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

Jinsoo Jang

Sunjune Kong

Minsu Kim

Daegyeong Kim

Brent Byunghoon Kang

KAIST

KAIST

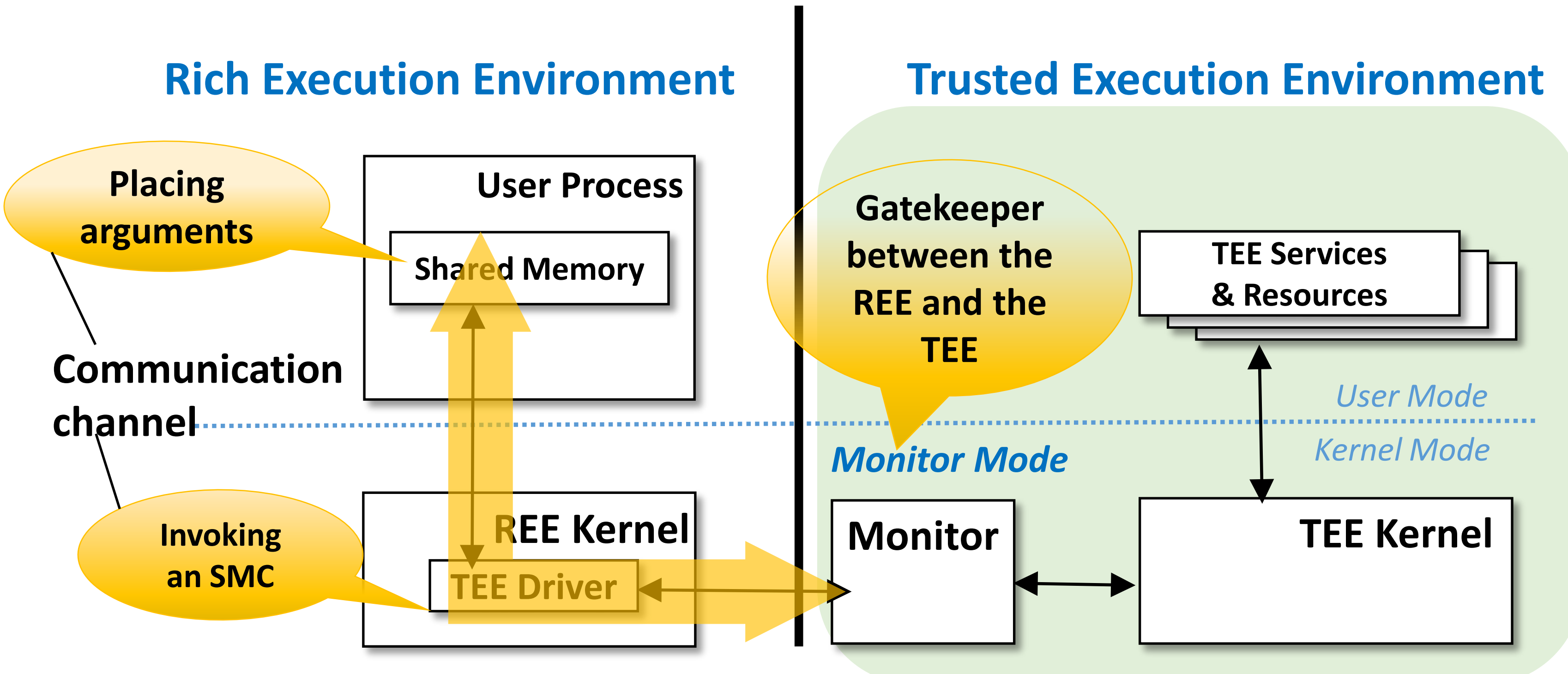
CySecLab

