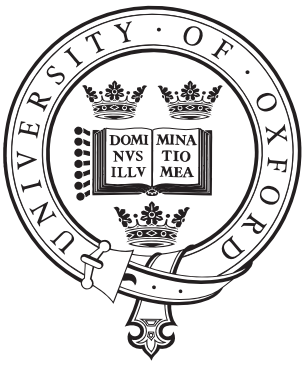
Software Engineering Group

The ten-page introduction to *Trusted Computing*

Andrew Martin

CS-RR-08-11



Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD.

The ten-page introduction to *Trusted Computing*

Andrew Martin, University of Oxford

November 2008

Synopsis

Networked computer systems underlie a great deal of business, social, and government activity today. Everyone is expected to place a great deal of trust in their correct operation, but experience shows that this trust is often misplaced. Such systems have always been subject to failures due to oversights and mistakes by those who designed them; increasingly such failures are exploited by those with malicious intent.

The concept of *Trusted Computing* has been present in the computer security literature for quite some time, and has influenced the design of some high-assurance solutions. These ideas are now becoming incorporated in mainstream products — PCs, mobile phones, disc drives, servers — and are the subject of much discussion and sometimes misinformation.

Trusted computing implies a re-design of systems architecture in such a way as to support its factorization into relatively discrete components with well-defined characteristics. This permits, in particular, rational decisions based upon reasonable expectations of behaviour. Any such systems thinking must be motivated by an analysis of risks — so that effort is expended where it may give the best return — and an awareness of the limitations of such risk assessment (because frequently the raw data and parameters are simply not available, and because security properties are typically not compositional).

The approach described here is largely the result of the work of an industry consortium (the *Trusted Computing Group*, TCG), itself informed by a history of research, largely in the area of high-assurance systems, from government and academe. TCG's approach is distinctive in that previous trusted systems were usually bespoke and highly expensive: the current work aims to touch every computing device.

This tutorial surveys the relevant notions of 'trust', exploring what this means for 'trusted computing'. We describe briefly the interventions needed in hardware and software required to give a stable platform upon which such systems can be constructed. In essence, this gives us two new systems characteristics: (a) a high degree of confidence in the *state* (configuration, running software, etc.) of a local computing system — and hence a measure of its relative freedom from unwanted intervention; (b) a relatively high degree of confidence in the state of a remote system (a property called 'remote attestation').

The first of those characteristics has perhaps always informed the way that users interact with desktop personal computers: many malware attacks exploit misplaced trust in the local system. The second characteristic is genuinely novel, and may be seen as an enabler of many new kinds of pattern of interaction in distributed systems. Knowing that a platform is in a particular state is neither a necessary nor sufficient condition for trustworthiness (or, indeed, security) — but helps to inform decisions about that. We explore how these capabilities are constructed, and some uses to which they might be put. We also briefly describe the state of deployment of these technologies, and some current areas of research.

1 Introduction

Much of what we do in today's society involves us typing things into a computer (or, sometimes now, a mobile phone), seeing them appear on the screen, and thereby interacting with someone or something at a distance. How do we know something desirable will happen when we press 'Go', without something bad happening now or in the future?

By and large, we don't. But we make some kind of informal risk assessment, and live our lives in the light of it. In particular, we tend to trust the computer on the desk in front of us to do the right thing. Or rather, we may not truly trust it, but we have little alternative. This is our starting point for thinking about trusted systems.

We probably know where the computer on the desk (or in our hands) came from: either we bought it from a reputable supplier, or someone we trust installed it for us. We may believe that we know what software is installed on it: the operating system and the applications. But there our problems start: one program may appear like another; life on the world-wide web entails all manner of downloads, some safer than others; we know to our cost that software is imperfect, and these days we expect vendors to patch it periodically, often to rectify problems with its security. If someone else has ever used the computer, they may, by accident or design, have installed software without our knowledge.

Most operating systems are designed around the idea of an all-powerful, professional system administrator. For a great many deployed desktop systems, we know to our cost that the administrative power is in the hands of someone who does not necessarily have the time, or the expertise, or the tools, with which to take administrative and configuration decisions. And yet these systems

are increasingly repositories for content of considerable value — whether purchased digital music, photograph collections, or downloaded software, or keys to online banking applications, tax returns, or medical records.

The same challenges apply to any system we interact with across the network, with the added complexity that because we cannot see or touch the servers involved, we have to take a whole lot more on trust. We generally have to trust — or assume — that those who administer those servers are both wise and good: wise that they will install only 'safe' software, and good in that they will (because of altruism, a contract, or a regulator) always act in our best interests. But intuitively we know that neither assumption is necessarily valid.

Of course, we manage complexity by building software and systems in layers. From a security perspective, we build protections into each layer, appropriate to the functionality it offers. But one recurring strategy of those who wish to attack our systems is to attack 'the layer below'. We may use a strongly-encrypted link to interact with our online bank accounts, but if the library which implements it has been replaced by a rogue, our secrets can still be compromised. Our files may be organised into directories with elaborate access control files attached, but as many public sector organisations have lately demonstrated, such protections avail nothing if the disc holding that filing system itself falls into the wrong hands. We may install a fresh operating system from trustworthy media, but many pieces of software run before a single character appears on the screen; they can readily be corrupted.[1] One such 'rootkit' is sufficient to infect every operating system installed on the corrupted platform, and so to, say, report every keystroke to a third party (whilst our anti-virus product is fooled into thinking that all is well).

2 Trust

We have used the term 'trust' informally above. This notion is, of course, the subject of stacks of literature: in computing, in sociology and psychology, and so on. Indeed, the word is sufficiently over-loaded that Gollman (2004) was able to argue that 'Trust is Bad for Security'. He notes that in the 1980s, the term was synonymous with *multi-level security* (the longstanding goal of many military and intelligence IT systems), and a decade later, synonymous with *digital signatures* (as in code signing).

