenges of technological change. The Royal Academy of Engineering, 29 Great Peter Street, London, SW1P 3LW (2007)
12. ABI Research: Twenty percent of mobile handsets will include near field communication by 2012. London, UK (2007) (accessed 10.10.2007), `http://www.abiresearch.com/abiprdisplay.jsp?pressid=838`
13. Java Community Process: Contactless Communication API, JSR 257, v. 1.0. Nokia Corporation, Espoo, Finland (2006) (accessed 10.10.2007), `http://www.jcp.org/en/jsr/detail?id=257`.
14. Java Community Process: Security and Trust Services API (SATSA) for Java^{TM}2 Platform, Micro Edition, v. 1.0. Sun Microsystems, Inc., Santa Clara, CA, USA (2004) (accessed 10.10.2007), `http://www.jcp.org/en/jsr/detail?id=177`
15. IBM Zurich Research Laboratory: JCOP Tools 3.0 (Eclipse plugin). technical brief, revision 1.0 (accessed 10.10.2007), `ftp://ftp.software.ibm.com/software/pervasive/info/JCOPTools3Brief.pdf`
16. Santesson, S., Polk, W., Barzin, P., Nystrom, M.: Internet X.509 public key infrastructure qualified certificates profile. Network Working Group, Request for Comments 3039 (2001) (accessed 10.10.2007)

Para-Virtualized TPM Sharing

Paul England and Jork Loeser

Microsoft Corporation,
One Microsoft Way, Redmond, WA 98052, USA
{paul.england,jork.loeser}@microsoft.com

Abstract. We introduce a technique that allows a hypervisor to safely share a TPM among its guest operating systems. The design allows guests full use of the TPM in legacy-compliant or functionally equivalent form. The design also allows guests to use the authenticated-operation facilities of the TPM (attestation, sealed storage) to authenticate themselves and their hosting environment. Finally, our design and implementation makes use of the hardware TPM wherever possible, which means that guests can enjoy the hardware key protection offered by a physical TPM. In addition to superior protection for cryptographic keys our technique is also much simpler than a full soft-TPM implementation.

An important contribution of this paper is to show that a current TCG TPM 1.2 compliant TPM can be multiplexed easily and safely between multiple guest operating systems. However, the peculiar characteristics of the TPM mean that certain features (in particular those that involve PCRs) cannot be exposed unmodified, but instead need to be exposed in a functionally equivalent para-virtualized form. In such cases we provide an analysis of our reasoning on the right balance between the accuracy of virtualization, and the complexity of the resulting implementation.

1 Introduction and Problem Statement

Hardware virtualization is about to become commodity on computers ranging from servers to handhelds. With this change security concerns arise, including the question of how to establish trust in operating systems running in these virtualized environments. One straightforward approach is to extend the established TPM-based model of a trusted software stack to the hypervisor (Figure 1). Such techniques should prevent adversarial virtualization by hyper-root-kits [1], and provide a platform for advanced security services [2].

However, to deliver on this vision we need mechanisms to extend the TPM notion of OS authentication into guest operating systems. In particular we must offer ways for guests to authenticate themselves (attestation) and to protect secrets from other operating systems and hypervisors that might run on the platform (sealing).

Two approaches to providing trusted computing services to guest operating systems have been advocated. Goldman and Berger describe enhancements to the current TPM design that allows the creation and maintenance of full "virtual TPM" contexts inside a physical TPM [3]. The other approach—also explored by authors at IBM—is to build a virtual TPM in software in the virtualization layer [4].

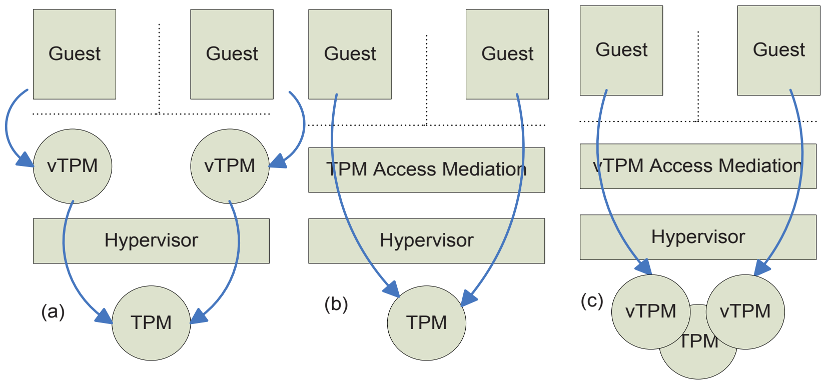


Fig. 1. Schematics illustrating extensions of the trusted software stack into the setting of a virtual machine. In 1(a) the hypervisor creates a virtual TPM for each guest. The hypervisor also creates and certifies the vTPMs, and the hypervisor and vTPM can be used to authenticate the running guest. Guests have no direct access to the hardware TPM in this case. In 1(b) the hypervisor maintains the chain of trust all the way to the guest, and provides mediated/paravirtualized hardware TPM access for each guest (the solution described in this paper). In 1(c) we illustrate a hardware TPM that has been enhanced to support virtual machines.

This paper introduces a third approach: we describe how an existing version 1.2 TPM can be safely shared amongst two or more guest operating systems in such a way that each can use the full complement of TPM functions. The approach has the advantages of simplicity and preservation of hardware protection for TPM-protected keys. Unfortunately, aspects of the TPM design preclude us from exactly maintaining the TPM1.2 interface. However, as we will show, the resulting design supports most legacy applications without changes: only (typically system) software that uses PCRs or related services needs to change to use our para-virtualized TPM design.

2 Architecture

2.1 Architectural Alternatives

To expose a TPM or TPM-like services in a virtualized environment, three approaches are worth exploring:

SoftTPM: Implement a piece of software that provides an interface very close to a real hardware TPM. The SoftTPM implementation might internally use a real hardware TPM, e.g. to secure its own secrets or to provide a proven boot log, but most SoftTPM operations are implemented entirely in software.

TPM Hardware Context Extension: Modify the hardware TPM to provide a separate context for each guest. These guest TPM contexts could be saved and loaded, so that the hypervisor could transparently provide isolated TPM sessions for its guests. [3]

TPM Para-Virtualization: Para-virtualize the TPM and mediate guest accesses by a software component that enables safe and fair device sharing. This design will pass through most of the functionality of a real TPM, but some aspects of the device interface will change.

These options are illustrated in Figure 1. One problem of the software TPM solution is the complexity of the hypervisor and the TPM implementation, and the resulting probability of exploitable bugs. In contrast—as we will show—a para-virtualized TPM solution requires far less added complexity. Also, a physical TPM provides much greater protection against physical tampering than a TPM implemented in software. For instance is it very hard to steal a private key from a hardware-based TPM whereas keys can be extracted from main-CPU code by simple hardware probes.

Extending the existing TPM design through the addition of guest contexts in hardware seems a viable approach for providing TPM functionality in a virtualized environment. This approach can provide a familiar legacy interface to guests, while providing greater protection for keys than that which is practical in the SoftTPM solution. However, virtualization-supporting TPMs are not available, and the need for security in virtualized environments is pressing.

We claim that TPM para-virtualization provides close to the best of both worlds: By multiplexing an existing hardware TPM, it provides the maximum "value pass-through" of the TPM to virtualization guests while preserving TPM key storage robustness of a physical TPM. However - as we will show – the special characteristics of the TPM mean that it is not possible to provide a full high-fidelity legacy TPM interface: instead we must expose an interface that is functionally equivalent, but is not fully compatible. This means that some programs must be ported to use the para-virtualized interface. Our analysis suggests that most user-level applications can run on the paravirtualized interface without modification, but certain OS functions and system services must be changed (in particular those using PCRs).

We also note that given the services provided by the para-virtualization, it should be possible to implement a SoftTPM (or more advanced functionality) as an additional service. This argues that para-virtualization is appropriate as a lowest-level service on which richer functionality can be built if needed.

2.2 TPM Virtualization Requirements

In this section we elaborate our requirements. The goals that motivate our design are:

1) To provide the maximum "safe" exposure of the TPM to upper-level software,
2) To isolate the activities of each guest from the actions of the others
3) To support authenticated boot of guests (recursively), and
4) To work with existing TPM devices (version 1.2)

In this context "safe" means that a guest using the TPM cannot impersonate or manipulate the identity of the hypervisor or other guests.

To share the TPM we must provide a service that schedules access to the underlying hardware, virtualizes limited TPM internal resources, and provides guest authentication services. Important TPM resources that must be shared or virtualized include

(1) Platform configuration registers (PCRs), (2) The TPM owner, (3) The Endorsement key (EK), (4) The storage root key (SRK), (5) Other keys in protected storage, (6) monotonic counters, (7) the delegation table, (8) authorization and transport sessions, (9) non-volatile storage, (10) the random number generator.[1] In the remainder of this section we describe these resources and considerations for para-virtualization.

PCRs: As individual guests have individual virtualized hardware configurations and OS images, we clearly have to virtualize PCRs if the TPM is to provide these guests with software authentication services. To do so, we propose para-virtualizing PCRs into virtual PCRs (vPCRs). Section 3.1 describes the details of our approach.

TPM Owner: The TPM is designed to be able to authenticate the authorized owner and protect itself from programs with direct access to the TPM launching a denial of service or other attack on the TPM. It does so by means of an access-control model in which an authenticated entity known as the owner has privileges to make configuration changes that have security, availability, or privacy implications. In a hypervisor setting we can continue to rely on the TPM itself for the owner security model, or we can expose a subset of TPM functionality based on a hypervisor-notion of which guests are permitted administrative access. In the case of the former, the TPM delegation capabilities allow guests to be granted a subset of full owner rights. For example, a guest may have the ability to create AIKs, but not clear the TPM. Conversely, if the actual owner password is revealed to the guest it can perform all owner-authorized actions—such guests have privileges to deny service, e.g. by clearing the TPM.

Endorsement Key (EK): Our model of para-virtualization is such that all guests safely share the underlying TPM and the endorsement key contained within it. Thus anyone authorized (with the owner password or an authorized delegation) should be able to use the EK. This design differs from soft-TPM or hardware TPM designs enhanced to support virtualization, which are likely to have per-VM endorsement keys. We note that designs that have per-guest endorsement keys need mechanisms to have the endorsement keys meaningfully certified for chains of trust to hardware to be maintained.

Storage Root Keys (SRK): The storage root key (and the associated TPM-proof key) provides for cryptographic protection of keys held outside the TPM. There are essentially two choices for allowing key-storage services to virtualization guests:

1) Everyone can share the hardware SRK. This implies that everyone must know the SRK authorization.
2) The TCB provides each guest with a child storage key off the SRK to use as "their SRK". Unfortunately, AIKs can only be created as children of the hardware-SRK, so this approach denies access to important TPM functionality. We believe that the AIK parentage restriction is burdensome and unnecessary, and should be removed in a later version of the TPM specification, but at the present time this does not appear to be an acceptable approach.

[1] A few features like self test and the get-capability family of operations are omitted here for brevity.

The first solution appears most attractive, but we note that solution (2) (with the fix noted) would allow VM migration by migrating an associated child-SRK to a target platform.

Other Keys: Essentially everything in “protected storage” in the TPM is based on a loaded key, and works for any key in the hierarchy. This means that once a guest has loaded a key against the real SRK (or an SRK child), it can do everything it needs to do without the involvement (or even knowledge) of any other guest on the platform. This should not be unexpected: although the TPM does not appear to be designed to support simple virtualization, it was designed to be shared amongst independent applications each using different sets of keys.

Note however that keys must be loaded into the TPM to be used, and they are loaded into a storage area of finite size. Keys are assigned a handle by the TPM when they are loaded, and this handle must be used in subsequent operations (e.g. signing, and unloading of the key). The TPM provides facilities for loading and unloading contexts that allow the key slots to be transparently virtualized by the TCB.

Counters: We see no way that a counter can be transparently shared. If non-trivial changes of the counters consumers are acceptable, trusted counter schemes such as presented in [7] can be applied without increasing the TCB.

If monotonic counters with similar behavior to the TPM counters are required, the TCB must implement monotonic counters in software (and use hardware counters to prevent state rollback as well). This implementation however is much closer to full virtualization than para-virtualized sharing.

We argue in Section 0 that although it is possible to provide a virtual monotonic counter implementation, it is probably better for the TCB to provide a semantically richer service such as general-purpose storage that is rollback protected.

Delegation Table and Delegation: Most delegation operations are virtualizable: If a guest has a key or owner authorization (possibly delegated), it can create a delegation blob with an associated password (or PCR set) than can grant another entity the guest’s rights of a subset thereof.

The exception is the family table and the delegation table, which are finite shared resources. If we take the view that there is a single TPM owner that is granting delegations to guests and users then this owner is responsible for the maintenance (sharing or partitioning) of the shared tables.

Authorization Sessions: OIAP and transport sessions can be context-saved and context-loaded by the TCB in the same manner as key slots, and hence are virtualizable. The TPM limits the number of saved context sessions, and this may be problematic under heavy load. Library code should be provided to recover in cases where hardware limits result in session loss (or the TPM design should be enhanced to remove this limitation).

Non-volatile Storage and DIR: Non-volatile (NV) storage is mainly provided to hold initial machine setup information. Virtualization environments provide far better and flexible solutions for guest-data-storage so we believe that simple partitioning of the NV regions is appropriate. DIR registers provide an alternative storage mechanism:

once again we see little benefit in virtualization and believe simple partitioning is preferred.

Random number generator: It is safe to share the RNG.

From this analysis we conclude that that most TPM functionality and state can be safely multiplexed or partitioned without significant loss of functionality. However, in order for the authenticated operation component of the TPM (sealing, attestation, etc.) to be exploited we need to provide a mechanism for PCR virtualization. We describe a solution to this problem in the next section.

3 TPM Virtualization Design

In this section we provide the high-level design that results from the requirements and other considerations of the last section and the interface to the resulting virtualized TPM. A summary of the implementation costs is given in the next section.

The actual TPM para-virtualization code can be located at various places as long as it is well isolated from non-TCB components. We have chosen to implement this functionality in the hypervisor since our implementation is simple and is a TCB security service, but it could also be part of a trusted guest.

Fig. 2. shows the high-level view of the resulting major software modules and their interconnection.

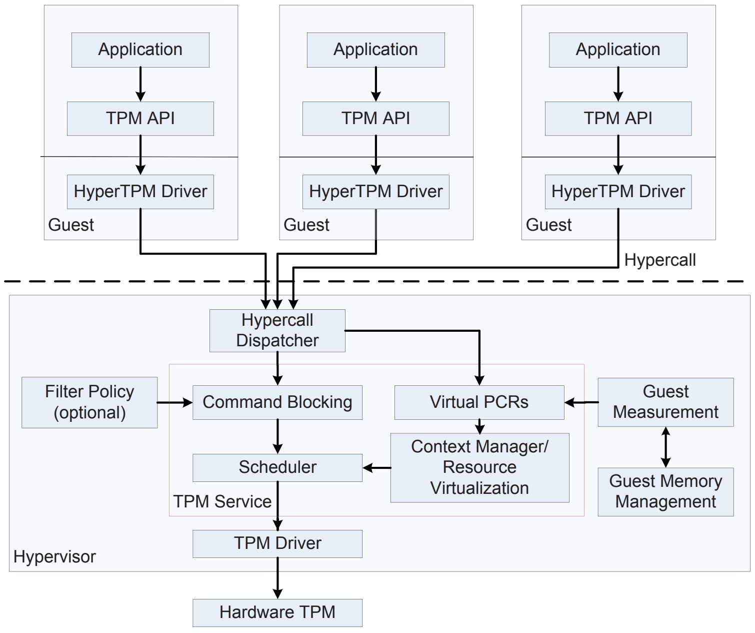


Fig. 2. TPM Support in the Hypervisor

At the lowest level, the **TPM Driver** interfaces with the actual TPM hardware.

Communicating directly with the TPM driver is the **TPM services component** in the hypervisor. It contains the following subcomponents:

- **Scheduler** – Schedules shared access to underlying TPM hardware.
- **Command Blocking** – To preserve the integrity of operations, certain TPM commands must not be executed by arbitrary software on the platform (this is described more in Section 3.4). The command blocking module filters commands (based on a list of allowed commands or on more advanced filter policy) to provide the maximum "safe" exposure of the TPM to upper-level software as discussed in Section 2.2.
- **Virtual PCRs** – Provides one set of hypervisor-managed "vPCRs" for each guest. Section 3.1 discusses PCR virtualization in more detail.
- **Context Manager** – Handles contexts of concurrent TPM clients. In order to ensure that different entities (guests) cannot access each other's resources, it is necessary to associate each command submitted to the TBS with a context specific to that guest. Section 3.5 discusses the details.
- **Resource Virtualization** – Virtualizes certain limited TPM resources, including key-slots, authorization session slots and transport sessions and binds them to a context provided by the context manager.

The **Guest Measurement** component is responsible for measuring the state of a guest in the same way that platform firmware is responsible for measuring the state of system code on a physical platform. Results of that measurement are a guest identity, which are stored in a virtual PCR. From there, they can be used similarly to hardware measurements placed in real PCRs for sign/unseal operations. Section 3.7 discusses guest measurement in more detail.

The **TPM API** library provides a programmatic interface for application access of the TPM. Many different levels of abstraction or possible including those described in the Trusted Computing Group's TSS specification [5].

The **HyperTPM Driver** within the guests communicates command byte streams and results to and from the TPM service component using a defined hypercall.

3.1 PCR Virtualization

As discussed in Section 2.2, we propose para-virtualizing the platform configuration registers to provide guests with VM-level attestation and sealed-storage services. This section first recaps PCRs and then introduces our virtualization approach.

PCRs are a set of hash-sized registers in the physical TPM. Their purpose is to provide a simple and flexible mechanism for representing the software running on a platform. Since the TPM is a passive device that cannot observe code executing elsewhere on the platform, firmware on compliant platforms is responsible for measuring and reporting the identity of software as it is initialized. The initial state of PCR registers is architecturally defined (generally zero). Most PCRs are only reset when the containing platform is itself reset to an architecturally defined starting state, but some PCRs can be reset by programs running on the platform. The only other way to modify a PCR is by means of the *Extend* function, which updates the value to the cumulative hash of the previous value and the parameter of the Extend function.

Semantically, PCRs are a compact representation of a secure log: once something is logged, the log entry cannot be removed until the platform is reset. This behavior was designed to support authentication of the platform's trusted computing base since.

A small number of resettable PCRs were introduced in the 1.2 version of the TPM specification. We will use this functionality to provide guests with guest-level authentication services that are compatible with other TPM functions.

In keeping with the TPM design, the vPCR module in the hypervisor provides each guest with a software implementation of PCR registers with identical behavior to those implemented in hardware. We call these *virtual PCR registers* or *vPCRs*. Since vPCRs are external to the TPM they cannot be directly used in quoting or other operations. However, the TCB can load a representation of a guest vPCR (or collection of vPCRs) and their contents into a resettable hardware PCR when that guest has TPM-focus. Once loaded, quote and similar operations can reference this PCR (and the associated vPCRs) by standard hardware PCR-selection. Because we are using a resettable hardware PCR, the contents can be swapped when another guest gets focus, or when the guest selects a different set of vPCRs.

Guests are free to select any combination of hardware and software PCRs to suit their needs. For example, if a guest needs to authenticate the hypervisor on which it is running it may select the appropriate hardware PCRs in a call to "quote." If it wishes to authenticate itself independent of the hypervisor it instructs the vPCR module to select appropriate virtual PCRs, and then calls quote "specifying" only the *resettable* hardware PCR used by the vPCR module. Finally, if an OS needs to authenticate itself and its hosting environment it can select both vPCRs and appropriate hardware PCRs. Further details are provided in the next section.

If this functionality is to be useful, the hypervisor must measure guests as they are initialized and store the results in the guest's vPCR registers. The problem of defining the secure representation of a guest is very similar to defining the secure representation of a classical (non-virtualized) OS, with the same security and flexibility considerations. Our design is covered in Section 3.7.

In the following sections we provide more details of our implementation and how vPCRs are used.

3.2 Virtual PCR Usage

Guests use vPCRs, or any combination thereof, by "selecting" them into their TPM context when issuing a TPM command. The hypervisor is then responsible for loading an actual resettable physical PCR with a representation of the selected vPCRs, so that *Sign*/*Seal*/*Unseal* TPM commands can make use of the virtual registers.

If a vPCR set is selected into a context, then a hash of a host-independent representation of the data structure containing the names and contents of selected VPCRs will be loaded into a resettable TPM/hardware PCR by the hypervisor (this is in direct analogy to the way that physical PCRs are selected into operations like seal and quote). To use that hashed vPCR set, a subsequent raw TPM command selects the resettable hardware PCR into its PCR set. Fig. 3. gives an example for a quote command using two vPCRs named "vPCR_A" and "vPCR_B".

We emphasize that a guest can include both vPCRs and physical PCRs in TPM operations—a raw TPM command can include PCRs that identify the hypervisor and

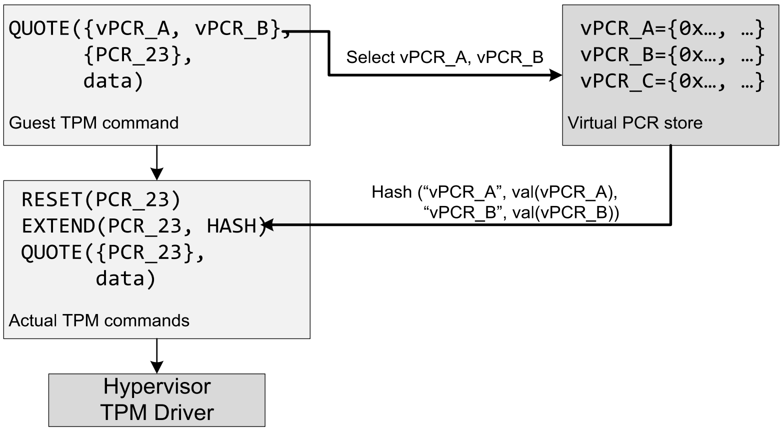


Fig. 3. vPCR support example inside the hypervisor: A guest issues a quote command that includes two vPCRs (vPCR_A and vPCR_B). The hypervisor looks up the contents of the two vPCRs, hashes the vPCR names and contents, and loads physical PCR_23 with the result. The raw TPM command selects PCR_23 for quoting.

firmware, if so desired. Alternatively, if the quest or other relying party trusts that only authorized hypervisors can host the guest then use of virtual PCRs alone may suffice.

