TRUSTED COMPUTING/PLATFORMS

Overview

- Intuitive model of trusted computing
- Practical applications using trusted computing to improve security
- Hardware versus software
- Software secured execution environment:
  - Java, STIP, .NET,
  - OSes and Virtualization: Linux LSM, SELinux
- Trusted Computing Group (TCG): TPMs
- Hardware secured execution environment: TrustZone
- Special trusted computing devices
  - HSM

New Security Challenges

- Computing devices are becoming distributed, unsupervised, and physically exposed
  - Computers on the Internet (with untrusted owners)
  - Embedded devices (cars, home appliances)
  - Mobile devices (cell phones, PDAs, laptops)
  - Base stations and wireless access points

- Attackers may physically tamper with devices
  - Invasive probing
  - Non-invasive measurement
  - Install malicious software

From trusted computing to trusted platform

- Trusted Computing
  - Requires that the (application) software can be trusted
  - SW security: Not the subject of this lecture
  - Requires that the underlying system can be trusted
  - Trusted Platforms
Useful Benefits of a trusted platform

When a platform can prove that it is running the expected executable
- Third-party (grid) computing
  - Produce correct results
- Supporting Bring Your Own Device (BYOD) policy
  - Data of different stakeholders is kept safe on the device
- Electronic payments
  - Correct amount, anonymous (to some extend), Trusted UI
- Digital Rights Management (DRM)
  - Enforce copyright on content (music, video, programs, etc)
- Sensor and surveillance
  - Can rely on the data that is received
- AND …

But not always needed

- If we can do use (almost) homomorphic encryption techniques then we can outsource processing (certain types) of sensitive data to an untrusted compute pool!
- For example CryptDB from MIT.
  (in cryptDB information on stored data still may leak during processing, but the idea is very nice, and it works pretty efficient) css.csail.mit.edu/cryptdb/

Intuitive platform models

- Open platform
  (e.g. PC, PDA, iOS, Android, Linux device)
  - General purpose computing platform
  - No a priori trust with third party

- Closed platform
  (e.g. ATM, set-top box, game console, satellite receiver, most older (pre iOS, Android) mobile phones)
  - Special purpose computing device
  - Can authenticate itself to a remote party using a secret key and ensure well-behaved operation
  - Hardware tamper resistance to protect the secret key (embedded during manufacturing)

Intuitive models cont’d

Trusted computing combines best properties of

- Open: allow applications from many different sources to run on same platform
- Closed:
  - Only "known" software can execute
  - Isolation of sw components to limit propagation of malicious code
  - remote parties can determine what software is running and whether to expect the platform to be well behaved
    - Done through attestation
Why is Trusted Computing so hard?

Attacks

- SW only based attacks
- HW based attacks
- Other (unexpected) ways

Hybrids

- Often the opponent can use the attack tools he/she wants
- The unexpected

SW exploits

Most common (and most “interesting”) way of attacking

- Due to design misses
- Careless programming: e.g. buffer overflow, XSS, (no input validation), ….
Example of the unexpected Attacks on smartcards

- Khokar et al., June 1998: Measure instantaneous power consumption of a device while it runs a cryptographic algorithm
- Different power consumption when operating on logical ones vs. logical zeroes.

Hardware vs Software

- Functionality in Hardware
  - hard/costly to change
  - high performance possible
- Functionality in Software
  - Easy to change
  - Difficult to hold private keys
Trusted Systems in Software

- Possible but we have limitations
  - owner of the device on which software runs should not be an attacker
    (he/she and the device “work together”/“have the same interests”)
  - Does not work when the device in the “enemy’s territory”
  - But “software only” is sometimes the only implementation option: e.g. virtual platforms

Execution platforms: OSes

- OSes come in many different versions; Linux, Windows X, Android, MeeGo, Symbian, Palm-OS, iOS, etc
  - Most simple OSes have no means to securely enforce (multi-user) access control. (relevant for small embedded systems)

- Even if so
  - Correct configuration OS environment is not trivial
  - Without protection user can attack system from “below”

OS for trusted computing

- Primary security objective of OS in a trusted platform = secure isolation
  - Mainstream OS (e.g. Windows, Linux)
    - Large and complex code base
    - Optimized for ease of use, performance and reliability
  - Trusted OS (e.g. Trusted Solaris, SE-Linux, SPEF)
    - Not user friendly (because of security policy)

OS for trusted computing

- Solutions to have best of both
  - Hypervisor (also called Virtual Machine Monitor (VMM))
    - attestation through virtual device
  - Modify OS
    - try to create isolation (VMs or OS features)
  - Change existing hardware
    - attestation done by hardware module
    - add secure execution mode to CPU
Virtualization

- Abstraction of computer resources
- Pioneered by IBM to keep using legacy system solutions on new hardware without rewriting code
- Turned up to have stability and security benefits (isolation) (at the expense of performance)
- There are many ways to do this and there exist therefore many different types of approaches to virtualization

Virtualization approaches

Type 1 and Type 2 virtualization
- Type 1: runs on "bare metal"
- Type 2: runs on host

Full/Pure vs impure/para virtualization
- Full/pure virtualization: ensure that sensitive instructions are not executable within the virtual machine, but instead invoke the hypervisor: needs hardware support
- Impure virtualization: remove sensitive instructions from the virtual machine and replace them with virtualization code.

