SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment

Jinsoo Jang
Sunjune Kong
Minsu Kim
Daegyeong Kim
Brent Byunghoon Kang

KAIST
Need for a Trusted Execution Env.

• Rich Execution Environment (REE)
  ▪ For versatility and richness
  ▪ Runs rich OSes: Android, Windows

• Trusted Execution Environment (TEE)
  ▪ Protection of Assets
    ✓ User credentials
    ✓ Crypto keys
  ▪ Safe execution of security critical services
    ✓ Mobile Banking
    ✓ Mobile Payment
    ✓ Digital Right Management
ARM TrustZone

• Provides a TEE for embedded devices
• Communication channel:
  • Invoking SMC instruction with arguments

Rich Execution Environment

- User Process
- Shared Memory
- REE Kernel
- TEE Driver
- Gatekeeper between the REE and the TEE

Trusted Execution Environment

- Monitor Mode
- TEE Services & Resources
- Monitor
- TEE Kernel

Communications channel

Placing arguments

Invoking an SMC
Weakness of TrustZone

- Communication channel is vulnerable
  - No way to authenticate the messages from the REE
  - Integrity of the messages is not guaranteed
Attack Model

• Attackers have kernel privileges

• Attackers exploit the communication channel to
  ▪ access to critical resources in the TEE
  ▪ perform a brute force attack against services in the TEE
  ▪ analyze the behaviors in the TEE
  ▪ find out the vulnerability of the TEE services
Our Goal & Assumption

• Securing the channel between the REE and the TEE
  ▪ Provide a session key to the REE processes
  ▪ Protect the session key from attackers

• Assumption
  ▪ Secure boot
  ▪ Critical resources are properly classified and located in TrustZone
  ▪ A list of pre-authorized REE processes is maintained in TrustZone
  ▪ Kernel’s static region in the REE is protected by active monitoring
    ✓ TZ-RKP (CCS ‘14), SPROBES (MoST ‘14)
SeCReT - Overview

- Framework to provide and protect the session key in the REE
Session Key Life Cycle (1/5)

• Secure boot
  ▪ Calculate the code hash based on the granularity of the small page

Rich Execution Environment

User mode

Kernel mode

Per pages

Hash

Temporally loaded pre-authorized Processes

Save the hash values

Trusted Execution Environment

Monitor mode

SeCRET_M

SeCRET Hash Storage

SeCRET_T

SeCRET_M

Hash

Save the hash values
Session Key Life Cycle (2/5)

- Execution of the pre-authorized process
  - Create an APC for the process
Session Key Life Cycle (3/5)

- Session-key creation
  - Set the access permission & generate the key value
• Using the session key
  ▪ Access control based on the occurrence of a data-abort exception
Session Key Life Cycle (5/5)

• Process termination
  ▪ Remove the APC of the process
How to Protect the Key?

• SeCReT interposes with every mode switch

• Access control to the session key
  ▪ Key assignment on legitimate access to the key
  ▪ Key flush in every mode switch to kernel

• Coarse-grained Control-flow Integrity
  ▪ Shadow stacks for critical registers
Interposition with Mode Switches

• SeCReT is enabled by exception-vector remapping
• Interposition with every mode switch between user and kernel
Interposition with Mode Switches

- SeCReT is enabled by exception-vector remapping
- Interposition with every mode switch between user and kernel
Interposition with Mode Switches

- **SeCReT_EXV**: New exception vector for SeCReT
  - Trampoline code is inserted to the starting point of
    - Handler code for user mode exceptions (User $\rightarrow$ Kernel)
    - Switch-to-user code (Kernel $\rightarrow$ User)
Interposition with Mode Switches

- **SeCReT_EXV**: New exception vector for SeCReT
  - Trampoline code is inserted to the starting point of
    - Handler code for user mode exceptions (User $\rightarrow$ Kernel)
    - Switch-to-user code (Kernel $\rightarrow$ User)
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

Control-flow for the access control to the session key

User Process

```assembly
...  
0x836c: MOV R3,#0  
0x8370: LDR R3,[R4]  
0x8374: XOR R1, R3  
...  
0x8400: SVC #0  
...  
```

Exception Vector

- SeCReT_T
- SeCReT_T
- SeCReT_T

SeCReT_M

- APC
- SessionKeyInfo
- KeyRequestFlag
- Fault Verifier
- Access Control
Access Control to the Key