However, vPCR-extension and -reset are limited to a guest's own set of vPCRs (subject to further limitations mentioned above): The physical PCRs express general machine state (hardware configuration, BIOS, original boot loader), but not guest state. It is also obvious that isolation requirements forbid any modifications of physical PCRs on behalf of a guest.

Most vPCRs provided to guests are not resettable for the life of the virtual machine. However guests are also furnished with a few resettable vPCRs that the guest can use for recursive virtualization. This is described in Section 3.6.

3.3 HyperTPM Guest Interface

The TPM hypercall interface exposed to guest OSes is very similar to original TPM "flat-wire" protocol. The command/response parameter format is in fact the same as if the guest application were communicating with the TPM directly. This is necessary because many TPM operation byte-streams are cryptographically protected (covered by the authorization HMAC), which forbids much parameter manipulation. However, we include additional header information in commands for specific hyperTPM commands and a newly introduced context. The context may include the virtual PCR information if any vPCR is used by the command.

Figure 4 shows the components of a hyperTPM command. The *Hypercall Layer* contains general information for wrapping data into a hypercall, such as size and address of result buffers. Data in the *HTPM Layer* defines the class of hyperTPM command to execute: This can either be a raw (native) TPM command, a vPCR command such as *vTPM reset* or *vTPM extend*, or a hyperTPM context command to

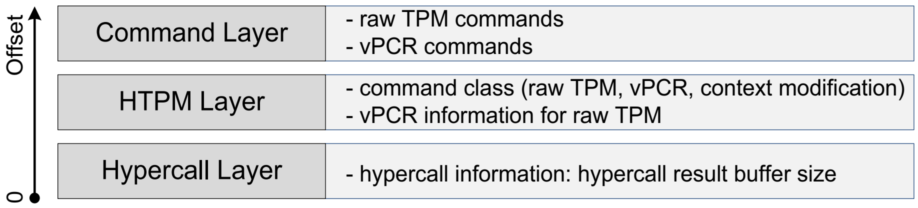

Offset
0
Command Layer
- raw TPM commands
- vPCR commands
HTPM Layer
- command class (raw TPM, vPCR, context modification)
- vPCR information for raw TPM
Hypercall Layer
- hypercall information: hypercall result buffer size

Fig. 4. HyperTPM Command layering. A raw TPM command effectively has two additional headers: The hypercall layer and the HTPM layer.

manage the guest session. In case of a raw TPM command, the HTPM Layer contains optional information on what vPCRs are to be selected for the raw TPM command encoded in the command layer. The *Command Layer* describes the actual action to take: A raw TPM command to execute or one of the vPCR/hyperTPM context commands.

We note that the command marshalling and response un-marshalling is done by the TPM API library in the guest operating system. This way the hypervisor exposes a rich set of TPM functions to the guests without needing to understand and interpret all the details of each command (but the hypervisor does need to extract limited/fixed information for command blocking and resource virtualization). Existing TPM software that does command marshalling and abstraction can be reused with little or no modification. Consequently, the additional code within the hypervisor can be kept small (see Section 4 for details).

3.4 Command Blocking

Direct PCR manipulation by guests (e.g. extend, reset) can compromise platform safety and/or availability. For example, if a guest were to *extend* a hardware PCR with a random number, then the hypervisor and other guests could no longer authenticate themselves. Similarly, if a guest could direct reset the resettable PCR used by the vPCR module, then reliable guest authentication may be compromised. For these reasons the hypervisor filters and denies certain operations.

This is simple: each TPM command received by the hypervisor is inspected for its command ID (for transport sessions, the inspection additionally breaks up the transport session wrapper). If the command is denied an error is returned to the guest.

3.5 Context Management

The physical TPM is accessed by multiple guests, and each of the guests loads its own resources (i.e., keys, transport and authorization sessions) into the TPM. However TPM internal space is limited so the vTPM component demand-loads and unloads TPM contexts when resources are low. The TPM context-save and context-load operations are used for this purpose.

As a consequence of the context management, resource handles need to be virtualized as well. As such, the context manager inside the hypervisor also inspects the handles created by TPM commands, stores them internally, and replaces them with a virtual handle. Whenever a guest issues a TPM command using that virtual handle, the context manager replaces it with the current handle values of the actual TPM resources[2].

3.6 Recursive TPM and vPCR Virtualization

In the previous section we described how one or more virtual PCRs can be used to represent guest state so that the TPM can provide attestation and similar services. This is generally sufficient for operating system use (for example, the Bitlocker drive-encryption product), but is inadequate if we need to extend the virtualization model to either (a) applications that need to use the TPM for authentication, or (b) further virtualization software running in a guest operating system.

The guest-resettable vPCRs described in the previous sections are introduced to allow recursive virtualization. A guest operating system provides its own para-virtualized interface to the TPM in a manner identical to that described in earlier sections. The OS can further create resettable vPCRs that it uses to encode its representation of the guest or application that has access to the TPM and load these into the hypervisor-provided resettable vPCRs. This recursive virtualization can be nested indefinitely.

It is interesting to consider the circumstances when a guest running at layer *n* of a virtualization stack needs to authenticate all layers beneath it (we have termed this “deep attestation”) and when it is sufficient to only authenticate itself (“shallow attestation.”) using the services of its hosting layer. In general we note that it is important to maintain a separation of concerns so that applications are relatively independent of the OS running beneath. This argues that shallow attestation is more manageable. However, clearly the trusted computing base—however many layers deep—has profound security implications for applications running on a platform, so (at least) system code and bootstrapping code will need to perform deep attestation (or recursive shallow attestation) to maintain an initial state of trust.

3.7 Guest Measurement

Platform firmware on a hardware (non-virtualized) system is responsible for measuring and reporting early boot code (e.g. the disk master-boot-record or MBR), and is also responsible for reporting platform hardware and firmware that might affect platform security (e.g. expansion card option ROMs). Similar considerations apply in virtualized setting.

The analog of early boot code depends on the details of guest initialization, but it seems reasonable that guest measurement is comparable to hardware measurement: a (typically small) bootstrapping component is hashed by the hypervisor, and gets control of the virtual processor. This component is free to measure and report additional components or settings through extending non-resettable vPCRs.

Virtual platform hardware must also be reported, as must any aspect of the guest execution environment that is configurable and might affect the security of the running operating system. Settings that affect security are hypervisor dependent, but

[2] This level of handle manipulation is allowed because resource IDs are not part of the HMAC’d part of authorized commands.

might include the initial program counter of virtual processors, whether other guests have read, write or debug access to a guest, and the guest's virtual memory configuration.

4 Prototype Implementation

We extended a hypervisor to support TPM para-virtualization according to the design described in the previous sections. Table 1 shows the number of lines of code we had to add/modify in order to add the TPM support and para-virtualization.

Table 1. Code size requirements for TPM para-virtualization

Component	**LOC**
TPM driver	1351
Scheduler	939
Command blocking	121
Virtual PCRs	685
Context management	240
Resource Virtualization	845
Guest Measurement	274
Misc (hypervisor glue, libc-equiv)	2550
Sum	7005

For comparison, Stamer et al. describe a software TPM emulator that supports parts of the functionality of a version 1.2 hardware TPM [8][3]. With 20kLOC, the emulator is already substantially more complex than our para-virtualization prototype.

5 Programming Models for Para-Virtualized TPM Usage

The para-virtualized TPM described in the last section allows maximum safe exposure of the TPM to guest operating systems: Our design exposes essentially all TPM functionality in native or in functionally equivalent form. In this section we review the circumstances under which direct TPM access is appropriate, and the cases where higher-level abstractions should be developed.

The first consideration is that the TPM is a memory- and price-constrained device, and this has resulted in a design that provides necessary security services, but rarely in a form that are particularly easy to use. For example, seal and unseal provide access-protected storage, but a better abstraction for an OS to use might be an encrypted block-storage device. Needless to say such services can be built into a virtualization platform given access to the underlying TPM.

[3] According to the paper, the following functionality is not yet supported by the emulator: Migration, Delegation, NV-Storage, as well as some other features around locality, capabilities and many optional commands.

The second consideration is that many of the advantages of virtual machine technology arise from creating virtual devices with advanced functionality like the ability to snapshot, roll-back and hot- and cold-migrate a running virtual machine. In general giving guests direct access to the underlying TPM interferes with this behavior. For example, a migrated OS image will no longer run if its keys are not migrated with it, and a guest that uses monotonic counters will fail if it is snapshotted and rolled back.

Of course in such cases, one man's bug is another man's feature, so assessing appropriate programming models in this case is more subtle. Some applications demand that application keys should be reliably bound to the hardware (for example, a key used by a corporate laptop used to identify the device during VPN access). Other keys have legislative requirements on their hardware protection such as transaction authorization keys covered by EU signature directives. But still other applications will benefit from migration flexibility over strong binding (for example a web services front-end using an SSL key for site identification).

Secure migration can be supported using native TPM key-migration facilities (although the endorsement key cannot be migrated). This will require that guest operating systems and applications adopt conventions about the keys that they use. Of course the alternative is that higher-level security services implemented in software are used by guests, and these services implement their own security polices to evaluate the circumstances that migration is allowed.

We believe that in most circumstances higher level services should be developed and exposed to guest operating systems and applications. This maintains the design principle of the TPM as a "root of trust" that bootstraps higher-level security services. These services can use the para-virtualized TPM interface that we have described, and (if needed) has code to support migration and other advanced services.

6 Summary

We have described a new procedure for safe and simple para-virtualized sharing of the TPM. The technique provides almost full exposure of the TPM to guest operating systems, and only those services that manipulate or use hardware PCRs need to be changed to use the defined interface and functionality. The technique is particularly attractive because it does not need any modifications to the (now widely deployed) TPM hardware.

We believe that this design is the optimal "lowest-level" exposure of the TPM to guest operating systems, but we stop short of arguing that this is the preferred programming model for application software. Instead we believe that system software using the para-virtualized TPM we have defined should provide richer services for use by applications.

For example, a cryptographically protected file-system or block-storage device is likely more useful than the "seal" primitive. And in general we should discourage applications running high in the software stack from being fragile to changes in software low in the stack (which might legitimately be updated or replaced). We encourage others to research higher-level services that provide semantically similar functionality to the TPM, but also provide security, usability, and manageability.

References

1. King, S.T., Chen, P.M., Wang, Y.M., Verbowski, C., Wang, H.J., Lorch, J.R.: SubVirt: Implementing malware with virtual machines. In: IEEE Symposium on Security and Privacy, Oakland (May 2006)
2. Sadeghi, A.R., Christian Stüble, C.: Property-based Attestation for Computing Platforms: Caring about properties, not mechanisms. In: New Security Paradigms Workshop (September 2004)
3. Goldman, K.A., Berger, S.: TPM Main Part 3 – IBM Commands, http://domino.research.ibm.com/
4. Berger, S., Cáceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: Virtualizing the Trusted Platform Module. In: 15th USENIX Security Symposium, Vancouver, Canada (August 2006)
5. Trusted Computing Group: TCG Software Stack (TSS) Specification – Version 1.10 Golden (2003)
6. Balacheff, B., et al.: Trusted Computing Platforms: TCPA Technology in Context. Prentice Hall (2002)
7. van Dijk, S., et al.: Virtual Monotonic Counters and Count-Limited Objects using a TPM without a Trusted OS (Extended Version). Technical Report MIT-CSAIL-TR-2006-064, MIT (September 2006)
8. Stamer, H., Strasser, M.: A Software-Based Trusted Platform Module Emulator. In: TRUST 2008, Villach, Austria (March 2008)

Slicing for Security of Code

Benjamin Monate and Julien Signoles[⋆]

CEA LIST, Software Reliability Labs,
91191 Gif-sur-Yvette Cedex, France
`Benjamin.Monate@cea.fr`, `Julien.Signoles@cea.fr`

Abstract. Bugs in programs implementing security features can be catastrophic: for example they may be exploited by malign users to gain access to sensitive data. These exploits break the confidentiality of information. All security analyses assume that softwares implementing security features correctly implement the security policy, *i.e.* are security bug-free. This assumption is almost always wrong and IT security administrators consider that any software that has no security patches on a regular basis should be replaced as soon as possible. As programs implementing security features are usually large, manual auditing is very error prone and testing techniques are very expensive. This article proposes to reduce the code that has to be audited by applying a program reduction technique called *slicing*. Slicing transforms a source code into an equivalent one according to a set of criteria. We show that existing slicing criteria do *not* preserve the confidentiality of information. We introduce a new automatic and correct source-to-source method properly preserving the confidentiality of information *i.e.* confidentiality is guaranteed to be exactly the same in the original program and in the sliced program.

1 Introduction

Bugs in programs implementing security features can be catastrophic: they may be exploited by malign users to gain access to sensitive data for example. These exploits break the confidentiality of information. All security analyses assume that softwares implementing security features correctly implement the security policy, *i.e.* are security bug-free. This assumption is almost always wrong and IT security administrators consider that any software that has no security patches on a regular basis should be replaced as soon as possible. As programs implementing security features are usually large, manual auditing is very error prone and testing techniques are very expensive. This article proposes to reduce the code that has to be audited by applying a program reduction technique called *slicing*. Slicing transforms a source code into an equivalent one according a set of criteria (see [13,14,12,16] for surveys about slicing). We show that existing slicing criteria do *not* preserve the confidentiality of information. We introduce a new automatic and correct source-to-source method properly preserving the

⋆ This work has been supported by the french RNTL project CAT ANR05RNTL00301 and the european ITEA2 €-Confidential project ITEA05011/N062930329.

confidentiality of information *i.e.* confidentiality is guaranteed to be exactly the same in the original program and in the sliced program. In the following sections this method will be called *confidentiality slicing criterion.* Thanks to this criterion, very drastic reductions can occur because usually only small parts of a program have an impact on security.

The advantages of these reductions are twofold. Firstly, they ease all auditing activities and help to focus on pertinent parts of the considered program. Secondly, whenever the confidentiality slicing criterion is not efficient (that is whenever the code is barely reduced), it points out that security features are scattered everywhere in the program and therefore are very error prone [1]. Moreover if a well-identified security-relevant part of the program is sliced, then it certainly contains major security issues because all security related parts of the program are kept unsliced.

For these purposes, we focus on an automatic over-approximated-but-correct source-to-source slicer. So, regarding the confidentiality criterion, it automatically transforms a compilable program p into another one p' which:

– is equivalent to p (in particular p' is compilable and has the exact same level of confidentiality ensured); and
– may contain useless code from a confidentiality point of view. This is the usual price for automation and correctness for undecidable problems.

This article focuses on confidentiality and integrity in source codes but deliberately ignores programming language level security issues like invalid pointer and array accesses also known as *buffer overflows.* One of our goals is indeed to formally study all security properties of *C* code as described by G. Heiser [6], but buffer overflows can be checked by standard *safety* verification techniques [3,8].

Outline. Firstly, we give a running and motivating *C* code example and describe its expected security properties. Secondly, we explain why the standard slicing methods cannot deal with these properties, especially confidentiality. Thirdly, we propose a new slicing technique to perform code reduction according to these properties.

2 Running Example

The *C* code in Figure 1 is used as a running example in the following sections. It contains the definition of function `check_and_send` which sends a message `msg` under some specific conditions.

This code contains special comments between `/*@` and `*/`. The first one is above the prototype of function `send` and describes its expected security property: that is the sent `data` has to be `public` (*i.e.* readable by anyone if you are interested in confidentiality). The second one is above the prototype of function `publicise` and explains that, after a call to this function, its argument becomes `public`. The last special comment is a special cast `/*@ (public) */`: that is a *declassification* operator which explicitly makes a data `public`. By default, all the values are considered private, *i.e.* non-public.

```
/*@ requires security_status(data) == public; */
void send(const void *data, const int dst[4]);

/*@ ensures security_status(data) == public; */
void publicise(char *data);

int check_and_send(char msg[]) {
  int src[4];
  int dst[4];
  int result = 0;

  compute_src(src);
  compute_dst(dst);

  if (dst[0] >= 190 && dst[0] <= 200) {   // quite dummy sanity check
    if (src[0] >= 190 && src[0] <= 200) { // another dummy sanity check
      send(/*@ (public) */src, dst);
      if (msg != 0) {
        compute_private_msg(msg);
        publicise(msg);
      } else
        msg = /*@ (public) */"hello world";
    } else
      result = 2;
  } else
    result = 4;

  send(msg, dst);

  compute_src(src);
  compute_dst(dst);

  send(/*@ (public) */src, dst);

  if (src[0] == 190)
    result = 2 * result + 1;

  return result;
}
```

Fig. 1. Running example: `check_and_send`

Two different properties can be studied on the function `check_and_send`:

Definition 1 (integrity). *A program fulfills the* integrity *property if no private data can be modified by an intruder which can intercept any sent message.*

Definition 2 (confidentiality). *A program fulfills the* confidentiality *property if no private data can be discovered by an intruder which can intercept any sent message.*

As for the integrity property the function `publicise` shall be a sealing primitive whereas for the confidentiality property it shall be a cyphering primitive.

Integrity is ensured by the `check_and_send` function because the only values which can be modified by an intruder are explicitly declassified or publicised. A quick similar reasoning seems to indicate that confidentiality is also ensured. Unfortunately, it is not true: the last instruction `send` emits the public data `src` which can securely be read by anyone but from which an intruder can deduce the parity of the private (*i.e.* confidential) data `result` returned by the function. This is a security leak induced by the information flow of the function.

In conclusion slicing preserving integrity can securely remove all the parts concerning the variable `result`, but slicing preserving confidentiality has to keep all the lines 1. To our knowledge there isn't any standard slicing criterion that is correct regarding the confidentiality criterion.

3 Standard Slicing Methods Do Not Preserve Security

In this section we explain the issues with the preservation of security criteria. Indeed there is no problem with the integrity criterion because preserving it is equivalent to preserve all the calls to the function `send`: only sent values are modifiable by an intruder. For this purpose, using the classical "preserve the accessibility of lines calling `send` and the value sent at these statements" criterion works fine.

The confidentiality criterion is hard to preserve because an intruder can discover confidential information through program flow. For example with the following lines of code

```
if (public_info) secret = choice1; else secret = choice2;
send(/*@ (public) */ public_info, dst);
```

the secret escapes because an intruder can *indirectly* deduce the value of `secret` from the non-confidential data `public_info` sent on the network: if the intruder intercepts the sent message, he discovers whether `public_info` is 0 or not and therefore whether the value of `secret` is `choice1` or `choice2`. So slicing according to confidentiality has to keep both lines of code, and not only the second one.

The issue remains if we simply swap the two lines of code:

```
send(/*@ (public) */ public_info, dst);
if (public_info) secret = choice1; else secret = choice2;
```

It is still possible to indirectly deduce the value of `secret` from the publicly sent data `public_info`. The confidentiality leak of our running example (Figure 1) follows this second scheme: the parity of `result` can be deduced from the last emission of `src`.

These issues bring forward a wider concern: the code that is potentially influenced by an intercepted sent value can be placed before or after — according to the program flow — the call to `send`. So security leaks may be quite difficult to identify: performing a backward or forward analysis won't work properly.

One can show analogically that confidentiality is not a compositional property: the confidentiality cannot be established for a function without knowing all its callers.

4 Security Guided Slicing: Criticality Components

Sections 2 and 3 have implicitly distinguished two different kinds of potential confidentiality leaks: *direct* and *indirect* leaks.

Direct leaks are related to program points modifying variables with values which can be *directly* deduced from a call to function `send` (the unsecured point of the program) because some modified bits may be sent by this call. For example if the function `compute_src` used in Figure 1 is defined like

```
void compute_src(int src[4]) {
  ...
  src[0] = ···;
  for(i = 1; i < 4; i++) src[i] = ···;
}
```

an intruder may deduce the values of `src[i]` ($i \in \{0, 1, 2, 3\}$) from the last call to function `send` *via* the call to function `compute_src`[1].

Indirect leaks are related to program points potentially impacted by the sent values and from which an intruder can *indirectly* deduce information from program flow. This is the problem described by all the examples given in Section 3.

In the last part of this section, we focus on a way to compute the sets of statements required to discover direct and indirect leaks from a given call to the function `send` (indeed and more generally from any statement s). We call these sets respectively the *direct and indirect criticality components* of s. In order to illustrate these computations, we consider the toy program given in Figure 2. Beforehand we introduce the necessary notions and notations.

```
   y = ...; e = y;
A: b = ...; B: c = ...; C: x = ...; D: y = ...;
E: a = x;
F: if (c) { G: y = x; H: a = 1; }
I: z = y;
J: send(z);
   t = e;
K: if (a) { L: b = 2; }
M: d = a + b;
```

Fig. 2. Illustrating example: different kinds of dependencies

[1] There is indeed no direct security leak at this call site because the sent value `src` is declassified. But this article is not dealing with verification but with slicing.

4.1 Program Dependence Graph and Notations

We use $\mathcal{S}$ to denote the set of all the statements of a program. If s is a statement, we denote by $\text{reads}(s)$ the set of the memory location read by s and by $\text{writes}(s)$ the set of the memory location modified by s.

The computation of the criticality components is based on the notion of Program Dependence Graph (PDG) [9,4] which shows dependence relations between program statements. Here we distinguish two different kinds of dependencies: *data* and *control* dependencies.

There is a *data dependency* from statement s_1 to statement s_2 according to the memory location l (a variable, a pointer access, ...) if s_2 modifies l which is used by s_1 and there is at least one path in the control flow graph from s_2 to s_1 on which l is not modified. We denote by $\text{dpds}_d(l, s)$ the set of all data dependencies of the statement s according to the memory location l. We denote by $\text{codpds}_d(l, s)$ the data *co-dependencies* of s according to l, that is $\{s \in \mathcal{S} \mid \exists s' \in \mathcal{S}, \text{dpds}_d(l, s')\}$. For example on the illustrating example (Figure 2), $\text{dpds}_d(\texttt{x}, \texttt{C}) = \{\texttt{E}, \texttt{G}\}$ and $\text{codpds}_d(\texttt{y}, \texttt{I}) = \{\texttt{D}, \texttt{G}\}$.