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

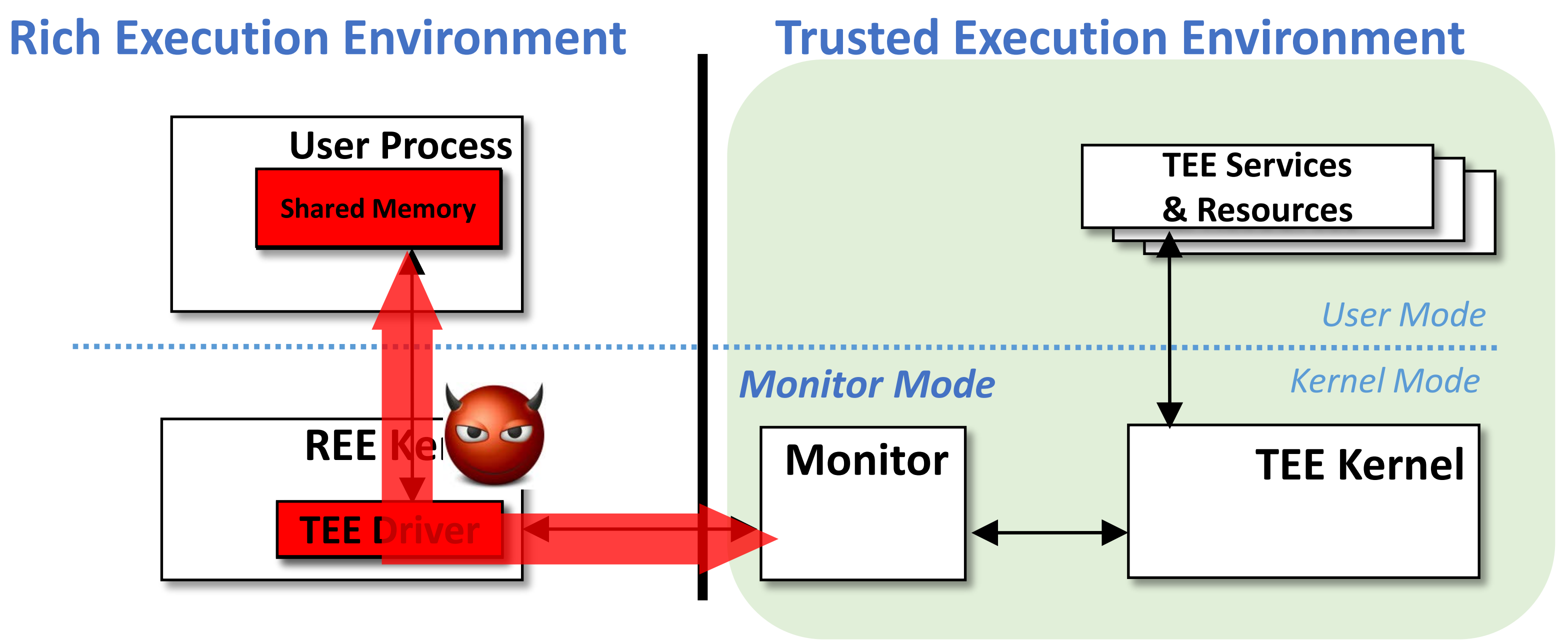
ARM TrustZone

- Provides a TEE for embedded devices
- Communcation channel :
  - Invoking SMC instruction with arguments