Hypervisors (and micro Kernels)

- Execute in priveledge mode
- Schedule the systems(OSs) that execute on it
Pure Virtualization

- Most instructions are executed directly on the hardware.
- All sensitive instructions are privileged. They are trapped and instead executed by the hypervisor that runs in privileged space (kernel mode, superuser mode, etc).
- Needs hardware support (Modern main CPUs have this)

Impure Virtualization

- Most instructions are executed directly on the hardware.
- All sensitive instructions rewritten (e.g. during load time or during porting (para virtualization)): either trap to hypervisor or jump to a user-level emulation code

Hypervisors and micro-kernels

- Virtualization is done often through a hypervisors
  - Thin layer
- But can also done via micro-kernels
  - Interesting for small systems? (However micro-kernels are common also as core part in large Oses)

Xen

virtualization of x86, x86_64, IA64, ARM, and other CPU architectures

http://xen.org/
Example of a trusted microkernel: L4

- Used (with modifications) in various Qualcomm platform based phones. Available for Linux and Symbian. Used in some smart cards
- Academic work ongoing: to proof the kernel is correct
- Free to use
- Commercial: e.g. OpenKernel Labs

JAVA AS TRUSTED EXECUTION ENVIRONMENT

Example of Trusted Computing in SW

Outline

- components of Java
- Java security models
- main components of the Java security architecture
  - class loaders
  - byte code verification
  - the Security Manager

Java language features

- object-oriented
- multi-threaded
- strongly typed
- exception handling
- very similar to C/C++, but cleaner and simpler
  - no more struct and union
  - no more (stand alone) functions
  - no more multiple inheritance
  - no more operator overloading
  - no more pointers
- garbage collection
- objects no longer in use are removed automatically from memory
The Java Virtual Machine (JVM)

- **class file verifier**
  - checks untrusted class files
  - size and structure of the class file
  - bytecode integrity (references, illegal operations, …)
  - some run-time characteristics (e.g., stack overflow)
  - A class is accepted only if it passes the test

- **native method loader**
  - native methods are needed to access some of the underlying operating system functions (e.g., graphics and networking features)
  - once loaded, native code is stored in the native method area for easy access

- **the heap**
  - memory used to store objects during execution
  - how objects are stored is implementation specific

**JVM cont’d**

- **execution engine**
  - a virtual processor that executes bytecode
  - has virtual registers, stack, etc.
  - performs memory management, thread management, calls to native methods, etc.
**JVM cont’d**

- **Security Manager**
  - enforces access control at run-time (e.g., prevents applets from reading or writing to the file system, accessing the network, printing, ...)
  - application developers can implement their own Security Manager
  - or use the policy based SM implementation provided by the "standard" JDK

**Java security models**

- the sandbox (Java 1.0)
- the concept of trusted code (Java 1.1)
- fine grained access control (Java 2)

**Java 1.0: The sandbox**

*idea:* limit the resources that can be accessed by applets (this creates an execution sandbox)

- local code had unrestricted access to resources
- downloaded code (applet) was restricted to the sandbox
  - cannot access the local file system
  - cannot access system resources,
  - can establish a network connection only with its originating web server

**Java 1.1: The concept of trusted code**

*idea:* applets that originate from a trusted source could be trusted

- applets could be digitally signed
- unsigned applets and applets signed by an untrusted principal were restricted to the sandbox
- local applications and applets signed by a trusted principal had unrestricted access to resources
Java 2: Fine grained access control

- Every code (remote or local) has access to the system resources based on what is defined in a policy file.

- A protection domain is an association of a code source and granted permissions.
- The code source consists of a URL and an optional signature.
- Permissions granted to a code source are specified in the policy file.

```java
grant CodeBase "http://java.sun.com", SignedBy "Sun" {
    permission java.io.FilePermission "/home/user/\*", "read, write";
    permission java.net.SocketPermission "localhost:1024-", "listen";
};
```

The four pillars of Java security

- The Security Manager
- Class loader
- The bytecode verifier
- The JVM

The Java runtime (sloppy called the JVM)

- Java code executes in the VM and execution is isolated from external (execution) environment.

- A Java runtime is the program that implements the JVM specification. This JVM specification defines what threading features a runtime must support, but does not mandate a particular implementation. So thread behavior depends on the implementation and on the capabilities of the underlying external environment (typically an OS).

Java’s impact

- The Java system has been an example for many other languages, execution environments, systems.
- E.g.
  - ActiveX
  - .Net
  - STIP
  - Android
Comparison with STIP

- STIP is pretty much like Java
- STIP needs much smaller VM
- STIP is not under control of SUN (or Microsoft)
- STIP is better on access control
- STIPlets can be executed directly from ROM/(NOR) Flash memory (no need to load first into RAM)
- STIP is for small devices (e.g. smart card terminals)
- STIP is limited in MMI APIs


Android

- Java in Android is different. It borrows many ideas from ordinary Java but in Android one has a totally different security architecture
  - E.g. digital signatures of apps are not used to verify origin but to have a proof that they originate from same issuer or not which affects how apps are isolated

Android: Linux access control also in "Java" env

SELINUX

Trusted systems: trusted OS
Example of a trusted OS: SELinux

Motivation

- Discretionary Access Control (DAC) in Linux provides not enough choices for controlling objects.