The US DoD gave us the 'Orange Book' *Trusted Computer Systems Evaluation Criteria* (DoD, 1985) in the 1980s, formalizing the *Trusted Computing Base* as the totality of protection measures within a system: that part of the system (hardware, software, other) which you simply have to rely upon, and whose failure almost inevitably leads to compromise.

For us, then, trusted systems are those upon whose correct (or predictable) operation we rely. If they fail to live up to our expectations, we may expect bad consequences. In a strong sense, this idea of trust is somewhat orthogonal to that of *security*: we may use trusted components to build secure systems. Trust on its own does not entail security: merely, predictable behaviour. We would not generally try to build secure systems from untrustworthy components, but the author can think of a handful of examples where this may work.

Careful speakers distinguish

- trusted systems
- trustworthy systems
- trustable systems

but if we understand the notion of 'trust' in use here, the meaning of these terms follows naturally.[2]

Graeme Proudler says (Mitchell, 2005, chapter 2) that it is safe to trust something when:

1. it can be unambiguously identified, and
2. it operates unhindered, and
3. the user has first-hand experience of consistent, good, behaviour *or* the users trusts someone who vouches for consistent, good, behaviour.

The first two points are easily overlooked, but they are of course crucial. If a rogue replaces a good device, the behaviour will be different. If a piece of software has been tainted by the introduction of a virus — or a replaced link library, or other change of context — normal good behaviour can easily turn bad.

Ideal grounds, perhaps, for building an expectation of good behaviour, are proofs of *correctness*. A system which is provably or demonstrably correct (i.e. it meets its specification) will be trustworthy – at least within the parameters of its specification.[3] Generally, this turns out to be of theoretical value only: a system on the scale of a modern desktop computer is much too complex to be amenable to formal specification, still less, proof.

Instead, we must usually fall back on surrogates for such proof: extensive testing, long experience of use, third party evaluation and certification (with or without liability!) and so on. This is not, of course, out of keeping with the notion of 'trust', which is seldom absolute. The crucial feature is that we are given useful grounds upon which to base the decision about the *degree* of trust.

All of this is summed up (some would say very inadequately) in the *Trusted Computing Group*'s working definition:

> An entity can be trusted if it always behaves in the expected manner for the intended purpose. (TCG 2004)[4]

[1] This is not mere paranoia or speculation (Heasman, 2006).

[2] Having said that, many of those responsible for popularizing 'Trusted Computing' now wish that they had described it more accurately as 'Trustable Computing', but the die is already cast.

[3] Notice that the classical notion of correctness gives us no help at all in establishing the first two steps to trust.

[4] This definition is widely quoted as '(TCG 2004)' but, despite

3 Trusted Computing Platforms

There are many ways in which a computing platform may fail to enable the steps to trust outlined above. (For the purposes of this section, we will consider mainly the PC platform; later sections will broaden that notion). In order to facilitate the first two steps, we will require platforms which:

1. strongly identify themselves — using public key cryptography, involving a secret key strongly tied to the platform itself, and
2. strongly identify their current configuration and running software — using cryptographic hashes of object code, and other mechanisms.

We may regard the platform identity as a cryptographic serial number, but mindful of privacy concerns raised when integrated circuit manufacturers introduced unique serial numbers for their CPUs, we must invest additional effort to protect this information from abuse (see below). For a 'strong tie' to the platform, we hope for solder and some form of tamper-evidence, at least. The strength needs, of course, to be commensurate with the risk entailed: locked cases work well in some environments, circuit boards encased in resin, in others.

In order to make use of cryptographic identity, we shall need some hardware support akin to that sometimes offered by a cryptographic co-processor (or a smart card): namely, the ability to sign data with one or more keys, without any danger that that key value can be extracted and re-used elsewhere.

Discovering, recording, and reporting the configuration of the platform might entail careful 'measurement' of all those elements — firmware and software: BIOS, 'option ROMs', loaders, configurations, kernels, libraries, applications — which contribute to the operating state of the platform, and hence the trustworthiness of the computations it performs and the extent to which it enforces a desired security policy. In this context, we use the term 'measurement' in a specific but possibly unusual manner: it means the action of making a cryptographic hash of the item to be measured. In this way, we reduce every one of the components mentioned above to a single, effectively unique representative value.[5]

3.1 Trusted Platform Module

The foregoing discussion implies a need for a change — a relatively minor one — to the architecture of the PC. A Trusted Platform Module (TPM) has been defined (as a logical bundle of functionality), together with a means to embed it in the PC architecture (for the time being, as a discrete chip, residing on the 'LPC' bus; but this may change in future). The functional behaviour of the TPM can be implemented wholly in software, but some of its behaviours — such as the strong protection of a platform-specific unique secret key — require protections which can be achieved only through a hardware device.

We might go so far as to say that for these particular pieces of functionality, we require a device which is *not* like a Turing machine: a general read/write capability for storage locations gives an attacker too much scope. In particular, we require *protected storage* (some of it non-volatile) which is accessible through certain special interfaces and not subject to the normal read-write protocols of a register or memory location. Such capabilities are inherent in the design of many smartcards, and it may be helpful to consider the TPM as like a smartcard soldered to the PC's motherboard. By combining the small quantity of non-volatile storage in the TPM with the ability to store *wrapped* keys (i.e. keys encrypted under keys available only within the TPM) we may build a hierarchy of protected storage of arbitrary size.

