# Roadmap

<table>
<thead>
<tr>
<th>Defining</th>
<th>• trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describing</td>
<td>• Trusted Infrastructure</td>
</tr>
<tr>
<td>Detailing</td>
<td>• Trusted Platform Module</td>
</tr>
<tr>
<td>Enumerating</td>
<td>• Trusted Infrastructure Ecosystem Components</td>
</tr>
</tbody>
</table>
Basic Questions

Can I believe what I read?

What happens to the things I type?

Who else has seen this?

Are these really my results?
More questions

- Who built it?
- Who delivered it?
- Who installed it?
- Who manages it?
- Who has tampered with it?
- Is it patched?
- Is there a virus? a rootkit?
Protections and Attacks

Hardware
- physical access
- passwords, jumpers
- CPU rings
- cross-domain authentication
- process and account separation

Firmware
- BIOS re-flash
- power analysis/bus probing

OS
- user accounts
- cross-domain authentication
- CPU rings
- user accounts,
- cross-domain authentication
- process and account separation

Middleware
- key logging
- boot-sector virus
- password brute-force
- phishing

Application Software
- cross-site scripting
- macro virus/trojan
- key logging
- boot-sector virus
- password brute-force
Who protects us?

Traditional role of operating system:
- process isolation
- privilege management; authorization

but may be tainted by:
- malware
- system managers
- service providers

and in a distributed context, how will you know?
Add-ons

- anti-virus
- anti-spyware
- firewalls
- token-based/two-factor authentication
- digital certificates
- code-signing
- sandboxing, managed code
- virtual private networks
- ...
Can they be effective?

mostly reactive:
- no protection against unknown vulnerabilities, zero-day attacks.

subject to the same controls as the software they protect:
- can be switched off by software
- no optional (transparent) circumvention

So we seek security measures which are subject to managed enforcement

For general-purpose commodity computing, this is a significant change to systems architecture.
Correctness

| correctness is relative to a specification | there is no way to tell whether a remote system is correct (and it may be correct malware) | the number of real-world pieces of correct software is more-or-less zero |

If every piece of software were perfectly *correct*, would that solve all our security problems?
Trust

Correctness is also an abstraction

- good for comprehensibility, modelling, and analysis
- problematic in security
  - the attacker doesn’t have to use your abstractions

So we need an additional notion of fitness for purpose.

- **Trust** is the subject of stacks of literature, in computing, in psychology, and so on.
  - Dieter Gollman: *Why Trust is Bad for Security*
  - Ken Thompson: *Reflections on Trusting Trust*
  - US DoD: *Trustworthy Systems Evaluation Criteria*
Trusted Systems

- Trusted systems are those upon whose correct (or predictable) operation we simply rely.
- If they fail to live up to our expectations, bad consequences will follow.
- Careful speakers distinguish
  - trusted systems
  - trustworthy systems
— we could have either without it being the other.
In practice, most of us trust most systems most of the time. *Degree or kind* of trust varies.

**Trusted Computing** is about

- giving us better grounds for trust; and
- reducing the scale and complexity of the things we need to trust.

Seek well-abstracted components with

- hard-to-break abstractions
- simple interfaces
- manageable behaviours
The Trusted Computing Base (TCB) of a computer system is “the totality of protection mechanisms within it, including hardware, firmware, and software, the combination of which is responsible for enforcing a computer security policy.” [Orange Book]

- why make it as small as possible?
- think of a system you use. What is the extent of its TCB?
TCB Components

- A Typical TCB:
  - may involve *trusted hardware* — military designs might use custom cryptographic devices
  - will probably involve a trusted operating system kernel
  - might involve *trusted paths* — a guarantee that your interactions are reaching the trusted kernel, and not some imposter
  - might involve *trusted output* etc.
Graeme Proudler says it is safe to trust something when:

- it can be unambiguously identified, and
- it operates unhindered, and
- the user has first-hand experience of consistent, good, behaviour.

or the user trusts someone who vouches for consistent, good, behaviour.
Questions

- How is trust related to correctness?
- How is trust related to security?
- Can we distinguish social trust from technical trust?
Words like ‘Trust’ get some people very excitable. Our objective here is not to explore the philosophy, sociology, or even psychology of trust, merely to use the term as a shorthand for a particular collection of reasonably well-defined technical concepts.
trust 1. (I) /information system/ A feeling of certainty (sometimes based on inconclusive evidence) either (a) that the system will not fail or (b) that the system meets its specifications (i.e., the system does what it claims to do and does not perform unwanted functions).