- Mandatory Access Control (MAC) allows you to define permissions for how all processes (called subjects) interact with other parts of the system such as files, devices, sockets, ports, and other processes (called objects in SELinux).

SELinux Development History

- Primarily developed by the US National Security Agency
- SELinux implements Flask

Security goal for SELinux

- General: Minimize the effect of malicious code and exploits
  - "Sandboxing" programs and data
  - control access rights, minimize the privileges

Policy in MAC system (SELinux)

- The behavior or control what is allowed or not is handled through a policy file.

- In SELinux where MAC is implemented on top of a DAC system. The DAC control goes first and then the policy is enforced.

  The specification of the policy (file) is the heart of the MAC system.
Reference Policies

- Maintained by NSA and FC Mailing Lists
- Compiles into three versions
  - Strict, Targeted, MLS (Multi Level Security)
- Statistic:
  - Version .18
  - Object Classes 55
  - Common Permissions 3, Permission 205
  - Types 1589
  - allow 372755, auditallow 12, dontaudit 238663
  - type_transition 2657, type_change 68
  - roles 6, RBAC allow 6, role_transition 97, users 3
  - bools 70

Time to play: SELinux distributions

- Fedora Core 3 and later
- Debian
- Gentoo
- SuSe
- SE-BSD
- SE-MACH

SELinux and Android

- Android 4.4 is adopting SELinux components
  - Android sandbox reinforced with SELinux. Android now uses SELinux in enforcing mode.

THREE APPROACHES TO HW SUPPORTED TRUSTED COMPUTING

- ARM TRUSTZONE
- Intel SGX
- TCG TPM
Recall security ring architecture

- Operating System Kernel
- Operating System Services
- Applications

Protection rings
- Dedicated
  - Instructions
  - Memory space

Level 0
Level 1
Level 2
Level 3

Separation of sensitive ops and data

- Since too much code in running in user space and even in the privileged space:

  Sensitive applications and data cannot be given good guarantees that other running code cannot tamper or get access.

ARM standard approach

- Operating System Kernel/Services
- Applications

Protection rings
- Dedicated
  - Instructions
  - Memory space

Privileged mode
- Supervisor mode

User mode

Security problem for applications

- Operating System Kernel/Services
- Applications

Protection rings
- Dedicated
  - Instructions
  - Memory space

Privileged mode
- Supervisor mode

User mode
- App1
- App2
**ARM TrustZone**

- A special mode of operation for the ARM11 processor
- Divides the SoC into “normal world” and “secure world”

<table>
<thead>
<tr>
<th>Normal world</th>
<th>Secure world</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Software Application</td>
<td>Demo Service API</td>
</tr>
<tr>
<td>Secure Service API</td>
<td>Native Service API</td>
</tr>
<tr>
<td>Stub Library</td>
<td>Interpreted Service API</td>
</tr>
<tr>
<td>TrustZone Software API</td>
<td>TrustZone Software Family</td>
</tr>
<tr>
<td>TrustZone Device Driver</td>
<td>HAL API</td>
</tr>
<tr>
<td>&lt;OS / Platform&gt;</td>
<td>HAL Implementation</td>
</tr>
</tbody>
</table>

**Basic idea**

- Introduce an NS-bit
  - use this bit to *tag* secure data throughout system
  - Buses
  - cache
  - pages

- Monitor
  - manages the NS-bit
  - manages transition in & out of security mode
  - Small fixed API (so we can better check/verify the code)

**Switching from Normal to Secure**

Normal

- Normal application
- Normal OS
- userspace
- priviledged

Secure

- Secure Service
- Secure Kernel
- Secure drivers
- Secure device
- userspace
- priviledged
- Monitor
- Boot loader

**TrustZone uses Hardware features - Example System**

TRustZone is used in many Android smartphone products:
SGX - ENCLAVES

Software Guard eXtensions

From: Innovative Instructions and Software Model for Isolated Execution, Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos Rozas, Hisham Shafi, Vedvyas Shanbhogue and Uday Savagaonkar, Intel Corporation

Overview

- SGX in a new technology introduced in Intel chipsets
- SGX architecture includes 17 new instructions, new processor structures and a new mode of execution.
- These include loading an enclave into protected memory, access to resources via page table mappings, and scheduling the execution of enclave enabled application. Thus, system software still maintains control as to what resources an enclave can access.
- An application can be encapsulated by a single enclave or can be decomposed into smaller components, such that only security critical components are placed into an enclave.

Enclaves

- Enclaves are isolated memory regions of code and data
- One part of physical memory (RAM) is reserved for enclaves and is called Enclave Page Cache (EPC)
- EPC memory is encrypted in the main memory (RAM)
- EPC is managed by OS/VMM
- Trusted hardware consists of the CPU Die only

Reduced attack surface with SGX

- Application gains ability to defend is own secrets
  - Smaller attack surface (App+processor)
  - Malware that subverts OS or VMM, BIOS, Driveers cannot steal app secrets of
SGX Programming Environment

Protected execution environment embedded in a process

- With its own code and data
- Provide confidentiality and integrity protection
- With controlled entry points
- Support for multiple threads
- With full access to app memory

OS
Enclave (DLL)
Enclave data
Enclave data
TCS (*n)

User process
Enclave

TCS= Thread Control Structure

The Enclave Page Cache (EPC) is protected memory used to store enclave pages and SGX structures. The EPC is divided into 4KB chunks called an EPC page. The Enclave Page Cache Map (EPCM) is a protected structure used by the processor to track the contents of the EPC.