Informally there is a *control dependency* from the statement s_1 to the statement s_2 if s_2 conditionally decides whether s_1 is to be executed or not. A more formal definition exists [4] but it is not required in our presentation. Similarly to the data dependencies, we use the notations $\text{dpds}_c(, s)$ and $\text{codpds}_c(, s)$ for control dependencies. Following the illustrating example2, $\text{codpds}_c(, \texttt{G}) = \{\texttt{F}\}$.

Furthermore in order to unify the notations, we use $\text{dpds}_\delta(l, s)$ with $\delta \subseteq \{c, d\}$ to denote the set of dependencies of s according to l and regarding the kind δ of dependencies. We use the same convention for $\text{codpds}_\delta(l, s)$. We can trivially prove that $\text{dpds}_{\delta_1 \cup \delta_2}(l, s) = \text{dpds}_{\delta_1}(l, s) \cup \text{dpds}_{\delta_2}(l, s)$. The same property holds for co-dependencies. Following the illustrating example, $\text{codpds}_{cd}(\texttt{x}, \texttt{G}) = \{\texttt{C}, \texttt{F}\}$.

4.2 Computation of Criticality Components

The computation of criticality components is divided in two parts. Firstly we focus on the computation of *direct* criticality components. Then we introduce *indirect* criticality components. Finally we explain how slicing works with these criticality components.

Direct criticality components. We first explain our algorithm for the computation of the *direct* criticality component of the statement J from Figure 2.

The direct criticality component, as it is informally defined in the beginning of this section, contains J itself and the set of statements that modify variables (and more generally memory locations) with some bits of values read in J. Thus it contains at least $\text{codpds}_d(l, \texttt{J})$ for each memory location l used in J (*i.e.* for $l \in \text{reads}(\texttt{J})$). In this case this is exactly the singleton containing I. I itself reads y and thus the component of J also contains $\text{codpds}_d(\texttt{y}, \texttt{I}) = \{\texttt{D}, \texttt{G}\}$. Similarly G reads x and therefore the component of J also contains $\text{codpds}_d(\texttt{x}, \texttt{G}) = \{\texttt{C}\}$. Hence the direct criticality component of J is $\{\texttt{J}, \texttt{I}, \texttt{D}, \texttt{G}, \texttt{C}\}$. This example shows that computing a direct criticality component of a set of statements S is an iterative process based on the data co-dependencies.

The formal definition of the direct criticality component of the set of statements S is denoted by $\uparrow_d^\star S$ and defined by

$$\uparrow_d S \triangleq \bigcup_{s \in S} \bigcup_{l \in \mathrm{reads}(s)} \mathrm{codpds}_d(l, s)$$

$$\uparrow_d^\star S \triangleq \mathrm{fix}(S \mapsto S \cup \uparrow_d S)$$

where fix is the fixpoint operator on the domain $(\mathcal{P}(\mathcal{S}), \subseteq)$. The existence and uniqueness of this fixpoint operator (as well as those of the others fixpoints in this section) are easy to prove using Tarski's domain theory theorem [11] (see for example Gunter's or Winskel's books [5,15]). Moreover on such a finite domain, fixpoints are easily computable.

Indirect criticality components. In Section 3, we assert that a backward or forward analysis won't work properly. We indeed distinguish two different kinds of indirect criticality components: *backward and forward indirect criticality components* .

Backward indirect criticality components. The standard scheme of code corresponding to this kind of components is described below.

```
secret = ...;
if (secret) public_info = choice1; else public_info = choice2;
send(/*@ (public) */ public_info);
```

In this scheme there is an indirect security leak because the value of `secret` escapes from the information flow: it can be deduced from the sent value `public_info` because there are control dependencies from the statements modifying `public_info` to the enclosing `if` statement.

Thus computing backward indirect criticality components is similar to computing direct criticality components but it has to take into account the control dependencies. It is indeed easy to generalise the $\uparrow_d$ and $\uparrow_d^\star$ operators in order to deal with any kind of dependencies and not only with the data dependencies. Furthermore the backward indirect criticality component of the set of statements S exactly contains the statements of its non direct backward criticality component. Hence it is formally defined by

$$\uparrow_{cd}^\star S \setminus \uparrow_d^\star S.$$

In the illustrating example 2, the backward indirect criticality component of `J` is $\{\mathtt{F}, \mathtt{B}\}$.

Forward indirect criticality components. The standard schemes of code corresponding to this kind of components are those of Section 3. We remind one of them below:

```
public_info = ...
send(/*@ (public) */ public_info);
if (public_info) secret = choice1; else secret = choice2;
```

In this scheme, there is an indirect security leak because the value of `secret` escapes from the information flow: its value can be deduced from the sent value `public_info` which is known thanks to the computation of the backward criticality component C of the call to `send`. One can indeed deduce information on any statement impacted by C. The impacted statements of a set of statements S are formally defined thanks to the dependencies of S in the following way:

$$\downarrow S \triangleq \bigcup_{s \in S} \bigcup_{l \in \mathrm{writes}(s)} \mathrm{dpds}_{cd}(l, s)$$

$$\downarrow^{\star} S \triangleq \mathrm{fix}(S \mapsto S \cup \downarrow S).$$

Note that $\downarrow$ and $\downarrow^{\star}$ are respectively dual operators of $\uparrow_{cd}$ and $\uparrow^{\star}_{cd}$.

Using this operator and backward criticality components, it is easy to define the forward indirect criticality components of the set of statements S: that is the impacted statements of the backward criticality component of S which do not belong to this backward criticality component. It is formally defined by:

$$\downarrow^{\star} (\uparrow^{\star}_{cd} S) \setminus \uparrow^{\star}_{cd} S.$$

In the illustrating example, the forward indirect criticality component of `J` is $\{\mathtt{E}, \mathtt{H}, \mathtt{K}, \mathtt{L}, \mathtt{M}\}$.

Slicing from criticality components. From the above definitions, the criticality component $\Psi(S)$ of a set of statements S is computable by firstly computing its backward criticality component and secondly computing the impacted statements of the resulting sets. So Ψ is formally defined by:

$$\Psi \triangleq \downarrow^{\star} \circ \uparrow^{\star}_{cd} .$$

In our illustrating example, the criticality component of `J` contains all the labeled statements: unlabeled statements are sliced.

$\Psi(S)$ does not yet define a full slicer (according to the confidentiality criterion) because the resulting statements are not compilable: pieces of code like declarations leak. One can yet use standard slicing techniques on this set of statements in order to complement our methodology.

A welcome side effect of our presentation is that the notion of criticality component is related to the notion of program dependency graph and not to the C programming language itself: it only uses the notions of dependencies, co-dependencies, read and written memory locations and control flow graph which are very common in imperative programming language.

5 Conclusion

We have shown that standard slicing methods are not correct regarding the confidentiality criterion.

In order to solve this problem, we have presented a new slicing technique which is automatic, over-approximated-but-correct and source-to-source. It deals with security criteria, especially confidentiality, and introduces notion of *criticality component* which can be used in any imperative programming language (even if we focus on *C* in this paper).

This security slicing performs code reductions which firstly ease all auditing activities and help to focus on pertinent parts of the considered program and, secondly, may pinpoint the quality of the security of the code. Besides the code reduction may be used as a security pre-analysis to be performed before running other expensive analyses such as formal security verification.

A prototype is under development as a plug-in of the *Frama-C* (Framework for Modular Analyses of *C*) platform [2]. It uses the following native plug-ins of *Frama-C*: the value analysis, the program dependency graph and the slicer. Thanks to these plug-ins the development contains around 500 lines of code in *Objective Caml* [7] and deals with all the *C* constructs except the recursive functions calls and the signals related features. The first results are stimulating. Our main goal is the slicing of the IPsec (IP secure) [10] implementation embedded in the Linux kernel regarding the confidentiality and the integrity criteria.

References

1. Bernstein, D.J.: Some thoughts on security after ten years of qmail 1.0. In: CSAW 2007: Proceedings of the 2007 ACM workshop on Computer security architecture, pp. 1–10. ACM, New York (2007)
2. CEA-LIST and INRIA-Futurs. Frama-C: Framework for Modular Analysis of C, `http://www.frama-c.cea.fr`
3. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th Symposium on Principles of Programming Languages, Los Angeles, Californie, États-Unis, pp. 238–252. ACM Press, New York (1977)
4. Ferrante, J., Ottenstein, K.J., Warren, J.D.: The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. 9(3), 319–349 (1987)
5. Gunter, C.A.: Semantics of Programming Languages: Structures and Techniques. In: Foundations of Computing. MIT Press, Cambridge (1992)
6. Heiser, G.: Your system is secure? prove it! USENIX;login: 32(6), 35–38 (December 2007)
7. Leroy, X., Doligez, D., Garrigue, J., Rémy, D., Vouillon, J.: The Objective Caml system, release 3.10 (May 2007), `http://caml.inria.fr`
8. Meyer, B.: Proving pointer program properties. part 1: Context and overview. Journal of Object Technology 2(2), 87–108 (2003), `http://www.jot.fm/issues/issue_2003_03/column8`
9. Ottenstein, K.J., Ottenstein, L.M.: The program dependence graph in a software development environment. In: Karl, J. (ed.) SDE 1: Proceedings of the first ACM SIGSOFT/SIGPLAN software engineering symposium on Practical software development environments, pp. 177–184. ACM Press, New York (1984)
10. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. Request for comments (rfc) 4301, Network Working Group (December 2005), `ftp://ftp.rfc-editor.org/in-notes/rfc4301.txt`

11. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5, 285–309 (1955)
12. Tip, F.: A survey of program slicing techniques. Journal of programming languages 3, 121–189 (1995)
13. Weiser, M.: Program slices: formal, psychological, and practical investigations of an automatic program abstraction method. PhD thesis, University of Michigan, Ann Arbor (1979)
14. Weiser, M.: Program slicing. In: ICSE 1981: Proceedings of the 5th international conference on Software engineering, Piscataway, NJ, USA, pp. 439–449. IEEE Press, Los Alamitos (1981)
15. Winskel, G.: The formal semantics of programming languages: an introduction. MIT Press, Cambridge (1993)
16. Xu, B., Qian, J., Zhang, X., Wu, Z., Chen, L.: A brief survey of program slicing. SIGSOFT Softw. Eng. Notes 30(2), 1–36 (2005)

Trusted Computing Serving an Anonymity Service*

Alexander Böttcher, Bernhard Kauer, and Hermann Härtig

Technische Universität Dresden
Department of Computer Science
Operating Systems Group
01062 Dresden, Germany
{boettcher,kauer,haertig}@os.inf.tu-dresden.de

Abstract. We leveraged trusted computing technology to counteract certain insider attacks. Furthermore, we show with one of the rare server based scenarios that an anonymity service can profit from trusted computing. We based our design on the Nizza Architecture [14] with its small kernel and minimal multi-server OS. We even avoided Nizza's legacy container and got a much smaller, robust and hopefully more secure system, since we believe that minimizing the trusted computing base is an essential requirement for trust into software.

1 Introduction

Anonymity while using the Internet is widely considered a legitimate and - for many use cases - essential requirement. A use case encompasses protection of privacy by avoiding traces and by preventing to reveal unnecessary private information. On the other hand, the Internet can be considered a least anonymous technology in wide use. Traces are left while visiting web sites on various levels, e.g. connection data on the network level and cookies as well as personal data on the application level. To enable network anonymity, anonymity services have been devised [8,11]. They obscure the real source address of a user by routing lot of users over one or more proxies.

However, these anonymity services can be compromised as the following scenario, based on a real incident [15] in the authors' group, will show. An unknown criminal to be investigated was supposed to use an anonymity service. The police and later on the German Federal Bureau of Criminal Investigation (FBCI) required to investigate a criminal that uses a known URL. In order to enable criminal prosecution by a given warrant the software was extended to log connection data [6,5]. This function of the anonymity software [3] was used only with a warrant but without directly notifying the users. After the warrant was reversed the FBCI showed up with a delivery warrant for a log record (which were later on reversed illegal [4]), first at the institute and later on at the institute's directors private home.

This incident shows a whole class of attacks on the anonymity and more general every computing service. Vendor and providers under constraint can reveal the service because of insider knowledge and physical access to those services. Besides reasons

* Part of this work has been sponsored by the European Research Council in the OpenTC and Robin project and by Intel.

like corruption or human social engineering, also external influence such as extortion can cause attacks, which under normal situations would never happen.

Until now changes undermining the anonymity cannot be detected by users relying on the service. Within this paper we investigate how technical solutions can provide users of the anonymity service with additional information about the used software. On the one hand we use trusted computing technology to verify the anonymity software stack and on the other hand we target on strong decomposition and minimizing the trusted computing base in order to minimize attack points and make trusting the software itself justifiable. The work in this paper aims at the server side of the anonymity service and excludes investigation at the client side. This would be a topic of research on its own.

The paper is structured as follows. In section 2 we describe the anonymity service and give a short introduction into trusted computing. In section 3 we look into related work, followed by the design and implementation of the extended anonymity service in section 4. In section 5 we evaluate the achieved protection against insider attacks and the size of the trusted computing base. The last section proposes future work and concludes.

2 Overview

2.1 Anonymizer service

We use the anonymizer service AN.ON [1] which is based on David Chaum's mix networks [8]. The general idea is to provide anonymous communication within a network by transferring and by recoding messages over several intermediate network nodes called mixes instead of directly transferring data from the source to the destination. By

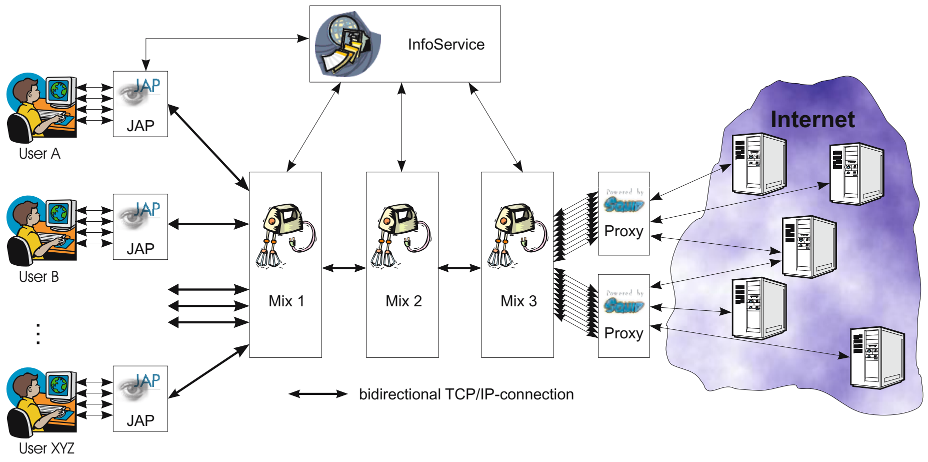


Fig. 1. High level overview of AN.ON (figure source: AN.ON Projekt home page at http://anon.inf.tu-dresden.de)

using mix network technologies senders, receivers or both sides are able to stay anonymous from each other. Mixes are message mediators similar to proxies, however mixes take care that received messages cannot be linked to messages they send forward. Basic methods to achieve this is to collect messages, to recode the message and to send the message in a different order than they were received.

AN.ON, an anonymizer service developed at the German universities of Regensburg and of TU Dresden, is used by thousands of users today. Figure 1 shows the general high level structure of the service, which consists of Java based clients called JAP, C++ based mixes, JAP based InfoServices and cache proxies. Single mix nodes are formed to a mix cascade and one mix serves one and only one cascade. InfoServices provide information about currently available cascades and provide meta information about the current number of users and data traffic workload of cascades. All information are visualized by JAP clients, where users can select a mix cascade. Web browsers of users are configured to use JAP as proxies, however JAP does not only forward messages like typical proxies do. JAP encrypts and decrypts all requests and responses according to the mix protocol, which will be described in detail below.

After a user has chosen a cascade, JAP connects to the first mix and establishes a channel. It uses a public key of the first mix to negotiate a symmetric key for encryption of data. The first mix then establishes a connection to the next mix and the JAP client also negotiates a symmetric key by using the public key of the second mix. This is done for all mixes until the last mix. After this setup phase a JAP client has symmetric keys negotiated with all mixes of the selected cascade. The user's web browser forwards requests locally to JAP which encrypts the data multiple times, first with the symmetric key of the last mix, then with the symmetric key of the mix before the last mix and so on. Finally JAP sends the encrypted packet to the first mix of the cascade. The first mix decrypts received packets and forwards them together with packets of other users to the next mix. Every mix of the cascade decrypts with the negotiated symmetric keys the received packets and forwards them to the next mix. The last mix obtains after decryption the original web browser request and sends it to the Internet. Relaying and encrypting replies of the Internet to users in the backward direction is done similarly but in the inverse order.

Currently the trust into a mix cascade can be based only on two things: a written statement of the mix providers, declaring the intent to do not log any data, and certificates of the public keys. The later ones are used to avoid man-in-the-middle attacks.

2.2 Trusted Computing

Trusted Computing [20,12] is a technology that gives us two mechanisms that are useful for our scenario: Remote Attestation and Sealed Memory.

Remote Attestation allows a third party to verify which software stack (e.g. BIOS, Bootloader, OS, Applications) is running on a remote computer.

Sealed Memory allows to seal secrets to a particular software stack, thus preventing the disclosure of private data from another - potentially untrusted - software stack running later on the same machine.

Although other ways were discussed in the literature [13,17,7], trusted computing based on a Trusted Platform Module (TPM) is now widely accepted and TPMs are deployed in the millions. The TPM, as defined by the Trusted Computing Group [25], is a smartcard-like low-performance cryptographic coprocessor, that is soldered on the motherboard.

In addition to cryptographic operations such as signing and hashing, a TPM can store a chain of hashes of the boot sequence. Using a challenge-response protocol, this chain of hashes can be signed by the TPM and verified by a remote entity. Similarly it can be used to seal data to a particular, not necessarily the currently running, software configuration. Unsealing the data is then only possible when this configuration was started.

A TPM based system has a couple of limitiations. The well known cryptographical limits (RSA assumption, SHA1 collision resistance) will not be targeted in this paper. Instead we have to keep in mind that the trusted computing specification explicitly excluded hardware attacks [16]. We will explain later in the evaluation section how this limit restricts the insider attacks we can tolerate.

3 Related Work

Garriss et al. [10] are describing an Internet kiosk scenario based on the IBM's Integrity Measurement Architecture (IMA) [21]. IMA extends a normal Linux with trusted computing functionality.

Another Linux based solution is ProxOS [24] which splits the operating system on the system-call level. System-calls can be redirected to various Linux instances responsible for appropriate tasks. Within this solution multiple specialized and stripped down Linux kernels can be used to serve an application.

We decided not to use a monolithic operating system approach for our scenario because of different reasons. First, solutions based on a big monolithic system, for example on Linux, will not achieve such a small trusted computing base as a decomposed approach promises. Second, monolithic systems make it hard to protect and encapsulate different services from each other at runtime. Successful attacks on a single service running in the kernel can compromise the whole system. Third, our application scenario does not require a lot of functionality provided by monolithic kernels. One example is multi-user support, a feature we do not need. Without it attacks like local privilege escalation are impossible. Finally unneeded functionality of such kernels cannot be arbitrarily decreased with a reasonable effort.

We aim to use a reduced and small computing base to make evaluation of each component possible and trust in the whole computing base revisable. In order to achieve this we base the anonymizer scenario on Nizza [14]. Nizza is an architecture describing how small trusted computing bases can be achieved by using a L4 microkernel and few basic L4 services, using trusted wrappers and legacy containers. The general idea is to split applications into security critical and non security critical parts and to execute them isolated from each other. Typically non critical parts are executed in legacy OS containers and the security critical parts are executed directly beside the legacy OS container on top of an microkernel.

Work based on the Nizza architecture were presented in [23] and [22]. The former work splits applications based on components borders into security critical and non-critical parts. In contrast the latter one splits the application based on the information flow of the scenario. Both reuse large parts of the application within OS legacy containers.

In this scenario we want to go a step further by avoiding the legacy OS container at all. Instead smaller decomposed services will be used. This results in better isolation and a much smaller computing base that is easier to handle, robust in execution and finally more secure. Most required services like network stack, driver support and file system services are already supported to run directly on top of the L4 microkernel Fiasco which makes an implementation of the anonymizer scenario feasible.

Another mechanism to support better isolation and providing a small trusted computing base is used by McCune et al. [18]. Instead of using isolation provided by a (monolithic) kernel they leverage new hardware features introduced by AMD and by Intel to execute security sensitive code underneath the OS. Although this is a nice idea that fills a research gap, a couple of limitations remain. They currently suffer for example from the speed of the TPM and their approach is not multiprocessor friendly as they have to stop all other CPUs while executing in the secure environment.

4 Scenario Design

4.1 The Trusted Computing Base

Figure 2 opposes the architecture of the Linux-based approch to the Nizza-based approach. The anonymizer mix is in both cases part of the trusted computing base (TCB) as it crypts user packets and routes them. The trusted computing support is currently part of the TCB as it sees the unsealed private key of the mix. Most of the basic L4 services are also included as either they have access to the memory the mix is using or

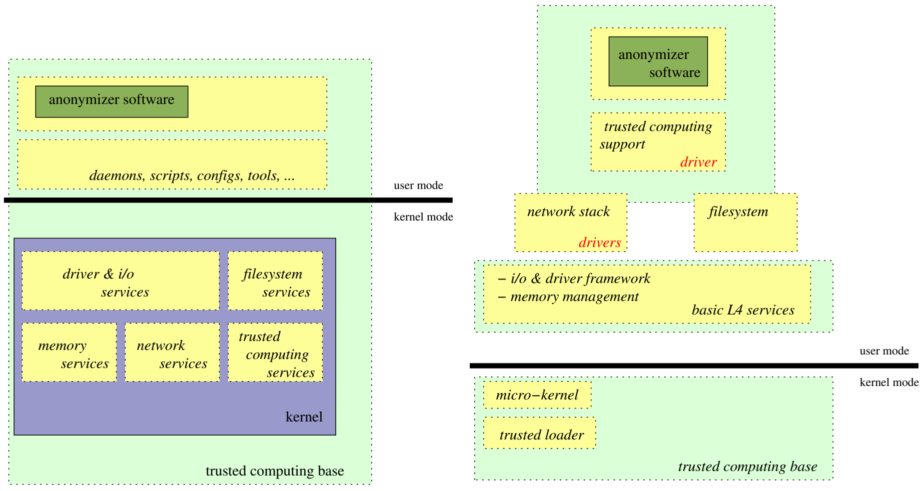


Fig. 2. Linux based and Nizza based architecture

they provide them with random numbers. Finally, the used L4 microkernel Fiasco itself is naturally also part of the trusted computing base.