The TPM, with its embedding in the PC platform, is then intended to provide three *roots of trust*:

Root of trust for measurement (RTM) a trusted implementation of a hash algorithm, responsible for the first measurement on the platform — whether at boot time, or in order to put the platform into a special, trusted state;

Root of trust for storage (RTS) a trusted implementation of a *shielded location* for one or more secret keys — probably just one, the *storage root key* (SRK);

Root of trust for reporting (RTR) a trusted implementation of *shielded location* to hold a secret key representing a unique platform identity, the *endorsement key*, (EK).

These are described as roots of trust because they are each inherently indivisible: all subsequent trust decisions are based upon these; they are the basis of an inductive chain.

The SRK and EK use asymmetric cryptography: the TPM's role is to protect the secret part of the key pair. The public part of the EK should be signed into a certificate by the manufacturer, and in most uses the key pair will remain fixed throughout the lifetime of the platform. The SRK, on the other hand, is established when someone 'takes ownership' of the platform, and may be re-initialised (loosing all the secrets it protects) when the platform passes to a new owner.

The RTM and RTR are the essential elements for the establishment of the 'steps to trust' described above, in relation to a third party. Via a process called 'sealing' (see below), the RTS helps to establish trust in the local platform. The RTS will also turn out to be valuable for

many inquiries, the author has been unable to find the document to which that reference refers.

[5] Such measurements are too fragile for many situations: two sequences of assembly instructions may produce identical behaviours on all inputs and in all contexts, yet hash to different values. We will return to this problem later.

the protection of privacy, as well as for building compartmentalized/isolated storage for mutually untrusting applications or users.

We can, then, describe a process of building a *chain of trust* forward from these roots. One approach to this is to thread the process of measurement alongside the platform boot process, to give a *measured boot* process. If we subsequently check those measurements, we shall have achieved an *authenticated boot* (though that checking requires care, lest the checker itself be under the control of an adversary). A measured boot proceeds as follows:

1. At power on (platform reset), the RTM records an indication of its own identity in a safe place — this might be a measurement of its own code, or simply an identifier.

2. Before dispatching the next item in the boot chain the RTM takes a hash of the next component to execute, and stores this value in a safe place. It then transfers control to that component.

3. Repeat step 2 for each link in the chain.

Then, any program, at any point in the chain, can gain confidence in (i) its own integrity, and (ii) the integrity of the components it relies upon, by comparing the stored values with the ones it was expecting. From a trust perspective, it is ideal to have an RTM which is immutable — burned into silicon once and for all.

The initial design for a trusted PC platform uses a so-called 'static' root of trust for measurement: the process described above whereby the entire boot chain is to be measured. This is problematic for several reasons: the number of elements in the boot chain is actually very large, which makes maintaining expected values for measurements relatively onerous; those elements are subject to frequent patching and updating (at the upper levels, we must include the boot loader configuration, say, and the operating system kernel, and some higher level libraries, and so on), so determining 'correct' values for them is non-trivial; also there are good reasons why some of the elements in the chain may be executed in different orders, which (because of the way we store the measurements) gives rise to different measurement values; it also introduces a performance hit (as each step in the chain requires storing away the measurement value in that 'safe place', which is not, for now, high-speed memory).

An alternative has been implemented by the major chip vendors, allowing a 'dynamic' root of trust; the 'late launch' of a measured environment (Grawrock, 2008; Strongin, 2005). Here, a special CPU instruction invokes a major change of platform state. A fixed piece of code is loaded from a trustworthy source which is able to measure and launch a nominated (white-listed) piece of software (recording the measurement, as before). This is, then, a very short chain of trust, and by having it launched by a piece of inherently trustworthy code (untainted by anything else which has happened since the platform was rebooted), the platform can jump directly into a trustworthy state — a 'measured launched environment'. The expected application of this capability is to launch a virtual machine monitor (see below), but it might also launch some special, short-lived program which requires a high degree of trust.

The 'safe place' referred to above is a collection of 'Platform Configuration Registers' within the TPM. These are special shielded locations: they can be directly read, but cannot be written with arbitrary values. Instead the only available update operation (beside 'reset', which for a static root of trust must coincide with a platform reset), is an operation called 'extend', which takes one input. It writes into the nominated PCR a cryptorgraphic hash of the combination of the input value and the previous value of that PCR. In this way, a single register can record a sequence of values — and a separate, unprotected storage area can list those values and their sources, if we wish. The PCR can be put into a particular state only by a particular sequence of *extend* operations — and so if a 'good' value is found there, we have the confidence that we need.

This process is not sufficient by itself: how is any high-level piece of software to know that the PCR values it sees are indeed those stored in the TPM, and not a fiction created by some rogue driver or other intervention? Two mechanisms help to achieve this:

1. The operation 'TPM quote' returns a (nonce-challenge protected) signed copy of nominated PCRs, the signature key protected by the TPM, and the calculation of the signature constructed entirely within the TPM. So any party able to verify the signature can have confidence in the 'current' state of the TPM (and hence the platform, if the elements in the chain of trust are relied upon to store trustworthy measurements there). Proving in this way to a third party that the platform is in a particular state is known as '*remote attestation*'.

2. Using asymmetric cryptography, we have the possibility of encrypting a block of data, and marking it for the TPM to say that its contents should be decrypted only when the PCRs match particular nominated values. This is described as '*sealing*' data to a particular platform state. We would, of course, expect the TPM to be managing the secret key used for the decryption, ensuring that it is not available for arbitrary cryptographic operations.

Both of these mechanisms use the protected storage hierarchy established from the RTS mentioned above. The TPM has of course limited storage space for keys: the storage root key is held in non-volatile memory; other keys are cached and decrypted within the TPM as needed. In this way, a tree of arbitrary size containing protected keys can be constructed.