Protection against Memory Snooping

1. Security perimeter is the CPU package boundary
2. Data and code unencrypted inside CPU package
3. Data and code outside CPU package is encrypted/integrity protected,
4. External memory reads and bus snoops tapping gives access to encrypted

Attestation and Sealing

- SGX supports also attestation and sealing
- What this is will be explained in the TCG slides
TCG

Trusted Computing Group

The three basic functions in a TCG trusted platform

- **Protected Capabilities**
  Protected capabilities is a set of commands that grant the user issuing the command access to protected locations, memory (storage), registers, etc.

- **Attestation**
  Attestation is the process of verifying the accuracy of information and the characteristics of the TPMs current state.

- **Integrity (Measurement and Reporting)**
  Integrity measurement is the process of obtaining metrics of the platform characteristics and storing the information digest in a Platform Configuration Register (PCR). Integrity reporting is to attest the integrity measurements that are recorded in the PCR register.

TCG Architecture (typically PC or Server)

- ![Diagram of TCG Architecture](http://www.trustedcomputinggroup.org/)
  - Founded in 1999 by Compaq, HP, IBM, Intel and Microsoft
  - Currently more than 200 members
  - Implies changes to the hw platform
    - Extra for advanced devices: Trusted Platform Module (TPM)
    - Extra for mobile devices: TPM Mobile
    - Software changes: BIOS + OS

Picture is not entirely correct as surrounding hw needs some changes
Trusted Platform Module (TPM)

- Cryptographic operations
  - Hashing: SHA-1, HMAC
  - Random number generator
  - Asymmetric key generation: RSA (512, 1024, 2048)
  - Asymmetric encryption/decryption: RSA
  - Symmetric encryption/decryption: DES, 3DES (AES)
- PCR Registers (≥16)
- Tamper resistant (hash and key) storage
- Slave device (i.e. must be driven from outside)
- Opt-in: TPM state

Asymmetric key generation
Signing and encryption
Random number generator
PCR registers (≥16)
I/O
Processor
Memory
Non-volatile memory (≥1280) bytes
TPM

Commercial TMP example - Infineon

Infineon Technologies Platform Module Solution Provides the Following Features

- 64 kilobytes of ROM & 8 kilobytes of RAM
- 64 kilobytes of EEPROM with 500 write-erase cycles
- 16 kilobytes of EEPROM for firmware secure updates
- RSA hardware accelerator for signature calculation and verification as well as 2048 bit key generation when using CRT
- World-leading security protection against SPA and DPA
- Low Pin Count (LPC) bus optimized
- Low power consumption

Software Architecture Overview:
- Embedded Secure Operating System
- Embedded Secure Application Support
- TCG Software Stack (TSS) compliant to current and released specifications
- TCG PC BIOS support available with design guide
- TPM cryptographic service providers for MS-CAPRI 2.0 and PKCS#11

EIT - Trusted Computing

TPM States

- Three operational states
  - (Enabled, Active, Owned)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>A</td>
<td>O</td>
</tr>
</tbody>
</table>
| ✓ | ✓ | ✓ | S1: Fully Operational State
| ✓ | ✓ |   | S2: Ownership is and can be set
| ✓ | ✓ |   | S3:
| ✓ | ✓ |   | S4: Ownership cannot be set
| ✓ | ✓ |   | S5: Local or remote ownership possible
| ✓ | ✓ |   | S6: Ownership can be set
| ✓ |   |   | S7:
|   |   |   | S8: All functions are off

Opt-In

- The TCG policy is that the TPM should be shipped “in the state that the customer desires”.
- Thus, it is up to the user to opt-in to use the TPM. Users are not forced to use trusted computing, they opt-in if they choose to do so by taking ownership of the device.
- The function of the Opt-In component is to provide mechanisms and protection to maintain the TPM state via the state of the corresponding flags.
TPM Counters

TPM has monotonic counters (at least 4)

- Increment rate: Every 5 secs for at least 7 years
  (so at least 26 bit counter needed)

- Can be used for anti-roll back protection
  - (old versions can be blocked from loading)

PCR is a register that contains a SHA1 hash and is used to accumulate “measurements”

- Can be read from the outside
- Are reset to zero at power up

We return to this later

PCRs

Main system components in a PC with a TPM

- Root of Trust for Measurement (RTM)
  - The Core Root of Trust for Measurement (CRTM)
    - Static and Dynamic RTM
    - The Trusted Building Block (TBB)

- Root of Trust for Storage (RTS)

- Root of Trust for Reporting (RTR)

- The TCG SW Stack (TSS)

The CRTM: S-RTM and D-RTM

- The CRTM is the \textit{a priori trusted code} that is part of the platform credential.