• Key assignment
  ▪ Data abort in the page reserved for key-assignment
  ▪ Hash-check for code area

• Key flush
  ▪ Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

---

Control-flow for the access control to the session key
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- **Key assignment**
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area

- **Key flush**
  - Every mode switch to kernel

---

**Control-flow for the access control to the session key**
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

*Control-flow for the access control to the session key*
Access Control to the Key

• Key assignment
  ▪ Data abort in the page reserved for key-assignment
  ▪ Hash-check for code area

• Key flush
  ▪ Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- **Key assignment**
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- **Key flush**
  - Every mode switch to kernel

**Control-flow for the access control to the session key**
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

**Control-flow for the access control to the session key**

User Process

0x836c: MOV R3, #0
0x8370: LDR R3, [R4]
0x8374: XOR R1, R3

Exception Vector

SeCreT_M

- APC
- SessionKeyInfo
- KeyRequestFlag
- Fault Verifier
- Access Control

Other exceptions

Key assignment
- Data abort in the page reserved for key-assignment
- Hash-check for code area

Key flush
- Every mode switch to kernel
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel

Control-flow for the access control to the session key
Access Control to the Key

- **Key assignment**
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- **Key flush**
  - Every mode switch to kernel

```plaintext
0x836c: MOV R3, #0
0x8370: LDR R3, [R4]
0x8374: XOR R1, R3
```

Control-flow for the access control to the session key
Coarse-Grained CFI (1/2)

• Attackers can try to exfiltrate the key by
  • Manipulating the process’ code area
    ➔ Hash-check for code area
  • Directly mapping the protected memory area
    ➔ Page-table update is not available in the REE

• Instead, manipulating the control flow to copy the key to unprotected memory area (e.g. ROP attacks)
  ▪ Critical values (e.g. return address to user mode)
Coarse-Grained CFI (2/2)

- Protection of user-mode context

**Rich Execution Environment**

- Process accessing TEE resources
- PC: 0x91FC
- LR: 0x8700
- SP: 0xbecd...

**TrustExd Execution Environment**

- APC for current process
- Mode_Switch_Flag
- Whitelist (Signal)
- 0x11000
- ... 0x11000

**Shadow stacks for protection of critical values**

- Shadow stack (SP_U)
  - Dummy
  - Original
  - ...
  - ...

- Shadow stack (LR_U)
  - Dummy
  - Original
  - ...
  - ...

- Shadow stack (Return address)
  - Dummy
  - Original
  - ...
  - ...

- Verifier

---

**CAUTION:**

- Do not analyze or distribute this document without permission.

- The content of this document is for educational purposes only.

- The information provided is intended for use by authorized personnel only.

- Any unauthorized use of the information contained herein is strictly prohibited.

---

**Disclaimer:**

- The information contained in this document is not intended for use in any manner that would constitute a violation of any applicable laws or regulations.

- The authors and contributors disclaim any liability for any loss, damage, or expense arising from the use or application of the information contained in this document.

---

**CySecLab**

18
Coarse-Grained CFI (2/2)

• Protection of user-mode context

Rich Execution Environment

- Process accessing TEE resources
  - PC: 0x91FC
  - LR: 0x8700
  - SP: 0xbecd...

- Kernel Stack
  - General Purpose Registers
  - Stack Pointer
  - LR_U
  - Return Address

- Mode Switch

- Shadow stack for protection of critical values

Trusted Execution Environment

- APC for current process

- Shadow stacks
  - Shadow stack (SP_U)
    - Dummy
    - Original
  - Shadow stack (LR_U)
    - Dummy
    - Original

- Verifier

SeCReT_M

- Verifier
- Verification of critical values
Coarse-Grained CFI (2/2)

- Protection of user-mode context

**Rich Execution Environment**

- Process accessing TEE resources

**Trusted Execution Environment**

- APC for current process
- Mode Switch Flag
- Whitelist (Signal)
- Shadow stack (SP_U)
- Shadow stack (LR_U)

**Mode Switch**

- Kernel Stack
  - General Purpose Registers
  - Stack Pointer
  - LR_U
  - Return Address

- Trusted Execution Environment

**Shadow stacks for protection of critical values**
Coarse-Grained CFI (2/2)

- Protection of user-mode context

**Rich Execution Environment**

- Process accessing TEE resources

**Trusted Execution Environment**

- Mode Switch

**Kernel Stack**

- General Purpose Registers
  - Stack Pointer: 0xAAAA
  - LR _U: 0xBBBB
  - Return Address: 0xCCCC

**Shadow stacks for protection of critical values**

- Shadow stack (SP _U)
  - Dummy: 0xAAAA
  - Original: 0xbecd...