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

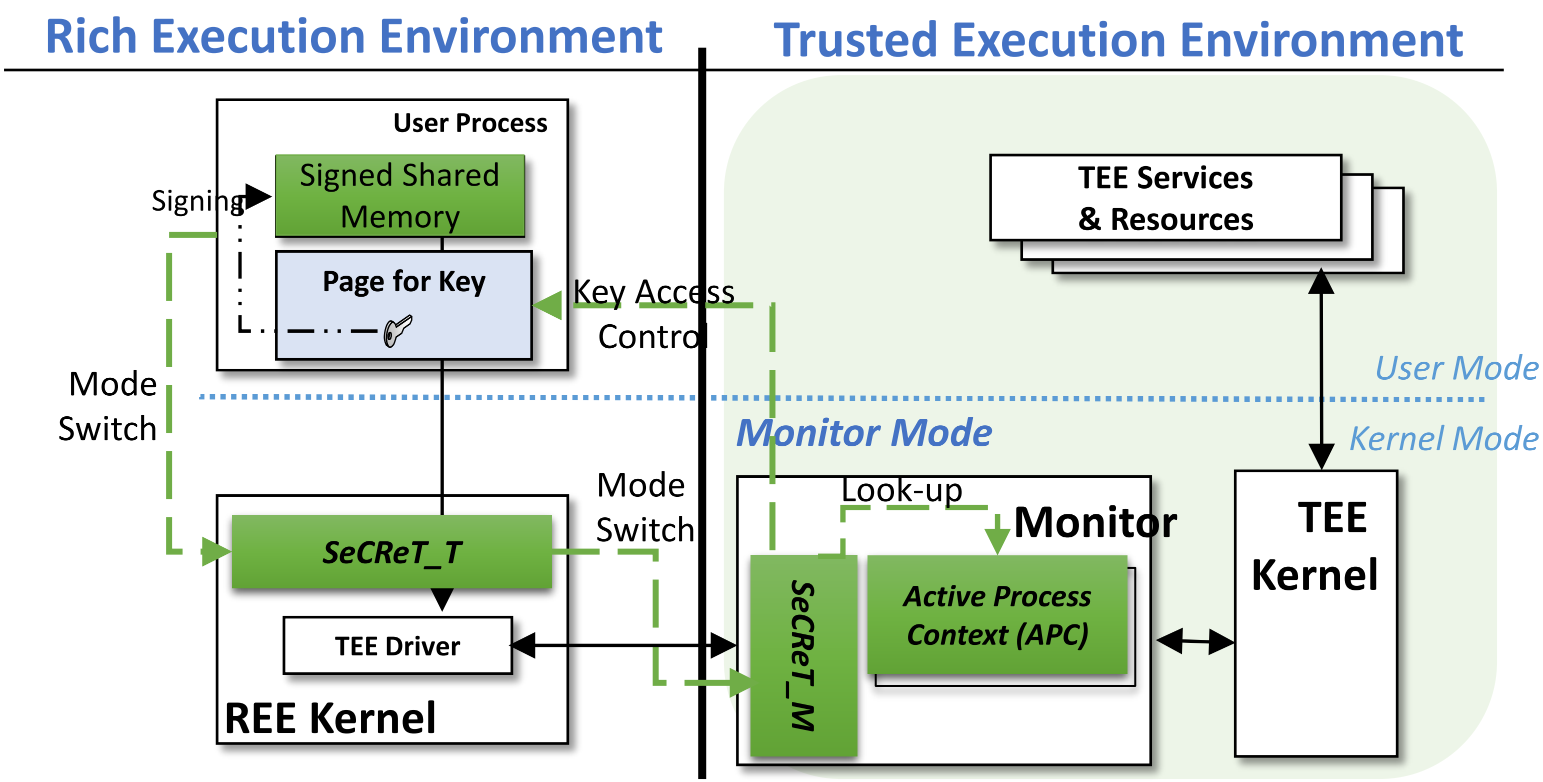
Our Goal & Assumption

- 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)