(See: trust level, trusted system, trustworthy system. Compare: assurance.)
trusted computer system

1. (I)
/information system/ A system that operates as expected, according to design and policy, doing what is required – despite environmental disruption, human user and operator errors, and attacks by hostile parties -- and not doing other things [NRC98]. (See: trust level, trusted process. Compare: trustworthy.)
trustworthy system  1. (I) A system that not only is trusted, but also warrants that trust because the system’s behavior can be validated in some convincing way, such as through formal analysis or code review. (See: trust. Compare: trusted.)
The Trusted Computing Group (TCG) defines specifications and standards in the area of Trusted Computing.

An entity can be trusted if it always behaves in the expected manner for the intended purpose.

(TCG 2004)
For us, then, Trusted Computing means an approach to building computer systems and infrastructure components which ...

- strongly identify themselves
- strongly identify their current configuration/running software
- allow us to make informed decisions about the level of trust to invest in them.

- platform identity will be based on public-key cryptography
- software identity will be based on cryptographic hashes of program object code
Some people now regret the name *Trusted Computing*:

| Trustworthy Computing could be a better title, |
| or maybe *Trustable Computing* |
| but it’s too late to change. |

Which is the better name?
Building Trust in a System

- We want to factorize the *trusted computing base*.

- We want to gain maximum benefit from the *hard-to-alter* characteristics of hardware.

- Focus for now on the identification and unhindered operation of systems.
Trustworthy state for a computing platform

- essence is to take a measurement (a *cryptographic hash*) of each component which contributes to the platform state
- use those measurements as the basis of deciding if the platform is in a state I trust

- firmware, kernel, library, application binary, configuration file, etc.
- the same state as last time it booted
- using components without known vulnerabilities
- etc.
Elements of a Trusted Infrastructure

- We also need:
  - (maybe) one or more parties to tell us which platforms have been implemented according to these principles (and, implicitly, which may be lying)
  - (maybe) one or more trusted parties to tell us which software measurements correspond to which versions of which software, and which are *good*.
Elements of a trusted infrastructure

- We may also need:
  - trusted system components and peripherals
    - with tightly constrained behaviours
    - and/or trustworthy implementation of selected security policies
  - networks of trusted hosts
    - where membership of the network is conditional upon trustworthiness
  - and this may be in a *virtualized* setting
Internet of Things

- In addition: more and more devices (and cars, books, clothing, sensors ...) are gaining network connectivity.
- Trusted networks of these things are clearly necessary.
- *Pervasive Trust* will be increasingly important.
An idealized objective

We could imagine a system with:

- trusted path from your fingers to
  - your screen, loudspeakers
  - service provider
  - services bootstrapped from your local roots of trust

- giving significant confidence about
  - integrity of data
  - confidentiality of input (e.g. passwords), and output (e.g. my banking data)
  - licence enforcement on remote hosts

Use hardware to hold secrets and enforce access control with much better guarantees than software can offer.
Reflection

Is this feasible?

Is it a worthwhile goal, or chasing a dream?

What are the obstacles to its full realization?

What potential unintended consequences might arise?
What kind of threats are we mitigating?

- *attacks against lower layers (root-kit, boot sector, BIOS virus)*
- *attacks based on theft of keys*
- *in consequence of these two, many impersonation attacks*
- *software-based attacks*
- *super-user/abuse of privilege attacks*
- *BORE (!)*
The threat landscape has evolved considerably since this kind of trusted platform was first proposed.

We should keep asking:

- Are we looking in the right place?
- Adopting these technologies will shift the attacks to the next-weakest spot: how well protected is it?

see: Rolf Oppliger and Ruedi Rytz, *Does Trusted Computing Remedy Computer Security Problems?*
Relative strengths

TC may be stronger for the local user because they might be able to see that their hardware has not been tampered with.

TC is weak for the remote user because they cannot see the state of the hardware.

TC is stronger for the remote system because it can run cryptographic protocols with the trusted platform.

