TRUSTED COMPUTING

CARSTEN WEINHOLD, HERMANN HÄRTIG
Goal: Understand principles of:

- Authenticated booting, relation to (closed) secure booting
- Remote attestation
- Sealed memory
- Dynamic root of trust, late launch
- Protection of applications from the OS
- Point to implementation variants (TPM, SGX, TrustZone)
Non-Goal:

- Deep discussion of cryptography
- Lots of details on TPM, TCG, TrustZone, SGX, ...
  → Read the documents once needed
Trusted Computing Base (TCB):
- Set of all components *(hardware, software, procedures)* that must be relied upon to enforce a security policy

Trusted Computing (Technology):
- Particular technology, often comprised of authenticated booting, remote attestation, and sealed memory

Trusted Computing Group (TCG):
- Consortium behind a specific trusted computing standard
Key Goals of Trusted Computing

- Prevent certain software from running
- Which computer system do I communicate with?
- Which stack of software is running ...
  - ... in front of me?
  - ... on my server somewhere?
- Restrict access to certain secrets to certain software
- Protect an application from the OS
Digital Rights Management (DRM):

- Vendor sells content
- Vendor creates key, encrypts content with it
- Client downloads encrypted content, stores it locally
- Vendor sends key, but wants to ensure that only specific software can use it
- Has to work also when client is offline
- Vendor does not trust the client
Virtual machine by cloud provider:

- Client rents compute and storage (server / container / virtual machine)
- Client provides its own operating system (OS)
- Needs to ensure that provided OS runs
- Needs to ensure that provider cannot access data
- **Customer does not trust cloud provider**
Industrial Plant Control:

- Remote operator sends commands, keys, ...
- Local technicians occasionally run maintenance / selftest software, install software updates, ...
- Local technicians are not trusted
Anonymity Service:

- Provides anonymous communication over internet (e.g., one node in mix cascade)
- User wants to be sure that the proxy actually anonymizes incoming and outgoing connections
- Anonymity-service provider not trusted
Trusted Computing Terminology

Measuring:
- Process of obtaining metrics of platform characteristics
- Example: Hash code of software

Attestation:
- Vouching for accuracy of (measured) information

Sealed Memory:
- Binding information to a (software) configuration
Notation: Hashes and Keys

**Hash:** \( H(M) \)
- Collision-resistant hash function \( H \) applied to content \( M \)

**Asymmetric key pair:** \( E_{\text{pair}} \) consisting of \( E_{\text{priv}} \) and \( E_{\text{pub}} \)
- Asymmetric private/public key pair of entity \( E \), used to either **conceal** (encrypt) or **sign** some content
- \( E_{\text{pub}} \) can be published, \( E_{\text{priv}} \) must be kept secret

**Symmetric key:** \( E \)
- Symmetric key of entity \( E \), must be kept secret ("secret key")
Notation: Result of Operations

Digital Signature: \{M\}E_{priv}
- \text{E}_{\text{pub}} \text{ can be used to verify that } \text{E} \text{ has signed } M
- \text{E}_{\text{pub}} \text{ is needed and sufficient to check signature}

Concealed Message: \{M\}E_{pub}
- Message \text{M} concealed (encrypted) for \text{E}
- \text{E}_{\text{priv}} \text{ is needed to unconceal (decrypt) } M
Identification of Software

Example: program vendor FooSoft (FS)

Software identity ID must be known

Two ways to identify software:

- By hash: $\text{ID}_{\text{Program}} = \text{H}(\text{Program})$

- By signature: $\{\text{Program}, \text{ID}_{\text{Program}}\}_{\text{FS}_{\text{priv}}}$
  - Signature must be available (e.g., shipped with program)
  - Use $\text{FS}_{\text{pub}}$ to check signature
  - $(\text{H}(\text{Program}), \text{FS}_{\text{pub}})$ can serve as $\text{ID}_{\text{Program}}$
Secure Booting ("Burn in the OS")