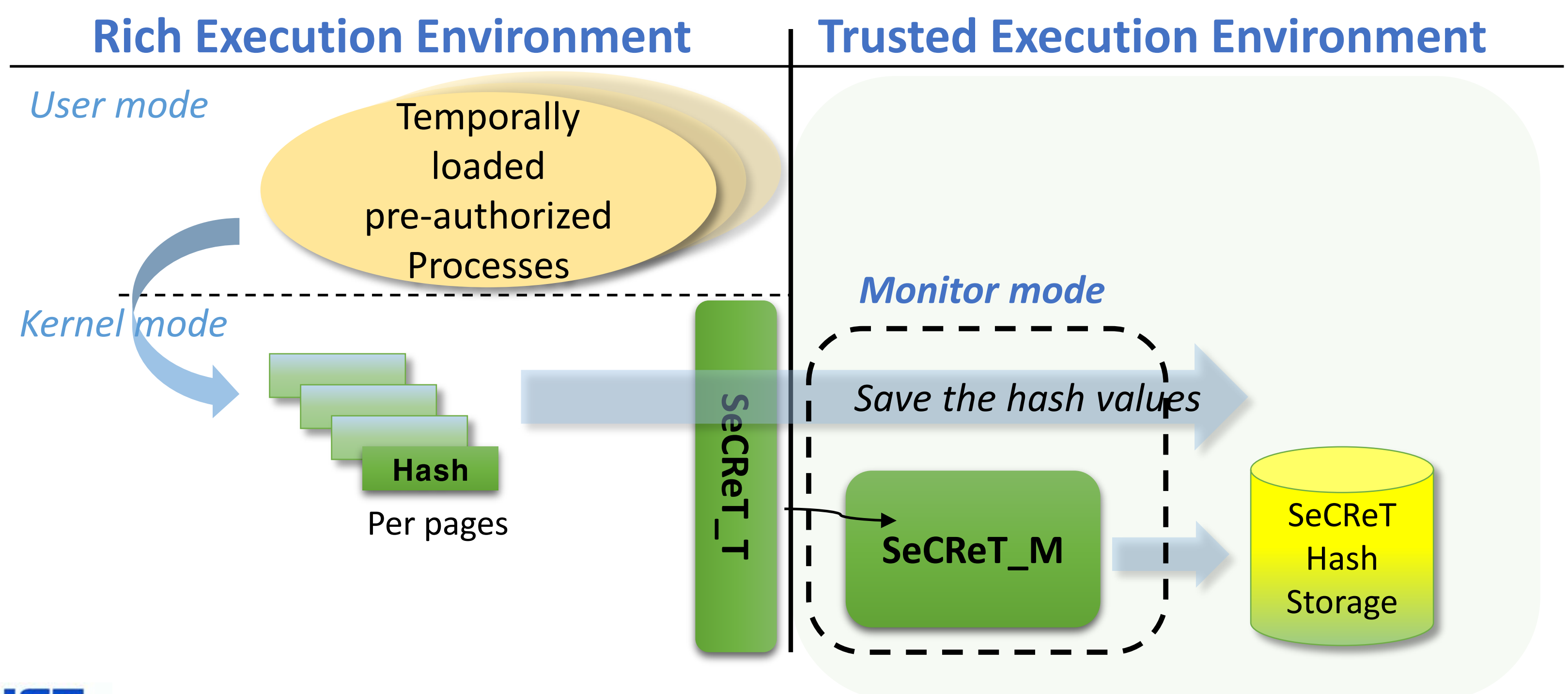
SeCReT - Overview

- Framework to provide and protect the session key in the REE



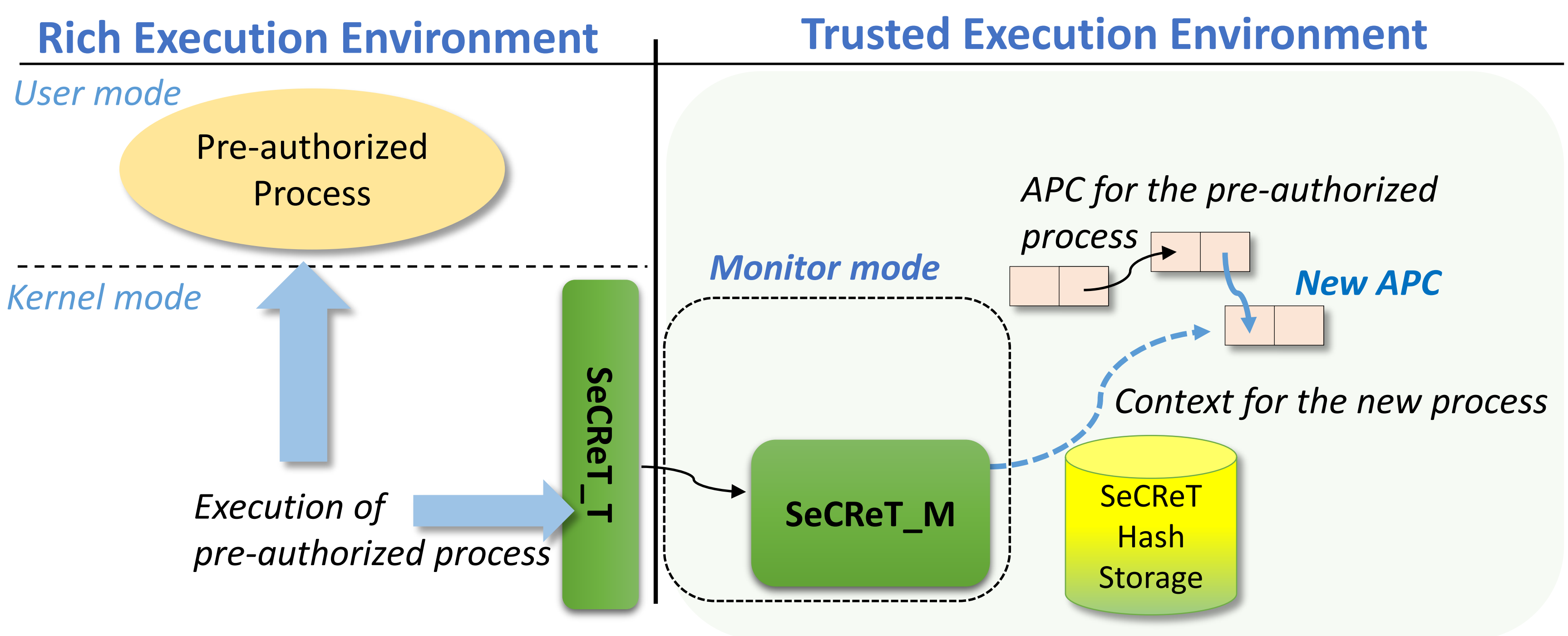
Session Key Life Cycle (1/5)