TC is weaker for the local user because the interface (and evidence of trustworthiness) is part of the platform.
Impact

Is it good for enforcing DRM?

the aim being to ensure that the platform is in a policy-enforcing state, before releasing protected data to it

Is it good for giving the user a safer runtime environment?

the aim being to prevent credentials and personal data from being misappropriated
Sober judgement

- Making anything but the smallest delta on a commodity platform price is very hard to justify
  - Incorporating TPM (see below) was designed to be no more than a $5 delta on the price of a platform; these chips currently cost less than $1.
- How much security do you get for $5?
  - Compare with the (very high) costs of bespoke national-security-grade components.
- **But smart use of technology can magnify the investment immensely:**
  - compare with easily-implemented strong cryptography.
  - *Cost to use vs Cost to break* can be very favourable.
Early proposals for trusted computing platforms met many criticisms

- some were clearly ill-founded
- some have been addressed in current designs
  - widely acknowledged and accepted
- some may persist
  - accepting that many technologies have both ‘good’ and ‘bad’ uses

Criticisms deserve to be taken seriously:

- Ross Anderson
  - ‘Trusted Computing’ Frequently Asked Questions
- Richard Stallman
  - Can you Trust your Computer?
Who decides what to trust?
  - should be platform/system owner, not vendor, etc.
    - private individual or corporate IT
    - may delegate this decision

Highly desirable to have no master key; no global secrets
  - most of the designs we shall explore this week have this property

Platform identity is highly sensitive
  - see later discussion of privacy protections
Owner control

- Owner controls the platform
  - but clearly some control may be given up
- Of course, the owner’s hands may be tied by the options available
  - software exploiting trusted platforms is being developed in both commercial and open-source settings
- A good outcome/aim is for owner to be able to make an informed choice about trading flexibility for security
security based on ‘patch and mend’ is problematic

we define a narrow technical meaning of the word ‘trust’

trusted systems are not inherently secure

but untrusted systems are inherently insecure

hardware-rooted trust helps to build trustworthy systems
Operating systems have many roles
  - including scheduling, device management, user interface control, and so on
They are also the principal control point for security:
  - process isolation
  - privilege separation
    - (kernel mode or user mode; user accounts)
  - reference monitor
Consequence

- Large parts of (or most, or all) of the operating system fall within the Trusted Computing Base (TCB).
- The operating system is inherently trusted – it is a great big root of trust.

**pro:** we invest security effort in a single component

**con:** a single vulnerability can compromise every process on every host running that OS
• also called trust anchor: A component that must always behave in the expected manner, because its misbehaviour cannot be detected. The complete set of Roots of Trust has at least the minimum set of functions to enable a description of the platform characteristics that affect the trustworthiness of the platform. [TCG]

• Iterative means to extend the trust boundary from the root(s) of trust, in order to extend the collection of trustworthy functions. Implies/entails transitive trust. [TCG, paraphrased]
Examples

- **PKI:**
  - root/anchor is CA’s self-signed certificate;
  - trusted components are intermediate/relying certificates

- **embedded device:**
  - root is factory-installed firmware;
  - updates implement a chain of trust
A goal

- We want to achieve firmware-like trust characteristics in a software programmable system.
  - implement hardware-based roots
    - control secret keys
    - control platform identity
  - build chains of trust which indicate what software is running
    - report platform state reliably
Measuring the state

- The state of the platform may depend on every item of code which has run since it was last reset
  - firmware, option ROMs, loaders, kernel, libraries, applications
  - their behaviour may in turn depend on configuration files, user inputs, etc.
- How to make a record of these?
  - use a cryptographic hash
  - so, “measure this software” = “compute a hash”

TPM v1.2 detail
registers for storing hashes are 160-bits: designed to hold SHA-1 outputs
Trusted Computing Group
PC Architecture: Roots of Trust

Root of Trust for Measurement (RTM)
- initiates process of recording what software is running
- implement in BIOS: in an immutable or securely updatable component

Root of Trust for Storage (RTS)
- implements shielded locations: registers with special integrity or confidentiality characteristics
- implement in Trusted Platform Module (TPM): in hardware for tamper-proofing

Root of Trust for Reporting (RTR)
- using cryptography to give assurances to third parties
- built from keys burned into TPM at manufacture time
Building a record of platform state

- is my application untainted?
- is the environment in which it runs untainted?
- how to obtain measurements?

application software
middleware
OS
firmware
hardware
Building a record of platform state

- concept is to have each component in the chain be measured by the preceding one
Remarks