Many applications are based upon keys which are trusted to reside at only one platform. Legitimate uses arise for the migration of keys from one platform to

another: for group work, or the passing of privilege, and also for maintenance and upgrade. Support for migration of keys from one TPM to another is arranged through suitably-protected protocols. Some keys can be marked as being prohibited from being migrated, or only to be migrated under a special backup procedure, designed to re-insert data into a new platform when a TPM device fails.

Because the TPM is participating in high-grade cryptography, it also implements key generation capabilities, based in turn upon a true random number generator. It also incorporates non-volatile monotonic counters, which can serve as a surrogate for a real time clock in many signature contexts, and can help to protect against other 'replay' style attacks. The embedding in the PC platform must pay careful attention to the hibernate/suspend/resume behaviours, and their potential impacts upon the availability and trustworthiness of measurements stored in the PCRs.

The TPM assists in the construction of a trusted *platform*. By enabling, in addition, key migration and *remote attestation*, we begin to be able to federate that trust; to support *trusted infrastructure*.

3.2 Privacy

The process of *remote attestation* is based upon the root of trust for reporting: this is implemented as the Endorsement Key (EK). The credentials accompanying the EK are the proof for a third party that the platform they are interacting with is indeed a trusted platform. We would expect to find credentials from the TPM manufacturer (to say that it has implemented the specification), the platform manufacturer (to say the it has followed the specification in the way that the TPM is embedded in the platform), third party evaluators, local IT services, and so on. There are pure software implementations of TPMs available, of course — and then most of those credentials will be missing, or limited in their assertions.

This is a crucial feature of the design: there is no master secret here, nor licensing authority. It is for the relying party to decide which credentials are acceptable, and so which platforms, which manufacturers, etc., are are trustworthy.

However, if the EK were simply used to sign each PCR quote operation, it would be a trivial matter to track all the remote interactions of a particular platform, and feasible for the platform manufacturer to tie those interactions to a named customer. This is unacceptable for many privacy-sensitive activities. Therefore, the TPM's design allows for a level of indirection in the creation of arbitrarily many *attestation identity key*s (AIKs) — and, indeed, prevents the EK from being used as a signature key.

In order to establish an AIK, the TPM generates a key pair, and then runs a protocol with a Certification Authority (a 'Privacy CA'). The EK is used to demonstrate that this is a *bona fide* trusted platform, but then its details do not need to be transcribed into the AIK credential: the AIK certificate binds a key to a trusted platform, but does not report to third parties *which* trusted platform this is. The user or application may, then, generate fresh AIKs as often as required — per application, per session, per protocol, etc.

It has been objected that the Privacy CA retains power because it is capable of associating a particular AIK with a particular platform. (The platform/owner is at liberty to select a different Privacy CA for each AIK, but that does not negate this point.) For those with stronger privacy requirements, an elaborate protocol called 'direct anonymous attestation' (DAA) (Brickell, Camenish and Chen, 2005), based upon advanced cryptography, is defined (but optional in implementation).

4 Other platforms

4.1 Mobile phone

The mobile platform is instructive because it comes with much stronger trust requirements than the PC. The radio frequency component is subject to substantial regulation: it is not acceptable to allow arbitrary software to change its operating parameters. Airtime is chargeable, and so must be unambiguously linked to the subscriber's identity. Moreover, the platform is subject to a much more complex pattern of ownership. The market has developed in such a way that end users are often given subsidized phones, either in a hire purchase agreement, or within some other contractual arrangement: the operator retains an interest, then, in the selection of network and services. Finally, mobile phones are far more subject to casual theft than full-sized computers, and a key to preventing this is seen as having the phone retain a fixed, unchangeable (and hence, blacklistable) identity reported to the network.

These platforms, then, have more interested stakeholders than PC platforms, and need to broker multiple trust relationships. From the perspective of the end user, they also wish to run arbitrary applications, in much the same way as a PC platform owner does.

In order to allow such patterns of trusted use, the designers of mobile phone platforms are in many ways ahead of the PC platform designs. There are widely adopted products in the market already (such as ARM TrustZone (Alves and Felton, 2004)) which provide trusted execution environments, whereby the RF and network stack can be strongly partitioned away from user space, and boot known good configurations.

There are also specifications for a *mobile trusted module* (MTM), which draws upon the TPM described above. Using the pre-existing trusted execution environments, it may optionally be implemented entirely in software — though an MTM chip may also emerge. The architecture allows for two separate suites of functionality: a 'remote owner' MTM (MTRM) and a 'local owner' MTM (MTLM). The latter is intended to offer TPM-like capabilities to user-level applications, and

so is virtually indistinguishable from the current TPM specification (for example, it allows the user's applications multiple pseudonymous identities via AIKs).

The MTRM requires a more constrained set of functionality: identity must be fixed, for example, since a design goal is to avoid having the phone masquerade as a different device. A small set of AIKs can therefore be fixed before delivery, and an EK may not even be needed. This aspect of the phone also requires 'secure boot' rather than the previously-described authenticated boot: if erroneous network software is detected, the boot must be aborted rather than allowed to proceed (else we run the risk of breaking regulatory compliance, and indeed becoming a local network jamming device). Secure boot requires a means to test acceptable code hashes (reference integrity metrics, 'RIM's), and so the MTRM needs recourse to two further roots of trust. A root of trust for verification (of RIM values; RTV) is the means by which the RIM values are reliably checked. A root of trust for enforcement (RTE) enables the construction of trustable components where dedicated devices do not exist. Again, we must build these in such a way as to make them tamper-resistant, but we have no independent way to check their validity in the field — hence their role as roots.

