DRM, Trusted Computing and Operating System Architecture

Jason F. Reid **William J. Caelli**

Information Security Research Centre
Queensland University of Technology
G P O Box 2434, Brisbane, Qld 4001, Australia

jf.reid@qut.edu.au, w.caelli@qut.edu.au

Abstract

Robust technological enforcement of DRM licenses assumes that the prevention of direct access to the raw bit representation of decrypted digital content and the license enforcement mechanisms themselves is possible. This is difficult to achieve on an open computing platform such as a PC. Recent trusted computing initiatives namely, the Trusted Computing Group (TCG) specification, and Microsoft's Next Generation Secure Computing Base (NGSCB) aim in part to address this problem. The protection architecture and access control model of mainstream operating systems makes them inappropriate as a platform for a DRM content rendering client because decrypted content cannot be protected against a privileged process. If a DRM client is to be deployed on an open computing platform, the operating system should implement the reference monitor concept, which underpins the mandatory access control model. The TCG model of trusted computing has important limitations when combined with an operating system enforcing discretionary access control. We argue that the TCG services of sealed storage and remote attestation which are important in DRM applications, cannot operate in a secure and efficient manner on such an operating system.

1 Introduction

Advances in digital compression technology coupled with the reduced cost and increased capacity of storage media and network bandwidth have combined to make the distribution of digital content over the Internet a practical reality. The owners of copyrighted works, particularly in the entertainment area, have become increasingly anxious to ensure that evolving digital technology does not limit or reduce their capacity to enforce their copyrights for financial reward. This concern has motivated a steadily growing interest in the field of Digital Rights Management (DRM).

DRM has been defined as "the management of rights to digital goods and content, including its confinement to authorized use and users and the management of any consequences of that use throughout the entire life cycle of the content" (CEN/ISSS, 2003).

Copyright © 2005, Australian Computer Society, Inc. This paper appeared at the *Australasian Information Security Workshop 2005 (AISW2005)*, Newcastle, Australia. Conferences in Research and Practice in Information Technology, Vol. 44. Reproduction for academic, not-for profit purposes permitted provided this text is included.

This definition encompasses two distinct aspects of DRM that are independently recognised as being worthy of protection in the World Intellectual Property Organisation's copyright treaty (WIPO, 1996): firstly, *rights management information*, which includes "information which identifies the work, the author of the work, the owner of any right in the work, or information about the terms and conditions of use of the work", and secondly, *technological enforcement*, which encompasses effective technological mechanisms used to enforce the terms and conditions of use for a work.

Although these two aspects of DRM cannot be cleanly separated, it is the second aspect that is the principle focus of this paper. In particular, we consider the difficult challenge of technological enforcement on open computing platforms such as a general purpose personal computer (PC).

The essential premise of DRM is that a rights owner wishes to license digital content (which is represented as binary digits or *bits*) to a licensee or *customer* who agrees to be bound by the terms of the license. Note that the customer is not buying the bits themselves. Rather, they are buying the right to use the bits in a defined and restricted manner, as authorised in the terms of the license. Hence the license defines a type of usage policy. A hypothetical license might authorise a single playing of the content on a nominated platform. Copying, multiple 'plays', redistribution and modification may be prohibited. Technological enforcement is necessary because the rights owner does not necessarily trust the customer, yet they would like to have a reasonable level of assurance that the license terms will be complied with even though the content is stored and used on devices that they do not own or control. Digital information can be protected from unauthorised access in transit and storage by well-understood cryptographic techniques. As Schneck (1999) argues, the more complicated challenge presented by DRM flows from the observation that the content bits must be *in the clear* (i.e., not protected by encryption) on the client platform in order to be rendered in a manner perceptible to a user. If the decrypted content bits can be accessed, (for example by using a kernel debugger or modified device driver) the technological enforcement of the license conditions can be circumvented. Once dissociated from its protection, the content can be freely copied, played, modified and redistributed, albeit in violation of the license terms.

Consequently, to reliably enforce typical DRM policies, it must not be possible for the platform user to access the plaintext bits that represent the content, despite the practical reality that the platform is under the user's direct control[1]. This is an access control problem that cannot be solved purely by cryptography. On open computing platforms that can run arbitrary software, it is a difficult problem to which there is currently no practical, deployed solution, particularly in terms of 'software-only' techniques. Recent trusted computing initiatives, namely Microsoft's Next Generation Secure Computing Base (NGSCB) (Microsoft, 2004) and the Trusted Computing Group (TCG) specification, (Trusted Computing Group, 2003) formerly known as TCPA aim in part, to address this issue through both hardware and software based methods (Anderson, 2003).

The goal of trusted computing is to deliver systems that are highly resistant to subversion by malicious adversaries, allowing them to operate reliably and predictably in almost any circumstance. Trusted computing is an important ingredient in DRM because it provides a sound basis for license enforcement. Given the way the NGSCB and TCG initiatives have been promoted, one could be forgiven for thinking that trusted computing is an entirely new concept. As we discuss in Section 3.1, trusted computing actually has a long history but the lessons this history can teach have been largely ignored over the last 20 years, particularly in the design of mainstream PC operating systems[2]. As a consequence, such systems are fundamentally ill equipped to provide the level of protection that a robust DRM system demands.

1.1 Contribution

In this paper we explain in detail why mainstream, commercial grade operating systems are an inappropriate platform on which to base a DRM client. We clarify why DRM platforms require an operating system that can enforce mandatory access control. We aim to address common misunderstandings as to the extent to which the TCG specification implements DRM. A key conclusion of our analysis is that the addition of TCG components to a discretionary access control enforcing operating system does not result in a 'trusted system' that can reliably enforce DRM licenses. We identify problems with DRM related applications of the TCG sealed storage feature that flow from the non-deterministic order of execution of a multi-tasking operating system. We highlight issues that undermine the effectiveness of the TCG remote attestation feature when it is deployed on mainstream operating systems.