4.2 Nizza Components

The following central services are needed to run an AN.ON mix on Nizza: network Internet protocol stack, a file system service and drivers for network and TPM.

In order to access network devices we use a framework called device driver environment (DDE), which offers a Linux compatible environment to reuse unmodified drivers. Network drivers are wrapped by the framework into a library which are then used by a network multiplexer ORe to access the devices.

On top of the network multiplexer ORe we run a stand-alone network stack called FLIPS (Flexible Internet Protocol Stack). This stack is an extracted legacy Linux 2.4 Internet protocol stack which is encapsulated with the help of DDE into a single L4 process. FLIPS contains only the network stack, accesses ORe via inter process communication (IPC) and itself exposes an IPC interface to user applications.

The AN.ON mix requires minimal file handling functionality to load and find the xml configuration file during startup. The L4VFS framework provides Posix like interfaces and services. Posix function calls are forwarded by the adapted lib-c for an application to appropriate L4VFS file-system services. The used L4VFS services are a file-system server providing the configuration of the anonymizer mix, a service providing stdin, stdout and stderr functionality and a service mounting the intital services in a file-system hierarchy. In our scenario we do not require drivers to access disks to load the configuration file. Instead it will be loaded during boot time into a RAM disk.

The TPM service (STPM) provides an IPC based interface to user applications that can be used to seal/unseal data or to request quotes from the TPM. The TIS driver from the OSLO project [19] is reused to drive the TPM.

4.3 Minimize the Trusted-Boot Chain

The software chain that needs to be trusted in the traditional trusted computing scenario starts with the BIOS and the bootloaders, which should not be considered secure [16].

To remove them from the trust chain we can use the Open Secure LOader [19]. OSLO is a bootloader that relies on new hardware features, mainly the `skinit` instruction of newer AMD processors, to start a trust chain later in the boot process. This allows to remove the bootloaders and main parts of the BIOS from the trust chain.

4.4 AN.ON Mix and Protocol Extensions

The private key of the mix, used for authentication to avoid man-in-the-middle attacks, is currently stored in the XML configuration file, which is obviously not a safe place. A more secure way would be to seal the key with the TPM. Another solution is to switch from a key external to the TPM to an internal one. This should avoid any propagation of the private key outside the TPM and it finally removes the trusted computing support out of the trusted computing base.

To be able to detect changes to the mix software we added remote attestation into the AN.ON protocol. This was achieved by embedding the TPM quotes as additional information besides similar XML meta-data in the AN.ON protocol.

The embedding of the quote is done in two places. First solely as hint in the data provided by the InfoService. These quotes can be used by JAP clients to determine whether they want to connect to a cascade at all.

Later a connected client can request its own quote from every mix. This operation will guarantee fresh quotes and avoids replay attacks.

If one of the mixes is rebooted after a symmetric key was negotiated, the specific mix will lose the key. Therefore a JAP client will detect this situation and can then renegotiate new symmetric keys and can request a new remote attestation.

4.5 Scenario Implementation

The effort to port a AN.ON mix to the microkernel based operating system (TUDOS) turned out to be moderate since almost all parts of required functionality for the scenario were available. For example the uclibc provides required Posix bindings such as memory management, socket and filesystem handling. The mix requires the openssl and xerces xml library which were adapted to the building infrastructure by solving the expected minor issues like configuration, compiling and linking. Changes on the sources of the both libraries were not necessary.

In order to access the trusted computing service (STPM) a small library was added to the AN.ON mix which provides the IPC bindings to request TPM quotes. In our scenario we require an 1.2 TPM in order to leverage the `skinit` facility of newer AMD CPUs. We had to extend the STPM services by 1.2 TPM commands since some older 1.1 TPM commands were deprecated and the TPM declined the execution.

We extended the client side JAP of our anonymizer scenario to evaluate the remote attestation embedded as additional meta-information within the XML messages exchanged between JAP clients and mix servers. A verification routine was added in Java which evaluates the quote by checking the signature and compares it to a user specified expected hash. The GUI of the JAP client presents the result of the verification result besides the normal CA related information.

5 Evaluation and Discussion

Within this chapter we evaluate the effects of different insider attacks and measure the size of the trusted computing base.

5.1 Attacker Profiles

We now describe some attacker profiles on our anonymizer scenario in which we assume that the attacker has some insider support. We describe how this can be detected and what can be done against such an attacker.

Trusted computing makes successful and unnoticed attacks harder, since remote attestation of the anonymity service AN.ON enables the users to verify which anonymity software and operating system is running as mix. Depending on this information they can decide on their own how trustworthy a mix is. Furthermore sealed memory is used to protect the private key of a mix during downtimes of the mix.

No physical access but software access. In the first scenario an attacker should have no physical access to the dedicated hardware of a mix, however will have access to the software. That means that an attacker can change the configuration or even boot other software on the mix. Possible imaginable cases are when an attacker compromised insider by extortion, due to social engineering or by corruption. Besides the fact that an attacker has to compromise all mixes of a cascade to be successful, changes on the software of the mix of our work would be detectable by users due to the remote attestation support. In order to avoid suspicion attackers, mix operators should generally provide some declaration why they modified software. Since the mix sources are completely publicly available, the changes can be reproduced by externals, institutions or experienced users. Unexperinced users could rely on a web of trust, similar to the PGP one, to rate the trustworthiness of changes and finally the whole software stack. As long as modifications are not publicly available and no public statement about altering the software is made a mix should be assumed as untrustworthy.

Physical access with limited knowledge about hardware. In the next scenario an attacker has physical access to the hardware on which the mix is running. Further an attacker should be able to exchange hardware, however he is not able, because of missing knowledge or physical protection, to modify or disassemble the security critical parts such as the TPM chip. In this scenario the attacker can exchange or add hardware components, for instance PCI cards or USB devices.

The major risk in this case are DMA attacks. A malicious device can read-out but also modify the memory the mix is using. Fortunately new chipsets can restrict DMA with their IOMMUs. Adding support for them into the OS would remove the threat.

Simple man-in-the-middle attacks, for example replacing the full machine, will not be successfull in this scenario, as the attacker can on one hand not clone the private key of the mix and on the other he will not be able to sign with the very same TPM key during remote attesation when using another TPM.

Physical access and hardware knowledge. If an attacker has full physical access and knowledge about attacking a TPM by exploiting implementation bugs or hardware limitations as described in [16], an attacker would be able to compromise the TPM in the end. Integrating the TPM into the chip-set and moving the TPM nearer to the CPU, the actual place where it is needed, will make successfull hardware attacks much harder. Even a good physical protection will help a lot against hardware attacks.

Exploiting software bugs with insider knowledge. In the next scenario an attacker has no physical access to the mix and no direct access to the software, however he is able to exploit bugs in the used operating services or the anonymizer software because of good insider knowledge of the sources. Depending on which service the attackers are able to compromise possible data about users of the anonymizer service can be leaked.

If attackers are able to exploit bugs in the kernel, the basic L4 services or in the anonymizer software, it is possible that they can enable logging or even get access to the session keys of the mix. Theses keys can be used to correlate the anonymous packets flowing in and out of the mix thus disclosing the anonymity.

If an attacker is able to exploit bugs in the network multiplexer ORe or the network IP stack FLIPS, an attacker would be able to trace source and destination of data packets that arrive and that are sent. As the mix is recoding and reordering the messages, these informations, that can also be collected by wiretapping on the network cable, should be worthless. An exception of this is the last mix of a cascade when users utilize unencrypted protocols like plain HTTP or POP3. Valuable private data like logins and passwords can be obtained at this point, as a recent case within the TOR network shows [26].

5.2 Managing Many Mix Hashes

An open question is how users know, which values of a remote attestation answer are ”good” mix services and which are not.

A possible solution is to embed expected hashes of a remote attestation of a mix in the certificate of the mix itself. However, here the responsibility of the certification authorities issuing such certificates would be to verify the hashes. This would include checking the operating system environment and the mix.

Another solution is to provide transparency about the procedure how the operating system and the AN.ON mix is built. If this procedure is reproducible by experienced institutions and experienced users then they could create identical binaries. Since the source code of the anonymity service and the used microkernel environment TUDOS is publicly available this should be possible. Booting the setup on own local machines and determining the final hash value of a remote attestation should match the one specified in the mix certificate or match the publicly known respectively. expected hash value.

Of course here a lot of issues will appear. First the exact build environment must be known, which comprises things like libraries, compiler, linker and so on. A simple difference in options, configuration or versions of the tools will result in different binaries. This is a big challenge for open source environments like Gentoo where everybody compiles on its own, perhaps slightly different, environment. Therefore some infrastructure is needed where the result of building the mix and of building the operating system is listed for example based on a fresh installed operating system distribution. Associated hashes of the binaries, hashes of resulting PCRs and the final remote attestation hash should be provided so that searching for differences is possible for institutions and for experienced users. If several independent people could report same results then such a mix could be considered well built and configured - and finally trustworthy.

5.3 Where Size Matters

We aimed for an minimal trusted computing base. To see how far we got we measured Source Lines of Code (SLOC) of the whole scenario. As SLOCs are only an indirect factor of complexity we also use a second metric: gzipped compressed binary sizes.

The SLOC in table 1 sum up to nearly 700 thousand lines. To understand this surprisingly large number we have to look a little bit deeper into them.

The biggest parts that contribute to the overall size are the libraries used by the AN.ON mix. The Xerces XML library is a full-fledged validating XML parser used

Table 1. SLOC of anonymity service scenario on top of TUDOS

Service class	kilo SLOC	gzipped kB
Trusted computing bootstrap (Oslo)	2	5
L4 Fiasco Microkernel and bootstrap	30	85
Basic L4Env services	55	340
Network services	116	700
- tg3 driver	12	
- dde 2.6	36	
- FLIPS, IP stack	13	
- dde 2.4	40	
- others	15	
Filesystem services	25	215
Trusted computing service	2	45
AN.ON mix	460	1300
- xerces xml lib	210	
- openssl lib	200	
- AN.ON protocol and routing	50	

to generate and parse XML documents. As the mixes use only a subset of XML for configuration files and meta data, a simplified, non-validating XML parser and generator could be used. The Bastei [9] XML parser (300 SLOC) for example is at least two orders of magnitude smaller.

The same could be said for the OpenSSL library. Instead of using the standard library for cryptographic operations an embedded version can dramatically reduce the size. For instance MatrixSSL (<10 kSLOC) was used in a similar scenario [23].

There are two additional places to cut down the source: First, the separate IP stack FLIPS is still based on the Device Driver Environment (DDE) 2.4 whereas the network switch ORe uses the new DDE2.6. The 40 kSLOC for DDE2.4 could be omitted if we port FLIPS to the new DDE. Second, the 25 thousand lines for Posix-like filesystem IO are only used to load the configuration file. Using a simpler interface or even linking the configuration to the application would further reduce the complexity.

To sum this up we can conclude that our anonymity-service scenario can be built with less than 300 kSLOC, from which the first third is the basic system, the second third the network environment and the last third the mix application itself. The trusted computing part adds less than 2% (5 kSLOC) to such a scenario.

If we finally compare the current binary size of the scenario against a Linux version we can see that our implementation is much smaller. The complete AN.ON scenario requires 2.8 MB on x86 in gzipped binary form. This is an order of magnitude less than a Linux based solution, as the publicly available AN.ON Live CD [2] for instance requires 57MB. These numbers are not absolute in terms of smallest size, however indicate the range of the different solutions.

6 Conclusion and Outlook

Within this work we presented a server side trusted computing scenario. We achieved a better appreciable anonymous service for users by leveraging trusted computing technology. Based on the described extensions of the service, users are able to verify which software is running on the server. Now they have an additional mechanism to judge the trustworthiness of a mix beside the traditional ones like certificates and written statements of the operators.

This additional criterion together with the reduced trusted computing base of the mixes will make undiscovered attacks harder and should therefore increase the trustworthiness of the service.

Future work are to further reduce the trusted computing base as indicated in the evaluation section and to research how to structure and set up an infrastructure to handle ”good” mix hashes and software updates. A real world deployment with a high number of users, should be able to detect performance bottlenecks for example on the TPM operations. Finally we would like to research how the client side of this scenario can benefit from trusted computing.

Acknowledgements. We would like to thank the reviewers for their suggestions to improve the paper. Special thanks go to Stefan Köpsell, a main developer of AN.ON, for information and discussions about internals of the AN.ON service.

References

1. Project: AN.ON - Anonymity, `http://anon.inf.tu-dresden.de`
2. MixOnCD: AN.ON Live-CD, `http://anon.inf.tu-dresden.de/operators/help/mixsetupLiveCD_deployment_en.html`
3. Anonymisier-Dienst JAP ist wieder anonym, `http://www.heise.de/newsticker/meldung/39813`
4. BKA-Vorgehen gegen Anonymisierdienst JAP rechtswidrig, `http://www.heise.de/newsticker/meldung/41690`
5. JAP, die Macher und die nicht mehr so ganz garantierte Anonymitt, `http://www.heise.de/newsticker/meldung/39531`
6. Nicht mehr ganz anonym: Anonymisier-Dienst JAP protokolliert Zugriffe, `http://www.heise.de/newsticker/meldung/39508`
7. Arbaugh, W.A., Farber, D.J., Smith, J.M.: A secure and reliable bootstrap architecture. In: 1997 IEEE Symposium on Security and Privacy, Oakland, CA, May 1997, pp. 65–71 (1997)
8. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM 24(2), 84–90 (1981)
9. Feske, N., Helmuth, C.: Design of the Bastei OS architecture. Technical Report TUD-FI06-07-Dezember-2006, TU Dresden (2006)
10. Garriss, S., Cáceres, R., Berger, S., Sailer, R., van Doorn, L., Zhang, X.: Towards trustworthy kiosk computing. In: Mobile Computing Systems and Applications, 2007. HotMobile 2007, March 2007, pp. 41–45 (2007)
11. Goldschlag, D., Reed, M., Syverson, P.: Onion routing. Commun. ACM 42(2), 39–41 (1999)
12. Grawrock, D.: The Intel Safer Computing Initiative, January 2006. Intel Press (2006)

13. Gross, M.: Vertrauenswürdiges Booten als Grundlage authentischer Basissysteme. In: Verläßliche Informationssysteme, Tagungsband, Informatikfachberichte 271. Springer, Heidelberg (1991)
14. Härtig, H., Hohmuth, M., Feske, N., Helmuth, C., Lackorzynski, A., Mehnert, F., Peter, M.: The nizza secure-system architecture. In: CollaborateCom (2005)
15. Bumler, C.G.H., Federrath, H.: Report on the proceedings by criminal prosecution authorities against the project an on anonymity online (2003)
16. Kauer, B.: OSLO: Improving the Security of Trusted Computing. In: 16th USENIX Security Symposium, pp. 229–237.
17. Lampson, B., Abadi, M., Burrows, M., Wobber, E.: Authentication in distributed systems: Theory and practice. ACM Transactions on Computer Systems 10(4), 265–310 (1992)
18. McCune, J.M., Parno, B., Perrig, A., Reiter, M.K., Isozaki, H.: An Execution Infrastructure for TCB Minimization. Technical Report CMU-CyLab-07-018, Carnegie Mellon University (December 2007)
19. OSLO - Open Secure LOader, `http://os.inf.tu-dresden.de/~kauer/oslo`
20. Pearson, S. (ed.): Trusted Computing Platforms. Prentice Hall International, Englewood Cliffs (2002)
21. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a tcg-based integrity measurement architecture. In: SSYM 2004: Proceedings of the 13th conference on USENIX Security Symposium, Berkeley, CA, USA, p. 16. USENIX Association (2004)
22. Singaravelu, L., Kauer, B., Boettcher, A., Härtig, H., Pu, C., Jung, G., Weinhold, C.: Enforcing configurable trust in client-side software stacks by splitting information flow. Technical Report GIT-CERCS-07-11, Georgia Institute of Technology, Atlanta, GA (May 2007)
23. Singaravelu, L., Pu, C., Härtig, H., Helmuth, C.: Reducing tcb complexity for security-sensitive applications: three case studies. SIGOPS Oper. Syst. Rev. 40(4), 161–174 (2006)
24. Ta-Min, R., Litty, L., Lie, D.: Splitting interfaces: making trust between applications and operating systems configurable. In: OSDI 2006: Proceedings of the 7th symposium on Operating systems design and implementation, Berkeley, CA, USA, November 2006, pp. 279-292. USENIX Association (2006)
25. TCG: Trusted Computing Group, `https://www.trustedcomputinggroup.org`
26. Embassy leaks highlight pitfalls of Tor, `http://www.securityfocus.com/news/11486`

Combining Biometric Authentication with Privacy-Enhancing Technologies

Konstantin Hyppönen[1], Marko Hassinen[1], and Elena Trichina[2]

[1] University of Kuopio, Department of Computer Science
POB 1627, FIN-70211, Kuopio, Finland
{Konstantin.Hypponen,Marko.Hassinen}@cs.uku.fi
[2] Spansion International Inc., Willi-Brandt-Allee 4, 81829 Munich, Germany
Elena.Trichina@spansion.com

Abstract. Although state of public research in privacy-enhancing technologies (PET) is reasonably good, they are not yet widely used in common electronic documents. We argue that low acceptance of PET is due to a large gap between ordinary paper-based documents and new e-ID schemes. We show how to make the gap narrower by introducing a mobile electronic identity tool with privacy-preserving biometric authentication scheme.

1 Introduction

Identification cards, driving licenses, and passports are common travel documents used worldwide. Considerable amount of information about a person's identity is often revealed to people who actually need only a small part of it. Various IDs are checked way too often for privacy to be properly protected. Privacy-related issues have been addressed by privacy-enhancing technologies (PET), an umbrella term for schemes providing various levels of anonymity and unlinkability [1] of transactions performed by users. The state of public research in PET is considered rather mature [2]; however, their use in common travel documents is scarce.

Arguably, poor acceptance of PET in travel documents is due to a huge gap between old "paper-based" documents and PET-based electronic identity tools. Indeed, among other differences, PET-based solutions lack traditional look and feel of "real documents" and do not feature biometric authentication of the holder. This leads to lower usability, convenience and universality. Moreover, most PET-based solutions are targeted at user authentication in online access control systems. To increase usability and technology acceptance, an e-ID scheme should enable identification to occur with equal facility in the electronic realm as in the face-to-face situation.

The main goal of this work is a mobile identity system that allows controlled identity revelation (which is one definition of privacy). We argue that such a system can replace usual IDs in many applications where officially recognised identification is required. In addition, PET can be successfully combined with biometric authentication, often required in real-life situations. We present a security architecture for the system, and discuss technological issues of its implementation using today's handset technology.

Our contributions. This paper presents a way of combining biometric authentication with PET in travel documents. We use an idea of turning a mobile phone equipped with a tamper-resistant security element (e.g., a SIM-card) into a "portable privacy proxy".

The rest of the paper is organised as follows. The following section briefly describes officially recognised e-ID schemes and discusses potential use of PET in them. In Sect. 3 we list security requirements for a mobile identity system with an emphasis on privacy, and describe a way of using biometric authentication in it. We show that the system can be implemented with today's handset technology and using existing tools and platforms in Sect. 4. A security analysis of the proposed solution is presented in Sect. 5. Section 6 concludes the paper.

2 State of the Art

The main objectives for the switch to e-IDs in many countries are their better resistance to forgery, and use in new electronic governmental services [3]. At the same time, privacy-related issues have been taken into account only marginally. In this section, we give a short overview of currently used official e-IDs, and discuss possibility of using PET in travel documents.

2.1 Currently Used Electronic IDs

Most of the European countries have introduced *electronic identity cards*, driven by the demands set by the European Digital Signature Directive [4]. Also in some Asian countries electronic identity cards are already in use, see `www.asiaiccardforum.org`. In the USA, the federal chip-based ID card is currently under development. However, many privacy-protecting organisations (see, e.g., `www.realnightmare.org`) are objecting to it.

Privacy considerations have been partly taken into account in the design of identity cards. For instance, in the Finnish electronic identity card (FINEID) electronic user identification numbers do not equal social security numbers (SSN), nor can they be used for deriving SSNs. On the other hand, anyone who gets physical access to the card still receives a big amount of information about the user.

New *biometric passports* (also called e-passports or Machine Readable Travel Documents) are now issued by many countries. All data that is included on the information page of the passport is also stored in a chip embedded into the page.

The machine readable zone (MRZ) is used for constructing an access key to the data stored on the chip. This mechanism prevents remote skimming of passport data: In order to access the chip, the reader must possess the physical document. However, it was shown in several studies [5,6,7] that part of the contents of the MRZ can be easily guessed, making it possible to guess the access key in a relatively short amount of time. A good overview of security issues in biometric passports, supported by many studies, is given by the FIDIS project [8], stating that new biometric passports "dramatically decrease security and privacy and increase the risk of identity theft."

The use of the *mobile phone as an ID token* for user authentication is common in many business applications. The SIM card can work also as an officially recognised ID. In Finland, a mobile subscription customer can request a Finnish e-ID -compatible PKI-SIM card from their mobile network operator. Such cards can be used for mobile user authentication, and for creating digital signatures legally equal to the handwritten ones.

Applets installed on the (U)SIM card or in the phone can be used not only for user authentication, but also for more flexible identity management. A travel document can therefore be developed into a powerful mobile identity tool.

2.2 Privacy-Enhancing Technologies

Issues of security, usability and trust are of major importance in mobile identity systems. Mobile phones are seen by many users as potentially insecure devices that can reveal sensitive information about one's identity to unauthorised parties through the network. Therefore, the implementations must ensure that the user has full control over data sent by the application. The user must be able to prevent disclosure of any information by the device. Moreover, the systems that need information about users must receive only a necessary minimum of it, and remove the information as soon as it is not needed [9].