4.2 Virtualized platforms

For a range of reasons, whole system virtualization is becoming a popular way to manage server environments, and increasingly desktop systems too. Running virtual machines can be migrated and copied, and allow for both good utilization of resources and also strong isolation among the contents of the virtualized containers.

The latter is a valuable security property. We often find it desirable to isolate software with different degrees of criticality, and different degress of trustworthiness. A game and an online banking application, for example, have little reason to interact with each other, so it is desirable to try to ensure that rogue software in the former cannot interfere with the correct operation of the latter.

Such isolation can be strengthened with the technologies described here. First, the 'late launch'/dynamic root of trust capabilities we described above can be used to launch a measured, trustworthy virtual machine monitor. Secondly, this VMM can measure a whole virtual machine before instantiating it. Taken together, these two ideas dramatically shorten the chain of trust: the DRTM launches the measured VMM; the measured VMM launches the measured VM. Of course, many applications require the VM to update its own state, and store this prior to shutdown. Many, however, require the VM to be brought up in the same state each time it is launched (or will only ever launch it once): for these, this approach is perfect.

In principle this approach can also help to limit the impact a platform administrator may have upon the data contained in a virtual machine: on some emerging designs, if the VMM is capable only of starting and stopping VMs, and allocating resources to them, then from outside the VM, there is no possibility of access to its internal state. That is, the contained VM may have access to secrets (confidential data, signing keys) not accessible to any other party using the (physical) platform.

This approach causes considerable complexity to the use of the TPM, however. The virtual machine will reasonably expect to address a (virtual) TPM: the guarantees on offer will depend upon how this is bound to the (physical) TPM. This problem becomes particularly complex when virtual machines are permitted to migrate from one physical host to another.

5 Impact of Trusted Computing

The approaches described above have the potential to alter radically the design both of end-user desktop systems, and distributed applications. It is certainly novel to have a strong guarantee of what software is running — locally and remotely — coupled with the necessary underlying cryptography to incorporate that guarantee into strong signatures.

Whether they will have such an impact or not depends of course on many factors, not least the commercial issues of whether there is demand for systems refactored into a number of relatively constrained components, communicating over well-defined interfaces, and having determinable trust characteristics. Across the broad sweep of computing technologies, such decompositions have been advocated for many different reasons, with mixed success.

The stated goal of the trusted platform approach is to prevent all *software-based* attacks — that is, there should be no way that the trusted platform can be compromised simply through participating in network protocols, say. Certain critical operations can be invoked only with the assertion of 'physical presence': the TPM can be configured, for example, so that physical presence is required to turn it on. The means by which physical presence is asserted is up to the platform designer, but could be accomplished by holding down a button, or through a BIOS set-up screen which is available only in a pre-boot environment.[6] No formal verification is on offer to demonstrate the achievement of the goal of defeating all software-based attacks, but the prospects seem good. Moreover, the absence of software attacks broadly implies a lack of *class* attacks: if one TPM is compromised through an expensive, equipment-intensive intervention, compromising subsequent TPMs will require the same amount of effort. Likewise, there are no 'global secrets' to be compromised: manufacturers' signing keys for TPM and

[6] Of course, many servers are configured to offer all such functions over a controlled network, or are connected to third party devices which achieve this, since in large server farm, it would be infeasible to configure each box by standing in front of it. Such concerns impact the design of a Server profile for Trusted Platforms.

platform endorsements are perhaps the most valuable, and these can be managed quite effectively in a PKI with revokation mechanisms. In order to mitigate the compromise of any endorsement key, relying parties (i.e. Privacy CAs) must be able to reject AIK requests based upon 'known bad' EKs.

Of course, there remains a gap between the low-level protections described in the foregoing account, and the architectures required by current network-based services in the cloud, and so on — see 'Active Research' below. The fragility of remote attestation is one of the biggest challenges. However, some further building-blocks which begin to raise the level of abstraction are described in Sections 5.2 and 5.3 below.

A number of authors have asked whether the approach taken here addresses the real security problems encountered today (Oppliger and Rytz, 2005; Arbaugh, 2002; Marchesini, Smith, Wild and MacDonald, 2003; Vaughan-Nichols, 2003). Although the trusted platform ideas have largely arisen in the context of the PC platform, it may be in embedded devices and environments that they find their best application: the mobile phone is a primary example. Environments like automotive and avionic systems are increasingly based around simple bus-based and network communications, where the correct (or trusted) operation of each component is critical. As more-controlled environments than a general-purpose PC, these have much better prospects of being able to manage configurations sufficiently well as to make attestation feasible.

5.1 Counter-arguments

Trusted Computing has attracted much criticism, not least from the open source community (Stallman, 2002). Some of the major themes of discussion are mentioned below. A bullish defence of Trusted Computing has been written by Safford (2002).

Privacy We have discussed privacy in Section 3.2. Those technologies seem to have addressed most of the stated concerns — although some authors seem yet to realise this. Of course, the question of whether vendors will choose to implement schemes such as DAA is a different matter entirely.

Vendor control The capability of the TPM to 'seal' data to a particular platform configuration raises the possibility that a software vendor could lock content (third-party content or user-generated content) to a particular application. This would give rise to *much* stronger 'vendor lock-in' than proprietary data formats ever have. An operating system vendor could exercise control over which applications could be run on that platform — as, indeed, several mobile phone operating systems have done. An application vendor could prevent migration to a competitor's product.

Anderson (2003) argues that such approaches have the potential to alter profoundly the economics of software production — probably to the detriment of the consumer and the stagnation of the market. It must be admitted that this is a possibility: it is by no means an inevitable consequence of trusted computing, however, and any such locked solutions would need to convince an increasingly discerning market that they offered, on balance, worthwhile benefits (as purchasers of Apple's iPhone, for example, appear to believe).