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



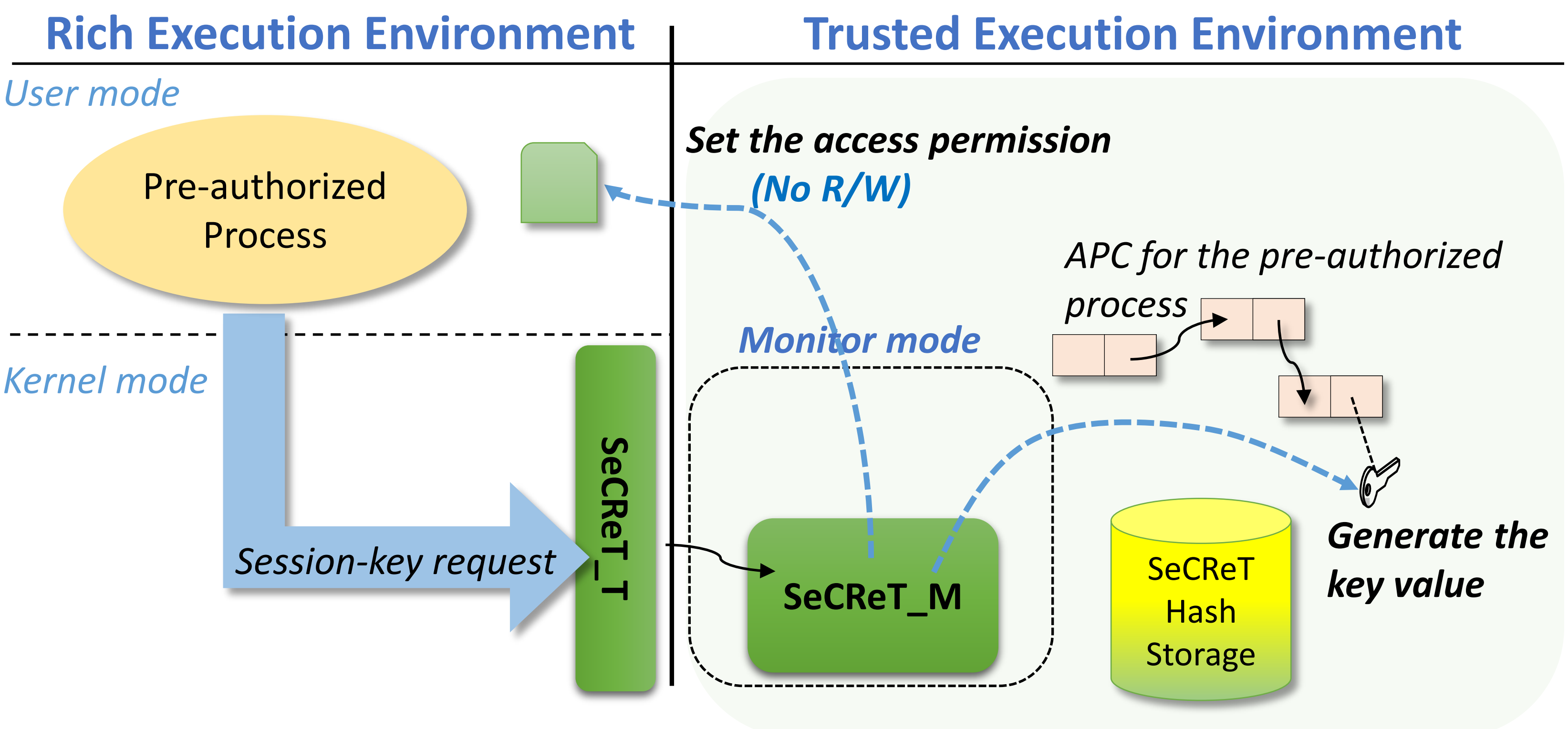
Session Key Life Cycle (2/5)