PET (see e.g. [10,11]) enable building identity systems whereby a person can make flexible proofs about the values of identity attributes. For instance, one can prove being over 22 and below 65 years old without surrendering any more information. Anonymous credential systems (e.g., [12]), make the proofs anonymous and unlinkable. However, in addition to attribute value proofs, most officially recognised travel documents require biometric authentication (usually, facial). Clearly, unlinkability stops here, as identity verifiers can easily cooperate by comparing digital biometric patterns read from documents and combining information associated with the same patterns. Therefore, combining PET with biometric authentication is not trivial. We also note that it is difficult to promote an "ID without a photo and leaving no traces" to decision-makers in governments, as it differs too much from documents used in the currently established identification practises.

3 Combining Biometric Authentication with PET

A privacy-preserving identification scheme should enable minimisation of data collection. User's consent has to be acquired prior to releasing any information. Just as in usual travel documents, authenticity and integrity of identity attribute values must be preserved. Exchanged information has to be protected from eavesdropping by third parties, and if possible, identity proofs should be made preserving the person's anonymity. To provide strong authentication of the person in face-to-face scenarios, biometric authentication must be used.

One can see that privacy requirements are set higher than in usual "paper-based" travel documents. Suitable PET schemes exist that implement all these requirements. We show now how to combine them with biometric authentication of the identity document holder.

The parties participating in the identification or transaction authorisation scenario are Prover (Peggy) and Verifier (Victor). Victor needs a certain proof about Peggy's identity or her rights. There is also a Trusted Third Party (TTP, Trent): an official certification authority (CA) that has issued Peggy her electronic ID.

The core component of the system is a mobile phone or PDA with a slot for a tamper-resistant security element. The security element is issued by the authorities, possibly in cooperation with the user's mobile network operator. Further on, we refer to it as a SIM card, although implementations may differ. At least the following information is stored on the SIM card:

1. Identity attribute names and values.
2. A public-key certificate of Trent.
3. A number of person's biometric patterns, signed by Trent.
4. A number of vouchers connecting the attribute values and corresponding biometric patterns.

A voucher could be implemented, in its simplest form, as a certificate issued by Trent to the biometric pattern and containing the attribute values. Depending on the credential system or PKI used, the SIM card can also contain:

– Trent's cryptographic commitments to the attribute values and corresponding secrets used by the card for making proofs about the attribute values.
– Certificates for either attribute values or Trent's commitments to them.
– Pseudonyms and pseudonymous certificates.
– Private and/or secret keys of Peggy.

The mobile phone is used as a proxy between the identity verifier and the SIM card. Software running on the mobile phone (we refer to it as to an *identity proxy*) mediates messages between Victor and Peggy's SIM card. Depending on the scheme, it also verifies requests submitted by Victor. In addition, it performs biometric authentication of Peggy, either automatically or with Victor's help.

The identity proxy application is provided by Trent and signed using his private key, corresponding to a public key certificate located on the SIM card. Exchange of messages between the identity proxy and the SIM card is allowed only after verification of the application signature against the public key. The verification is performed by the mobile phone (see Sect. 5 for details).

Biometric patterns are communicated by the SIM card to the identity proxy, but never to identity verifiers. For example, if the pattern is a photo of Peggy, the privacy proxy will display the photo on the screen of the mobile phone. Peggy will then show the phone to Victor, just as she would show a usual identity card. This step is performed after running the normal identity proof protocol.

Obviously, it must be ensured that an attacker cannot impersonate Peggy by running a malicious applet instead of the certified identity proxy. Although a malicious applet cannot access the SIM card, as it is not signed by Trent, it can be used for showing a forged photo (of, say, Mallory instead of Peggy). In order to prevent impersonation, the identity proxy calculates a short fingerprint (e.g., first 5 hexadecimal characters of a cryptographic hash) of all messages sent between Victor and Peggy. The fingerprint is added as a visible watermark on the photo, to ensure that both the photo and the proof come from the same place (the identity proxy).

Theoretically, a mobile phone camera could be used for face recognition, but in real-life situations the usability of such solution is much lower than simply showing a readily available photo on the screen. For other types of biometric patterns, such as fingerprints, embedded readers are needed. In this case, the identity proxy performs pattern matching and outputs its results together with the communication fingerprint. See Sect. 5 for details on how the results of matching should be output.

Biometric patterns come originally from the SIM card, but may be stored in the memory of the identity proxy to enable faster communication. To prevent possible tampering with the patterns in phone memory, the applet will retrieve the pattern signature from the SIM card and verify it every time before using it.

4 Pseudonymous Mobile Identity with Biometric Authentication

In this section, we give an example of a prototype built using the architecture presented in this paper. Note that the system implemented in the prototype is pseudonymous rather than anonymous, and therefore provides no unlinkability. However, the system enables partial revelation of identity. The main logical component of the system is the user's public-key certificate which, however, includes no plaintext data about the user. The certificate is issued to a biometric pseudonym, namely, to a hash of one selected biometric feature. For the sake of clarity we assume that a digital photograph of the person is used as the biometric. In the description, we use notation given in Table 1.

The certificate contains masked values of identity attributes as X.509v3 certificate extensions. The user's identity attributes ATTR_U^i, $i \in I$, are masked as $H(\mathrm{ATTR}_U^i|\mathrm{MASK}_U^i)$, where $H(\cdot)$ is a cryptographic hash function. The certificate is stored on the (U)SIM card, as part of a mobile identity management applet. The applet contains the following information:

- The user's certificate CERT_U, signed by the CA.
- The user's private key SKEY_U corresponding to the public key PKEY_U in the certificate.
- The CA certificate.
- The user's photo, BIO_U, hash of which, $H(\mathrm{BIO}_U)$, is included in the "subject info" field of the certificate.
- A number of identity attributes $\mathrm{ATTR}_U^{i_j}$, $i_j \in I$, such as name, surname, date of birth, social security number, nationality, and so on.
- For each attribute $\mathrm{ATTR}_U^{i_j}$, the attribute mask $\mathrm{MASK}_U^{i_j}$ used in the certificate.

Table 1. Protocol Notation

V, P, T	Parties: Victor (verifier), Peggy (prover), Trent (CA)
BIO_X	Biometric characteristic (e.g., photo) of subject X
I	Set of identity attribute types; a naming system (e.g., Object Identifiers, OID) is used for distinguishing the types
ATTR_X^i	Identity attribute of type $i \in I$ of subject X
MASK_X^i	Mask (random bit sequence) of attribute ATTR_X^i. Length l of the mask must be such that 2^l hash function evaluations is not feasible.
SKEY_X	The secret RSA key of subject X
PKEY_X	The public RSA key related to SKEY_X
CERT_X	The public key certificate of subject X
$H(m)$	A preimage (both first and second) resistant cryptographic hash (e.g., SHA-1) of the message m
TIME_X	A timestamp generated by subject X
SIG	A digital signature of the message (based on the private key of the sender)
$+$	A separator between message contents and message signature

In addition, the applet contains a program for operating the information listed above. Information is provided in form of *identity proofs*, structure of which is described later on. The proofs are signed with SKEY_U.

4.1 Protocol Description

The following protocol is used in identity proof transactions.

1. Request composition and transfer. Victor prepares a request for Peggy's personal information. The request contains a number of attribute identifiers, Victor's public-key certificate, and a timestamp. Victor signs the request with his private key, and sends it to Peggy.

$$\text{Victor} \rightarrow \text{Peggy}: \quad i_1\ [\ |\ i_2 \ldots]\ |\ \text{CERT}_V\ |\ \text{TIME}_V\ +\ \text{SIG}$$

2. Consent acquirement. Identity proxy at Peggy's phone verifies Victor's certificate and signature. If verification is successful, the phone asks Peggy for a confirmation, showing her Victor's request. If Peggy chooses to comply with the request, she enters her PIN code. For example, it may be written on the screen "Victor requests information on your first name and surname. Shall I provide it?". If Peggy has entered her PIN code correctly, the request is forwarded to the (U)SIM card. In addition, a timestamp TIME_P is attached to the request.

3. Proof construction. Having received the request, the (U)SIM card decodes it and constructs an identity proof of the form

$$\begin{aligned}\text{Peggy} \rightarrow \text{Victor}: \quad & \langle i_1\ |\ \text{ATTR}_P^{i_1}\ |\ \text{MASK}_P^{i_1}\rangle\ [\ |\ \langle i_2\ |\ \text{ATTR}_P^{i_2}\ |\ \text{MASK}_P^{i_2}\rangle \ldots]\ | \\ & \text{CERT}_P\ |\ \text{TIME}_P\ |\ \text{BIO}_P\ +\ \text{SIG}\end{aligned}$$

The identity proof is sent by the (U)SIM card to the identity proxy, which encrypts the proof with Victor's public key and forwards it to Victor.

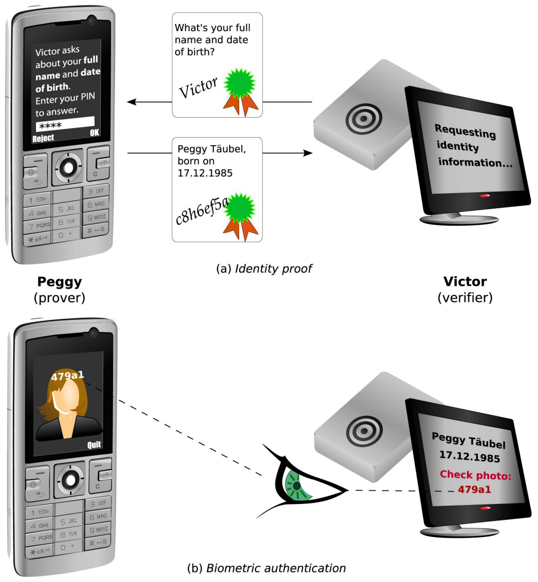

Victor asks about your full name and date of birth. Enter your PIN to answer.
****
Reject
OK
What's your full name and date of birth?
Victor
Peggy Täubel, born on 17.12.1985
c8h6ef5a
Requesting identity information...
(a) Identity proof
Peggy (prover)
Victor (verifier)
479a1
Quit
Peggy Täubel 17.12.1985
Check photo: 479a1
(b) Biometric authentication

Fig. 1. An example of an identity proof scenario

4. Proof verification. Having received the identity proof from Peggy, Victor first checks the validity of the certificate presented by Peggy, and verifies her signature of the proof. Victor computes a cryptographic hash of messages sent in stages 1 and 3 of the protocol. The identity information and 5 first hexadecimal characters of the hash are displayed on the screen of Victor's terminal.

5. Biometric authentication. The identity proxy on Peggy's phone verifies the integrity of Peggy's photo by computing a hash of the photo and comparing it against the hash value in the subject name field of the certificate. It then computes a proof fingerprint by calculating a cryptographic hash of messages sent in stages 1 and 3 of the protocol and taking first 5 hexadecimal characters of it. Peggy's photo is displayed on the phone screen, with the proof fingerprint added as a visible watermark to it. Peggy shows her phone to Victor. If the photo matches with Peggy's appearance, Victor accepts the identity proof.

An example of an identity proof scenario implemented by this protocol is presented in Fig. 1.

4.2 Implementation

We have developed a proof-of-concept implementation of the mobile identity tool described above. We use Near Field Communication (NFC) (see `www.nfc-forum.org`) as the interconnection technology between Peggy and Victor. The reasons for this choice (instead of, for example, Bluetooth) are usability and speed.

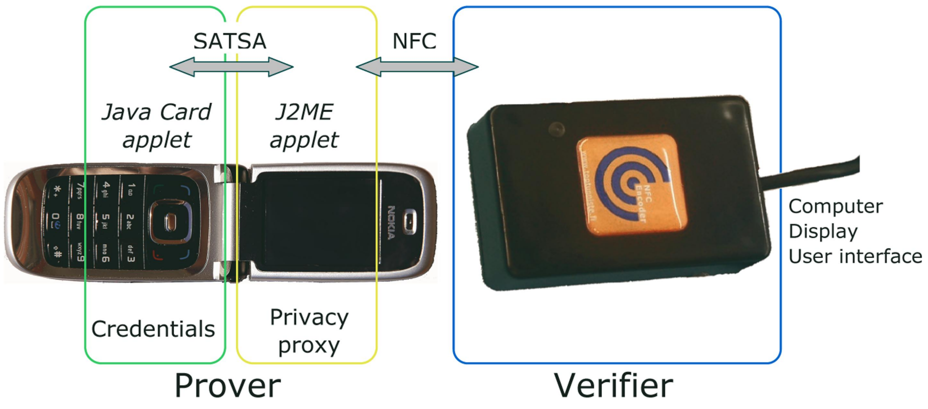


Fig. 2. Implementation: main idea and technologies involved

We have used J2ME with Contactless Communications API [13] and Security and Trust Services API (SATSA) [14] for the implementation of the identity proxy, and Java Card for the implementation of the software on the (U)SIM card. The SATSA API is used to facilitate exchange of messages between the J2ME applet and the SIM card. At the moment, none of the mobile phones that support NFC have support for SATSA, and vice versa. Therefore, we had to test the application using a mobile phone emulator (Nokia 6131 NFC SDK). For testing the application with real NFC phones, we have also implemented a version which does not use SATSA API, but instead stores all user identity information in the J2ME applet. As the identity field codes, we used standard X.509 attribute codes [15]. Devices and technologies involved in the implementation are outlined in Fig. 2.

5 Security Evaluation

In Sect. 3 we provided an outlook on requirements that a mobile identity tool must implement. Obviously, the implementation of these requirements depends on the PET used in the system. In our prototype, the requirement on minimisation of data collection is implemented partly: Identity attribute values can be either revealed or held back, but flexible proofs about values (e.g., belonging to an interval) cannot be performed. Our prototype meets the requirements on user consent, authenticity, and integrity. For revocation, the same type of blacklisting as that for usual credit cards can be used.

Just as in case of all other e-IDs, Peggy does not have control over what information is stored in Victor's system after a transaction. Ideally, Victor's software and hardware has to be evaluated by an independent laboratory and certified to meet privacy standards, to make sure that only necessary information (if any) is stored therein.

The prototype of a mobile identity tool presented in this paper is pseudonymous rather than anonymous. Indeed, Victor can cooperate with other identity verifiers to get more information about Peggy, by collecting all information that corresponds to a given hash of the photo. By using a suitable PET based on cryptographic commitments and zero-knowledge proof protocols (e.g., [11,16,12] identity proofs can be made anonymous and unlinkable. It is a question of discussion whether such systems are suitable for travel documents.

One attack idea (targeted at Peggy) is to install a "virus" to Peggy's mobile phone, to have all information about her identity transmitted to the attacker. However, this attack is hard to implement due to a protection provided by SATSA. To connect to an applet in the SIM card, the J2ME applet ("virus", in this case) must be signed using the operator or TTP domain certificate. An access control entry with a hash of this certificate is stored on the SIM card. The implementation of SATSA on the mobile phone compares this hash with that of the certificate used for signing the J2ME applet. The J2ME applet is allowed to exchange messages with the applet on the SIM only if the hashes match. Therefore, the attacker must have the "virus" signed by the operator or TTP before it can be used for identity theft. Another option would be to find a serious vulnerability in the implementation of SATSA and exploit it. On emerging mobile secure environments (e.g., M-Shield, `www.ti.com/m-shield`) and with the use of trusted mobile platforms [17] this risk could be minimised.

The identity proof fingerprint prevents an attacker from impersonating Peggy: if the attacker quickly switches from the identity proxy to another application for showing the attacker's photo in step 5 of the protocol, he has to also show the correct fingerprint. The security architecture of J2ME uses the sandbox model, preventing applications from accessing memory space of other applications, therefore the attacker's program cannot intercept data sent by the identity proxy. The fingerprint calculation includes two digital signatures made with private keys unknown to the attacker; this essentially cuts the chances of guessing the fingerprint to 1 in $16^5 = 1048576$.

Malware with screen capture capabilities could potentially be used to make a screenshot of the identity proxy in stage 5 of the protocol, extract the identity proof fingerprint and place it on another photo, to impersonate the legitimate user. To prevent these "screenshot attacks", the following techniques can be used:

– Visual distortion of the identity proof fingerprint (using, e.g., CAPTCHA-like techniques [18]).
– Placing the fingerprint in several random locations.
– Slight randomised distortion of the user's photo (using, e.g. Stirmark [19]). This is useful for preventing the attacker from analysing a captured watermarked image against a non-watermarked one, for extracting the identity proof fingerprint. The non-watermarked image could be obtained by the attacker by using collusion attacks on a number of watermarked images.

Similar techniques could be used for outputting the results of biometric authentication based on other biometric patterns (e.g., fingerprints). The results should be output on randomly generated backgrounds, designed for preventing optical character recognition.

6 Conclusions

This paper described how to use a mobile phone as an electronic ID, combining privacy-enhancing technologies with biometric authentication. Using the presented architecture, a travel document with controlled identity revelation and other privacy-protecting features can be implemented. With the use of a governmental PKI this ID can become officially recognised so that it can be used in various official purposes.

A prototype of such electronic ID, related protocols and implementation issues have been described. We have also provided security evaluation of the architecture and discussed potential attacks and protection against them.

References

1. Pfitzmann, A., Hansen, M.: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology. version v0.29 (2007)
2. Pfitzmann, A.: Multilateral security: Enabling technologies and their evaluation. In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 1–13. Springer, Heidelberg (2006)
3. CEN/ISSS Workshop eAuthentication: Towards an electronic ID for the European Citizen, a strategic vision. Brussels (2004) (accessed 10.10.2007), `http://europa.eu.int/idabc/servlets/Doc?id=19132`
4. The European Parliament and the Council of the European Union: Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. Official Journal L 013, 0012–0020 (2000)
5. Witteman, M.: Attacks on digital passports. Talk at the What The Hack conference (2005) (Accessed 10.10.2007), `http://wiki.whatthehack.org/images/2/28/WTH-slides-Attacks-on-Digital-Passports-Marc-Witteman.pdf`
6. Juels, A., Molnar, D., Wagner, D.: Security and privacy issues in e-passports. In: Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference, pp. 74–88 (2005)
7. Hoepman, J.H., Hubbers, E., Jacobs, B., Oostdijk, M., Schreur, R.: Crossing borders: Security and privacy issues of the european e-passport. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., Kawamura, S. (eds.) IWSEC 2006. LNCS, vol. 4266, pp. 152–167. Springer, Heidelberg (2006)
8. FIDIS - Future of Identity in the Information Society: Budapest declaration on machine readable travel documents (MRTDs) (2006) (Accessed 10.10.2007), `http://www.fidis.net/fileadmin/fidis/press/budapest_declaration_on_MRTD.en.20061106.pdf`
9. The Royal Academy of Engineering: Dilemmas of privacy and surveillance: Challenges of technological change. The Royal Academy of Engineering, 29 Great Peter Street, London, SW1P 3LW (2007)

10. Brands, S.A.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. The MIT Press, Cambridge (2000)
11. Li, J., Li, N.: A construction for general and efficient oblivious commitment based envelope protocols. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 122–138. Springer, Heidelberg (2006)
12. Camenisch, J., Herreweghen, E.V.: Design and implementation of the idemix anonymous credential system. In: CCS 2002: Proceedings of the 9th ACM conference on Computer and communications security, pp. 21–30. ACM, New York (2002)
13. Java Community Process: Contactless Communication API, JSR 257, v. 1.0. Nokia Corporation, Espoo, Finland (2006) (Accessed 10.10.2007), `http://www.jcp.org/en/jsr/detail?id=257`
14. Java Community Process: Security and Trust Services API (SATSA) for Java^{TM}2 Platform, Micro Edition, v. 1.0. Sun Microsystems, Inc., Santa Clara, CA, USA (2004) (accessed 10.10.2007), `http://www.jcp.org/en/jsr/detail?id=177`
15. Santesson, S., Polk, W., Barzin, P., Nystrom, M.: Internet X.509 public key infrastructure qualified certificates profile. Network Working Group, Request for Comments 3039 (2001) (accessed 10.10.2007)
16. Boudot, F.: Partial revelation of certified identity. In: Domingo-Ferrer, J., Chan, D., Watson, A. (eds.) CARDIS. IFIP Conference Proceedings, vol. 180, pp. 257–272. Kluwer, Dordrecht (2000)
17. Trusted Computing Group: TCG mobile trusted module specification, version 1.0, revision 1. TCG published (2007)
18. von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: CAPTCHA: Using hard AI problems for security. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 294–311. Springer, Heidelberg (2003)
19. Petitcolas, F.A.P., Steinebach, M., Raynal, F., Dittmann, J., Fontaine, C., Fates, N.: Public automated web-based evaluation service for watermarking schemes: StirMark benchmark. In: Wong, P.W., Delp III, E.J. (eds.) Security and watermarking of multimedia contents III: SPIE proc. ser., vol. 4314, pp. 575–584. SPIE (2001)

A New Direct Anonymous Attestation Scheme from Bilinear Maps

Ernie Brickell[1], Liqun Chen[2], and Jiangtao Li[1]

[1] Intel Corporation
{ernie.brickell,jiangtao.li}@intel.com
[2] HP Laboratories
liqun.chen@hp.com

Abstract. Direct Anonymous Attestation (DAA) is a cryptographic mechanism that enables remote authentication of a user while preserving privacy under the user's control. The DAA scheme developed by Brickell, Camenisch, and Chen has been adopted by the Trust Computing Group (TCG) for remote anonymous attestation of Trusted Platform Module (TPM), a small hardware device with limited storage space and communication capability. In this paper, we propose a new DAA scheme from elliptic curve cryptography and bilinear maps. The lengths of private keys and signatures in our scheme are much shorter than the lengths in the original DAA scheme, with a similar level of security and computational complexity. Our scheme builds upon the Camenisch-Lysyanskaya signature scheme and is efficient and provably secure in the random oracle model under the LRSW (stands for Lysyanskaya, Rivest, Sahai and Wolf) assumption and the decisional Bilinear Diffie-Hellman assumption.

Keywords: direct anonymous attestation, elliptic curve cryptography, bilinear map, trusted platform module, the Camenisch-Lysyanskaya signature scheme.