**OS stored in read-only memory (flash)**

Hash $H(\text{OS})$ in TRB NVM, preset by manufacturer:
- Load OS code, compare $H(\text{loaded OS code})$ to preset $H(\text{OS})$
- Abort if different

**Public key $FS_{pub}$ in TRB NVM, preset by manufacturer:**
- Load OS code, check signature of loaded OS code using $FS_{pub}$
- Abort if check fails
Authenticated Booting ("Choose your OS")

**Steps:**

1) Preparation by OS and TRB vendors
2) Booting & measuring
3) Remote attestation
1a) Preparation by OS vendor:
- Certifies: \{"a valid OS", H(\text{OS})\}_{\text{OSVendors}_{\text{priv}}}
- Publishes identifiers: \text{OSVendors}_{\text{pub}} and H(\text{OS})
Tamper-Resistant Black Box (TRB)

TRB (Conceptual View)

Processor

Non-Volatile Memory (NVM)

Memory

Platform Configuration Register (PCR)
1b) Preparation by TRB vendor:

- TRB generates "Endorsement Key" pair: $\text{EK}_{\text{pair}}$
- TRB Stores $\text{EK}_{\text{priv}}$ in TRB NVM
- TRB publishes $\text{EK}_{\text{pub}}$
- TRB vendor certifies: 
  \{
    "a valid EK", $\text{EK}_{\text{pub}}$
  \}$_{\text{TRBVendor}_{\text{priv}}}$
2) Booting & measuring:

- TRB resets
- TRB computes ("measures") hash $H(\text{OS})$ of loaded OS
- Records $H(\text{OS})$ in platform configuration register PCR
- TRB starts OS

**Note:** PCR not directly writable
3) Remote Attestation:

- Remote computer sends "challenge": NONCE
- TRB signs $\{\text{NONCE, PCR}\} \cdot \text{EK}_{\text{priv}}$ (evidence or attestation report) and sends it to "challenger"
- Challenger evaluates evidence: checks signature, decides if OS identified by $H(\text{OS})$ in PCR is OK
Problem: Time-of-check, time-of-use (TOCTOU) attack possible

Solution: Create new key pair for protecting data until next reboot
At each boot, TRB does the following:

- Computes $H(OS)$ and records it in PCR
- Creates two key pairs for the booted, currently active OS:
  - $ActiveOSAuthK_{\text{pair}}$ /* for authentication (signing) */
  - $ActiveOSConK_{\text{pair}}$ /* for concealing (encryption) */
- TRB certifies:
  $\{ActiveOSAuthK_{\text{pub}}, ActiveOSConK_{\text{pub}}, H(OS)\}EK_{\text{priv}}$
- Hands over to booted OS, to be used like "session keys"
Remote Attestation:

- Challenger sends: NONCE
- Currently booted, active OS generates response:
  \[\{\text{ActiveOSConK}_{\text{pub}}, \text{ActiveOSAuthK}_{\text{pub}}, \text{H(OS)}\}_{\text{EK}}_{\text{priv}}\]
  \[\{\text{NONCE}\}_{\text{ActiveOSAuthK}}_{\text{priv}}\]

Client sends data over secure channel:

- \[\{\text{data for active OS}\}_{\text{ActiveOSConK}}_{\text{pub}}\]
Authenticated booting and remote attestation as presented are secure, if:

1) TRB can protect $E_{K_{priv}}, PCR$

2) OS can protect "Active OS" keys

3) Rebooting destroys content of:
   - PCR
   - “Active OS keys” in memory
Software Stacks and Trees

ROOT

Boot Loader

OS Kernel

OS Services

Application

ROOT

Boot Loader

OS Kernel

Some Services

Application 1

ROOT

Boot Loader

OS Kernel

Other Services

Application 2

Some Services

Application 1

Other Services

Application 2
Software Stacks and Trees

Two Concerns:

- Remote attestation of one process (leaf in tree)
- Very large Trusted Computing Base (TCB) for booting (including device drivers, etc.)
Extend operation:

$$\text{PCR}_n = H(\text{PCR}_{n-1} \ || \ \text{new component})$$  \quad [\text{PCR}_0=0]