- this process gives us a *measured* boot process
  - any component in the chain can gain confidence about the components below/before it by querying the TPM
  - implies *transitive trust* in components
- considerable complexity
  - around 200 measurements in a typical boot to a general-purpose operating system
- **Question:** how could we convert this to a *trusted* (or *assured*) boot?
Role of TPM

Provides tamper-proof store for these measurements

- we can record additional measurements, but not overwrite old ones, nor undo their recording, without a platform reset

Needs to provide authenticated reports, too

- consider an application asking a TPM driver to report on the platform state
- driver could lie to application, unless TPM output is itself strongly authenticated
- how to communicate this to the desktop user?
Late Launch/Dynamic Root of Trust

- Building trust into the bootstrap process is hard and complex
- Intel and AMD have both developed (similar) alternative approaches instead.
- Involve a ‘late launch’ or ‘dynamic root of trust for measurement’ (DRTM):
  - use a special instruction to put the platform directly into a trusted state
  - intended to launch a trustworthy virtual machine monitor/hypervisor
Using virtualization

- Virtual machines (VMs) have the potential to simplify the measurement process
  - especially with a late-launched measured virtual machine monitor
  - measure the whole VM in one step (from the VMM), rather than each component as it loads
  - this approach is ideally-suited to VMs which are reset to the same state at each boot
    - special-purpose banking VM
    - remote working ‘terminal’ with VPN connection
    - personal firewall
    - etc.
Embedding

- Anything could **claim** to be a TPM, or a trusted platform
  - *root of trust for reporting* is intended to substantiate claims
- To trust the platform we need
  - assurance that it contains a correctly-implemented TPM
  - evidence that the embedding of that TPM within the platform conforms to an evaluated design
- Here is a role for
  - platform manufacturers
  - third-party accreditation
Credentials

- We might hope for a platform to ship with a stack of digital certificates:
  - to say that the TPM is implemented according to the TPM specification;
  - to say that the embedding of the TPM within the PC platform conforms to the TPM PC specification;
  - etc.

- Central is the endorsement credential
  - platform supplier points to all these other certificates
  - signs a public endorsement key (EK)

- Relying party decides which manufacturers’ endorsement credentials it trusts.
  - This is the root of trust for reporting
Platform Identity and Endorsement

- The Endorsement Key (EK) is held in the TPM:
  - gives the platform a unique identity
    - the EK is typically* fixed for the lifetime of the platform
- asserts the platform credentials:
  - secret key is the proof of possession for endorsement credentials, conformance, etc.
- Secret part of EK must not be known outside the TPM
  - but the specification allows it to be generated outside the chip, during manufacture
- These features give rise to significant challenges for manufacture, supply chain management, and provisioning.

TPM v1.2 detail

EK is a 2048-bit RSA key pair
Privacy Concerns

- a unique platform identity is bad for privacy
  - every transaction on the web, say, would be traceable to the PC which generated it
  - actions could be correlated
    - with or without “real” identity information
  - manufacturers could trace their platforms’ activities after delivery
    - on a shared platform, one user’s behaviour might be correlated with, or mistaken for, another’s

- TPM needs very careful design to avoid these pitfalls.
Attestation Identity Key (AIK)

- Solution is to allow the platform to have arbitrarily many *attestation identity keys* (AIKs)
- Process for signing these involves EK — so can check platform credentials.
- In use, the AIK has no reference to EK
  - but would generally assert that this AIK belongs to a TPM
- Each AIK is strongly bound to the platform, and protected by the root of trust for storage (RTS)
- AIK is certified by a *Privacy CA*
- Alternative is *Direct Anonymous Attestation (DAA)*
  - advanced zero-knowledge protocol
  - resource-intensive; optional implementation
Privacy CA

- generate AIK key-pair
- $AIK_{pub}$, endorsement cert, etc., including $EK_{pub}$
- certificate for $AIK_{pub}$ encrypted under $EK_{pub}$
- decrypt using $EK_{priv}$
- sign certificate for $AIK_{pub}$
Privacy CA

Privacy CA is a potential weak point

- must be trusted to protect privacy
- could make a merit instead of destroying records after certificates are issued
  - i.e. to forget EK–AIK association
  - removes possibility of revocation

Potential open choice of privacy CAs

- depends on what the relying party will accept
- might differ, for different applications
Noteworthy features