- Execution of the pre-authorized process
  - Create an APC for the process



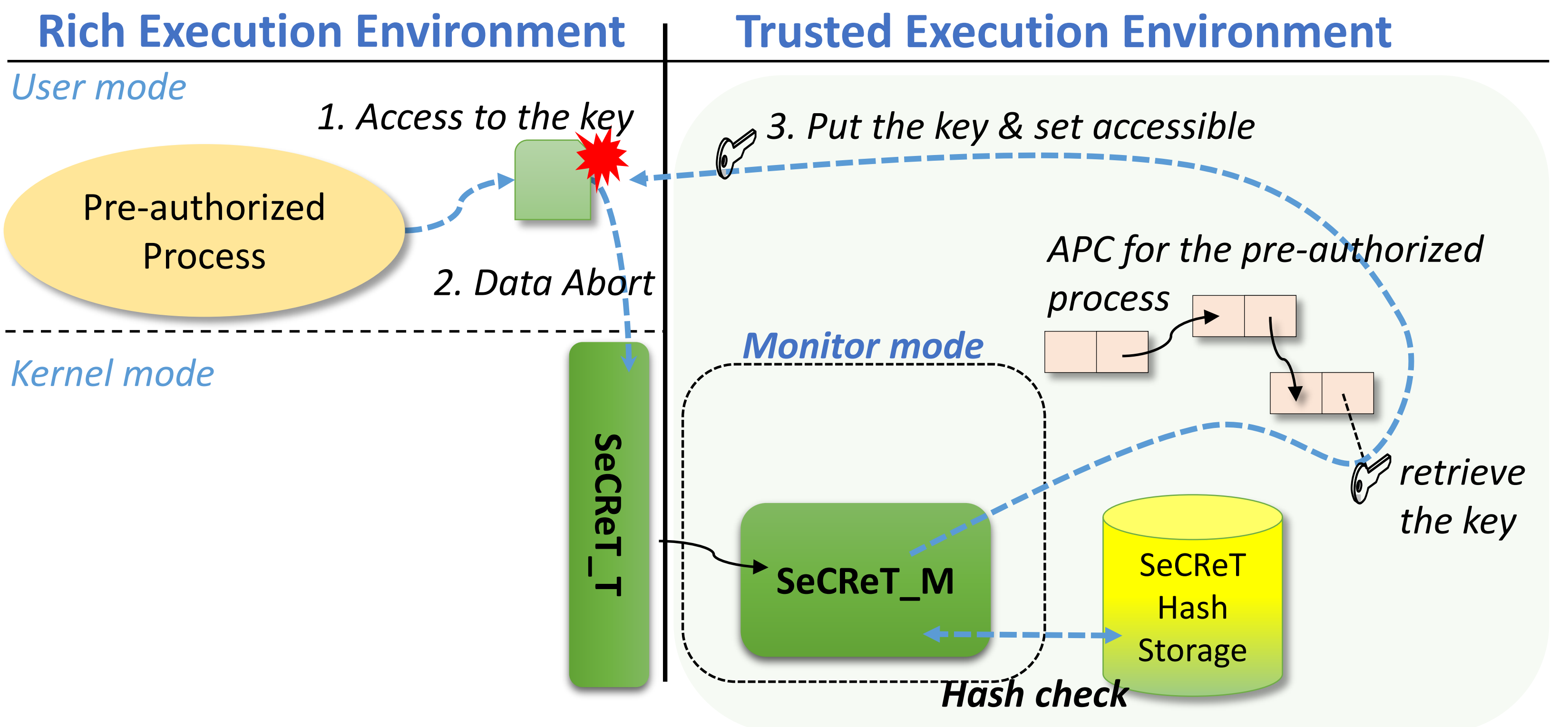
Session Key Life Cycle (3/5)