Digital Rights Management A related concern is that Trusted Computing is entirely designed to enforce 'Digital Rights Management': a trusted execution environment is certainly a good candidate location for the critical, key-handling parts of DRM. We have noted that the TCG design for trusted platforms is designed to defeat all software-based attacks. It is inherently weaker at protecting against hardware-based attacks.[7] Another way to view this is to say that the design is very good for convincing the platform owner that their own platform is in an untainted state (since they have not taken the cover off and interfered with it): it can never be *as* good at proving to some party across the network that a platform is untainted, since, with hardware access and enough resources the protections *can* be defeated.

It remains to be seen how the risk profile for such protections may play out: it is crucial to understand the ratio of the value of the data being protected to the cost of defeating the rights management regime. In the context of digital mass media (music, movies, games), the rewards associated with being able to unlock protected content may be very great, so it may be cost-effective to produce compromised hosts to receive licenced content and strip off the protections. Some other data (patient records, say, or medium-grade corporate secrets) may have a higher unit value, but a much smaller market, and a much smaller exploitation opportunity, and the degree of cost-effectiveness may differ.

Attestation The approach we have described to remote attestation is exceedingly fragile. Hundreds of components contribute to the high-level operating state of a PC platform. Subtle timing issues affect the order in which they are executed — and so the order in which they are measured and extended into PCRs. Some configuration components may be critical, and yet updated at each boot. The system, its drivers and applications, may be subject to patching on a regular basis: there is an immense combinatorial explosion of possible acceptable configurations (well as an enormous collection of known bad ones).

The question of what, then, to attest, and how to attest the *semantics* of software — its behaviour, rather than

[7] That is not to say that it is 'weak'. The TPM is intended to be sufficiently strong that a well-equipped physics lab is needed to defeat its protections. The intention of a trusted PC platform is that 'corner-shop' attacks should fail, although version 1.1 of the TPM has been subject to a few of these, prompting some careful re-evaluation of version 1.2 — and ongoing consideration of the best location for this functionality within the PC architecture. The absence of class attacks is intended to keep the costs high for *each TPM attack*.

its binary signature — is a topic for active research. The 'late launch' capabilities we described above, with a dynamic root of trust, help to simplify some parts of the attestation challenge, but their main design goal is to facilitate the use of whole system virtualization, which adds complexities all of its own.

Symmetric Cryptography The TPM design discussed above, despite implementing many elements of a cryptographic co-processor, exposes *no* symmetric (bulk) cryptographic functions. This is necessary, in order to satisfy many countries' import/export controls, but relates also to the issue below.

Law Enforcement Trusted Computing is part of a big shift towards greater use of encryption for everyday secrets. This has significant benefits for the individual and corporate user, especially in an ever-increasingly hostile network environment; indeed, it seems unavoidable. Used well, however, the technologies of encryption are exceptionally strong, and this benefits criminal and terrorist conspiracy, as well as legitimate users. When this was anticipated in the 1990s, western governments contemplated key escrow for a while, before rejecting it as unworkable or unduly injurious to privacy and/or the development of commercial solutions. Finding an acceptable balance between law enforcement, practicality, privacy, and innovation continues to be the topic of much debate, and many would expect responsible systems designers to take account of the legitimate needs of law enforcement when designing architectures. It is worth noting that the TPM specification intentionally *requires* that there be no 'back door' built into compliant devices (Trusted Computing Group, 2005, Section 3).

5.2 Trusted Network Connect

With more and more mobile and portable devices seeking intermittent access to networks, the problem of *network access control* is increasingly acute. Connection to a particular network segment often conveys — or helps to convey — access privilege for local resources (or, for example, in the case of publishers locking access to IP address ranges, to licensed remote resources). Moreover, a rogue device may introduce malware to the local network, or implicate the network owner in criminal activity elsewhere.

Trusted Network Connect (TNC) (TCG, 2008) is an architecture and suite of protocols to facilitate policy-based network access control. The variety of those policies is in the hands of the system designer and network manager. Of interest for our present purposes, an option exists to allow that policy to refer to the *attested state* of the platform seeking a connection.

In the first instance this allows a strong assertion of platform identity — something missing from most other connection protocols, since the MAC address of most ethernet cards, for example, is software-settable. Moreover, it allows the network owner to specify that connecting clients must run an approved operating system, with specified patches, approved anti-virus, personal firewall, and so on, with good and up-to-date configurations. In other settings, a supplicant could mis-represent its state in order to gain access. Requiring an attested platform state removes this possibility.

The network policy enforcement point (the switch) has the capability to route traffic from the supplicant onto its main, unrestricted network, or onto a special remediation network where it can obtain the necessary patches, anti-virus signatures, and so on, necessary to meet the policy requirements for connection to the main network.

5.3 Trusted Storage

There are many products on the market which will encrypt one's storage devices (whether discs, tapes, memory sticks, etc.), and increasingly many application domains are seeing a need for this. The approach outlined above can strengthen such approaches quite substantially, and are increasingly doing so. We explore two major lines of investigation.

Operating System-based Encryption One approach is to implement drivers and supporting paraphernalia to enable all data to be encrypted as it is written to disc. The novelty that a Trusted Platform can bring to this approach is that the key for encrypting the drive can be stored using the TPM, and sealed so that it is released only when the legitimate platform configuration is seen. This is intended to mean that a stolen disc (or a disc extracted from a laptop) cannot be decrypted, because the correct context to do so will not exist at the attacker's system. The stolen laptop will still boot, but the attacker needs login credentials in order to access the disc's contents.[8] This is the approach taken by Microsoft's *Bitlocker*.