1 Introduction

The concept and a concrete scheme of Direct Anonymous Attestation (DAA) were first introduced by Brickell, Camenisch, and Chen [6] for remote anonymous authentication of a hardware module, called Trusted Platform Module (TPM). The DAA scheme was adopted by the Trusted Computing Group (TCG) [30], an industry standardization body that aims to develop and promote an open industry standard for trusted computing hardware and software building blocks. The DAA scheme was standardized in the TCG TPM Specification Version 1.2 [29]. A historical perspective on the development of DAA was provided by the DAA authors in [7].

A DAA scheme involves three types of entities: a DAA issuer, DAA signers, and DAA verifiers. The issuer is in charge of verifying the legitimation of signers and of issuing a DAA credential to each signer. A DAA signer can prove membership to a verifier by signing a DAA signature. The verifier can verify the membership credential from the signature but he cannot learn the identity of the signer. DAA is targeted for implementation in TPM which has limited storage space and computation capability. For this purpose, the role of the DAA signer is spilt between a TPM and a host that has the TPM "built in". The TPM is the real signer and holds the secret signing key, whereas the host helps the TPM to compute the signature under the credential, but is not allowed to learn the secret signing key and to forge such a signature without the TPM involvement.

The most interesting feature of DAA is to provide differing degrees of privacy. A DAA scheme can be seen as a special group signature scheme without the feature of opening the signer's identity from its signature by the issuer. Interactions in DAA signing and verification are anonymous, that means the verifier, the issuer or both of them colluded cannot discover the signer's identity from its DAA signature. However, the signer and verifier may negotiate as to whether or not the verifier is able to link different signatures signed by the signer.

DAA has drawn a lot of attention from both industry and cryptographic researches. Pashalidis and Mitchell showed how to use DAA in a single sign-on application [23]. Balfe, Lakhani and Paterson utilized a DAA scheme to enforce the use of stable, platform-dependent pseudonyms and reduce pseudo-spoofing in peer-to-peer networks [2]. Leung and Mitchell made use of a DAA scheme to build a non-identity-based authentication scheme for a mobile ubiquitous environment [19]. Camenisch and Groth [9] found that the performance of the original DAA scheme can be improved by introducing randomization of the RSA-based Camenisch-Lysyanskaya signature (CL-RSA) [10]. Rudolph [26] pointed out that if the DAA issuer is allowed to use multiple public keys each for issuing one credential only, it can violate anonymity of DAA. It is obviously true. Like a group signature scheme, the anonymous property is relied on the reasonable large size of the signer group that is associated with a single group manager's public key. Smyth, Chen and Ryan discussed how to ensure privacy when using DAA with corrupt administrators [28]. Backes et al. [1] presented a mechanized analysis of the original DAA scheme. Ge and Tate [18] proposed a very interesting DAA scheme with efficient signing and verification implementation but inefficient joining implementation. The security of this scheme, as the same as the original DAA scheme, is based on the strong RSA assumption and the decisional Diffie-Hellman assumption.

One limitation of the original DAA scheme [6] is that the lengths of private keys and DAA signatures are quite large for a small TPM, i.e., around 670 bytes and 2800 bytes, respectively. In this paper, we present a new DAA scheme from elliptic curve cryptography and bilinear maps. This DAA scheme builds on top of the Camenisch and Lysyanskaya signature scheme [11] based on the LRSW problem [21] (CL-LRSW). Like most other pairing-based cryptographic mechanisms, our scheme requires a much shorter key length compared with the original integer factorization based DAA scheme. The lengths of private keys and signatures in our new scheme are approximately one third and one fifth of the original DAA scheme, respectively, with a similar level of security and computational complexity. We prove that this DAA scheme is secure in the random oracle model and under the LRSW assumption and the decisional Bilinear Diffie-Hellman assumption.

Rest of this paper is organized as follows. We first recall the formal specification of DAA in Section 2. We then review the notations on bilinear maps, some relative security assumptions, and some known cryptographic building blocks in Section 3, which will be used in the description of our new DAA scheme in Section 4. In the end, we consider how to implement the new DAA scheme in Section 5 and conclude our paper in Section 6.

2 Form Specification and Security Model of DAA

In this section we recall the formal specification of DAA [6], which uses an ideal-system/real-system model to prove security of DAA based on security models for multi-party computation [13] and reactive systems [24].

We review the basic ideas of the ideal-system/real-system model as follows. In the real system there are a number of players who run some cryptographic protocols with each other, an adversary $\mathcal{A}$ who controls a set of dishonest players, and an environment $\mathcal{E}$ that (1) provides the players with inputs and (2) arbitrarily interacts with $\mathcal{A}$. The environment provides the inputs to the honest players and receives their outputs, and interacts arbitrarily with $\mathcal{A}$. The dishonest players are fully controlled by $\mathcal{A}$, who monitors all the messages sent to the dishonest players and generates output messages for them. In the ideal system, we have the same players. However, they do not run any cryptographic protocols but send all their inputs to and receive all their outputs from an ideal trusted third party $\mathcal{T}$. This party computes the output of the players from their inputs, i.e., applies the functionality that the cryptographic protocols are supposed to realize. A cryptographic protocol is said to implement securely a functionality if for every adversary $\mathcal{A}$ and every environment $\mathcal{E}$ there exists a simulator $\mathcal{S}$ controlling the same players in the ideal system as $\mathcal{A}$ does in the real system such that the environment $\mathcal{E}$ can not distinguish whether it is run in the real system and interacts with $\mathcal{A}$ or whether it is run in the ideal system and interacts with $\mathcal{S}$.

We now specify the functionality of DAA [6]. There are five kinds of players: an issuer $\mathcal{I}$, a TPM $\mathcal{M}_i$, a host $\mathcal{H}_i$, a verifier $\mathcal{V}_j$, and a revocation oracle $\mathcal{O}$. $\mathcal{M}_i$ and $\mathcal{H}_i$ form a platform in the trusted computing environment and share the role of a DAA signer. In the following specification, a basename `bsn` is used to provide the property of a possible link, which is controlled by a signer and a verifier, between multiple DAA signatures signed under the same DAA key. In the ideal system we support the following operations:

Setup. Each player indicates to $\mathcal{T}$ whether or not it is corrupted. Each TPM $\mathcal{M}_i$ sends its unique identity $\mathtt{id}_i$ to $\mathcal{T}$ who forwards it to the respective host $\mathcal{H}_i$.

Join. The host $\mathcal{H}_i$ contacts $\mathcal{T}$ and requests to become a member. Thus $\mathcal{T}$ sends the corresponding TPM $\mathcal{M}_i$ and asks it whether it wants to become a member. Then, $\mathcal{T}$ asks the issuer $\mathcal{I}$ whether the platform with identity `id` can become a member. If $\mathcal{M}_i$ was tagged rogue, $\mathcal{T}$ also tell $\mathcal{I}$ this. If the issuer agrees, $\mathcal{T}$ notifies $\mathcal{H}_i$ that it has become a member.

Sign/Verify. A host $\mathcal{H}_i$ wants to sign a message m with respect to some basename $\texttt{bsn} \in \{0,1\}^* \cup \{\perp\}$ for some verifier $\mathcal{V}_j$. So $\mathcal{H}_i$ sends m, $\texttt{bsn}$ to $\mathcal{T}$. If $\mathcal{H}_i/\mathcal{M}_i$ are not a member, then $\mathcal{T}$ denies the request. Otherwise, $\mathcal{T}$ forwards m to the corresponding $\mathcal{M}_i$ and asks it whether it wants to sign. If it does, $\mathcal{T}$ tells $\mathcal{H}_i$ that $\mathcal{M}_i$ agrees and asks it w.r.t. which basename $\texttt{bsn}$ it wants to sign (or whether it wants to abort). If $\mathcal{H}_i$ does not abort, $\mathcal{T}$ proceeds as follows

- If $\mathcal{M}_i$ has been tagged rogue, $\mathcal{T}$ lets $\mathcal{V}_j$ know that a rogue TPM has signed m.
- If $\texttt{bsn} = \perp$ then $\mathcal{T}$ informs $\mathcal{V}_j$ that m has been signed w.r.t. $\texttt{bsn}$.
- If $\texttt{bsn} \neq \perp$ then $\mathcal{T}$ checks whether $\mathcal{H}_i/\mathcal{M}_i$ have already signed a message w.r.t. $\texttt{bsn}$. If this is the case, $\mathcal{T}$ looks up the corresponding pseudonym P in its database; otherwise $\mathcal{T}$ chooses a new random pseudonym P. Finally, $\mathcal{T}$ informs $\mathcal{V}_j$ that the platform with pseudonym P has signed m.

Revoke. $\mathcal{O}$ tells $\mathcal{T}$ to tag of the platform with identity $\texttt{id}$ as a rogue. If the TPM with identity $\texttt{id}$ is not corrupted, $\mathcal{T}$ denies the request. Otherwise, $\mathcal{T}$ marks the TPM with identity $\texttt{id}$ as rogue.

Observe that the ideal system model captures both unforgeability and anonymity/pseudonymity. More precisely, a signature can only be produced with the involvement of a TPM that is a member and is not tagged rogue. Furthermore, signatures involving the same TPM using the same basename are linkable to each other via a pseudonym, but if they are done with regard to different basenames or no basename then they cannot be linked. These two properties hold, regardless of whether or not the corresponding host is corrupted. Anonymity/pseudonymity is only guaranteed if both the host and the TPM are honest, as a dishonest host can always announce its identity and the messages it signed.

3 Background and Building Blocks

3.1 Background on Bilinear Maps

As in Boneh and Franklin's identity-based encryption scheme [5] and the Camenisch and Lysyanskaya (CL-LRSW) signature scheme [11], our new DAA scheme makes use of a bilinear map $e : G \times G \rightarrow \mathsf{G}$, where G and G denotes two groups of prime order q. The map e satisfies the following properties:

1. Bilinear. For all $P, Q \in G$, and for all $a, b \in \mathbb{Z}_q$, $e(P^a, Q^b) = e(P, Q)^{ab}$.
2. Non-degenerate. There exists some $P, Q \in G$ such that $e(P, Q)$ is not the identity of G.
3. Computable. There exists an efficient algorithm for computing $e(P, Q)$ for any $P, Q \in G$.

A bilinear map satisfying the above properties is said to be an admissible bilinear map. Such bilinear map is also known as the symmetric pairing. In Section 5, we will give a concrete example of groups G, G and an admissible bilinear map between them. The group G is a subgroup of the group of points of an elliptic curve $E(\mathbb{F}_p)$ for a large prime p. The group G is a subgroup of the multiplicative group of a finite field $\mathbb{F}^*_{p^2}$. We shall use the Tate pairing to construct an admissible bilinear map between these two groups.

In general, one can consider bilinear maps $e : G_1 \times G_2 \rightarrow \mathsf{G}$ where G_1, G_2, G are cyclic groups of prime order q. However in our paper, we limit ourself to symmetric pairing where $G_1 = G_2$, because our scheme builds upon the CL-LRSW signature scheme, which uses the symmetric pairing.

3.2 Cryptographic Assumptions

The security of our DAA scheme relies on the Decisional Bilinear Diffie-Hellman (DBDH) assumption and the Lysyanskaya, Rivest, Sahai, and Wolf (LRSW) assumption. We now state these two assumptions as follows:

Assumption 1 (DBDH Assumption). *Let $G = \langle g \rangle$ be a bilinear group defined above of prime order q. For sufficiently large q, the distribution $\{(g, g^a, g^b, g^c, e(g, g)^{abc})\}$ is computationally indistinguishable from the distribution $\{(g, g^a, g^b, g^c, e(g, g)^d)\}$, where a, b, c, d are random elements from $\mathbb{Z}_q$.*

The DBDH assumption is a natural combination of the Decisional Diffie-Hellman assumption and Bilinear Diffie-Hellman assumption. It has been used in many cryptographic schemes, e.g., Boneh and Boyen's construction of secure identity based encryption scheme without random oracles [4].

Assumption 2 (LRSW Assumption). *Let $G = \langle g \rangle$ be a cyclic group, $X, Y \in G$, $X = g^x$, and $Y = g^y$. Suppose there is an oracle that, on input $m \in \mathbb{Z}_q$, outputs a triple (a, a^y, a^{x+mxy}) for a randomly chosen $a \in G$. Then there exists no efficient adversary that queries the oracle polynomial number of times, and outputs (m, a, b, c) such that $m \neq 0$, $b = a^y$ and $c = a^{x+mxy}$ where m has not been queried before.*

The LRSW assumption was introduced by Lysyanskaya et al. [21] and was shown that this assumption holds for generic groups. This assumption is also used in the CL-LRSW signature scheme [11].

3.3 Protocols for Proof of Knowledge

In our scheme we will use various protocols to prove knowledge of and relations among discrete logarithms. To describe these protocols, we use notation introduced by Camenisch and Stadler [12] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For example, $PK\{(a, b) : y_1 = g_1^a h_1^b \wedge y_2 = g_2^a h_2^b\}$ denotes a proof of knowledge of integers a and b such that $y_1 = g_1^a h_1^b$ and $y_2 = g_2^a h_2^b$ holds, where $y_1, g_1, h_1, y_2, g_2, h_2$

are elements of some groups $G_1 = \langle g_1 \rangle = \langle h_1 \rangle$ and $G_2 = \langle g_2 \rangle = \langle h_2 \rangle$. The variables in the parenthesis denote the values the knowledge of which is being proved, while all other parameters are known to the verifier. Using this notation, a proof of knowledge protocol can be described without getting into all details. In the random oracle model, such proof of knowledge protocols can be turned into signature schemes using the Fiat-Shamir heuristic [16,25]. We use the notation $SPK\{(a) : y = z^a\}(m)$ to denote a signature on a message m obtained in this way.

In this paper, we use the following known proof of knowledge protocols:

– Proof of knowledge of discrete logarithms. A proof of knowledge of a discrete logarithm of an element $y \in G$ with respect to a base z [27] is denoted as $PK\{(a) : y = z^a\}$. A proof of knowledge of a representation of an element $y \in G$ with respect to several bases $z_1, \ldots, z_v \in G$ [15] is denoted $PK\{(a_1, \ldots, a_v) : y = z_1^{a_1} \cdot \ldots \cdot z_v^{a_v}\}$.
– Proof of knowledge of equality. A proof of equality of discrete logarithms of two group elements $y_1, y_2 \in G$ to the bases $z_1, z_2 \in G$, respectively, [14] is denoted $PK\{(a) : y_1 = z_1^a \wedge y_2 = z_2^a\}$. Such protocol can also be used to prove that the discrete logarithms of two group elements $y_1 \in G_1$ and $y_2 \in G_2$ to the bases $z_1 \in G_1$ and $z_2 \in G_2$, respectively, in two different groups G_1 and G_2 are equal [8].

3.4 Camenisch-Lysyanskaya Signature Scheme

Our DAA scheme builds upon the Camenisch-Lysyanskaya (CL-LRSW) signature scheme [11]. Unlike most signature schemes, this one is particularly suited for our purposes as (1) it uses bilinear maps and (2) it allows for efficient protocols to prove knowledge of a signature and to obtain a signature on a secret message based on proofs of knowledge [11]. We now review the CL-LRSW signature scheme as follows.

Key Generation. This algorithm chooses two groups $G = \langle g \rangle$ and $\mathsf{G} = \langle \mathsf{g} \rangle$ of prime order q and an admissible bilinear map e between G and G. Next choose $x \leftarrow \mathbb{Z}_q$ and $y \leftarrow \mathbb{Z}_q$, and set the public key as $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$ and the secret key as (x, y), where $X = g^x$ and $Y = g^y$.

Signature. On input a message m, the secret key (x, y), and the public key $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$, choose a random $a \in G$, and output the signature $\sigma = (a, a^y, a^{x+mxy})$.

Verification. On input the public key $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$, the message m, and the signature $\sigma = (a, b, c)$ on m, check whether the following equations hold $e(Y, a) \stackrel{?}{=} e(g, b)$, $e(X, a) \cdot e(X, b)^m \stackrel{?}{=} e(g, c)$.

Observe that from a signature $\sigma = (a, b, c)$ on a message m, it is easy to compute a different signature $\sigma' = (a', b', c')$ on the same message m without the knowledge of the secret key: just choose a random number $r \in \mathbb{Z}_q$ and compute $a' := a^r$, $b' := b^r$, $c' := c^r$.

Theorem 1 ([11]). *The CL-LRSW signature scheme described above is secure against adaptive chosen message attacks under the LRSW assumption.*

4 New DAA Scheme from Bilinear Maps

4.1 The Scheme

We now present a new DAA scheme from bilinear maps based on the CL-LRSW signature scheme [11]. In the DAA protocol, there are three types of entities: an issuer $\mathcal{I}$, signers, and verifiers $\mathcal{V}$. Each signer (a trusted platform) consists of a host $\mathcal{H}$ and a TPM $\mathcal{M}$. All communications between $\mathcal{M}$ and $\mathcal{I}$ are through $\mathcal{H}$. For any computation performed by a signer, part of it is performed on the $\mathcal{M}$ while the rest of the computation is done on $\mathcal{H}$. Since the TPM is a small chip with limited resources, a requirement for DAA is that the operations carried out on the TPM should be minimal. We have the following four operations:

Setup. Let ℓ_q, ℓ_H, and ℓ_ϕ be three security parameters, where ℓ_q is the size of the order q of the groups, ℓ_H is the output length of the hash function used for Fiat-Shamir heuristic, and ℓ_ϕ is the security parameter controlling the statistical zero-knowledge property. The issuer $\mathcal{I}$ chooses two groups $G = \langle g \rangle$ and $\mathsf{G} = \langle \mathsf{g} \rangle$ of prime order q and an admissible bilinear map e between G and G, i.e., $e : G \times G \to \mathsf{G}$. $\mathcal{I}$ then chooses $x \leftarrow \mathbb{Z}_q$ and $y \leftarrow \mathbb{Z}_q$ uniformly at random, and computes $X := g^x$ and $Y := g^y$. For simplicity, throughout the scheme specification, the exponentiation operation h^a for $h \in G$ and an integer a outputs an element in G. $\mathcal{I}$ sets the group public key as $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$ and its private key as (x, y), and publishes the group public key. Let $H(\cdot)$ and $H_{\mathsf{G}}(\cdot)$ be two collision resistant hash functions, such that $H : \{0,1\}^* \to \{0,1\}^{\ell_H}$ and $H_{\mathsf{G}} : \{0,1\}^* \to \mathsf{G}$. Observe that the correctness of the group public key can be verified, e.g. by the host $\mathcal{H}$, by checking whether each element is in the right groups or not.

Join. We assume that the signer and the issuer $\mathcal{I}$ have established a one-way authentic channel, i.e., $\mathcal{I}$ needs to be sure that it talks to the right signer (i.e. a platform with a specific TPM). The authentic channel can be achieved in various ways. The one TCG recommended is that every message sent from $\mathcal{I}$ to the signer is encrypted under the TPM endorsement key [29]. Let $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$ be the group public key, K_I be a long-term public key of $\mathcal{I}$. Let `DAAseed` be the seed to compute the secret key of the signer. Note that we do not include the issuer's basename $\mathtt{bsn}_I$ in this operation, since we assume that the value g is unique for the issuer. However, if a different $\mathtt{bsn}_I$ is required, it can be added easily in the same way as in the original DAA scheme [6]. The join protocol takes the following steps:

1. $\mathcal{M}$ first computes

$$f := H(\mathtt{DAAseed} \| K_I) \bmod q, \qquad\qquad F := g^f,$$

where $\|$ stands for the operation of concatenation. $\mathcal{M}$ then chooses a random $r_f \leftarrow \mathbb{Z}_q$, computes $\tilde{T} := g^{r_f}$ and sends $\tilde{T}$ and F to $\mathcal{H}$.
2. The issuer chooses a random string $n_I \in \{0,1\}^{\ell_H}$ and sends it to $\mathcal{H}$.
3. $\mathcal{H}$ computes $\mathfrak{c}_h := H(q\|g\|\mathsf{g}\|X\|Y\|F\|\tilde{T}\|n_I)$ and sends it to $\mathcal{M}$.

4. $\mathcal{M}$ chooses a random string $n_T \in \{0,1\}^{\ell_\phi}$ and computes $\mathfrak{c} := H(\mathfrak{c}_h \| n_T)$ and $s_f := r_f + \mathfrak{c} \cdot f \bmod q$. $\mathcal{M}$ sets f as its private key and sends $(F, \mathfrak{c}, s_f, n_T)$ to $\mathcal{I}$ through $\mathcal{H}$.
5. $\mathcal{I}$ checks its record and policy to find out whether the value F should be rejected or not. If F belongs to a rogue TPM or does not pass the issuer's policy check, e.g. having been required for an credential too many times, $\mathcal{I}$ aborts the protocol.
6. $\mathcal{I}$ computes $\hat{T} := g^{s_f} F^{-\mathfrak{c}}$, and verifies that $\mathfrak{c} \overset{?}{=} H(H(q\|g\|\mathsf{g}\|X\|Y\|F\|\hat{T}\| n_I)\|n_T)$. If the verification fails, $\mathcal{I}$ aborts. Otherwise, $\mathcal{I}$ chooses $r \leftarrow \mathbb{Z}_q$, and computes

$$a := g^r, \qquad b := a^y, \qquad c := a^x F^{rxy}.$$

Observe that $c = a^x g^{frxy} = a^{x+fxy}$ and therefore (a,b,c) is a CL-LRSW signature on f. $\mathcal{I}$ sets (a,b,c) to be the credential for $\mathcal{M}$, and sends (a,b,c) to the signer, $\mathcal{M}$ and $\mathcal{H}$.
7. If verifying (a,b,c) is required, $\mathcal{M}$ computes $d = b^f$ and sends it to $\mathcal{H}$, who then verifies whether $e(Y,a) = e(g,b)$, $e(g,d) = e(F,b)$, and $e(X,ad) = e(g,c)$ hold.