1.2 Overview

The remainder of this paper is organised as follows. Section 2 provides background on the relevance of trusted systems to DRM. Section 3 examines operating system properties that are necessary to support secure DRM client applications, highlighting the deficiency of mainstream operating systems in relation to these requirements. Section 4 presents an analysis of the TCG specification and the degree to which it can improve the trustworthiness of mainstream operating systems. Section 5 examines Microsoft's NGSCB initiative and Section 6 provides conclusions to this analysis.

2 DRM Client Security and Trusted Systems

Stefik (1997) argues that *trusted systems* are a basic requirement for robust DRM. Stefik defines a trusted system as one "that can be relied on to follow certain rules. In the context of digital works, a trusted system follows rules governing the terms, conditions and fees for using digital works." The rules or *license terms* are expressed in a computer-interpretable format, (known as a rights expression language) and the content rendering client ensures that the content is protected and manipulated in a manner consistent with the rules. Stefik does not describe the properties a trusted system should have to reliably enforce the rules. However it is implicit from the functional description that the trusted system must protect the underlying bit representation of the content from direct access by a user. If this were not the case, bypassing rule enforcement would be trivial.

According to Lacy *et al.* (1997), the license interpretation and enforcement components of the client's content rendering platform, (which we will refer to as the *DRM client*) must be implemented within the boundaries of a *Trusted Computing Base* (TCB). Lacy does not consider the complications that are introduced if the DRM client is an open computing device capable of running arbitrary software.

Biddle *et al.* (2003) identify the following necessary enforcement properties for a DRM client:

1. *The client cannot remove the encryption from the file and send it to a peer.*
2. *The client cannot 'clone' its DRM system to make it run on another host.*
3. *The client obeys the rules set out in the DRM license.*
4. *The client cannot separate the rules from the payload.*

When the DRM client is implemented in application software on a mainstream operating system, Hauser and Wenz (2003) contend that policy enforcement can be bypassed with relative ease. They document a number of successful attacks on deployed schemes. Properties 1, 3 and 4 are particularly difficult to implement in application level clients that run on mainstream operating systems. In the next section we discuss reasons for the inadequacy of

[1] An alternative policy enforcement strategy relies on the detection of policy descriptive watermarks embedded in content. To be effective, all content rendering devices must detect these watermarks and process the content accordingly. Biddle *et al.* (2003) argue that the requirement for all computing devices to implement and enforce such protections is impractical.

[2] We adopt the term *mainstream operating system* to refer to popular commercial operating systems such as Windows 95, NT, 2000 and XP from Microsoft Inc. (USA), Linux from various distributions of this open source system and various versions of Unix that implement a discretionary access control policy.

application level policy enforcement, in the absence of certain operating system features.

3 DRM, Trusted Systems and Mandatory Access Control

This section examines the relationship between operating system architecture and the effectiveness of DRM license enforcement.

According to Loscocco *et al.*, (1998) "current security efforts suffer from the flawed assumption that adequate security can be provided in applications with the existing security mechanisms of mainstream operating systems." They identify two important operating system mechanisms that are necessary for effective application-level security policy enforcement: firstly the ability to enforce a mandatory security policy; and secondly, a trusted path for user input and output. Since neither property is implemented in current mainstream operating systems they argue that the security of applications based thereon can be nothing more than "a fortress built on sand". This contention is certainly consistent with the results reported by Hauser and Wenz (2003).

The terms mandatory security policy and Mandatory Access Control (MAC) are used in a broader sense than their more common association with Multi-Level Security (MLS) systems that are based on the work of Bell and La Padula (1973). As Loscocco *et al.* (1998) describe, a system must be capable of enforcing a defined security policy in a mandatory fashion. This means that policy enforcement must be assured. To achieve mandatory security, security policy must be established, configured and maintained by an authorised security policy administrator (as opposed to Discretionary Access Control (DAC) systems, which allow ordinary users to configure access policy). The policy may be expressed in MLS terms or it may be role based[3] or it may be based on Domain and Type enforcement (DTE)[4]. The key requirement is that the operating system and underlying hardware are designed so that it is not possible for software or users to reconfigure or subvert the enforcement of the policy. Loscocco *et al.*, argue that carefully controlled memory segregation is critically important in achieving this requirement. When it is attained, mutually distrustful applications are able to execute on the same platform without being able to access each other's resources (if the policy does not allow it). This is a crucial requirement for DRM clients if Biddle's enforcement properties, as described in Section 2, are to be met.

3.1 Architecture Flaws in Mainstream OSs

A key limitation of mainstream operating systems that renders them inappropriate as platforms for a DRM client is their enforcement of an identity based discretionary security policy rather than a mandatory security policy. The DAC model is incompatible with the trust relationship that exists between a content provider and a content consumer, who is not assumed to be trustworthy. Since content is locally rendered, the content provider is forced to rely on the customer's DRM client to enforce their license terms but the environment that this client operates in is under the customer's control. In a DAC system, control of the execution environment gives the customer the ability to subvert the policy enforcement mechanisms of their DRM client[5]. The system owner can run privileged software such as a modified device driver that is able to access the memory space of the DRM client[6]. In a MAC system based for example on DTE, it is possible to configure the MAC policy so that the DRM client's memory space cannot be accessed, even by other parts of the operating system. If the content provider can evaluate the MAC policy configuration to confirm that it enforces DRM client isolation, they can establish a degree of trust in the DRM client's ability to enforce the license terms – on the grounds that the application's technological enforcement measures cannot be tampered with or bypassed and its address space cannot be snooped by other privileged processes to copy decrypted content bits.