- In the \textbf{Static RTM Model}, this MUST be the very first piece of code executed on power on or upon reset of the server or complete physical hardware environment.
  - Note: at startup the CRTM will check for physical presence of the TPM
  - REMEMBER: TPM is not the root-of-trust but trust starts with the CRTM

- In the \textbf{Dynamic RTM model} the hardware is designed to support that while running a trusted execution thread can be started:
  - Intel call their implementation (Intel) \textit{Trusted eXecution Technology}
  - AMD: DRTM instruction, SKINIT
Trusted Building Block (TBB)

- The combination of the CRTM, TPM functions, connection of the CRTM to the system/platform, and the connection (or binding) of a TPM to the system/platform makes up the TBB.

- Trust in the connection of the CRTM to the TPM is transitive, and relies upon trust in the CRTM connection to the system/platform and in the TPM connection to the system/platform.

- Server platforms may differ from other platforms in that TPM functionality may be physically distributed within the platform. However, the TBB encompasses the RTM

BOOT Process with TPM

1. The Root of Trust for Measurement (RTM) measures the BIOS metric.
2. Measured value is reported to the Trusted Platform Module (TPM).
3. BIOS is executed.
4. Operating System (OS) Loader metric is measured.
5. Measured value is reported.
6. OSLoader is executed.
7. OS metric is measured.
8. Measured value is reported.
9. OS is executed.
10. An application is measured.
11. Measured value is reported.
12. Application is executed

How does it work - overview

- Protected TPM keys
- TPM secure storage
- Secure boot
- TPM in mobiles
- Software stack for TPM
- TPM in windows
TPM Protected storage - keys

- TPM holds only two permanent keys
  - The EK (sort of TPM private key)
  - The SRK
- Internal storage is smartcard alike
  - Persistent, secure storage of keys
  - Not 100% hardware tamper resistant
- Other keys are stored encrypted outside and loaded internally as needed during processing

Main keys – always remain in TPM

1. **Endorsement Key (EK)** (2048-bit RSA)
   - Created at manufacturing time. Cannot be changed.
   - Used for “attestation” (described later)
2. **Storage Root Key (SRK)** (2048-bit RSA)
   - Used for implementing encrypted storage
   - Created after running `TPM_TakeOwnership(OwnerPassword, ...)`
   - Can be cleared later with `TPM_ForceClear` from BIOS
3. **OwnerPassword** (160 bits) and persistent flags

Private EK, SRK, and OwnerPwd never leave the TPM

TPM Key types

- **Endorsement Key (EK)**
- **Storage Root Key (SRK)**
- **Attestation Identity Keys (AIK)**
  - Sign data from the TPM, such as quotes or certificates, a TPM can have many identities!
- Storage: encrypt data, including other keys, (SRK is a special storage key)
- Signing: key only for signing
- Binding: decrypt data (usually from remote platforms)
- Certified Migration Key (CMK), is of one above type but tagged as migratable
- Legacy: signing or encryption (TPM v1)

Key Hierarchies

Can only be created once.

```
<table>
<thead>
<tr>
<th>EK (Endorsement key)</th>
<th>SRK (Storage Root Key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take Ownership</td>
<td></td>
</tr>
<tr>
<td>AIK (Attestation Id Key)</td>
<td></td>
</tr>
<tr>
<td>“your keys” (----)</td>
<td></td>
</tr>
</tbody>
</table>
```

These two will stay always inside TPM in non-volatile mem.

“make key/ and certify that key belongs To SRK

Question: How prove that SRK inside TPM?
TPM Protected storage - data

- The TPM
  - can wrap data
  - “Sealing”: binds data to a certain value of the PCR. Then the TPM can only decrypt (unseal) if the PCR value(s) is the same as when encryption happened (seal)

- Management: migration, backup

Secure bootstrap: secure vs authenticated boot

- Integrity metric
- Platform Configuration Register (PCR)

- Two methods of booting
  - Secure Boot: boot can be halted when check fails
  - Authenticated (or Measured) Boot: just reporting

Secure bootstrap: PCRs and measuring

- Extending a PCR
  PCR_Extend(n,data): PCR[n]←SHA1(PCR[n] || data*)

- When booting
  1. Reset PCRs
  2. PCR_Extend(n,<Bios Code>)
  3. PCR_Extend(n,<MBR>)
  4. etc

Using PCRs for sealing

Encrypted storage (TCG lingo = sealed storage).

Step 1: TPM_TakeOwnership( OwnerPassword, … )
  - Creates 2048-bit RSA Storage Root Key (SRK) on TPM
  - Cannot run TPM_TakeOwnership again without OwnerPwd:
    - Ownership Enabled Flag ← False
    - Done once by laptop owner or IT department (hmm).

(optional) Step 2: TPM_CreateWrapKey / TPM_LoadKey
  - Create more RSA keys tied to this TPM, protected by SRK (stored outside TPM)
  - Each key is identified by a 32-bit key handle
**Protected Storage: SEAL**

**Main Step:** Encrypt data using RSA key on TPM

**TPM_Seal**

- **INPUT:**
  - KeyHandle: which TPM key to encrypt with
  - KeyAuth: password for using key with id `KeyHandle`
  - PcrList: list with indices J and PCR[i] i ∈ J to be embedded in output sealedData
  - data: at most 256 bytes (2048 bits) (typically used to encrypt symmetric key (e.g. AES))

- **OUTPUT:**
  - sealedData RSA encrypted data (and PcrList)

**Example: Use case of a sealed key**

- **Problem:** We want at machine start read out a secret that decrypts some site specific TLS client certificates (containing secret PKI keys) and puts it into RAM but when the OS kicks-in the secret should not be recoverable anymore.

