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 against 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 (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)
- Law enforcement can request introduction of surveillance functionality (software change)
- 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")
Digital Signature: \(\{M\}E_{priv}\)
- \(E_{pub}\) can be used to verify that \(E\) has signed \(M\)
- \(E_{pub}\) is needed and sufficient to check signature

Concealed Message: \(\{M\}E_{pub}\)
- Message \(M\) concealed (encrypted) for \(E\)
- \(E_{priv}\) 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: ID_{Program} = H(\text{Program})
- By signature: \{\text{Program}, ID_{Program}\}_{FS_{priv}}
  - Signature must be available (e.g., shipped with program)
  - Use FS_{pub} to check signature
  - (H(\text{Program}), FS_{pub}) can serve as ID_{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: \{\text{“a valid OS”, } H(\text{OS})\}\text{OSVendor}_\text{priv}
- Publishes identifiers: \text{OSVendor}_\text{pub} \text{ 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: $\text{EK}_{\text{pair}}$
- TRB Stores $\text{EK}_{\text{priv}}$ in TRB NVM
- TRB publishes $\text{EK}_{\text{pub}}$
- TRB vendor certifies:
  $\{\text{"a valid EK"}, \text{EK}_{\text{pub}}\}_{\text{TRBVendor}_{\text{priv}}}$
2) Booting & measuring:

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

**Note:** PCR not directly writable, more on this later
3) Remote Attestation:

- Remote computer sends "challenge": NONCE
- TRB signs \{NONCE, PCR\}E_{K_{priv}} and sends it to "challenger"
- Challenger checks signature, decides if OS identified by $H(OS)$ 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
Booting (Considering Reboot)

At each boot, TRB does the following:

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

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

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 $E_{K_{\text{priv}}}$, PCR

2) OS can protect "Active OS" keys

3) Rebooting destroys content of:
   - PCR
   - "Active OS keys" in memory
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 \text{Root} to \( \text{PCR}_{\text{OS}} \), then diverge to leafs at \( \text{PCR}_{\text{App}_1}, \text{PCR}_{\text{App}_2}, ... \)
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 “root of trust” 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

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>

**PCR**

- $H(OS) = H(PlayOS)$

**Sealed Memory**

- 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(Windows)
Sealed Memory Principle

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

Other slots inaccessible due to PCR mismatch

Can be accessed by currently active OS

ExpectPCR | Data | NVM
---|---|---
H(Linux) | Linux | 
H(PlayOS) | PlayOS | 
H(Windows) | Windows | 
H(L4) | L4 | 

PCR = H(OS) = H(L4)
- TRB creates secret symmetric key \textit{SealK}
- TRB encrypts (\textit{Seal}) and decrypts (\textit{Unseal}) data using \textit{SealK}
- \textit{Seal}(\text{ExpectPCR, data}) \rightarrow \{\text{ExpectPCR, data}\}\textit{SealK}
- \textit{Unseal}(\{\text{ExpectPCR, data}\}\textit{SealK}) \rightarrow \text{data}
  - \textbf{iff} current PCR == \text{ExpectPCR}
  - \textbf{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)

\[
\{\text{H}(\text{Linux}), \text{Linux}\} \text{SealK} \quad \{\text{H}(\text{PlayOS}), \text{PlayOS}\} \text{SealK} \\
\{\text{H}(\text{Windows}), \text{Windows}\} \text{SealK} \quad \{\text{H}(\text{L4}), \text{L4}\} \text{SealK}
\]
Windows: Seal \( H(\text{PlayOS}, \text{PlayOS}_\text{Secret}) \)  
\[ \rightarrow \text{sealed} \_ \text{message} \] (store it on disk)

L4: Unseal (sealed\_message)  
\[ \rightarrow \text{PlayOS}, \text{PlayOS}_\text{Secret} \]  
\[ \rightarrow \text{ExpectPCR} \neq \text{PlayOS} \]  
\[ \rightarrow \text{abort} \]

PlayOS: Unseal(sealed\_message)  
\[ \rightarrow \text{PlayOS}, \text{PlayOS}_\text{Secret} \]  
\[ \rightarrow \text{ExpectPCR} = \text{PlayOS} \]  
\[ \rightarrow \text{emit} \text{PlayOS}_\text{Secret} \]
Tamper Resistant Black Box?

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

**In practice:**

- Additional physical protection (e.g., IBM 4758, → Wikipedia)
- Hardware support:
  - Separate “Trusted Platform Module (TPM)”: often insufficiently integrated, TRB functionality breaks when replacing BIOS, etc.
  - Add a new privilege mode: ARM TrustZone
  - Shielded more for user processes: Intel SGX
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

AIK
- SK
- AES
- Random Number Generator
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

- Hardware
- Linux
- X11
- App
- Helper
Trusted Computing Base: Only Hardware?

- App
- X11
- Linux
- Helper
- Hardware
“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 (partially implemented in CPU microcode).
iPhone

- Application Processor
- iOS Kernel
- GUI, etc.
- App

Service
- L4-Based OS
- Secure Enclave Processor
- Service
Important Foundational Paper:


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