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
Terminology

- Secure Booting
- Measured / authenticated Booting
- (Remote) Attestation
- Sealed Memory
- Late Launch / dynamic root of trust
- Trusted Computing (Group)
- Trusted Computing Base

Beware of terminology chaos!
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**
Usage Examples (2)

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
Usage Examples (3)

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)
- Law enforcement can request introduction of surveillance functionality (software change)
- Anonymity-service provider not trusted
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
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")
Digital Signature: \( \{M\}E_{\text{priv}} \)
- \( E_{\text{pub}} \) can be used to verify that \( E \) has signed \( M \)
- \( E_{\text{pub}} \) is needed and sufficient to check signature

Concealed Message: \( \{M\}E_{\text{pub}} \)
- Message \( M \) concealed (encrypted) for \( E \)
- \( E_{\text{priv}} \) is needed to unconceal (decrypt) \( M \)
Example: program vendor FooSoft (FS)

Software identity ID must be known

Two ways to identify software:

- By hash: $\text{ID}_{\text{Program}} = H(\text{Program})$
- By signature: $\{\text{Program}, \text{ID}_{\text{Program}}\}_{FS_{\text{priv}}}$
  - Signature must be available (e.g., shipped with program)
  - Use $FS_{\text{pub}}$ to check signature
  - $(H(\text{Program}), FS_{\text{pub}})$ can serve as $\text{ID}_{\text{Program}}$
Tamper-Resistant Black Box (TRB)

TRB (Conceptual View)

- Processor
- Memory
- Non-Volatile Memory (NVM)
- Platform Configuration Register (PCR)
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 $F_{\text{pub}}$ in TRB NVM, preset by manufacturer:
- Load OS code, check signature of loaded OS code using $F_{\text{pub}}$
- 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{OSVendor}}^{\text{priv}}
- Publishes identifiers: \text{OSVendor}^{\text{pub}} and H(\text{OS})
Tamper-Resistant Black Box (TRB)

TRB (Conceptual View)

- Processor
- Memory
- Non-Volatile Memory (NVM)
- Platform Configuration Register (PCR)
1b) Preparation by TRB vendor:
- TRB generates "Endorsement Key" pair: $E_{K_{\text{pair}}}$
- TRB Stores $E_{K_{\text{priv}}}$ in TRB NVM
- TRB publishes $E_{K_{\text{pub}}}$
- TRB vendor certifies: 
  \{"a valid EK", $E_{K_{\text{pub}}}$\} 
  $TRB_{\text{Vendor}_{\text{priv}}}$
2) Booting & measuring:

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

- **Note:** $\text{PCR}$ not directly writable, more on this later
3) Remote Attestation:

- Remote computer sends "challenge": NONCE
- TRB signs $\{\text{NONCE, PCR}\}E_{\text{Kpriv}}$ and sends it to "challenger"
- Challenger checks signature, decides if OS identified by $H(\text{OS})$ in reported signed 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:
  \[ \{ActiveOSAuthK_{\text{pub}}, \, ActiveOSConK_{\text{pub}}, \, H(OS)\}EK_{\text{priv}} \]
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:

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

1) TRB can protect $\text{EK}_{\text{priv}}$, PCR
2) OS can protect "Active OS" keys
3) Rebooting destroys content of:
   - PCR
   - “Active OS keys” in memory
Software Stacks and Trees

Application

GUI et al.

OS-Kernel

Boot-Loader

ROOT

Application 1

Some Services

OS-Kernel

Boot-Loader

ROOT

Application 2

Other Services

OS-Kernel

Boot-Loader

ROOT
Two Concerns:

- Very large Trusted Computing Base (TCB) for booting (including device drivers, etc.)
- Remote attestation of one process (leaf in tree)
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 \textbf{Root} to \( \text{PCR}_{\text{OS}} \), then diverge to leafs at \( \text{PCR}_{\text{App1}}, \text{PCR}_{\text{App}}, \ldots \)
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}
**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)
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>
</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>

- **Add/remove/read/write** "Sealed Memory" slots
- Can be accessed by currently active OS
- Other slots inaccessible due to PCR mismatch
Sealed Memory Principle

<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>

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
Sealed Memory Principle

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

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

Other slots inaccessible due to PCR mismatch

Can be accessed by currently active OS
- TRB creates secret symmetric key **SealK**
- TRB encrypts (Seal) and decrypts (Unseal) data using **SealK**
- \(\text{Seal}(\text{ExpectPCR}, \text{data}) \rightarrow \{\text{ExpectPCR}, \text{data}\}\text{SealK}\)
- \(\text{Unseal}(\{\text{ExpectPCR}, \text{data}\}\text{SealK}) \rightarrow \text{data}\)
  - iff current \(\text{PCR} = \text{ExpectPCR}\)
  - 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)

\[
\{H(\text{Linux}), \text{Linux}\}\{\text{SealK}\}
\{H(\text{Windows}), \text{Windows}\}\{\text{SealK}\}
\{H(\text{PlayOS}), \text{PlayOS}\}\{\text{SealK}\}
\{H(L4), \text{L4}\}\{\text{SealK}\}
\]
Windows: Seal (H(PlayOS), 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: ARM TrustZone, Intel SGX
  - Add encrypted VMs: Intel TDX, AMD SEV, ...
TCG PC Platform: Trusted Platform Module (TPM)
Trusted Platform Module

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

- RAM
  - AIK
  - AIK
  - SK
  - Embedded Processor

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

- Random Number Generator
- AES
- Key Gen
- RSA
- SHA-1 / SHA-2
- AIK
- AIK
- SK
- Embedded Processor
- Firmware
- Counters
- EK
- SRK
- Non-Volatile Memory

I/O

SHA-1 / SHA-2

RSA

Key Gen

Random Number Generator

AES

I/O

SHA-1 / SHA-2

RSA

Key Gen

Random Number Generator

AES
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
Trusted Computing Base: Small OS

- App
- X11
- Linux

Small Microkernel-based OS

Hardware

- App
- Helper
Trusted Computing Base: Only Hardware?

- App
- X11
- Linux

Helper

Hardware
ARM TrustZone

Normal World

Secure World

PL0

App

App

PL1

OS Kernel

OS Kernel

PL2

Hypervisor

Monitor

Trusted Services

Trusted OS Kernel

App?
Intel SGX

Diagram showing the following components:
- Hardware
- Helper
- Linux
- X11
- App
- App Enclave
“Enclaves” for applications:

- Established per special SGX instructions
- Measured by CPU
- Provide 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**
  - 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