Encrypting Disc Drives An alternative is to build a bulk encryption capability into the firmware of the disc drive. Data may travel to the drive unencrypted on the host's bus, but will be encrypted on the drive itself. The latter would be of no benefit if the drive could be attached to any host of an attacker's choosing, so the drive runs a protocol with the host (and in particular, the host's TPM): a key protecting the encryption on the device is released for use in the drive only when it is connected to a host with which it has been registered. A lost/stolen drive will be of no value, even if the chassis is disassembled and the platters used in a different context.

Both of these approaches offer an added benefit that when the disc is to be decommissioned (at end of life,

[8] Attacks against login credentials usually entail extraction of the 'password' file from an un-booted disc, or its replacement prior to boot. Such attacks fail here.

or upon passing to a new owner), in order to prevent its secrets being divulged, it suffices to destroy the disc's encryption key: no elaborate erasure or reformatting process is needed.

6 State of the Art

As of late 2008, more than 200m TPM chips have been shipped in laptops, worldwide. It is estimated that a large majority of all new 'business class' laptops are now equipped with this device, and a large proportion of those sold in the last three years.[9]

Windows Vista offers limited support for the TPM: those instances of Vista which incorporate Bitlocker will at least populate the PCR values during the boot process. A few OEM-specific tools use the TPM to provide a strong storage location for passwords and keys. Linux drivers for most TPM devices are available, together with a kernel patch which supports measured boot, a 'Trusted GRUB' boot-loader, capable of establishing the PCR values in the chain of trust, and a DRTM-enabled boot-loader OSLO (Kauer, 2007).

It must be remarked that some estimates suggest that as few as 5% of TPMs are turned on: the major vendors' investment over a ten year period in developing these technologies has yet to pay off.

Fully encrypting disc drives are now on the market, and are expected to dominate within a few years. Some interoperable TNC implementations have been developed, including an Open Source version from THH Hanover.

7 Active Research

Although we have remarked that this paradigm of trusted computing is at its strongest when assuring local owners (or users) of the state of their platforms, it is notable that much of the active research in this area relates to the use of remote attestation in one form or another.

Perhaps there are two reasons for this: one, the integrity of the local platform is largely a matter for the operating system. This, in turn, is a relatively specialist area, not particularly amenable to those who are not deeply involved in it. Secondly, as we have remarked, attestation is a much more novel concept than integrity, and so as a new primitive gives rise to a range of genuinely innovative lines of inquiry.

A project which aims to address the operating system integration issues (in the open source arena) and thereby to facilitate remote attestation based upon trusted virtualization, is the European project OpenTC[10]. In doing so, it is developing prototype implementations: for example its first demonstration was for a home banking scenario: a specialized virtual machine was launched to host the home banking client. This VM would be attested by the bank (and the bank by the VM) to prevent not merely impersonation attacks (which mutual authentication would solve) but also invidious interference in the network stack or keyboard driver (Kuhlmann, Lo Presti, Ramunno, Vernizzi, Bayer, Katrcolu and Gngren, 2008).

A similar scenario involves building a virtual machine to sit at the client end of a VPN. Most VPN solutions are intended to prevent mis-behaving clients from compromising the server, but this is hard to ensure. An attested client, perhaps fully isolated from the rest of the host system, is potentially much better able to enforce the security policy required by the server.

A more complex realization of this kind of thinking comes in the High Assurance Platform (HAP) Programme[11]. This is aiming to build, using trusted virtualization, a secure client execution environment, capable of participating in multiple isolated virtual security domains. A goal — which the government and industry partners appear to believe is achievable — is to enable operation to be sufficiently partitioned as to able systems safely to process material classified 'Top Secret' on a host also connected to the Internet. This has been an aspiration of high-assurance systems for decades.

Our own aims have been at a rather more modest level of assurance. We have seen the technologies described here as ideal for implementing something we might call a 'trusted grid', or, in the broader sense of service orientation, trusted services. Grid computing implies having work done on a possibly-distributed collection of hosts, outside one's direct control. The full abstraction of the grid would mean *not knowing* where one's job (or service) might be executed. In such a setting, questions of data and code confidentiality, and of results integrity become difficult.

The author came to this problem whilst considering the security properties for climate*prediction*.net. This system, in the style of `SETI@home`, distributed hundreds of thousands of climate modelling tasks to users around the world. It was under a duty to protect the climate model code, but also, of course, needed to offer reassurance that the data collected really arose from genuine climate model runs. Although there are more actors involved in such a system than in the kind of grid constructed from supercomputers and clusters, the same kind of questions arise.

We have explored the requirements (Martin and Yau, 2007), and the architecture necessary to isolate and attest grid jobs (Cooper and Martin, 2006a), and to enforce access control policies on data as it travels within the grid (Cooper and Martin, 2006b). Finally, as an example of how to undertake privileged processing of sensitive data, we are investigating how to undertake logging in a trustworthy manner in such distributed settings (Huh and Martin, 2008). Some of these ideas are presently being prototyped.

[9] TCG Marketing WG anticipates TPMs in over 70% of laptops in 2008; almost 90% in 2009.

[10] `www.opentc.net`

[11] `http://www.nsa.gov/ia/industry/HAP/HAP.cfm?MenuID=10.2.1.6`

8 Closing remarks

For the reasons discussed above, the greatest strength of trusted computing is in providing components which can be used to give assurances to desktop users about the integrity of their operating environments. This may be achieved through providing extra capabilities for the local operating system and applications to test whether they are running in untainted configurations. The idea of factorizing a computing platform into a 'trusted computing base' and the rest is far from new. Now, there is growing evidence that we need this, not just in the high assurance domains where the TCB concept has been widespread, but in every-day end-user systems also.