Sign. Let m be the message to be signed (as the same as in the original DAA scheme, m is presented as $b\|m'$ where b $= 0$ means that the message m' is generated by the TPM and b $= 1$ means that m' was input to the TPM), $\mathtt{bsn}_V$ be a basename associated with $\mathcal{V}$, and $n_V \in \{0,1\}^{\ell_H}$ be a nonce provided by the verifier. $\mathcal{M}$ has a secret key f and a credential (a,b,c), whereas $\mathcal{H}$ only knows the credential (a,b,c). The signing algorithm takes the following steps:
1. Depending on whether $\mathtt{bsn}_V = \perp$ or not, $\mathcal{H}$ computes B as follows

$$\mathsf{B} \overset{R}{\leftarrow} \mathsf{G} \qquad \text{or} \qquad \mathsf{B} := H_{\mathsf{G}}(1\|\mathtt{bsn}_V),$$

where $\mathsf{B} \overset{R}{\leftarrow} \mathsf{G}$ means that B is chosen from G uniformly at random. $\mathcal{H}$ sends B to $\mathcal{M}$.
2. $\mathcal{M}$ verifies that $\mathsf{B} \in \mathsf{G}$ then computes $\mathsf{K} := \mathsf{B}^f$, and sends K to $\mathcal{H}$.
3. $\mathcal{H}$ chooses two integers $r, r' \leftarrow \mathbb{Z}_q$ uniformly at random and computes

$$\begin{array}{lll} a' := a^{r'}, & b' := b^{r'}, & c' := c^{r'r^{-1}}, \\ \mathsf{v}_x := e(X, a'), & \mathsf{v}_{xy} := e(X, b'), & \mathsf{v}_s := e(g, c'). \end{array}$$

4. $\mathcal{H}$ sends v_{xy} back to $\mathcal{M}$ who later verifies that $\mathsf{v}_{xy} \in \mathsf{G}$.
5. $\mathcal{M}$ and $\mathcal{H}$ jointly compute a "signature of knowledge" as follows

$$SPK\{(r,f) : \mathsf{v}_s^r = \mathsf{v}_x \mathsf{v}_{xy}^f \ \wedge\ \mathsf{K} = \mathsf{B}^f\}(n_V, n_T, m).$$

(a) $\mathcal{H}$ chooses a random integer $r_r \in \mathbb{Z}_q$ and computes $\tilde{\mathsf{T}}_{1t} := \mathsf{v}_s^{r_r}$.
(b) $\mathcal{H}$ computes $\mathfrak{c}_H := H(q\|g\|\mathsf{g}\|X\|Y\|a'\|b'\|c'\|\mathsf{v}_x\|\mathsf{v}_{xy}\|\mathsf{v}_s\|\mathsf{B}\|\mathsf{K}\|n_V)$ and sends $\mathfrak{c}_H$ and $\tilde{\mathsf{T}}_{1t}$ to $\mathcal{M}$.

(c) $\mathcal{M}$ chooses a random integer $r_f \leftarrow \mathbb{Z}_q$ and a nonce $n_T \in \{0,1\}^{\ell_\phi}$ and computes

$$\tilde{\mathsf{T}}_1 := \tilde{\mathsf{T}}_{1t}\mathsf{v}_{xy}^{-r_f}, \qquad \tilde{\mathsf{T}}_2 := \mathsf{B}^{r_f},$$
$$\mathfrak{c} := H(\mathfrak{c}_H\|\tilde{\mathsf{T}}_1\|\tilde{\mathsf{T}}_2\|n_T\|m), \qquad s_f := r_f + \mathfrak{c}\cdot f \bmod q.$$

(d) $\mathcal{M}$ sends $\mathfrak{c}$, s_f and n_T to $\mathcal{H}$.

(e) $\mathcal{H}$ computes $s_r := r_r + \mathfrak{c}\cdot r \bmod q$.

6. $\mathcal{H}$ outputs the signature $\sigma = (\mathsf{B}, \mathsf{K}, a', b', c', \mathfrak{c}, s_r, s_f)$ along with n_T.

Verify. The input to this program is the group public key $(q, g, G, \mathsf{g}, \mathsf{G}, e, X, Y)$, a message m, two nonces n_V and n_T, the basename $\mathtt{bsn}_V$, a candidate signature $\sigma = (\mathsf{B}, \mathsf{K}, a', b', c', \mathfrak{c}, s_r, s_f)$ on (m, n_V, n_T), and a list of rogue secrete keys, the verifier $\mathcal{V}$ does the following:

1. If $\mathtt{bsn}_V \neq \perp$, $\mathcal{V}$ verifies that $\mathsf{B} \overset{?}{=} H_{\mathsf{G}}(1\|\mathtt{bsn}_V)$, otherwise, $\mathcal{V}$ verifies that $\mathsf{B} \overset{?}{\in} \mathsf{G}$.
2. For each f_i in the rogue key list, $\mathcal{V}$ checks that $\mathsf{K} \overset{?}{\neq} \mathsf{B}^{f_i}$. If K matches with any f_i in the rogue list, $\mathcal{V}$ outputs `reject` and aborts.
3. $\mathcal{V}$ verifies that $e(a', Y) \overset{?}{=} e(g, b')$ and $\mathsf{K} \overset{?}{\in} \mathsf{G}$.
4. $\mathcal{V}$ computes

$$\hat{\mathsf{v}}_x := e(X, a'), \qquad \hat{\mathsf{v}}_{xy} := e(X, b'), \qquad \hat{\mathsf{v}}_s := e(g, c'),$$
$$\hat{\mathsf{T}}_1 := \hat{\mathsf{v}}_s^{s_r}\hat{\mathsf{v}}_{xy}^{-s_f}\hat{\mathsf{v}}_x^{-\mathfrak{c}}, \qquad \hat{\mathsf{T}}_2 := \mathsf{B}^{s_f}\mathsf{K}^{-\mathfrak{c}}.$$

5. $\mathcal{V}$ verifies that $\mathfrak{c} \overset{?}{=} H(H(q\|g\|\mathsf{g}\|X\|Y\|a'\|b'\|c'\|\hat{\mathsf{v}}_x\|\hat{\mathsf{v}}_{xy}\|\hat{\mathsf{v}}_s\|\mathsf{B}\|\mathsf{K}\|n_V)\|\hat{\mathsf{T}}_1\|\hat{\mathsf{T}}_2\|n_T\|m)$.
6. If all the above verifications succeed, $\mathcal{V}$ outputs `accept`, otherwise $\mathcal{V}$ outputs `reject`.

4.2 Security Proof

The following theorem establishes the security of our scheme. Due to the page limitation, we omit the detailed proof, which will be given in the full version of this paper.

Theorem 2. *The protocols described above securely implement a DAA system under the LRSW assumption and the DBDH assumption in the random oracle model.*

We now give a very brief sketch of the proof. Following the security model and the proof of the original DAA scheme in [6], the proof of Theorem 2 consists of the description of a simulator and arguments that the environment cannot distinguish whether it is run in the real system interacting with an adversary and the real parties, or in the ideal system interacting with a simulator and the ideal parties. The simulator has black box access to the adversary. In the DAA-signing protocol, the simulator forges signatures by using the zero-knowledge simulator and the power over the random oracle. In cases where the adversary manages to do any of the followings: to sign a signature on behalf of a honest signer, or to tag an honest TPM as a rogue, or to create a signature which should link to another signature since they are associated with the same basename and signing key but the linkability does not exist, the simulator fails. We can show that these cases cannot occur; otherwise the adversary can forge a CL-LRSW signature, which is contradiction to the LRSW assumption based on Theorem 1. We then show that if the simulator does not fail, under the DBDH assumption, the environment will not be able to tell whether or not it is run in the real system interacting with the adversary or the ideal system interacting with the simulator.

5 Consideration on Implementing the DAA Scheme

In this section, we show how to implement the proposed DAA scheme. There exist a number of bilinear maps, which match with the security requirements of our scheme. In this paper, however, we recommend a well-known construction of an admissible bilinear map from the Tate pairing, since it is the most efficient one based on the authors' knowledge. We first recall the pairing construction, describe how to choose the corresponding security parameters, and then analyze the efficiency of our DAA scheme when uses this bilinear map. Finally we discuss various constructions of the point B in the DAA scheme.

5.1 An Admissible Bilinear Map from the Tate Pairing

We now recall the description of an admissible bilinear map [3,17,20] from the Tate pairing. Let p be a prime satisfying $p = 3 \bmod 4$ and let q be some prime factor of $p+1$. Let E be the elliptic curve defined by the equation $y^2 = x^3 - 3x$ over $\mathbb{F}_p$. $E(\mathbb{F}_p)$ is supersingular and contains $p+1$ points and $E(\mathbb{F}_{p^2})$ contains $(p+1)^2$ points. Let $g \in E(\mathbb{F}_p)$ to a point of order q and let G be the subgroup of points generated by g. Let G be the subgroup of $\mathbb{F}_{p^2}^*$ of order q.

Let $\phi(x, y) = (-x, iy)$ be an automorphism of the group of points on the curve $E(\mathbb{F}_p)$, where $i^2 = 1$. Then ϕ maps points of $E(\mathbb{F}_p)$ to points of $E(\mathbb{F}_{p^2})\backslash E(\mathbb{F}_p)$. Let f be the Tate pairing, then we can define $e : G \times G \rightarrow \mathsf{G}$ as $e(P, Q) = f(P, \phi(Q))$, where e is an admissible bilinear map.

5.2 Choices of Security Parameters

To choose right sizes of p and q, we must ensure that the discrete log problem in G is hard enough. As the discrete log problem in G is efficiently reducible to discrete log in G [22], we need to choose p large enough such that the discrete log in $\mathbb{F}_{p^2}^*$ is hard to compute. For our DAA scheme, we choose p as a 512-bit prime and q as a 160-bit prime as follows [20]: (I) Choose a 160-bit prime q. Careful choices of q would speed up the Tate pairing operation substantially, e.g., choose q with low Hamming weight. (II) Randomly generate a 352-bit r where $4 \mid r$. (III) Compute $p := rq - 1$ and check whether p is a prime. Note that $p = 3 \bmod 4$. If p is not prime, repeat the previous step.

5.3 Efficiency of Our DAA Scheme

Let $\ell_H = 256$, $\ell_p = 512$, $\ell_q = 160$ and $\ell_\phi = 80$. Given a concrete DAA scheme described above using the Tate pairing, we summarize the efficiency of our scheme as follows:

– Since p is 512-bit, we can use 513 bits to present a point in $E(\mathbb{F}_p)$. The size of the DAA public key is 2211 bits or 277 bytes. Each membership private key is 1699 bits or 213 bytes. Each signature is 4163 bits or 521 bytes.
– To compute a signature, the TPM needs to perform 5 exponentiations and the host needs to perform 5 exponentiations and 3 pairings. To verifier a signature, the verifies needs to 7 exponentiations and 5 pairings.

Note that the host can pre-compute $A = e(X, a)$, $B = e(X, b)$ and $C = e(g, c)$, and store the triple (A, B, C). In each signing process, the host can compute $\mathsf{v}_x := A^{r'}$, $\mathsf{v}_{xy} := B^{r'}$, and $\mathsf{v}_s := C^{r'r^{-1}}$ to avoid expensive pairing operations.

5.4 Constructing the Point B

How to construct $\mathsf{B} \overset{R}{\leftarrow} \mathsf{G}$ and $\mathsf{B} = H_{\mathsf{G}}(m)$ given the input m is a sensitive part of the scheme implementation. To implement $\mathsf{B} \overset{R}{\leftarrow} \mathsf{G}$, we suggest creating a random generator of G, which is a q-th root of unity in $\mathbb{F}_{p^2}$. This can be done by choosing a random $x \in \mathbb{F}_{p^2}$, then computing $\mathsf{B} := x^{(p^2-1)/q}$. We suggest the following various ways to construct $\mathsf{B} = H_{\mathsf{G}}(m)$.

1. Use a collision resistant hash function $H(m)$, which maps the value m to an element in F_{p^2}, then compute $\mathsf{B} = H(m)^{(p^2-1)/q}$.
2. Use a MapToPoint function H, as described in [5], to map the value m to an element of G, that can guarantee nobody knows the discrete log relation between g and $H(m)$, and then compute $\mathsf{B} := e(H(m), H(m))$.
3. As in the original DAA scheme [6], make a cyclic group different from either G or G, in which the discrete logarithm problem is hard, and then compute B in this group instead of G.

6 Conclusion

In this paper, we proposed an efficient DAA scheme from bilinear pairings, which has a much shorter key length and signature size compared with the original integer factorization based DAA scheme [6] with similar level of security. The security of our new DAA scheme is based on the LRSW assumption and the DBDH assumption in the random oracle model.

Acknowledgements

We would like to thank Frederik Armknecht, Hans Löhr, Paul Morrissey, Kenny Paterson, Mark Ryan, Ahmad-Reza Sadeghi, Nigel Smart, Ben Smyth and Robert Squibbs for helpful discussions and comments on this work. We also thank the anonymous referees for their comments.

References

1. Backes, M., Maffei, M., Unruh, D.: Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestaion protocol. Cryptology ePrint Archive, Report 2007/289 (2007), `http://eprint.iacr.org/`
2. Balfe, S., Lakhani, A.D., Paterson, K.G.: Securing peer-to-peer networks using trusted computing. In: Mitchell, C. (ed.) Trusted Computing, ch.10, pp. 271–298. IEE, London (2005)
3. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–368. Springer, Heidelberg (2002)
4. Boneh, D., Boyen, X.: Efficient selective-ID secure identity based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
5. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
6. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 132–145. ACM Press, New York (2004)
7. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation in context. In: Mitchell, C. (ed.) Trusted Computing, ch.5, pp. 143–174. IEE, London (2005)
8. Brickell, E., Chaum, D., Damgård, I., van de Graaf, J.: Gradual and verifiable release of a secret. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 156–166. Springer, Heidelberg (1988)
9. Camenisch, J., Groth, J.: Group signatures: Better efficiency and new theoretical aspects. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 120–133. Springer, Heidelberg (2005)
10. Camenisch, J., Lysyanskaya, A.: A signature scheme with efficient protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003)
11. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)
12. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: CAIP 1997. LNCS, vol. 1296, pp. 410–424. Springer, Heidelberg (1997)
13. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13(1), 143–202 (2000)
14. Chaum, D.: Zero-knowledge undeniable signatures. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 458–464. Springer, Heidelberg (1991)
15. Chaum, D., Evertse, J.-H., van de Graaf, J.: An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In: Price, W.L., Chaum, D. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 127–141. Springer, Heidelberg (1988)
16. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
17. Galbraith, S.D., Harrison, K., Soldera, D.: Implementing the Tate pairing. In: Proceedings of the 5th International Symposium on Algorithmic Number Theory, London, UK, pp. 324–337. Springer, London (2002)

18. Ge, H., Tate, S.R.: A direct anonymous attestation scheme for embedded devices. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 16–30. Springer, Heidelberg (2007)
19. Leung, A., Mitchell, C.J.: Ninja: Non identity based, privacy preserving authentication for ubiquitous environments. In: Krumm, J., Abowd, G.D., Seneviratne, A., Strang, T. (eds.) UbiComp 2007. LNCS, vol. 4717, pp. 73–90. Springer, Heidelberg (2007)
20. Lynn, B.: On the implementation of pairing-based cryptosystems. PhD thesis, Stanford University, Stanford, California (2007)
21. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In: Heys, H.M., Adams, C.M. (eds.) SAC 1999. LNCS, vol. 1758, pp. 184–199. Springer, Heidelberg (2000)
22. Menezes, A., Vanstone, S., Okamoto, T.: Reducing elliptic curve logarithms to logarithms in a finite field. In: Proceedings of the 23rd annual ACM Symposium on Theory of Computing (STOC), pp. 80–89. ACM Press, New York (1991)
23. Pashalidis, A., Mitchell, C.J.: Single sign-on using TCG-conformant platforms. In: Mitchell, C. (ed.) Trusted Computing, ch. 6, pp. 175–193. IEE, London (2005)
24. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 184–200. IEEE Computer Society Press, Los Alamitos (2001)
25. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)
26. Rudolph, C.: Covert identity information in direct anonymous attestation (DAA). In: Proceedings of the 22nd IFIP TC-11 International Information Security Conference (SEC 2007) (2007)
27. Schnorr, C.P.: Efficient identification and signatures for smart cards. Journal of Cryptology 4(3), 161–174 (1991)
28. Smyth, B., Chen, L., Ryan, M.: Direct anonymous attestation (DAA): ensuring privacy with corrupt administrators. In: Stajano, F. (ed.) ESAS 2007. LNCS, vol. 4572, pp. 218–231. Springer, Heidelberg (2007)
29. Trusted Computing Group. TCG TPM specification 1.2 (2003), `http://www.trustedcomputinggroup.org`
30. Trusted Computing Group website, `http://www.trustedcomputinggroup.org`

On a Possible Privacy Flaw in Direct Anonymous Attestation (DAA)

Adrian Leung[1,⋆], Liqun Chen[2], and Chris J. Mitchell[1]

[1] Information Security Group
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK
{A.Leung,C.Mitchell}@rhul.ac.uk
[2] Hewlett-Packard Laboratories
Filton Road, Stoke Gifford, Bristol, BS34 8QZ, UK
Liqun.Chen@hp.com

Abstract. A possible privacy flaw in the TCG implementation of the Direct Anonymous Attestation (DAA) protocol has recently been discovered by Rudolph. This flaw allows a DAA Issuer to covertly include identifying information within DAA Certificates, enabling a colluding DAA Issuer and one or more verifiers to link and uniquely identify users, compromising user privacy and thereby invalidating the key feature provided by DAA. In this paper we argue that, in typical usage scenarios, the weakness identified by Rudolph is not likely to lead to a feasible attack; specifically we argue that the attack is only likely to be feasible if honest DAA signers and verifiers never check the behaviour of issuers. We also suggest possible ways of avoiding the threat posed by Rudolph's observation.

Keywords: Direct Anonymous Attestation, DAA, Privacy, Trusted Computing.

1 Introduction

Direct Anonymous Attestation (DAA), proposed by Brickell, Camenisch and Chen, [1,2], is a special type of group signature scheme that can be used to anonymously authenticate a principal, also referred to as a prover, to a remote verifier. DAA has been adopted by the Trusted Computing Group[1] in version 1.2 of the Trusted Computing Trusted Platform Module (TPM) Specifications [3]. The key features provided by DAA are the capability for a prover (a group member) to anonymously convince a remote verifier that:

- It is in possession of a DAA Certificate obtained from a specific DAA Issuer, without having to reveal the DAA Certificate to a verifier (as would be necessary for a signature-based proof of knowledge);

⋆ This author is supported by the British Chevening/Royal Holloway Scholarship, and in part by the European Commission under contract IST-2002-507932 (ECRYPT).
[1] http://www.trustedcomputinggroup.org/

- A DAA Signature computed by a prover on a message m, has been generated using a valid DAA Certificate issued by a specific DAA Issuer; colluding verifiers are unable to link two different DAA Signatures created by the same prover, and, in particular, verifiers are not given the DAA Certificate.

Moreover, the DAA scheme provides a flexible way of achieving a number of different levels of 'linkability'. Under an agreement between the prover and verifier, DAA Signatures can be either 'random-base' or 'name-base'. Two random-base signatures signed by the same prover (TPM) for the same verifier cannot be linked. However, name-base signatures are associated with the verifier's name; as a result, two name-base signatures signed by the same prover (TPM) for the same verifier can be linked.

These features help to protect the privacy of a prover. Another important feature of DAA (distinguishing it from other types of group or ring signature schemes) is that the powers of the supporting Trusted Third Party (i.e. the DAA Issuer in its role as the group manager) are minimised, as it cannot link the actions of users (i.e. provers) and hence compromise the user's privacy even when it colludes with a verifier. This unlinkability property is the key feature of DAA.

However, an attack was recently discovered by Rudolph [4] which potentially compromises the unlinkability property of DAA. The attack could allow a DAA Issuer to embed covert identifying information into DAA Certificates (of provers) and to subsequently link the transactions of the users/provers to whom the DAA Certificates belong [4]. As a result, DAA Signatures originating from the same users/provers become linkable, and users can thereby be uniquely identified. In this paper, we argue that Rudolph's attack may be infeasible in practice, and we discuss why an attempt to launch such an attack could easily be discovered in an environment where there is at least one honest verifier. We also propose approaches which could prevent the attack from taking place.

The remainder of this paper is organised as follows. In Section 2 we briefly describe the workings of DAA as well as outlining the privacy attack. In Section 3 we explain why the attack is likely to be unrealistic in many practical scenarios, and, in Section 4, we discuss possible modifications to the use of DAA in the TCG specifications that can prevent the attack. Finally, conclusions are drawn in Section 5.

2 The Privacy Attack on DAA

In this section, we only describe those aspects of the DAA protocol necessary to understand the Rudolph attack. For full details of DAA, including a proof of its security, see [1] and chapter 5 of [2]. A brief description of the attack then follows.

2.1 DAA Overview

We first introduce the entities involved in the DAA protocol and the roles they play.

- The *Certification Authority (CA)* acts as a Trusted Third Party (TTP). Its role is to certify the authenticity of the DAA Issuer's longer-term public key, CK_I. It does not directly participate in the DAA protocol.
- The *DAA Issuer* (or just the Issuer) is a third party that issues DAA Certificates (i.e. anonymous credentials) to provers. It must be trusted by all other participants in the DAA protocol to perform its role in a trustworthy manner.
- The *Prover* (or the User) generates DAA Signatures that are verified by verifiers. In the context of Trusted Computing, the prover is the TPM.
- The *Verifier* verifies DAA Signatures computed by provers.

Note that, apart from the CA, all the entities above take direct part in the DAA protocol. Also, in normal circumstances, the numbers of CAs and DAA Issuers are likely to be very small by comparison with the number of provers.

The DAA Scheme consists of two sub-protocols (or phases), namely the *DAA Join Protocol* and the *DAA Sign Protocol*. In the Join Protocol (shown in Figure 1), a prover interacts with a DAA Issuer I in order to obtain an anonymous credential on a secret value f (also referred to as the DAA Secret), known only to the prover. This anonymous credential, also known as the DAA Certificate, is jointly computed by the DAA Issuer and the prover as a function of a blinded value of f, the shorter-term public key of I, PK_I, and other parameters. The DAA Certificate is later used by a prover during the DAA Sign Phase in the DAA Signature computation. As part of the DAA Join protocol, the prover is authenticated to the DAA Issuer using its unique and long lived Endorsement Key (EK). Note that the public part of the EK can be used to uniquely identify a prover. The Issuer authenticates itself to the prover using its shorter-term public key PK_I, which the prover verifies using a certificate signed by the Issuer with its longer-term public key CK_I, which is in turn certified by the CA.