- **Solution:** Seal the secret to a PCR that “measures” the machine state during boot. When the boot comes to the correct point its TPM can do unseal and we load the RAM with our client cert. Then we erase the secret, verify the OS code, update the PCR and start the OS. Now we no longer can unseal the secret and is protected against wrong doings by the OS and the software that is running on the OS.

**MAC protected list by TPM_Seal**

- **TPM_Seal(J , data) →**
  - $C, \text{MAC}((\text{SRK,)((i_1, PCR[i_1]), (i_2, PCR[i_2]),...)))$

  Protection of PCR list

  Ex: C=RSA(SK,data) and SK is storage key

**Protected Storage: UNSEAL**

**Main Step:** Decrypt data using RSA key on TPM

**TPM_UnSeal**

- **INPUT:**
  - KeyHandle: which TPM key to decrypt with
  - KeyAuth: password for using key with id `KeyHandle`
  - sealedData: RSA decrypted data and PcrList

- **OUTPUT:** IF and Only IF
  - data: ∀ i ∈ J current PCR[i] = PCR in MAC protected list

- **EIT - Trusted Computing 2015 - B. Smeets**
Remote measurement of TPM state

- This is called attestation

- The goal is to securely determine remotely the state of a machine. The approach is to record the state of the machine in the PCRs and to have a mechanism by which one can interrogate a TPM to get the PCR values.

TCG: Attestation

- Integrity reporting: report the value of the PCR
- Challenge-response protocol

Challenger \(\text{nonce}\) Trusted Platform Agent
\[\text{Sign}_{\text{ID}}(\text{nonce, PCR, log}), C_{\text{ID}}\]

- TPM Identities (pseudonyms)
  - Endorsement certificate embedded in TPM
  - Privacy CA
  - Use different identity for every challenger

Basic model for attestation (1/2)

- Certificate \(C_{\text{hw}}\) (\(\text{hw}=\text{hardware}\)) related to
  - signing key \(K_{\text{sign}}\)
- Application A generates key pair \(PK_A, SK_A\)
- Certificate \(C_A\) related
  - \(PK_A\) and signature of hash of executable image of A using \(K_{\text{sign}}\)
- A attests its validity to a remote server by
  - sending \(C_{\text{hw}}\) and \(C_A\)
  - proving knowledge of \(SK_A\)
- Remote server checks certificates and whether the application hash in \(C_A\) is trusted

Basic model for attestation (2/2)

\(C_{\text{hw}}: K_{\text{sign}}\)

Application A: ImageA PKA, SKA

CA:
1. PKA
2. hash(imageA) using Ksign

Prove & REPORT

SERVER

Attestation
Attestation: how it works

- Recall: EK private key on TPM.
  - Cert for EK public-key issued by TPM vendor.

- Step 1: Create Attestation Identity Key (AIK)
  - Details how this is done are not important.
  - AIK Private key known only to TPM
  - AIK public cert issued only if EK cert is valid (how to check?)

Step 2: sign PCR values (after boot)

Call `TPM_Quote` (some) Arguments:
- `keyhandle`: which AIK key to sign with
- `KeyAuth`: Password for using key `keyhandle`
- `PCR List`: Which PCRs to sign.
- `Challenge`: 20-byte challenge from remote server
  - Prevents replay of old signatures.
- `Userdata`: additional data to include in sig.
- Returns signed data and signature.

Attestation: how it (should) work

In an actual system

- Generate pub/priv key pair
- `TPM_Quote(AIK, PcrList, chal, pub-key)`
- Obtain cert

(SSL) Key Exchange using Cert

Communicate with app using SSL tunnel

Remote Server

- Attestation must include key-exchange
- App must be isolated from rest of system

Attestation: how it works

- Step 2: sign PCR values (after boot)

Validate:
1. Cert issuer,
2. PCR vals in cert

Remote Server

RemoteServerPC

TPM

OS

App

Attestation Request (20-byte challenge)

Initially a privacyCA was architectured through which AIKs could be bind to a specific TPM.

Yet there were concerns that the binding compromises anonymity and therefore TCG has implemented a more advanced attestation method based on zero-knowledge techniques: Direct Anonymous Attestation (DAA).

DAA is a cryptographic protocol which enables the remote authentication of a trusted platform yet preserving the user’s privacy.
TPM Mobile

- 2006 the Mobile Trusted Module (MTM) was standardised. The purpose of the mobile phone architecture was to support multiple stakeholders in a single device. No market interest.