- Platform specification requires that manufacturers ship platforms with TPMs switched off
  - further to address privacy concerns
- Before use, TPM must be enabled, and then ‘take ownership’ completed
  - Creates a fresh encrypted key hierarchy
- Many see this as a barrier to adoption
Trusted Platforms: Summary

cryptographic hash is our ‘measurement’ of software

use a chain of measurements to demonstrate that platform is in a known state

separate hardware – Trusted Platform Module – helps to assure this

reporting the state uses cryptography, and needs indirection to protect privacy
TPM

- TPM is the component at the heart of this vision of Trusted Infrastructure
  - root of trust for storage
  - root of trust for reporting
  - root of trust for measurement
    - with BIOS or other chipset components
- Logical bundle of functionality — a state and some operations — could be hardware or software.
- Tamper-resistance requires hardware
- Virtual TPM necessarily implemented partially (at least) in software
The TPM (v1.2) Specification divides the TPM into ten components, together with volatile storage.

- Input and Output
- Cryptographic Co-Processor
- Key Generation
- HMAC Engine
- Random Number Generator
- SHA-1 Engine
- Power Detection
- Opt-In
- Execution Engine
- Non-Volatile Memory
Protected Storage

Major functionality of TPM

Volatile and non-volatile elements

Register locations with unusual interface

Shielded storage locations

Platform configuration registers
# Protected Storage

**non-volatile**

- storage root key (SRK)
- endorsement key (EK)
- monotonic counters

**volatile**

### platform configuration registers (PCRs)

<table>
<thead>
<tr>
<th></th>
<th>.hash values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### temporary storage for keys

<table>
<thead>
<tr>
<th></th>
<th>key structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**space for certificates and other data**
Hierarchy of Keys

- **storage root key (SRK)** generated and held in TPM
- private part cannot be extracted
  - can be used to *decrypt* (see below) only
- key blobs can be encrypted for storage in untrusted locations
- TPM implements ‘LoadKey’ operation to import an encrypted blob, and hold it in temporary store
- so TPM can protect an arbitrarily large collection of keys
TPM non-volatile storage

TPM volatile storage
Key Hierarchy

- SRK keypair generated when new entity ‘takes ownership’
- destroy SRK => destroy whole key hierarchy
- using a key requires authorization data:
  - 20 bytes — SHA1 output
  - subordinate keys can have different authN
  - SRK owner is not ‘super-user’
  - TPM to protect against brute force attacks
The *KeyStructure* mentioned above will record the type of key being stored/processed:

- **storage keys** are used to store things (!), whether other storage keys; binding or signature keys. These are 2048-bit RSA keys.
- **binding keys** are intended to be used to store *symmetric* keys
- **identity keys** are special keys for signing identity assertions (see PCRs, below)
- **signature keys** are regular RSA signature keys: the TPM must be able to use key sizes of at least 2048-bits.

*Signature keys cannot be used for encryption, and vice-versa.*
### Migratable and non-migratable keys

<table>
<thead>
<tr>
<th>Would want to migrate keys:</th>
<th>Would not want to migrate keys:</th>
</tr>
</thead>
<tbody>
<tr>
<td>to permit <em>group</em> use of keys, in some applications</td>
<td>to have confidence that some keys are forever bound to a particular TPM</td>
</tr>
<tr>
<td>practicality and compatibility with non-TPM software</td>
<td>desirable that some keys are <em>guaranteed</em> to exist in only one place (at any one time, anyway)</td>
</tr>
<tr>
<td>users move!</td>
<td>danger of keys being available outside the control of <em>any</em> TPM</td>
</tr>
<tr>
<td>because hardware doesn’t last forever (*)</td>
<td></td>
</tr>
</tbody>
</table>
Migratable and non-migratable Keys

- Key type structure records whether keys are migratable or not.
- Migratable keys can be created inside or outside the TPM.
- Non-migratable keys must be created inside the TPM.
- Migrate a storage key ⇒ migrate the keys below it in the hierarchy.
- Migratable key cannot be parent of (hence ancestor of) non-migratable key.
- SRK is non-migratable.
  - * except for special maintenance procedure.
- Identity keys are non-migratable (see above).
- Special protocols for migration, with and without guarantees of moving to another TPM: logical copy or logical move are supported.
Platform Configuration Registers (PCRs)

implement trustworthy storage of measurements

- output of *hash* function (SHA1)
- hash of program code
- hash of configuration file
- hash of password (if needed)

cannot be directly written, only ‘extended’:

- \( \text{extend}(i, v) := pcr[i] \leftarrow \text{hash}(pcr[i], v) \)
Using PCRs

- large number of measurements in a typical boot process
  - small number of PCRs (24 in TPM v1.2)
- nominate particular PCRs to hold sequence of measurements at a particular phase in the boot
- use separate integrity measurement log to record which hashes contributed to each PCR value
  - does not need to be secured – result of hashing together the log entries should be the same as the PCR value
- this approach can enable us to achieve authenticated boot
Platform configuration registers (PCRs)

PCRs
- cannot be directly written,
- but can be read

PCR read operation:
- $TPM \text{PCRRead}(n) := \text{output } PCR[n]$

PCR ‘quote’ operation, for nonce $i$: “give me a signed, current record of the contents of the PCRs nominated”
PCR **Quote** operation enables us to build protocols which **attest** a platform’s state to a third party

Nonce value allows third party to know that quote is ‘fresh’

Signing key needs to be certified as belonging to a TPM, to the third party’s satisfaction
Sealed Storage

- combine cryptographic capabilities with PCRs to give a novel capability: sealed storage
  - ‘sealing’ operation nominates a key, target PCR values, and some data
    - target values need not be current PCR values
  - result is a blob which can
    - only be unsealed only the TPM which sealed it
    - can be unsealed only if the current PCR values match the target PCR values
Sealed Storage: applications

- reassure user – or other software – that a trusted application is running, in a familiar state
  - otherwise, authenticated boot is useless: rogue could impersonate good software or configuration
- system-relevant keys locked to OS state
  - hard disc encryption ties disc
    - to platform
    - to operating system
    - etc.
  - locked backups (do you want this?!)

Sealed Storage: applications

- strongly password-locked blobs
  - extend password into an unused PCR
  - can then implement ‘password safe’ functionality
  - fairly good brute-force resistance
- data ‘locked’ to a particular piece of software
  - software which refuses to export data
  - more generally, software which enforces a particular access control policy
  - enables undesirable anti-competitive behaviour
  - enables very desirable privacy-protecting behaviour
Other features

- TPM has a full RSA crypto engine
  - normal placement in PC architecture means it cannot be judged a *high speed* facility
  - deliberately, no bulk symmetric crypto functionality is exposed
- also has built-in genuine random number generation
  - and key generation
Monotonic counters

- TPM is a passive device
  - it would complicate matters to include a real-time clock
- Danger of, for example, certain replay attacks
  - non-volatile monotonic counters help to defeat these
- Suite of commands covers:
  - establishing, incrementing, reading, a counter
- Small number of independent counters
  - complicated by burn-out of NV storage: TPM in PC must support increment every five seconds for seven years.
- Use a signature to tie a ‘tick counter’ value to an external clock
  - can ensure sequencing of collection of signatures
  - can tie a particular tick counter value to a particular real-time interval
Trusted Software Stack (TSS)

- Provides standard API for addressing the functions of the TPM.
- Some operations are performed in software, and not on the TPM.
- TSS has over 200 commands (some optional).
- ‘official’ version is in C++ (and WSDL ??!), but implementations exist for other languages
- Good topics for research:
  - what makes for a good API for trusted computing?
  - how to map TPM functionality onto standard crypto APIs?
Attacks

- Few known attacks against the functional spec
  - intention is that there should be no software attacks
- In just about every system, the TPM is not going to be the weakest point
  - attacks are presently of largely theoretical interest
- Attacks against the secret keys are expected to require large physics lab capabilities
  - recent attack (Feb 2010) achieved this
  - Question: what is the impact of such an attack?
- Question: besides attacking the TPM itself, how else might we compromise the operations it is intended to protect?
Attacks

- Worst known attack (?) to date is a TPM reset attack (TPM v1.1b)
  - use a wire to connect pins
  - TPM is reset (PCRs zeroed), without platform being reset
  - rogue software can extend ‘made up’ values into PCRs
  - thus fool third party into thinking platform is in a different state

- Subsequent (current) design inhibits this
  - successfully?
Summary

**Trusted Platforms based on TPMs**

‘big news’ is protected storage/shielded locations

RNG, key generation, and crypto features are of course critical too

ongoing research regarding how much/what functionality should be in a next version
Trusted Virtualization
Trusted Network Connect
Trusted Storage