A MAC capability provides a sound basis for policy enforcement through rigorous control over information flows between subjects and objects (see Badger *et al.*, 1995). MAC systems rely on the concept of a reference monitor, (Anderson, 1972) which is responsible for enforcing the policy. The reference monitor mediates every access to system resources and data, (collectively known as objects) deciding whether the requested access is consistent with the policy. To ensure that every access is mediated, it must not be possible to bypass the reference monitor. The reference monitor mechanism also needs to be protected against tampering to ensure that an attacker cannot subvert or influence its access decisions. Finally, the reference monitor needs to be small enough to allow it to be validated via analysis and testing. These properties can be achieved by software in concert with hardware protection mechanisms (Schell *et al.*, 1985). The totality of software and hardware that is responsible for enforcing the security policy is known as the Trusted Computing Base (TCB).

A further critical weakness in mainstream operating systems is their inability to implement the security principle of *least privilege*, described for example, by Saltzer and Schroeder (1975). As the name suggests, least privilege requires a program or user to be given only the minimum set of access rights necessary to complete a task. To achieve this, a system needs to be able to express and enforce fine-grained access rights. In today's mainstream operating systems, privileges are bound to so called user 'IDs' so access decisions are based on user identity. As a consequence, all of a user's privileges are granted to each program running on behalf of that user.

[3] See: Ferraiolo, *et al.* (1995)

[4] See: Badger *et al.* (1995)

[5] A DAC architecture cannot reliably enforce a mandatory security policy (Harrison *et al.*, 1976).

[6] TCG integrity measurement capabilities do not effectively address this problem. See Section 4.3

There is no efficient mechanism to reduce the set of available privileges to those that are actually needed.

To make matters worse mainstream operating systems have only two major categories of users: the root or super-user, and normal users. As the name 'super-user' implies, processes with super-user privilege cannot be constrained by access controls as there is no reference monitor. This operating system architecture creates a serious problem for DRM applications, because the platform owner typically has access to the super-user account. Without access controls, a DRM license cannot be enforced against the super-user and plaintext content bits cannot be reliably protected. Observance of the principle of least privilege and enforcement of MAC underpin effective domain confinement through reliable control over information flows within the operating system (Saltzer and Schroeder, 1975). A DRM client based on an open computing platform cannot successfully maintain Biddle's minimum enforcement properties, (listed in Section 2) without the confinement and information flow control that a reference monitor enables.

Device drivers in mainstream operating systems present a particular problem because they must be totally trusted but they are not necessarily trustworthy. Drivers are tailored to a specific piece of hardware (e.g., a particular sound or graphics card) so they are normally provided by the hardware vendor. This creates a problem of trust. Solomon and Russinovich (2000) note: "device driver code has complete access to system memory space and can bypass Windows 2000 security to access objects." So device driver code is like application code in that it is supplied by a range of sources. But it is effectively operating system code since it has unrestricted access to system resources. To further complicate matters, device drivers can be dynamically loaded at runtime. Thus, a malicious or buggy driver can be used for example, to compromise cryptographic keys and the plaintext of protected content. A digitally signed driver provided by a trusted vendor is a common approach to combat this problem. Unfortunately this offers only a partial solution because drivers are typically too large and complex to evaluate to attain a reasonable degree of assurance that they do not contain exploitable bugs or unexpected behaviours. A signature does not guarantee correct operation. It is also difficult to ensure that the integrity of the signature verification mechanism and signer's public key are protected.

Early trusted systems such as Multics, (Corbato *et al.* 1972) addressed the device driver trust issue via a hierarchy of hardware enforced execution domains known as *rings*. Building on the Multics approach, Intel x86 processors have supported a ring based protection architecture (Figure 1) since the 286 chip. This four level design, with ring 0 being the most privileged and ring 3 the least, is intended to allow the operating system kernel, (which implements the reference monitor) to operate at a higher level of hardware enforced privilege than device drivers, other operating system components and code libraries which in turn can have higher privilege than users and application processes. The higher level of privilege ensures that the reference monitor mechanism is more resistant to bypass or tampering by other less trusted processes running in rings of lesser privilege. The quantity and complexity of code that must be trusted to enforce the mandatory security policy is thereby substantially reduced. This makes it easier to establish confidence in its correctness. The x86 hardware architecture is capable of supporting highly secure and trustworthy operation. When correctly utilised, its ring-based architecture combined with its fine-grained memory segmentation allow it to enforce effective domain separation and confinement at the hardware level. Unfortunately, with rare exceptions, (e.g. the GEMSOS OS described in Schell *et al.* (1985)) the protection rings and memory segmentation/capability features of the Intel x86 have not been used by mainstream, general purpose operating system designers as they were intended. Mainstream operating systems use only the most and least privileged rings for system and user space respectively, emulating two state machines. While PC operating systems may not have been designed with security as a high priority, the same cannot be said of the processor on which they are based.

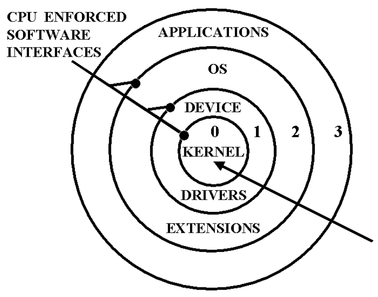


Figure 1: The Intel x86 Ring Architecture

The failure of mainstream operating systems to correctly utilise the ring structure of the x86 processor explains Intel's announced intention to release a new chip with what is effectively a 'ring –1'. According to Peinado, *et al.*, (2004) the reason for this is that Microsoft's new NGSCB trusted computing architecture requires a higher level of privilege to enable effective domain separation within ring 0 which, for reasons of backward compatibility must continue to host device drivers of questionable provenance and other potentially insecure operating system components. This is discussed in more detail in Section 5.