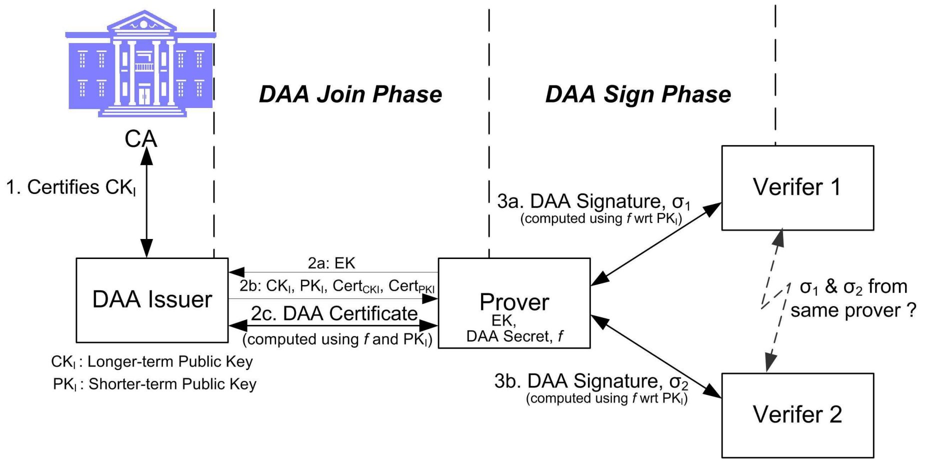

DAA Join Phase
DAA Sign Phase
CA
1. Certifies CK_I
DAA Issuer
CK_I : Longer-term Public Key
PK_I : Shorter-term Public Key
2a: EK
2b: CK_I, PK_I, Cert_CKI, Cert_PKI
2c. DAA Certificate
(computed using f and PK_I)
Prover
EK,
DAA Secret, f
3a. DAA Signature, σ1
(computed using f wrt PK_I)
3b. DAA Signature, σ2
(computed using f wrt PK_I)
Verifer 1
Verifer 2
σ1 & σ2 from same prover ?

Fig. 1. The DAA Scheme

In the DAA Sign Phase, the prover DAA-signs a message m, using its DAA Secret f, the DAA Certificate, and other public parameters. The output of the DAA-signing process is known as the DAA Signature σ. This DAA Signature enables the prover to prove to a verifier (using a signature-based proof of knowledge) that (i) it is in possession of a DAA Certificate, and (ii) the DAA Signature on message m was computed using its DAA Secret f, the DAA Certificate in (i), and other public parameters. Verifying a DAA Signature requires knowledge of the DAA Issuer's public key PK_I (i.e. the public key of the DAA Issuer that was used to create the DAA Certificate). Hence prior to running the DAA Sign protocol, a verifier must have obtained an authentic copy of PK_I.

The DAA Sign Protocol has the property that colluding verifiers are unable to link different DAA Signatures originating from the same prover (as shown in Figure 1). This property applies even if a DAA Issuer colludes with a verifier, i.e. they are still unable to link and uniquely identify a particular prover.

2.2 The Rudolph Attack

The privacy breaching attack on DAA proposed by Rudolph [4] operates under an assumption about the use of DAA, which we first describe. Specifically, Rudolph assumes that the DAA Issuer's longer-term public key CK_I, as well as the (shorter-term) public key PK_I (along with its certificate chain) used to compute the DAA Certificate, are communicated to the verifier via the prover during the DAA Sign Phase (as shown in Figure 2). Whether or not this is a realistic assumption is not clear; other possibilities include use of a publicly accessible certificate repository. In any event, as we describe below, the verifier will need to know which of the DAA Issuer's shorter-term keys has been used to create the DAA Certificate on which the DAA Signature is based.

The attack works as follows. As shown in Figure 1, during the DAA Join Phase the DAA Certificate is computed using the DAA Issuer's public key PK_I and other parameters. The key PK_I is a shorter-term public key which is certified

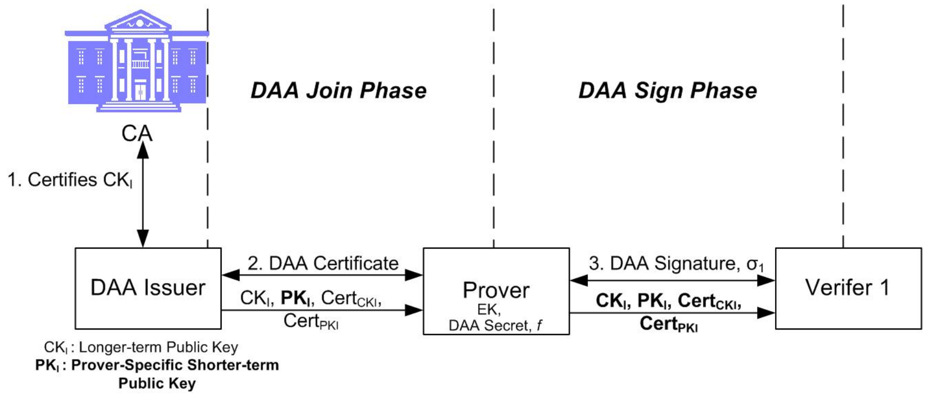

CA
1. Certifies CK_I
DAA Join Phase
DAA Sign Phase
DAA Issuer
2. DAA Certificate
CK_I, PK_I, Cert_CKI, Cert_PKI
Prover
EK, DAA Secret, f
3. DAA Signature, σ1
CK_I, PK_I, Cert_CKI, Cert_PKI
Verifer 1
CK_I : Longer-term Public Key
PK_I : Prover-Specific Shorter-term Public Key

Fig. 2. The Rudolph Attack

using the longer-term public key CK_I, which in turn is certified by a trusted CA (as shown in Figure 1). This key hierarchy is an intentional design feature of DAA, chosen to make the TPM's key life cycle and the Issuer's key life cycle independent. This is because the TPM computes its DAA private key as a digest of a secret seed and the Issuer's longer-term public key. This ensures that the TPM uses the Issuer key that matches its DAA private key. If the Issuer had only a single key, then, when the Issuer changed its key, every TPM would also have to update its key. To avoid this problem, the Issuer is given the two types of key described above.

Unfortunately, this flexibility can potentially be exploited by a curious DAA Issuer to compromise the privacy properties of DAA. As observed by Rudolph, a DAA Issuer could embed covert identifying information into a public key without the knowledge of an honest prover. The Issuer simply uses a different public key PK_I to generate DAA Certificates for each prover with which it interacts. As a result, the Issuer will be able to compile a table mapping between a prover's public EK (its unique key) and the public key used to generate the DAA Certificate for this prover.

Suppose a verifer has obtained the Issuer's public key PK_I from the prover. Whenever a prover executes the DAA Sign protocol with a colluding verifier (i.e. one that colludes with the Issuer to identify a prover), and assuming that the public key PK_I used to generate its DAA Certificate is communicated to a verifier via the prover, the DAA Issuer and the colluding verifier can easily link the transactions of a prover and uniquely identify it. A verifier needs only inform the DAA Issuer of the value of PK_I. The DAA Issuer can then consult the EK-PK_I mapping it has compiled, and determine the prover's EK. Similarly, with the aid of the Issuer, colluding verifiers are able to uniquely identify a particular prover.

3 How Realistic Is the Rudolph Attack?

We now consider how the Rudolph attack might work in practice, and in particular we examine two possible attack scenarios.

3.1 Scenario 1: Linking Large Numbers of Users

In this scenario, the aim of the DAA Issuer is to identify large numbers of provers. We believe that this attack scenario is infeasible in practice. We demonstrate (i) why this attack scenario is unrealistic (even if all the verifiers collude) because of the communications and computation burden involved in performing such an attack; and (ii) how the attack can easily be detected if there is at least one honest verifier:

(i) The Rudolph attack only works if the DAA Issuer and all the verifiers collude. Suppose we have a scenario where there is one DAA Issuer, n provers (all joining the network or system at different times), and k verifiers (which all collude with the Issuer in an attempt to link or uniquely identify the provers). For the attack to work, the colluding verifiers have to shoulder an additional communication and computational burden (see Table 1 for a summary of the communication and computational overheads). First and foremost, to be able to link all the n provers, the DAA Issuer would need to use n different public keys PK_I, one for each prover. These n public keys would also need to be communicated to each of the colluding verifiers. If a trusted directory is used to hold copies of the public keys, then the Issuer would need to upload a total of n different public keys to this directory (if the provers join at different times then this may involve sending up to n separate upload messages). If the verifiers obtain the public keys from the Issuer directly, then the total communications overhead for an Issuer may be up to nk messages (as compared to k messages if the Issuer only uses one key).

A colluding verifier, regardless of the mechanism used to retrieve Issuer public keys, will need to obtain up to n Issuer public keys for the n provers. This means that the communications overhead for a verifier may be up to n messages. Furthermore, whenever there is a new prover, the verifiers might need to obtain the new public key for this prover.

In terms of computational overheads, we now point out why it is by no means trivial to launch the Rudolph attack. Firstly, the generation of the DAA Issuer public key PK_I involves performing a non-interactive zero knowledge proof (using the Fiat-Shamir heuristic) [1]. This potentially involves the DAA Issuer performing at least 160 modular exponentiations (which could go up to 6×160, one for each of the public key components g, h, S, Z, R_0 and R_1). This contradicts Rudolph's claim that the process of generating a large number of public keys can be performed efficiently [4].

Secondly, the work to be performed by the verifier in trying to identify the prover may become infeasibly large, depending on how Issuer public keys are distributed. If the prover sends the Issuer-signed certificate for the Issuer public key PK_I to the verifier as part of the DAA Sign protocol, as assumed by Rudolph, then there is no problem. However, if the key is obtained from a

Table 1. Communications and computation costs for honest and colluding entities

Type of Costs	Costs Incurred By			
	Honest Issuer	Honest Verifier	Colluding Issuer	Colluding Verifier
Communication (no. of messages)	1	1	nk	n
Computation (no. of DAA Sign Verifications)	-	1	-	n

directory, then significant computational problems arise. This is because the verifier will have no way of knowing which of the n Issuer public keys have been used to create the DAA certificate, and hence which of them should be used to attempt to verify the DAA Signature. The only solution would be for the verifier to attempt to verify the signature using every possible key, which would involve up to n verifications. Given the ubiquity of trusted computing hardware, a typical value of n might be 10^5 or 10^6, which would make such a process computationally infeasible.

(ii) The attack will easily be discovered if there is at least one honest verifier or if Issuer public keys are stored in public directories, as we now describe.
   - Consider first an environment in which there is at least one honest verifier. When the honest verifier attempts to verify a DAA Signature, it first needs to retrieve the Issuer's public key from the Issuer (or from a trusted directory). If the Issuer (or the trusted directory) submits a large number of public keys to the verifier, then suspicions about its trustworthiness will immediately be aroused. Even if the honest verifier is given the Issuer public key by the prover rather than retrieving it from a directory, then it could still detect misbehaviour if it keeps a log of all the Issuer public keys that it has been passed. If one particular Issuer is using large numbers of different public keys, then this will quickly become obvious to such a verifier.
   - Suppose an Issuer uploads multiple public keys to a directory or other repository. This will immediately be obvious both to the operator of the directory (which may report suspicious behaviour) as well as to any user of the directory.

3.2 Scenario 2: Linking a Small Set of Users

On the other hand, if the aim of the DAA Issuer is to link all transactions involving a single user (e.g. a high-profile user or one that makes high value transactions), or a very small set of users, then the attack is much more likely to succeed in a way that is hard to detect. For example, if a DAA Issuer only wants to distinguish transactions involving one user, then the Issuer only needs to have two public keys PK_I. Hence, the communication and computation problems discussed in the previous section would not be an issue in this attack scenario, nor would there be a large number of Issuer public keys in circulation to arouse suspicions.

Nevertheless, if a verifier deals with many provers who are clients of the same Issuer, the suspicions of an honest verifier might be aroused if one Issuer public key, or some small set of such keys, is used much less than others. In particular, if the DAA Signatures are of the name-base type, allowing a verifier to link DAA Signatures signed by the same prover (TPM) for the same verifier, then the verifier will be able to observe significant differences in the numbers of clients for an Issuer's public keys.

4 Preventing the Rudolph Attack

Despite the issues raised in the previous section, the Rudolph attack will work if a verifier obtains the Issuer's public key from a prover (as described in [4]), and if no honest verifier keeps track of the Issuer public keys it has received or if the goal of the attackers is only to track a few users. Even worse, a prover would be completely oblivious to such an attack, as there is no way for a prover to tell if a DAA Issuer is embedding covert identity information into the public key that is used to generate its DAA Certificate (e.g. by using a different public key for each prover).

We now examine a number of possible ways of preventing the Rudolph attack. We also discuss the limitations of these approaches.

4.1 Modifying the TCG Specifications

We first observe that addressing the root cause of the problem would involve changing the TCG specifications to prevent a DAA Issuer from self-certifying an arbitrary number of public keys. This could be achieved by requiring the Issuer to use a private key for which the public key has been directly certified by a third party CA (in the notation used above, this would mean that the issuer key CK_I would be used to generate DAA certificates).

There are two problems with such a approach. Firstly, the CA would then need to be trusted not to generate large numbers of certificates for an Issuer. Whilst this could be supported by requiring any CA that generates Issuer certificates to adhere to an appropriate Certification Practice Statement, it still means that a significant amount of trust is placed in a single third party, a situation which the design of DAA seeks to avoid.

Secondly, as explained above, this means that the key life cycle of the TPM and the Issuer become linked. Addressing this issue would require further changes in the operation of the TPM.

An alternative approach would retain the two levels of Issuer public keys, but would require both types to be certified by a CA. As in the existing scheme, the first level key would be used to compute the DAA secret, and the second level key would be used to create the DAA Certificate. Certificates for second level keys could mention the first level key to link the two together. This could address the Rudolph attack without causing a key life cycle issue.

4.2 Using a Trusted Auditor

We next explore another possible approach which does not involve modifying the TCG specifications too much (or at all). We propose that a prover obtains DAA Certificates only from DAA Issuers that use the same public key for a very large set of users. Thus the challenge is to enable a prover to determine the key usage behaviour of a DAA Issuer.

If a prover is able to obtain assurance that a specific Issuer's public key has being used more than a certain number of times (i.e. to generate a certain number of DAA Certificates), then it is immediately able to derive confidence that it will not be uniquely identified, and at the same time gain information about the level of anonymity that it is being afforded. For example, if a prover knows that the public key used to generate its DAA Certificate has been used to generate a thousand other DAA Certificates, then it knows that it cannot be distinguished from a thousand other entities. On the other hand, if it knows that a particular public key has only been used to generate three other DAA Certificates, then the level of anonymity afforded to it is potentially very limited.

We now suggest two possible approaches designed to give a prover this type of assurance. We also discuss the possible limitations of the suggested approaches.

A modification to the use of DAA. This approach requires the introduction of a new type of trusted third party, which we refer to as a *Trusted Auditor* (TA). We make use of the TA (which is not necessarily unique) to give provers assertions about the "trustworthiness" of individual DAA Issuers. Since the CA needs to be trusted by the protocol participants, and since it is already employed to certify the longer-term public key CK_I of the Issuer, a CA could act as a TA, although this does need to be the case.

We propose that the following additional steps be performed by a prover during the DAA Join protocol. During the Join phase, and after a DAA Certificate has been successfully created, the prover establishes a secure channel with the TA. This can be achieved by the prover first establishing a unilaterally authenticated secure channel using a public key certificate for the TA, e.g. using SSL/TLS. The prover can then send its EK credential to the TA via this channel, and use this credential to authenticate itself, e.g. by decrypting data sent to it encrypted using the EK. Finally, the prover uses this channel to send a statement that a specific DAA Issuer has used a particular public key PK_I to derive its DAA Certificate, i.e.

$$Prover \rightarrow TA : ID_{Issuer}, CK_I, PK_I$$

Using these messages, the TA can compile a list of the form given in Table 2. A copy of this list signed by the TA (to prove authenticity) can then be communicated to a prover prior to the Join Phase. Since it is desirable not to publicise the public EK values of individual trusted platforms, this information can be removed from the list before it is distributed. The list of public EKs can be replaced by the total number of different EKs for which credentials have been generated using a particular Issuer public key.

This approach suffers from one problem. The public part of the Endorsement Key (EK) of every prover is revealed to the TA. Since the EK pair for a trusted platform is fixed, the public EK functions as a fixed identifier for a platform, and hence revealing it is not desirable. Indeed, DAA was introduced to avoid the need to reveal the link between the public EK and other prover public keys to a Privacy CA. Nevertheless, this scheme does not pose the same threat to user privacy as the use of a Privacy CA, since the TA does not learn the link between the EK and any other prover keys.

Table 2. Mapping of EKs to a PK_I for individual Issuers

No.	Issuer Name	Longer-term CK_I	Shorter-term PK_I	Users
1.	Alice	CK_I^A	PK_I^{A1}	EK_1 ⋮ EK_{2000}
			PK_I^{A2}	EK_1 ⋮ EK_{10}
2.	Bob	CK_I^B	PK_I^{B1}	EK_1 ⋮ EK_{200}

An alternative approach. A possible alternative to the above approach avoids revealing the EK to a third party, although it still relies on a TA to provide assertions regarding the trustworthiness of the DAA Issuer (i.e. reporting on the number of DAA Certificates generated for a particular public key for a specific Issuer). A prover and DAA Issuer run the DAA Join protocol as normal, thereby obtaining a DAA Certificate (for the prover's unique DAA Secret f). The prover then conducts an instance of the DAA Sign protocol with the TA, which acts as the verifier. The prover DAA-signs the following information sent to the TA (acting as verifier), so that the TA can compile a table similar to that shown in Figure 2.

$$Prover \rightarrow TA : ID_{Issuer}, CK_I, PK_I$$

One possible problem with this approach is that, because no use is made of the EK, a malicious Issuer could fabricate DAA-signed messages of the above form, and send them to the TA. The DAA signature signed for the TA could be of the name-base type, which will guarantee that each DAA secret can only provide one piece of evidence. However, the messages could be based on any number of 'fake' DAA Certificates, that are valid in that they have been created by the DAA Issuer, but have never been sent to a genuine prover. Such messages will be indistinguishable from messages sent from genuine provers, and hence the number of uses of a key can be artificially inflated.

Nevertheless, a table created in this way will still reveal if an Issuer has created more public keys than would be expected in 'normal' behaviour; this may be sufficient to deter an Issuer from using the Rudolph attack on a large scale.

4.3 A User-Centric Approach

To determine the trustworthiness of an Issuer, two or more users could collaborate to compare the PK_I values that they have obtained from a particular Issuer. If all of them have the same key PK_I, then there is good chance that the

Issuer is using the same PK_I for a large set of users. However, if the users find that two or more different keys PK_I have been used, then the trustworthiness of the issuer is immediately called into question. This approach is suited to a distributed or peer-to-peer environment, and does not require the involvement of a trusted third party. Clearly, the effectiveness of the technique will depend on the number of cooperating users.

5 Concluding Remarks

A privacy flaw in DAA was recently pointed out by Rudolph [4]. In this paper we have analysed the feasibility of attacks exploiting this property. We then examined possible approaches which could be used to prevent (or reveal) the attack as well as the limitations of these approaches. These approaches could make a successful attack very difficult to perform; however, all of the suggestions have certain drawbacks. It remains an open problem to find a 'perfect' solution to the Rudolph attack. Indeed, the DAA scheme itself cannot stop an issuer from using a different key with each TPM, no matter whether the key is certified by the Issuer's longer-term key or by another CA. It is a very tough challenge for any application to completely avoid such a threat.

It should be noted that protocols and applications (such as those given in [5,6,7]) which employ DAA as a building block are not affected by this attack, as it is not an attack on the DAA protocol itself, but is rather a weakness introduced in the particular use of DAA. In fact, an implementation of an arbitrary group signature scheme might have a similar problem if the size of a group is very small, e.g. for groups containing just a single member. However, in other implementations of group signatures, a group manager might not have any motivation to break the anonymity of its members, because the manager has the ability to open the identity of a signer from its signature.

Acknowledgements

We thank Heiko Stamer and Stéphane Lo-Presti for their valuable comments, and in particular to Heiko Stamer for pointing out the true complexity of generating DAA Issuer key pairs. We also thank Rene Mayrhofer for bringing the Rudolph attack to our attention.

References

1. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, Washington DC, USA, October 25–29, 2004, pp. 132–145. ACM Press, New York (2004)
2. Mitchell, C.J. (ed.): Trusted Computing. IEE Press, London (2005)
3. Trusted Computing Group (TCG): TCG Specification Architecture Overview. Version 1.2, The Trusted Computing Group, Portland, Oregon, USA (2004)

4. Rudolph, C.: Covert identity information in direct anonymous attestation (DAA). In: Venter, H., Eloff, M., Labuschagne, L., Eloff, J., von Solms, R. (eds.) 22nd IFIP TC-11 International Information Security Conference (SEC 2007) on New Approaches for Security, Privacy and Trust in Complex Environments, Sandton, South Africa, May 14-16, 2007. IFIP International Federation for Information Processing, vol. 232, pp. 443–448. Springer, Boston (2007)
5. Balfe, S., Lakhani, A.D., Paterson, K.G.: Trusted computing: Providing security for peer-to-peer networks. In: Proceedings of the Fifth International Conference on Peer-to-Peer Computing (P2P 2005), Konstanz, Germany, August 31–September 2, 2005, pp. 117–124. IEEE Computer Society, Los Alamitos (2005)
6. Leung, A., Mitchell, C.J.: Ninja: Non identity based, privacy preserving authentication for ubiquitous environments. In: Krumm, J., Abowd, G.D., Seneviratne, A., Strang, T. (eds.) UbiComp 2007. LNCS, vol. 4717, pp. 73–90. Springer, Heidelberg (2007)
7. Leung, A., Poh, G.S.: An anonymous watermarking scheme for content distribution protection using trusted computing. In: Proceedings of the International Conference on Security and Cryptography (SECRYPT 2007), Barcelona, Spain, August 28–31, 2007, pp. 319–326. INSTICC Press (2007)

Author Index