- Shadow stack (LR _U)
  - Dummy: 0xBBBB
  - Original: 0x8700
  - ...: ...

- Mode_Switch_Flag
  - 0x11000

- Whitelist (Signal)
  - ...: ...

- Shadow stack (Return address)
  - Dummy: 0xCCCC
  - Original: 0x9200
  - ...: ...

- APC for current process
  - APC

- SeCReT_M
  -_dummy_
Coarse-Grained CFI (2/2)

- Protection of user-mode context

**Rich Execution Environment**

- Process accessing TEE resources
- Mode Switch

**Trusted Execution Environment**

- APC for current process
- Shadow stack (SP_U)
  - Dummy: 0xAAAA
  - Original: 0xbecd...
- Shadow stack (LR_U)
  - Dummy: 0xBBBB
  - LR_U: 0x8700
  - Return Address: 0x9200
- Whitelist (Signal)
  - Signal: 0x11000

**Shadow stacks for protection of critical values**
Coarse-Grained CFI (2/2)

• Protection of user-mode context

Rich Execution Environment

<table>
<thead>
<tr>
<th>Process accessing TEE resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC: 0x91FC</td>
</tr>
<tr>
<td>LR: 0x8700</td>
</tr>
<tr>
<td>SP: 0xbecd…</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Purpose Registers</td>
</tr>
<tr>
<td>Stack Pointer: 0xbecd…</td>
</tr>
<tr>
<td>LR_U: 0x8700</td>
</tr>
<tr>
<td>Return Address: 0x9200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>APC</th>
</tr>
</thead>
<tbody>
<tr>
<td>APC for current process</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shadow stack (SP_U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy 0xAAAA</td>
</tr>
<tr>
<td>Original 0xbecd…</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shadow stack (LR_U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy 0xBBBB</td>
</tr>
<tr>
<td>Original 0x8700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Whitelist (Signal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0x11000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode_Switch_Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

| Trusted Execution Environment |

<table>
<thead>
<tr>
<th>SeCReT_M</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Shadow stack (Return address)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy 0xCCCC</td>
</tr>
<tr>
<td>Original 0x9200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>REE User Mode</td>
</tr>
<tr>
<td>REE Kernel Mode</td>
</tr>
</tbody>
</table>

Shadow stacks for protection of critical values
Coarse-Grained CFI (2/2)

• Protection of user-mode context

**Rich Execution Environment**

- Process accessing TEE resources
  - PC: 0x91FC
  - LR: 0x8700
  - SP: 0xbecd...

**Trusted Execution Environment**

- APC for current process
- Whitelist (Signal)
  - 0x11000
  - …

- Shadow stack (SP_U)
  - Dummy
  - Original
  - 0xAAAA 0xbecd...

- Shadow stack (LR_U)
  - Dummy
  - Original
  - 0xBBBB 0x8700

- Shadow stack (Return address)
  - Dummy
  - Original
  - 0xCCCC 0x9200

- Verifier

**Mode Switch**

REE User Mode
REE Kernel Mode

**Shadow stacks for protection of critical values**
Trusted Computing Base for SeCReT

• Active Monitoring as part of TCB
  ▪ Kernel code and system registers can be protected by Active Monitoring

<table>
<thead>
<tr>
<th>Type</th>
<th>Usage in SeCReT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel code Exception Vector</td>
<td>• SeCReT Trampoline</td>
</tr>
<tr>
<td>Kernel code process execution and termination</td>
<td>• SeCReT Trampoline</td>
</tr>
<tr>
<td>Register Translation Table Base Register (TTBR)</td>
<td>• APC lookup</td>
</tr>
<tr>
<td>Register Data Fault Status Register (DFSR)</td>
<td>• exception verification</td>
</tr>
<tr>
<td>Register Data Fault Address Register (DFAR)</td>
<td>• exception verification</td>
</tr>
<tr>
<td>Register Vector Base Address Register (VBAR)</td>
<td>• Exception vector remapping</td>
</tr>
<tr>
<td>Register System Control Register (SCTLR)</td>
<td>• Exception vector remapping</td>
</tr>
</tbody>
</table>
Implementation