In summary, mainstream operating systems lack the essential features that are required to protect decrypted content and thereby support the enforcement of DRM licenses. They do not enforce a mandatory access policy, and they fail to observe the principle of least privilege, which greatly magnifies the threat presented by software bugs and privileged but malicious code. In addition, the sheer volume and complexity of privileged code, (including device drivers) means that there is no possibility of gaining any reasonable level of assurance that a platform will obey DRM license terms. Trust mechanisms based on signed code and drivers do not alter this situation since the problem flows from the access control model and operating system architecture.

4 Trusted Computing Group - formerly TCPA

In response to myriad problems created by the insecurity of open computing platforms, the Trusted Computing Group (TCG) has proposed a trusted computing platform specification (Trusted Computing Group 2003). In this section we briefly describe key aspects of the specification. In the context of a DRM client application, we analyse the operating system features that are necessary to make meaningful use of TCG services, particularly *remote attestation* and *sealed storage*.

The Trusted Computing Group (TCG), successor to the Trusted Computing Platform Alliance (TCPA), is an initiative led by AMD, Hewlett-Packard, IBM, Intel, Microsoft, Sony, and Sun Microsystems. The TCG aims to "develop and promote open, vendor-neutral, industry standard specifications for trusted computing building blocks and software interfaces across multiple platforms"[7].

The novelty of the TCG architecture lies in the range of entities that are able to use TCG features as a basis for trust. These include not only the platform user and owner but also, remote entities wishing to interact with the platform. The mechanism of *remote attestation* allows remote third parties to challenge a platform to report details of its current software state. On the basis of the attestation, third parties can decide whether they consider the platform's configuration to be trustworthy. If correctly implemented, remote attestation promises to be an important feature for DRM clients on open platforms since it may assist a content provider in deciding whether the client is currently configured to enforce the license terms reliably before the content is actually provided.

A closely related TCG objective is to provide reliable, hardware-based protection for secrets such as cryptographic keys. Since open computing platforms can run arbitrary software, this objective aims to ensure that protected secrets will not be revealed unless the platform's software state meets clearly defined and accurately measurable criteria. TCG's *sealed storage* feature can be used to bind a protected secret such as a content decryption key to a particular software configuration. If the configuration is not as specified, the sealed key will not be released.

4.1 TCG Architectural Modifications

The architectural modifications required by the TCG specification include the addition of a cryptographic processor called a Trusted Platform Module (TPM). The TPM must be a fixed part of the computing device that cannot (easily) be transferred to another platform. The TPM provides a range of cryptographic primitives including random number generation, SHA-1 hashing, asymmetric encryption and decryption, signing and verification using 2048 bit RSA, and asymmetric key pair generation[8]. There is also a small amount of protected key storage. Currently available TPMs are based on smart card processors.

[7] See: https://www.trustedcomputinggroup.org/home

[8] See: Menezes et al., (1996) for a description of these terms

4.2 Integrity Measurement and Reporting

The TCG security services of remote attestation and sealed storage build on an integrity protected boot technique that was introduced by Arbaugh *et al.* (1997). Integrity protected booting is fundamental to the design of the TCG architecture. Figure 2 illustrates the process with numbers in parentheses denoting the sequence of events.

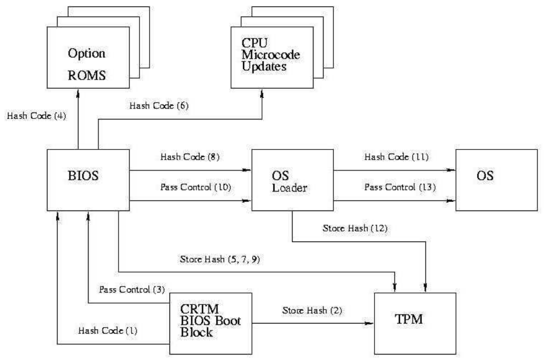


Figure 2: TCG Integrity Protected Boot Sequence

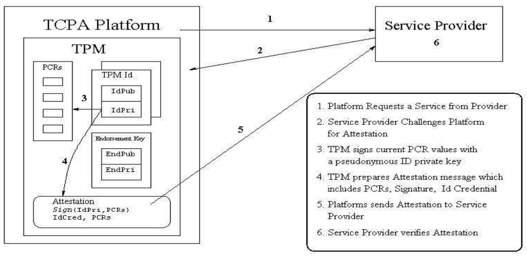
The boot process starts in a defined state with execution of the BIOS 'boot block' code. The BIOS boot block is called the Core Root of Trust for Measurement (CRTM). Since it initiates the booting and measurement process, it is implicitly trusted. The core idea behind integrity protected booting is that a precise hash-based measurement or *fingerprint* of all executable code in the boot chain should be taken and securely stored immediately before that code is given control of the processor. Accordingly, the CRTM takes a hash of the BIOS code (1) and stores the value in a protected hardware register in the TPM (2), called a Platform Configuration Register (PCR). PCRs cannot be deleted or arbitrarily overwritten within a boot cycle. They are 'update only' using a simple chained hash technique, based on the SHA1 secure hash algorithm that works as follows (where || denotes concatenation):

Updated PCR Value=Hash(Previous PCR Value || Current Measurement To Store)

This operation is known as *extending* a PCR. It allows a practically unlimited number of measurements to be stored or *committed* in a fixed size register.

The CRTM then passes control to the BIOS code (3) which stores measurements of option ROMS, CPU microcode updates and the OS loader before passing control to the latter. The boot process continues following the same pattern until the kernel is loaded. If any executable stage in this chain has been modified, the change will be reflected in the hash value. Since the PCRs can only be extended, not overwritten, the modified code cannot hide itself when it is given control of the CPU.

Remote attestation allows a TCG enabled platform to assert the state of its current software environment to a third party. The TPM uses a certified key pair that identifies the platform as a genuine TCG platform, to sign the current PCR values. Figure 3 illustrates the attestation procedure in a DRM setting. Before delivering protected content a content provider can challenge a requesting TCG platform to attest on its current configuration. The platform sends the signed PCRs together with a log containing each of the individual measurements that have been extended into the PCR hash chains. The provider reviews each individual measurement to ensure that it corresponds to a component that it considers 'trusted'. If satisfied with the platform's state, the content can be delivered.