Software Stack:
- 1 PCR value $\text{PCR}_n$ after n components have been measured

Software "Tree":
- 1 PCR value $\text{PCR}_n$ for each leaf at end of a branch of length n
- Needs multiple PCRs (1 per branch) that share state from Root to $\text{PCR}_{\text{OS}}$, then diverge to leaves at $\text{PCR}_{\text{App1}}$, $\text{PCR}_{\text{App}}$, ...
Key pairs per level of tree:

- OS controls applications → generate additional key pair per application
- OS certifies:
  - \{Application 1, App1K_{pub}\}ActiveOSAuth_{priv}
  - \{Application 2, App2K_{pub}\}ActiveOSAuth_{priv}
Late Launch/Dynamic Root of Trust

**Problem:** huge software to boot system

**Solution:** late launch

- Use arbitrary software to start system and load all software
- Provide specific instruction to enter “secure mode”
  - Put hardware in secure state (stop all processors, I/O, ...)
  - Measure software and record into PCR
- **AMD (skinit):** hashes arbitrary "secure loader" and start it
- **Intel (senter):** starts boot code (must be signed by Intel)
The Need for Trusted Storage

Use case from earlier example:
- Send data over secure channel after remote attestation
- Bind that data to software configuration via TRB

Problem: How to work with this data when offline?
- Must store data for time after reboot
- For example for DRM: bind decryption key for downloaded movie to specific machine with specific OS
Sealed Memory Principle

<table>
<thead>
<tr>
<th>ExpectPCR</th>
<th>Data</th>
<th>NVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>H(Linux)</td>
<td>Linux</td>
<td></td>
</tr>
<tr>
<td>H(PlayOS)</td>
<td>PlayOS</td>
<td></td>
</tr>
<tr>
<td>H(Windows)</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>H(L4)</td>
<td>L4</td>
<td></td>
</tr>
</tbody>
</table>

Add/remove/read/write "Sealed Memory" slots

Can be accessed by currently active OS

Other slots inaccessible due to PCR mismatch
Sealed Memory Principle

Add/remove/read/write "Sealed Memory" slots

Can be accessed by currently active OS

Other slots inaccessible due to PCR mismatch

<table>
<thead>
<tr>
<th>ExpectPCR</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>H(Linux)</td>
<td>Linux</td>
</tr>
<tr>
<td>H(PlayOS)</td>
<td>PlayOS</td>
</tr>
<tr>
<td>H(Windows)</td>
<td>Windows</td>
</tr>
<tr>
<td>H(L4)</td>
<td>L4</td>
</tr>
</tbody>
</table>

PCR

H(OS) == H(PlayOS)
## Sealed Memory Principle

Add/remove/read/write "Sealed Memory" slots

<table>
<thead>
<tr>
<th>ExpectPCR</th>
<th>Data</th>
<th>NVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>H(Linux)</td>
<td>Linux</td>
<td></td>
</tr>
<tr>
<td>H(PlayOS)</td>
<td>PlayOS</td>
<td></td>
</tr>
<tr>
<td>H(Windows)</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>H(L4)</td>
<td>L4</td>
<td></td>
</tr>
</tbody>
</table>

Can be accessed by currently active OS

Other slots inaccessible due to PCR mismatch
Sealed Memory Principle

Add/remove/read/write "Sealed Memory" slots

Other slots inaccessible due to PCR mismatch

Can be accessed by currently active OS

<table>
<thead>
<tr>
<th>ExpectPCR</th>
<th>Data</th>
<th>NVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>H(Linux)</td>
<td>Linux</td>
<td></td>
</tr>
<tr>
<td>H(PlayOS)</td>
<td>PlayOS</td>
<td></td>
</tr>
<tr>
<td>H(Windows)</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>H(L4)</td>
<td>L4</td>
<td></td>
</tr>
</tbody>
</table>