TRUSTED INFRASTRUCTURE
Virtualized Platforms

- investing trust in large, multi-purpose operating systems is problematic
  - too much state to attest
  - too many attack surfaces
  - too frequent vulnerability reports
- trust is about an expectation of behaviour
  - behaviour of large complex systems is hard to predict
### Virtualized Platforms

- use of virtual machines enables factorization
  - require trust in what could be a (relatively) small *virtual machine monitor (VMM)* or *hypervisor*.
  - separate VMs can perform distinct tasks: some trusted, some untrusted
- hardware support enables good performance
- hardware support assists launch of *measured execution environment*
  - see discussion of *late launch* and *dynamic root of trust for measurement*
Types of Virtualized Platform

- Conventional platform
- 'hosted' VMM (Type II)
- native VMM (Type I)

many variants and hybrids possible
Virtual machine (VM) is a good target for measurement and attestation
- capable of being ‘read only’ (or reset after each use)
- measure in one step
- chain of trust containing just VMM and VM (approximately)
physical platform has a physical TPM to provide RTS and RTR, and uses hardware to implement RTM.

for running software, virtual platform should behave ‘like’ physical platform

- needs some kind of virtual TPM (vTPM)

software emulator or pass-through driver?

- raises non-trivial questions

fertile research area (Ref: OpenTC project)
device drivers are a major source of system instability (or attack)
devices may be virtualized but still need native drivers
VMs sharing a device will generally share a native driver somewhere
hardware *is* offering increasingly good support for isolation and controlled sharing
Example: Xen 3.0 Architecture

Xen 3.0 Architecture

VM0
Device manager and control software
Guest OS (XenLinux)
Native Device Drivers

VM1
Unmodified user software
Guest OS (XenLinux)
front-end device drivers

VM2
Unmodified user software
Guest OS (Windows)
front-end device drivers

Hardware (SMP, MMU, physical memory, Ethernet, SCSI/IDE)

[after a diagram in Ian Pratt, *Xen and the Art of Virtualization*, Ottawa Linux Symposium 2006]
Virtualization for mandatory controls

- In most OS designs, the role of the OS as a *reference monitor* can be circumvented by a suitably-privileged process.

- Trusted VM or VMM can impose a control point outside the general OS.
- Trusted VM can provide policy decision point
  - or policy enforcement point if interactions are routed through it
- ‘out of band’ enforcement and management of:
  - VPN access
  - logging/audit
  - cryptographic key management
Application

- isolation of VMs at different levels of criticality
  - ‘home’ and ‘work’
  - ‘secret’ and ‘internet’
  - ‘internet’ and ‘intranet’
  - etc.
- trusted sandboxing
- application-independent policy enforcement
the generic issue: **network access control** (NAC)  
with an increasingly large range of portable devices, it is more and more important to be able to make careful decisions about network access  
simplest case is plugging a PC into a cabled network, or connecting a laptop to a wireless network  
unambiguous identification is a start; but insufficient  
attestation of certain properties is needed
NAC: Basic Functionalities

- **User Authentication, e.g.**
  - based on passwords or certificates
  - via VPN and IEEE 802.1X

- **Configuration Assessment**
  - Configuration measurement before network access
  - e.g. installed software like antivirus scanner and firewall
    - Compare measurements to policies of the network to assess Integrity check of the computer system
    - Re-assess accepted computer systems in regular intervals

- **Policy Enforcement**
  - Enforce policies to non-compliant computer systems
Dialogue

1. let me in!

3. certificate/nonce challenge response

5. Ubuntu, Apache, ...

2. who are you?

4. and what OS are you running? antivirus? patch level?