TCPA Platform
TPM
PCRs
TPM Id
IdPub
IdPri
Endorsement Key
EndPub
EndPri
Attestation
Sign(IdPri,PCRs)
IdCred, PCRs
Service Provider
1
2
3
4
5
6
1. Platform Requests a Service from Provider
2. Service Provider Challenges Platform for Attestation
3. TPM signs current PCR values with a pseudonymous ID private key
4. TPM prepares Attestation message which includes PCRs, Signature, Id Credential
5. Platforms sends Attestation to Service Provider
6. Service Provider verifies Attestation

Figure 3: TCG Remote Attestation Protocol

4.3 Operating System Support for TCG Trusted Computing

In this section we examine the types of operating system support that are necessary to meaningfully make use of remote attestation and sealed storage features. We highlight a number of serious challenges in applying TCG concepts of trusted computing to mainstream operating systems.

The TCG specification is claimed to be operating system neutral. It defines lowest common denominator functionality for a range of platform types including PDAs, mobile phones and PCs. Consequently, it does not deal with operating system architecture issues and it does not mandate or suggest operating system security features necessary for TCG services to work reliably. The specification defines requirements as far as the operating system (or bootstrap) loader. Despite the TCG specification's understandable silence in this regard, we contend that a multi-tasking operating system cannot meaningfully implement TCG functions, particularly remote attestation and sealed storage unless it has the 'classical' trusted computing features that were described in Section 3. The TCG's integrity verification approach is not a substitute for sound operating system architecture.

To understand why this is so, consider a typical DRM content delivery scenario wherein a TCG enabled DRM client platform wishes to connect to a content provider to download and subsequently view a movie. In exchange for payment, the provider will transfer an encrypted copy of the movie and a license for a single viewing. The license prohibits copying, modification and transfer to other platforms. Note that these license terms implicate all of Biddle's enforcement properties (listed in Section 2). Building on TCG features, the content provider takes a number of steps to ensure the integrity of the policy enforcement mechanisms on the client's platform. We use the term *integrity* to refer to the ongoing ability of the platform to reliably enforce the license terms. There are three main types of verification requirement to determine and preserve integrity:

1. The content provider requires client platforms to establish via remote attestation that they have booted according to TCG principles. The content provider assesses the booted software environment against a list of 'trusted' components to ensure that the client booted into a trusted state.
2. The content provider must ensure that the client has not executed untrusted software *after* the boot process completed but before the remote attestation was initiated.
3. The content provider must ensure that a trusted version of the DRM client is executing at the time of the remote attestation, and that client cannot execute untrusted software while the protected content is being accessed or viewed.

We will now analyse the practical implications of addressing these requirements in the context of a mainstream operating system.

The motivation for the first requirement is self-evident. However, it is worth emphasizing that the TCG design is based on the premise that a system can be trusted if the PCR registers match values expected by a relying party. The expected values must be those of a *known* secure configuration. This assumes that a secure and trusted configuration actually exists and that the configuration is trusted because it is *trustworthy*. As we argued in Section 3.1, this assumption is not well founded in the case of mainstream operating systems because complexity and architecture defeat assurance of integrity. TCG does not solve code quality or operating system architecture problems. This is a critical and often overlooked limitation of the TCG model of trusted computing.

The second and third requirements reflect the fact that the integrity of a platform (running a mainstream operating system) is dependent on its runtime behaviour. A theoretically trustworthy state immediately post-boot provides no guarantee that integrity will maintained, since software executing post-boot can affect the integrity of the platform. For example, a dynamically loaded kernel module, device driver or privileged application can potentially execute at any time and violate protection requirements. As we noted in Section 3.1, this is possible because there is no mandatory control over information flow among and within processes running at the highest level of privilege.

To support requirements two and three, the operating system itself needs to be modified to include a measuring function that fingerprints the executable code and configuration data of any system or user level process before it is launched. Sailer *et al.*, (2004) propose a 'TCG based integrity measurement architecture' based on

Linux in which they describe a number of operating system modifications to this end. They describe the *instrumentation* of functions that initiate the loading of code and data that may impact platform integrity. In a DAC-based operating system not enforcing the principle of least privilege, there are many such places where the measuring function must be called. Relatively simple candidates include functions that load kernel modules, dynamically loadable libraries, and user-level executables. Other more problematic candidates include script interpreters, virtual machines, (e.g., the java class loader) privileged applications and network services. The instrumentation itself is relatively simple. It requires a call to a 'hash and store' function immediately prior to code execution. The problem lies in the practical difficulty of ensuring that *all* code is instrumented – particularly privileged application code. The source code of legacy and proprietary applications is not always available and software vendors may not be motivated or able to issue instrumented versions of all software that is still in use. Arguably, the impracticality of instrumenting all script interpreters, virtual machines, just in time compilers, and privileged applications calls into question the viability of the whole approach. The failure to instrument results in potential avenues for integrity violation.

The third requirement is the most challenging since it requires that the PCR values *always* reflect the current configuration. Therefore, to be able to verify the integrity of any one process all other processes must be measured on an ongoing basis (Sailer *et al*., 2004). Since the integrity of a process may be violated at any moment, an attestation is only meaningful at the instant of the last measurement. By the time a challenger has evaluated an attestation response, the integrity status may have changed in a material way. The challenger has no way to know if this has happened except to continually rechallenge the platform, a highly inefficient and unsatisfactory option that in any case can only provide retrospective assurance.