- Will during 2013 be replaced by TPM mobile which is based on TPM v2.0 with some of the extensions of MTM.
  - Can be realized as physical component
  - Can also be realized in embedded fashion (e.g. using TrustZone)

TPM use in practice

- TPM software stacks
- TPM in MS Windows
- TPM and UEFI boot

TCG Stack vs. TPM Services Stack

- TPM applications use the TCG Service Provider (TSP) interfaces
- The TCG Core Services component (TCS) is ported to communicate with the TBS instead of the TCG Device Driver Layer (TDDL)

TPM experimenting

- You can experiment with TPM by using a TPM emulator and the TrouSerS (TSS) software stack

  - TPM Emulator:
    - http://tpm-emulator.berlios.de/

  - TSS Stack:
    - http://trousers.sourceforge.net/
    - Above link contains many other useful code

  - PrivacyCA: (for use with AIKs)
    - http://www.privacyca.com/
TPM in windows

- TPM support in Windows 7
- TPM use more driven in Windows 8

Use case
- Bitlocker (file encryption)
- Secure boot (UEFI boot)

Bitlocker (1/3)

- **AES in CBC mode with Elephant diffuser**
- Key escrow via Active Directory
- Three different modes are supported
  - **Transparent operation mode (with TPM):** The key used for the disk encryption is sealed by the TPM and will only be released to the OS loader code if the early boot successfully verified. The boot components of BitLocker do a Static Root of Trust Measurement.
  - **User authentication mode (with TPM):** This mode requires that the user provide some authentication to the pre-boot environment in order to be able to boot the OS. Two authentication modes are supported: a pre-boot **PIN** entered by the user, or a **USB key**.
  - **USB Key Mode:** The user must insert a USB device that contains a startup key into the computer to be able to boot the protected OS.

AES+CBC with Elephant (diffuser)

- AES CBC: cost 20 cycles per byte on Pentium4
- Diffusers: cost 10 cycles per byte
- Microsoft reports overall penalty of 5%

A diffuser

For $i = n - A_{\text{cycles}} - 1, \ldots, 2, 1, 0$

$\text{der} \leftarrow \text{der} - (d_{i-2} \oplus (d_{i-5} \ll R_{i \mod 4}^{(a)})$

B diffuser

For $i = 0, 1, 2, \ldots, n - B_{\text{cycles}} - 1$

$\text{der} \leftarrow \text{der} + (d_{i+2} \oplus (d_{i+5} \ll R_{i \mod 4}^{(b)})$

Figure 1: An overview of AES-CBC + diffuser

Bitlocker (2/3)

- Combinations
  - TPM only
  - TPM + PIN
  - TPM + PIN + USB Key
  - TPM + USB Key
  - USB Key
- At least two NTFS volumes needed: one for the OS and another unencrypted to boot the OS from.
Bitlocker (3/3)

- BitLocker encryption is transparent to OS
  - BitLocker decrypts on-disk files before the OS has loaded. Therefore, all file operations occur from the OS perspective as if there is no encryption.
  - Protection of the files from processes/users within the operating system can only be performed using encryption software that operates within Windows, such as Encrypting File System.

Windows secure boot

- UEFI secure boot
- OEM (manufacturer’s) role

UEFi Trusted Boot Architecture

UEFi Secure Boot Keys

- Platform Key (PK)
  - Only one
  - Allows modification of KEKs
- Key Exchange Key (KEK)
  - Can be multiple
  - Allows modification of db and dbx
- Authorized Database (db)
  - CA, key, or image hash to allow
- Forbidden Database (dbx)
  - CA, key, or image hash to block
**Keys required for Secure Boot**

<table>
<thead>
<tr>
<th>Key/db name</th>
<th>Owner</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKpub</td>
<td>OEM</td>
<td>Must be RSA 2048 or stronger</td>
</tr>
<tr>
<td>Microsoft KEK CA</td>
<td>Microsoft</td>
<td>Allows updates to db and dbx</td>
</tr>
<tr>
<td>Microsoft Windows Production CA</td>
<td>db</td>
<td>This CA in the allowed signature database (db) allows Windows 8 to boot</td>
</tr>
<tr>
<td>Forbidden Signature database</td>
<td>dbx</td>
<td>List of bad/compromised keys, CAs images from Microsoft</td>
</tr>
</tbody>
</table>

**+ Required for secure firmware updates**

<table>
<thead>
<tr>
<th>Key/db name</th>
<th>Owner</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure firmware update key</td>
<td>OEM</td>
<td>Should differ from PK. Must be RSA 2048 or stronger</td>
</tr>
</tbody>
</table>

**UEFi secure boot and TPM**

- Observe that actually the TPM is not needed for secure boot if one skips the requirement to support attestation. (one basically has no secrets to protect then).


**Manufacturing Requirements**

- Secure boot requires a computer that meets the UEFI Specifications Version 2.3.1, Errata C or higher.
- Secure boot is supported for UEFI Class 2 and Class 3 computers. For UEFI Class 2 computers, when Secure boot is enabled, the compatibility support module (CSM) must be disabled so that the computer can only boot authorized, UEFI-based operating systems.
- Secure boot does not require a Trusted Platform Module (TPM).

To enable kernel-mode debugging, enable TESTSHELL, or to disable NMI, you must disable Secure Boot. For more info, see Windows 8 Secure Boot Key Creation and Management Guidance.