6. prove it!
Architecture
Observation

- Much effort covering this for wireless already; wired connections matter as much.
- Same approach could be used to join other – more remote – networks, not just the local physical network.
- A well-managed context needs a remediation network, too
  - supply missing patches
  - supply missing antivirus signatures, etc.
  - ready the machine to request access again
One Topology

NAC server

policy manager

internet

remediation network

remediation server

Page 112
Trusted Network Connect (TNC)

- open architecture for NAC
  - published by TCG
  - wide vendor adoption
- multi-vendor support – critical for success
- supports existing technologies (802.1X, EAP)
- covers protocols for normal connection, and for quarantine and remediation
TNC Actors

- **Access Requestor (AR)**
  - supplicant/VNC client, etc; TNC client; integrity measurement collectors

- **Policy Enforcement Point (PEP)**
  - typically switch, network edge, firewall, VPN gateway

- **Policy Decision Point (PDP)**
  - TNC server; integrity measurement verifiers

- **Metadata Access Point (MAP)**
  - store of information provided by network security components – may provide context to policy decision
The TNC Architecture

[Figure 2 from TNC Architecture for Interoperability, Specification Version 1.3r6, TCG]
TNC Message Flow

[Figure 3 from TNC Architecture for Interoperability, Specification Version 1.3r6, TCG]
TNC with TPM

[Figure 5 from TNC Architecture for Interoperability, Specification Version 1.3r6, TCG]
Metadata Access Point

- collects metadata about access decisions
- can be used to inform dynamic firewall set-up
  - or other network topology issues
- interact with other sensors
  - use building access to determine network access (or enable/disable VPN access)
- interoperability is key
Motivation:
- increasingly high-value data on portable devices: problems of:
  - accidental loss/theft
  - intentional compromise of data
  - targetted theft
- equally serious problem of end-of-life disposal
- password locks for discs are cumbersome, and can be subject to brute-force attacks
- software-based encryption is sub-optimal
Aside: Microsoft Bitlocker™

- *non-example, but relevant anyway*
- “software-based encryption is sub-optimal” ... *but easy to deploy*
- Using Bitlocker, the Operating System (Microsoft *Windows Vista* and *Windows 7*) implements an encrypted disc filing system
- Can utilize TPM for secure key storage and boot environment authentication
  - key for disc encryption is released only if on the correct platform and with the pre-determined pre-boot environment
TCG Storage Encryption

- open standards approach
  - general-purpose,
  - operating system independent
- being supported by ('all six') disc manufacturers;
  - also solid state devices
- largely independent of connection technology (but standards compliant)
  - use for internal and external discs
Use cases

- Enrollment and connection trusted relationship between the storage device and the host
- Protected storage for storing sensitive data
- Locking and encryption mating a storage device and host; encrypting stored data at rest

- Logging for forensic purposes
- Cryptographic services supporting a variety of security services
- Authorizing storage device feature sets to hosts for trusted and exclusive use
- Secure download of firmware trusting firmware upgrades
Concept

- host
- TPM
- trusted storage device
- interface
- authenticate
- data
- encrypted data
Authentication

- **device-to-host**
  - prove knowledge of a secret set by the host

- **host-to-device**
  - need to know the device password/key
    - set at manufacture; changeable by OEM or owner
  - finer-grained control for access to particular ‘ranges’ (partitions), ‘security providers’
  - substantial key management potential:
    - store on device, on host, or with third party
Full Disk Encryption

- Fast disk disposal/repurposing
- Protect of data against computer theft
- Independent of operating system
- Lock individual drives to particular machines makes a hard drive useless to a drive thief
- Effectively extends system protection provided by TPM to a large data store
Full Disk Encryption

- enrollment: host stores public key at drive
- reconnection: host must prove possession of private key before drive (partition) is unlocked
- encryption provides protection beyond enrollment and connection: drive data is useless if decryption key is not available
Protected Storage

- additional capability to create ‘Secure Partitions’ (SPs)
  - secure data areas, separate from user data area
  - subject to host-managed access controls
  - use for:
    - private data
    - software licences
    - digital cash?
TCG Storage

SW and HW features and function (e.g., Crypto Calls)

ATA/S CSI I/F

Internet

mobile devices

end users

(service providers)

[after Thibadeau/TCG]
Secure Partitions

“The host platform, applications, devices, local end users, and remote users/service providers can gain exclusive control of selected features of the storage device. This allows them to simultaneously and independently extend their trust boundary into the storage device.”

[after Thibadeau/TCG]
Summary

key components for trusted infrastructure

- platform virtualization
- network access control (trusted network connect)
- trusted storage
- ... your research too
Andrew Martin, MA, DPhil, CEng, MBCS, CITP
Deputy Director, Software Engineering Programme

Wolfson Building, Parks Road, Oxford OX1 3QD, UK.
+44 (0) 1865 283605
Andrew.Martin@cs.ox.ac.uk
www.softeng.ox.ac.uk/andrew.martin