This problem also impacts sealed storage. A platform may have been in the configuration mandated in a sealed storage policy (as reflected in the required PCR values) at the time a protected cryptographic key was released to the operating system but this configuration can subsequently change (i.e., within the same boot cycle) putting the key and decrypted cipher text at risk of compromise.

In the absence of MAC based domain confinement, what is needed is a reliable way for the platform to revoke its trusted status and immediately purge any protected content or keys from memory if an integrity-relevant change occurs. The insurmountable difficulty lies in distinguishing a change that should result in trusted status revocation from an integrity preserving one. The reason we consider the problem to be insurmountable relates to how unrecognised fingerprints that have been extended into PCRs should be handled. This is discussed in the next section.

4.4 Maintaining Integrity Assurance

In the TCG remote attestation protocol, the challenged platform sends the current PCR values together with a log of the individual fingerprints that have been chained together to produce these values. The challenger can use the log to determine whether it trusts the components that are identified by each individual measurement that makes up a hash chain. To identify any tampering with the log, it can recalculate the chain to ensure it produces the reported PCR value. From a challenger's perspective, the presence of any unrecognised fingerprint in the log should result in the platform being considered untrustworthy - the unknown fingerprint could be that of a malicious component such as a modified device driver or a program that provides the same functionality as an instrumented program but without the instrumentation, e.g. a kernel module loader that can violate the integrity of the platform by loading malicious code that will not be reflected in the current PCR values.

This will only present a problem if unrecognised fingerprints can be legitimately expected from an otherwise 'trustworthy' platform. Unfortunately they are highly likely because loosely structured data, (including scripting files, configuration files etc.,) must be fingerprinted. This is necessary because the runtime behaviour of a program is commonly determined by the configuration files it parses at start up, and also, the data consumed as inputs once running. Therefore, to assess the integrity impact of executable code, its inputs must be measured. The measurement of semi-structured and unstructured input data and configuration files is particularly problematic in the context of the TCG architecture since they are not as amenable to integrity verification via fingerprinting, as is static code. Non-executable files of this type can tolerate subtle differences such as extra white space characters, comments or the same elements in a different order, with no impact on integrity. Nonetheless, any such difference will produce a completely different fingerprint. Differences of this type can be reasonably expected in the real world as users tailor system behaviour to meet their individual needs via configuration files. Without access to the measured file itself, the challenger will be unable to determine whether an unrecognised fingerprint results from an irrelevant formatting difference in a semi-structured file as opposed to a malicious component or an un-instrumented component loader. Hash-based integrity measurement may be practical for executable code but it is very unforgiving when applied to semi-structured data – arguably so unforgiving as to call the whole approach into question.

This line of reasoning applies equally to TCG's sealed storage feature. We noted that sealed storage allows the release of a protected key to be conditional on the current status of PCR registers matching some predefined and trusted values. The presence of an unknown measurement in the PCR hash chains will render a sealed object inaccessible. New measurements can be introduced by changes in the software or hardware configuration (e.g., an upgrade of a video card).

We believe the use of sealed storage for DRM content protection in mainstream operating systems is impractical for two reasons. The first relates to the order PCRs are extended in. With hash chains, the order of element chaining determines the resulting output value. Therefore, to access a sealed object, all the fingerprints that have been extended into a PCR must be trusted *and* they must be chained together in precisely the same order that produced the reference PCR values to which the object is sealed. In mainstream operating systems, the order of PCR extension is deterministic up to the operating system loader. After this point, order depends on individual runtime behaviour as the operating system kernel proceeds to launch multiple concurrent processes which themselves may be multithreaded. In such a multi-tasking environment, execution order is not deterministic.

The second reason why sealed storage is impractical flows from the observation in Section 4.3 that to maintain integrity all executable code needs to be measured before it is loaded. This means that PCRs continue to be extended as new applications are run. Therefore, in typical usage they do not stabilise to a predetermined value[9]. This problem could be ameliorated by sealing an object to a subset of PCR values that only reflects the early stages of the boot process, perhaps up to the loading of the operating system kernel. This would be more likely to produce the deterministic result that sealed storage requires. It will not however capture post boot platform configuration changes such as the loading of kernel modules that can materially impact integrity.

The impracticality of remote attestation and sealed storage on mainstream operating systems is a serious drawback. It underlines the fact that the TCG building blocks cannot remedy problems that flow from operating system architectural deficiencies. Secure DRM clients cannot be deployed on multi-tasking operating systems that are unable to provide isolation and confinement of mutually distrustful and potentially hostile processes. The application of TCG components does not change this fact. Sailer *et al.*, (2004) assert that "many of the Microsoft NGSCB guarantees can be obtained on today's hardware and today's software and that these guarantees do not require a new CPU mode or operating system but merely depend on the availability of an independent trusted entity, a TPM for example." We contend that while a number of guarantees may be possible, the important ones, (not necessarily delivered by NCSGB either) cannot be achieved, since critical elements of a trusted system must be enforced by a combination of the operating system mapped to CPU hardware-based protection structures. They cannot be provided by add-on hardware.

The addition of TCG components to mainstream operating systems does not result in a 'trusted system' in the traditional sense of the term. It does not introduce trusted paths, or a reference monitor, and it does not alter the problems created by the failure to observe the principle of least privilege. The TPM allows integrity measurements to be stored in a trusted fashion but it does not provide any mechanism to prevent integrity violations. It does allow known versions of compromised software to be identified by their fingerprints and this is definitely useful. It also provides a more secure environment for the storage of cryptographic keys, particularly asymmetric signing keys. However, effective domain confinement and mandatory access control are fundamental requirements for trusted computing and reliable license enforcement in DRM clients. For these reasons we believe that the security benefits gained by applying TCG components to mainstream operating systems that enforce purely discretionary access control are worthwhile but modest.