**Optional Keys for Secure Boot**

**+ Recommended for non WinRT Systems**

<table>
<thead>
<tr>
<th>Key/db name</th>
<th>Owner</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft UEFI driver signing CA</td>
<td>db</td>
<td>Microsoft signer of 3rd party UEFI binaries via DevCenter program</td>
</tr>
</tbody>
</table>

**+ Optional for Customization**

<table>
<thead>
<tr>
<th>Key/db name</th>
<th>Owner</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>OEM or 3rd party KEK</td>
<td>OEM/3rdP</td>
<td>Allows db/dbx updates, e.g. for alternate OS or Trusted 3rd party</td>
</tr>
<tr>
<td>OEM or 3rd party CA</td>
<td>OEM/3rdP</td>
<td>Allows 3rd party OS or drivers signed by Trusted 3rd party</td>
</tr>
<tr>
<td>Image Hashes</td>
<td>db</td>
<td>Hashes of images on PC that are allowed to execute even if not signed,</td>
</tr>
<tr>
<td>Forbidden Signature Database</td>
<td>dbx</td>
<td>List of bad/compromised keys, CAs images from OEM or partner</td>
</tr>
</tbody>
</table>

**OEM’s role**

- Thus the OEM generates and own the PK secret key.
  - Basically the OEM can decide what can be loaded/booted is defined in the boot policy

- However, Microsoft, can demand the OEM boot policy to comply with Microsoft requirements if the OEM want to run Microsoft software (say Windows 8)
UEFi checklist for OEMs

As explained by Microsoft

- Define your security strategy
- Identify roles
- Procure server and hardware for key management
  - Recommended solution – network or standalone HSM
  - Consider whether you will need one or several HSM’S for high availability and also your key back up strategy
- Set policy for how frequently will you be rekeying keys
- Have a contingency plan for Secure Boot Key compromise
- Identify how many PK and other keys will you be generating
- Use HSM to pre-generate secure boot related keys and certificates
- Get the Microsoft KEK and other Secure Boot related keys and certificates
- Sign UEFI drivers
- Update firmware with Secure Boot keys based on the system type
- Run tests including WHCK Secure Boot tests
- Deploy > Refine > Deploy > Refine...

References

- Microsoft Connect http://connect.microsoft.com/
- MSDN: http://msdn.microsoft.com/
  - Search on keyword “Secure Boot”
  - http://www.microsoft.com/security
- UEFI 2.3.1. Specification errata C: http://www.uefi.org
- Tianocore: http://www.tianocore.sourceforge.net

TCG/TPM Issues 1

- Often seen as DRM enabler (no longer true)
- Privacy issues
  - TPM has unique embedded key
  - Get pseudonyms in anonymous way (e.g. anonymous credentials)
- User loses control over own computer
  - Secure boot: refuse to boot own compiled open source kernel
  - Disable reverse engineering: need for hardware hack
- In a mobile device/phone we have multiple-stake holders
- TPM is complex

Mobile Trusted Module (MTM)
TCG/TPM Issues 2

- Current TPM 1.2 cannot be remotely managed.
  - IT-departments do not like this. Hard to use
- Integration of TPM in a product is not the same for different computer vendors (even true for PC/laptops)
- Why trust a US standard?

TPM V2.0

- New TPM version is on its way which differs in many aspects from the current TPM 1.2
  - Most notably
    - Compliant to current US NIST standards (1.2 is no any more)
    - Support for symmetric key algorithms
    - The cipher suite is in principle replaceable (e.g. Chinese version)
    - More powerful method to realize conditions for use of objects, functions of the TPM

Chinese TPM

- Chinese authorities: import control on equipment which uses crypto. Permission is needed.
- There is a Chinese TPM called TCM which has certain approaches in common with TPM v 2.

Special trusted computing devices

- Secure Cryptoprocessors
  - Dedicated microprocessor system with physical protection features
    - Tamper-detecting and tamper-evident containment
    - Automatic zeroization of secrets in the event of tampering
    - Chain of trust boot-loader which authenticates the operating system before loading it
    - Chain of trust operating system which authenticates application software before loading it
    - Hardware-based capability registers, implementing a one-way privilege separation model
    - Possibly battery backup
  - Smart cards
  - **History**: IBM 4758 PCI Cryptographic processor (PKCS#11 interface)
    - Attacked 2001 by PhD students at Cambridge University
    - Replaced by IBM 4764
- Security Modules:
  - Like Secure cryptoprocessors but more capable as system.
HSM
(Hardware (or Host) Security Modules)
- Special Computers with high-grade protection with purpose to store critical information and keys
- Some can be small – pci card/smartcard like
- Some can be large – desktop box like
- Interfaces
  - often non-standard or
  - PKCS#11

Smartcard architecture (example Infineon SLE66)
- Smart card IC processor product with advanced security mechanisms (cryptographic engine, physical protection)
- Certified according to EAL5

Elements of Java Card Application:
Realizing a secure system is very tricky

Intermezzo

Study questions

- Why not only depend on closed platforms for building mission critical systems?
- Is it possible to implement a secure system using only sw?
- What is the security relevance of virtualisation?
- What are the main security roles of a hypervisor?
- Why runs the guest OS kernel not in privileged mode?
- Explain the role of the app signatures and compare it with signed applets in Java
- What makes SELinux a special OS? It's pros and cons.
- Explain the three main use cases for a TPM.
- Which keys are permanently stored in a TPM?
- In connection to TPM use what is a sealed key?
- What is the difference between SRTM and DRTM?
- What is attestation?
- What keys are used for attestation?

- What are enclaves in the SGX Technology
- Secure boot vs authenticated boot
- Explain main security features of UEFI
- Describe ARM TrustZone
- What is an HSM and give two use cases.
- By what mechanisms are applications sandboxed in Android and in Java?