• On Arndale board
  ▪ Offering a Cortex-A15 dual-core processor

• Components in the REE
  ▪ Linux 3.9.1
    ✓ Trampolines and new exception vector

• Components in the TEE
  ▪ Monitor code
    ✓ Page access-control
    ✓ Hash calculation
  ▪ Data structure
    ✓ Active Process Context
Microbenchmarks

• LMBench
  - Null: mode switch overhead between user and kernel
  - Overhead is imposed by SeCReT’s intervention with switches in modes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linux</th>
<th>SeCReT</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>0.27</td>
<td>1.06</td>
<td>3.9259x</td>
</tr>
<tr>
<td>Read</td>
<td>0.33</td>
<td>1.23</td>
<td>3.7273x</td>
</tr>
<tr>
<td>Write</td>
<td>0.42</td>
<td>1.57</td>
<td>3.7381x</td>
</tr>
<tr>
<td>Open/Close</td>
<td>5.43</td>
<td>8.83</td>
<td>1.6264x</td>
</tr>
<tr>
<td>Fork</td>
<td>147.78</td>
<td>174.66</td>
<td>1.1819x</td>
</tr>
<tr>
<td>Fork/exec</td>
<td>160.32</td>
<td>189.03</td>
<td>1.1781x</td>
</tr>
</tbody>
</table>

*Lmbench Latency Microbenchmark Results (in microseconds.)*
Key Access-Control Overhead

• Measurement for Key access-control overhead
  ▪ Parses, encrypts, and prints an input payload

**Input:** An ascii payload of size: **128 to 8192** bytes
**Output:** Encrypted payload

```plaintext
*key = allocMemory()
if Key_Protection then
    assignKeyBySeCReT(key)
else
    *key=tempValue()
end if
payload = encrypt(payload, *key)
printString(payload)
```

<table>
<thead>
<tr>
<th>Test Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
</tr>
<tr>
<td>SeCReT-enabled Linux</td>
</tr>
<tr>
<td>SeCReT-enabled w/ key protection</td>
</tr>
</tbody>
</table>
### Key Access-Control Overhead

- Average latency after running 10 times for each payload

<table>
<thead>
<tr>
<th>Payload Size (Bytes)</th>
<th>Linux Time</th>
<th>SeCreT Enabled Time</th>
<th>SeCreT w/ Key Protection Time</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1334.6</td>
<td>1544.5</td>
<td>1979.0</td>
<td>15.73%</td>
</tr>
<tr>
<td>256</td>
<td>1642.5</td>
<td>1912.1</td>
<td>2425.8</td>
<td>16.41%</td>
</tr>
<tr>
<td>512</td>
<td>2279.4</td>
<td>2509.8</td>
<td>3068.2</td>
<td>10.11%</td>
</tr>
<tr>
<td>1024</td>
<td>3650.9</td>
<td>3822.6</td>
<td>4516.7</td>
<td>4.70%</td>
</tr>
<tr>
<td>2048</td>
<td>340225.7</td>
<td>340244.6</td>
<td>341531.4</td>
<td>0.01%</td>
</tr>
<tr>
<td>4096</td>
<td>679761.2</td>
<td>679818.7</td>
<td>681604.3</td>
<td>0.01%</td>
</tr>
<tr>
<td>8192</td>
<td>1693561.2</td>
<td>1693683.6</td>
<td>1696639.1</td>
<td>0.01%</td>
</tr>
</tbody>
</table>

**Benchmark of SeCreT Overhead compared to Linux (in microseconds.)**
Discussion

• Extension of SeCReT
  ▪ Protecting applications from untrusted kernel
  ▪ Protecting guest VMs from vulnerable hypervisors

• Attack against SeCReT
  ▪ Transient code modification in user mode
  ▪ Reverse-engineering the target binary

• Usability of SeCReT
  ▪ Protecting the session key: SeCReT library vs. Secure buffer
  ▪ Updating the list of pre-authorized applications in TrustZone
Summary

• SeCReT aims to generate a secure channel to reinforce the access control of the resources in TrustZone

• SeCReT extends the usage of TrustZone more flexibly, not limited to simply providing a TEE

• SeCReT can coordinate with already deployed TrustZone-based security solutions such as active monitoring