TCG components combined with MAC-based operating systems such as SELinux (Loscocco, 2001) have significant advantages over their DAC based counterparts – advantages that may make sealed storage and remote attestation considerably more practical. In a MAC based system a challenger needs to verify the integrity of the security policy interpretation and enforcement mechanism, the policy configuration itself and the DRM client application. If these components are trusted to enforce isolation of the DRM client, no further measurements are required to establish the *ongoing* integrity of the DRM client. Thus the impractical requirement for ongoing measurement in DAC architectures is avoided. Since integrity assurance is based on isolation, detailed measurements of other loaded executables and configuration data is not required.

4.5 Privacy Impacts

Remote attestation reveals particularly fine-grained details of a platform's configuration that are likely to be sufficient to distinguish different platforms from each other and to recognise the same platform across different sessions. In DAC enforcing architectures in particular, the privacy protection mechanisms detailed in the TCG specification, including the zero-knowledge authentication protocols introduced in version 1.2 are at great risk of being rendered ineffective.

5 Next Generation Secure Computing Base (NGSCB) formerly Palladium

'Next Generation Secure Computing Base' (NGSCB) is the name for Microsoft Inc.'s trusted computing program, formerly known as Palladium. According to the designers of the system Peinado *et al.*, (2004) NGSCB is a "high assurance runtime environment for trustworthy applications on a regular personal computer". NGSCB builds on the TCG specification and a number of significant planned modifications to the CPU and chipset that have been announced by Intel Inc. under their 'LaGrande' program. The chipset modifications are apparently designed to introduce a trusted path for keyboard, mouse and other input, and secure display output. The CPU modifications introduce a new mode akin to a 'ring –1' and a number of new instructions to support it. The new mode is required for reasons of

[9] Version 1.2 of the TCG Specification introduced PCRs that can be reset within a boot cycle.

backward compatibility with existing Microsoft 'Windows' operating systems.

Backward compatibility was stated as a key requirement for NGSCB. As we noted in Section 3.1, the Windows kernel, device drivers and operating system operate at the highest level of hardware privilege, in ring 0. Due to the untrustworthy bug-prone nature of some operating system components and drivers running in this ring, there is no way to introduce a "high assurance runtime environment" because there is no tamper resistant space in which to implement a domain separation mechanism. Virtual Machine Monitors (VMM), a well understood approach to isolating mutually distrusting systems on the same CPU are not an option in this case because, according to Robin and Irvine, (Robin and Irvine, 2000) the Intel x86 cannot be securely virtualised. In the simplest version of the VMM approach, multiple operating systems can be simultaneously hosted in ring 1 by a VMM that runs in ring 0. The VMM presents the same interface to the hosted operating systems as the native hardware and provides domain isolation between them.

Full virtualisation of the x86 chip may not be possible but, nonetheless, the type of isolation that NGSCB demands requires something roughly equivalent to a VMM and a VMM requires a higher level of privilege than the hosted operating systems in order to work. This is the reason behind the new mode, 'ring –1' that Intel announced for their 'LaGrande' project.

Peinado et al., (Peinado *et al*. 2004) describe the NGSCB equivalent of a VMM as an *isolation kernel*. The isolation kernel protects the memory and device resources of a domain from access by other domains. The isolation kernel is not itself a fully featured operating system, it merely manages the coexistence of multiple operating systems on the same machine. Again, for reasons of backward compatibility, the isolation kernel does not boot before the standard Windows operating system and it does not run beneath it. This somewhat unconventional approach is a key difference to the TCG style of trusted computing, which bases its assurance on an assessment of the integrity of the entire boot chain. NGSCB also takes a different tack in not assuming that the BIOS is trustworthy. In a sense, the NGSCB approach has abandoned any premise of making the traditional Windows operating system 'trusted' and opted instead to construct a secure environment that can run along side it – "a tight sanctuary on an otherwise bug-prone system" in the words of Peinado et al., (2004). This tight sanctuary can be launched and terminated multiple times in the same boot cycle and multiple sanctuaries can simultaneously coexist.

Like TCG, the NGSCB design supports remote attestation, and sealed storage. Unlike the basic version of TCG described in the previous section, it supports hardware enforced domain isolation and trusted paths. As between domains, this is a form of MAC since, once activated neither users nor administrators can turn the hardware protection off. Microsoft has asserted that the code for the isolation kernel will be sufficiently small and simple to provide assurance of its correctness through independent evaluation. Thus NGSCB's 'sanctuary' appears to embody the important features that are required of a trustworthy computing platform. However, Microsoft has not described in any detail the trusted operating system that the isolation kernel actually launches – the trusted sanctuary itself. This sanctuary is intended to support multiple applications or 'applets' which do not necessarily trust each other. Thus an operating system capable of the functions described in Section 3 will also be required.

According to Rooney (Rooney, 2004) the future of NGSCB is far from certain. Application developers have expressed reluctance to rewrite code to take advantage of NGSCB features and Microsoft is reportedly reviewing the design to make the task of application integration more streamlined. The impact that this will have on the trustworthiness of NGSCB is unclear. As at the time of writing, the Microsoft NGSCB web page states, "Microsoft is currently evolving the NGSCB architecture to expand the benefits to customers in terms of applicability and flexibility". (See: Microsoft, 2004). Experience has shown that flexibility tends to undermine security, but certainly, no conclusions can be drawn until the revised design is clarified. Rooney notes that Microsoft is shifting emphasis to support No Execute (NX) technology that has been implemented by Intel in its recent processors. NX allows memory pages to be marked as non-executable to limit common attacks that exploit buffer overflow. We note that a strategy based purely on NX falls well short of the requirements for trusted computing. It is also worth noting that if the Intel segmentation/capability architecture were correctly utilised, NX would be unnecessary as a no-execute control bit is already available at the segment level.