PCR:

\[ H(\text{OS}) = H(L4) \]
TRB creates secret symmetric key $\text{SealK}$

TRB encrypts (Seal) and decrypts (Unseal) data using $\text{SealK}$

Seal(ExpectPCR, data) $\rightarrow \{\text{ExpectPCR, data}\} \text{SealK}$

Unseal(\{ExpectPCR, data\}SealK) $\rightarrow$ data

iff current PCR $==$ ExpectPCR
else abort without releasing data
Sealed Memory Flexibility

- Sealed (encrypted) data can be stored outside of TRB, allows to keep NVM small
- When sealing, arbitrary "expected PCR" values can be specified (e.g., future version of OS, or entirely different OS)

\[
\begin{align*}
\{H(\text{Linux}), \text{Linux}\} & \text{SealK} \\
\{H(\text{Windows}), \text{Windows}\} & \text{SealK} \\
\{H(\text{PlayOS}), \text{PlayOS}\} & \text{SealK} \\
\{H(L4), \text{L4}\} & \text{SealK}
\end{align*}
\]
Example

- **Windows:** Seal \( H(\text{PlayOS}), \text{PlayOS\_Secret} \) → **sealed\_message** (store it on disk)

- **L4:** Unseal (sealed\_message)
  → PlayOS, PlayOS\_Secret
  → ExpectPCR != PlayOS
  → **abort**

- **PlayOS:** Unseal(sealed\_message)
  → PlayOS, PlayOS\_Secret
  → ExpectPCR == PlayOS
  → **emit PlayOS\_Secret**
Tamper Resistant Black Box?

**Ideally:** includes CPU, Memory, ...

**In practice:**
- Additional physical protection (e.g., IBM 4758, → Wikipedia)
- Hardware support:
  - Trusted Platform Module (TPM): requires careful design to allow firmware updates, etc.
  - Add a new privilege mode: Intel SGX, Arm TrustZone,...
  - Add encrypted VMs: Intel TDX, AMD SEV, Arm CCA, ...
TCG PC Platform: Trusted Platform Module (TPM)

- CPU
- Memory
- BIOS
- TPM

- CPU/mem interconnect
- PCIe
- Low Pin Count Bus
Trusted Platform Module

Non-Volatile Memory
- SRK
- EK
- Counters
- Firmware

RAM
- AIK
- AIK
- SK

I/O

SHA-1 / SHA-2
- RSA
- Key Gen
- Random Number Generator

SK

Embedded Processor

AIK

AES

RSA

Key Gen
Protection of Application

**Principle Method:**
- Isolate critical software
- Rely on small Trusted Computing Base (TCB)

**Ways to implement the method:**
- Small OS kernels: microkernels, separation kernels, ...
- Hardware / microcode support
Trusted Computing Base: Big OS

Hardware

Linux

X11

App

App
Trusted Computing Base: Small OS

Small Microkernel-based OS / Hypervisor

Hardware

Linux

X11

App

Helper

App
Trusted Computing Base: Only Hardware?

- App
- X11
- Linux

Hardware

Helper

Software?
ARM TrustZone

Normal World

- PL0: App, App
- PL1: OS Kernel, OS Kernel
- PL2: Hypervisor

Secure World

- Trusted Services
  - App?
- Trusted OS Kernel

Monitor
“Enclaves” for applications:

- Established per special SGX instructions
- Measured by CPU
- Provides controlled entry points
- Resource management via untrusted OS
Applications executing in enclaves benefit from hardware memory protection (also against OS and hypervisor); they are measured, seal and unseal data and request quote for remote attestation, all through CPU instructions (which are themselves entry points to firmware implemented as x86-64 code).
Apple Secure Enclave Processor

- App
- GUI, etc.
- iOS Kernel

- Service
- Service
- L4-Based OS
- Secure Enclave Processor

Application Processor
Important Foundational Paper:


Technical documentation:

- Trusted Computing Group's specifications
  https://www.trustedcomputinggroup.org
- ARM Trustzone, Intel SGX vendor documentation