Concerns of security and trying to create a trusted execution environment have re-invigorated interest in operating system designs. We may not yet be able to design out the all-powerful administrators, but trusted components can help to reduce the extent to which they must be wise, and help to show them up when they are not good.

Security properties are famously non-compositional, in general. And in common speech, trust is not transitive. But components like TPMs, trusted virtualization, and trusted storage devices do have the capacity to weaken the cross-linkages among components, to allow meaningful and testable chains of trust to be created, and thereby to remove some of the present fragility in operating systems, where a single exploited vulnerability leads to the whole platform being 'owned' by an attacker.

Remote attestation is genuinely a new security primitive. The ability to gain a medium-strong guarantee of the software running on a different host is novel and has many potential applications. Hardware attacks may be theoretically feasible, but the number of contexts in which they will prove cost-effective seems small.

As we saw in Section 6, the approaches described in this document are far from inevitable at present: the massive installed base of TPMs may turn out to be a proverbial white elephant. This would seem a shame to the author: security technologies tend to be thought of as *preventing* actions; there are novel concepts and ideas here which have the potential to *enable* many new patterns of interaction: this is a much more compelling concept.

Acknowledgements

The author's understanding of these topics has been aided by discussion with many people, not least Graeme Proudler, David Grawrock, Andy Cooper, Jun Ho Huh, John Lyle, and Joe Loughry, together with a number of excellent books on this subject. Review comments from Luc Moreau also helped to focus the text. This tutorial was partly written with the support of the eScience Institute's *Theme* on *Trust and Security in Virtual Communities*. It forms a basis for a number of courses and introductory lectures on this topic.

Thank you also to Robert and Anna Booy for providing a peaceful cottage in the Blue Mountains in which to write without distractions.

Bitlocker is a trademark of the Microsoft corporation.

References

Alves, T. and Felton, D. (2004). TrustZone: integrated hardware and software security, *White paper*, ARM.

Anderson, R. (2003). Cryptography and competition policy: Issues with trusted computing., *Proceedings of the Workshop on Economics and Information Security*.

Arbaugh, B. (2002). Improving the TCPA specification, *IEEE Computer* **35**(8): 77–79.

Brickell, E., Camenish, J. and Chen, L. (2005). The DAA scheme in context, *in* Mitchell (2005), chapter 5.

Cooper, A. and Martin, A. (2006a). Towards a secure, tamper-proof grid platform., *CCGRID*, IEEE Computer Society, pp. 373–380.

Cooper, A. and Martin, A. (2006b). Towards an open, trusted digital rights management platform, *DRM '06: Proceedings of the ACM workshop on Digital rights management*, ACM Press, New York, NY, USA, pp. 79–88.

DoD (1985). Department of Defense Trusted Computer System Evaluation Criteria, *DoD Standard 5200.28-STD*, DoD.

Gollman, D. (2004). Why trust is bad for security. *http://www.sics.se/policy2005/Policy_Pres1/dg-policy-trust.ppt*

Grawrock, D. (2008). *Dynamics of a Trusted Platform: A building block approach*, Intel Press.

Heasman, J. (2006). Implementing and detecting an acpi bios rootkit, presentation. *https://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Heasman.pdf*

Huh, J. H. and Martin, A. (2008). Trusted logging for grid computing, *3rd Asia-Pacific Trusted Infrastructure Technologies Conference*, China.

Kauer, B. (2007). Oslo: Improving the security of trusted computing, *Proceedings of the 16th USENIX Security Symposium*, Boston, Mass., USA. *http://os.inf.tu-dresden.de/papers_ps/kauer07-oslo.pdf*

Kuhlmann, D., Lo Presti, S., Ramunno, G., Vernizzi, D., Bayer, E., Katrcolu, M. A. and Gngren, B. (2008). Private electronic transaction (pet) proof-of-concept prototype documentation, *Deliverable 10c.3*, Open Trusted Computing Project.

Marchesini, J., Smith, S., Wild, O. and MacDonald, R. (2003). Experimenting with TCPA/TCG hardware, or: How I learned to stop worrying and love the bear, *Technical Report TR2003-476*, Department of Computer Science, Dartmouth College, Hanover, New Hampshire.

Martin, A. and Yau, P.-W. (2007). Grid security: next steps, *Information Security Technical Report* **12**(3): 113–122.

Mitchell, C. (ed.) (2005). *Trusted Computing*, The Institution of Electrical Engineers, London.

Oppliger, R. and Rytz, R. (2005). Does trusted computing remedy computer security problems?, *IEEE Security and Privacy* **3**(2): 16–19.

Safford, D. (2002). Clarifying misinformation on TCPA. *http://www.research.ibm.com/gsal/tcpa/tcpa_rebuttal.pdf*

Stallman, R. M. (2002). Can you trust your computer?, *in* J. Gay (ed.), *Free Software, Free Society: Selected Essays of Richard M. Stallman*, GNU Press, chapter 17, pp. 117–120. *http://www.gnu.org/philosophy/fsfs/rms-essays.pdf*

Strongin, G. (2005). Trusted computing using AMD 'Pacifica' and 'Presidio' secure virtual machine technology, *Information Security Technical Report* **10**(2): 120–132.

TCG (2008). TCG TNC architecture for interoperability, *Specification Version 1.3*, Trusted Computing Group. *https://www.trustedcomputinggroup.org/specs/TNC/TNC_Architecture_v1_3_r6.pdf*

Trusted Computing Group (2005). Tpm main: Part 1, design principles, TCG Specification Version 1.2, Level 2 Revision 85.

Vaughan-Nichols, S. J. (2003). How trustworthy is trusted computing?, *Computer* **36**(3): 18–20.