6 Conclusion

We began by emphasizing the importance of trusted computing for robust DRM license enforcement. The license validation, interpretation and enforcement functions of a DRM client need to implemented within the bounds of a trusted computing base. With reference to the early trusted computing literature, we reviewed the critical features that define a trusted computing platform. Enforcement of a mandatory access control policy, observance of the principle of least privilege and provision of trusted paths were identified as key requirements. These are absent in mainstream operating systems. For this reason, we argued that such operating systems are inappropriate to host DRM client applications. We then analysed the degree to which the TCG model of trusted computing can be applied to mainstream operating systems to improve their trustworthiness. We concluded that the addition of TCG components to such systems does not result in a trusted system. The TCG specification cannot substitute for the absence of MAC and the associated domain confinement it supports. We identified specific problems that make the TCG features of sealed storage and remote attestation impractical when they are used in conjunction with a DAC enforcing operating system. We concluded that the TCG specification in conjunction with a MAC enforcing operating system may offer a considerably more robust platform on which to deploy a DRM client.

7 References

Anderson, J.P. (1972). Computer Security Technology Planning Study. ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA.

Anderson, R.J. (2003). Cryptography and competition policy: issues with 'trusted computing'. In *Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing (PODC 2003),* July 13-16, Boston, Massachusetts, pp 3-10.

Arbaugh, W.A. Farber, D.J. et al. (1997). A secure and reliable boot strap architecture. In *Proceedings of 1997 IEEE Symposium on Security and Privacy*, pp 65-71, May 1997.

Badger, L. Sterne, D.F. *et al*. (1995). Practical domain and type enforcement for Unix. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, page 66, IEEE Computer Society.

Bell, D.E. and LaPadula, L.J (1973). Secure computer systems: Mathematical foundations and model. Technical Report M74 244, The MITRE Corp., Bedford MA.

Biddle, P., England, P., Peinado, M. and Willman, B. (2003). The Darknet and the future of content protection. In *Digital Rights Management-Technological, Economic, Legal and Political Aspects.* LNCS 2770, Springer, pp 344-365.

Corbato, F.J., Saltzer, J.H. and Clingen C.T. (1972). Multics - the first seven years. In *Proceedings of the Spring Joint Computer Conference*, pages 571-583, AFIPS Press.

CEN/ISSS, (2003). Digital Rights Management Report http://europa.eu.int/comm/enterprise/ict/policy/doc/drm.pdf. Accessed 18 August 2004.

Ferraiolo, D. Cugini, J. and Kuhn, R. (1995). Role Based Access Control (RBAC): Features and motivations. In *Proceedings of the Annual Computer Security Applications Conference*, New Orleans, Louisianna, December 11-15, IEEE Computer Society Press.

Harrison, M., Ruzzo, W. and Ullman, J. (1976). Protection in Operating Systems. *Communications of the ACM,* 19(8) pp. 461-471.

Hauser, T. and Wenz, C. (2003): DRM Under Attack: Weaknesses in Existing Systems. In *Digital Rights Management-Technological, Economic, Legal and Political Aspects.* LNCS 2770, Springer, pp 206-223.

Lacy, J., Snyder, J.H. and Maher, D.P. (1997). Music on the Internet and the intellectual property protection problem. In *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE 97),* Vol 1, pp 77-83.

Loscocco, P.A. Smalley, S.D. et al. (1998). The inevitability of failure: The flawed assumption of security in modern computing environments. In *21st National Information Systems Security Conference*, Arlington, VA., August.

Loscocco, P.A. Smalley, S.D. (2001). Meeting critical security objectives with Security-Enhanced Linux. In *Proceedings of the 2001 Ottawa Linux Symposium*.

Menezes, A.J. Van Oorschot, P.C. and Vanstone, S.A. (1996). *Handbook of Applied Cryptography,* CRC Press, ISBN 0849385237.

Microsoft, (2004), Next Generation Secure Computing Base – Product Information, available at: http://www.microsoft.com/resources/ngscb/productinfo.mspx, accessed 20 July 2004.

Peinado, M. Chen, Y. et al. (2004), NGSCB: A Trusted Open System. In *Proceedings of 9th Australasian Conference on Information Security and Privacy ACISP,* Sydney, Australia, July 13-15.

Robin, J. and Irvine, C. (2000). Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In *Proceedings of the 9th USENIX Security Symposium*, Denver, Colorado, USA, August 14-17.

Rooney, P. (2004). Microsoft Shelves NGSCB Project as NX Moves to Centre Stage. 5 May 2004, http://www.crn.com/sections/BreakingNews/dailyarchives.asp?ArticleID=49936 accessed 7 May 2004.

Sailer, R. Zhang, X. et al. (2004). Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th Usenix Security Symposium*, San Diego, August 9–13.

Saltzer, J.H. and Schroeder, M.D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278-1308, Sept.

Schell, R., Tao, T. and Heckman M. (1985). Designing the GEMSOS security kernel for security and performance. In *Proceedings of National Computer Security Conference*, Gaithersburg.

Schneck, P. (1999). Access Control to Prevent Piracy of Digital Information. *Proceedings of the IEEE*, 87(7):1239-1250.

Solomon, D.A., and Russinovich, M. (2000). *Inside Microsoft Windows 2000*. Microsoft Press, 2000. ISBN 0735610215.

Stefik, M. (1997). Shifting the Possible: How Trusted Systems and Digital Property Rights Challenge Us to Rethink Digital Publishing. *Berkeley Technology Law Journal* Vol. 12. No.1. pp 137-159.

Trusted Computing Group, (2003), Trusted Platform Module Main Specification, Part 1: Design Principles, Part 2: TPM Structures, Part 3: Commands, October 2003, Version 1.2, Revision 62, available at http://www.trustedcomputinggroup.org, accessed 26 May 2004.

WIPO, (1996). World Intellectual Property Organization. Wipo copyright treaty. http://www.wipo.org/eng/diplconf/distrib/treaty01.htm Accessed 15 April 2004.