- Session-key creation
  - Set the access permission & generate the key value



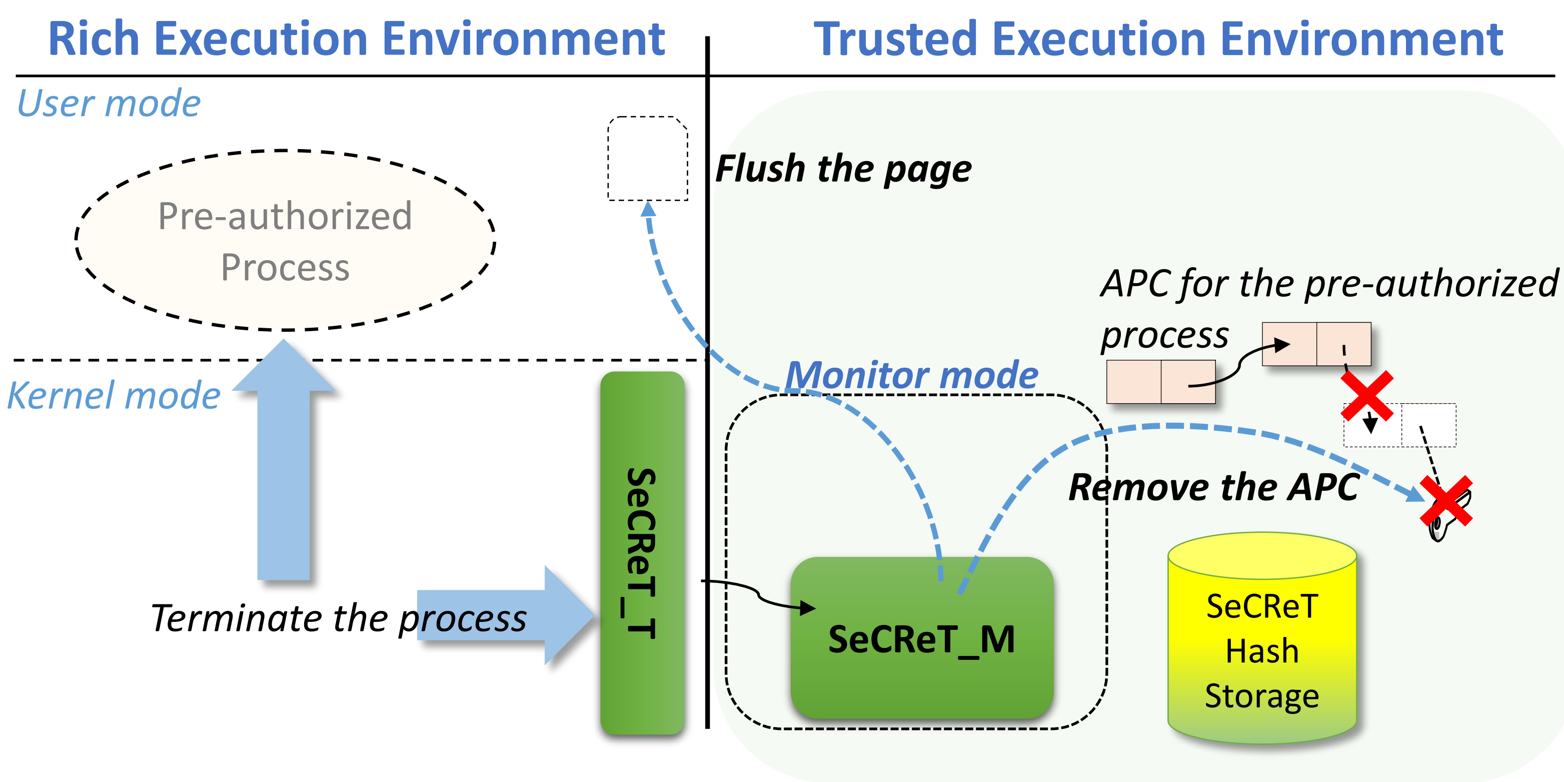
Session Key Life Cycle (4/5)

- 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



KAIST

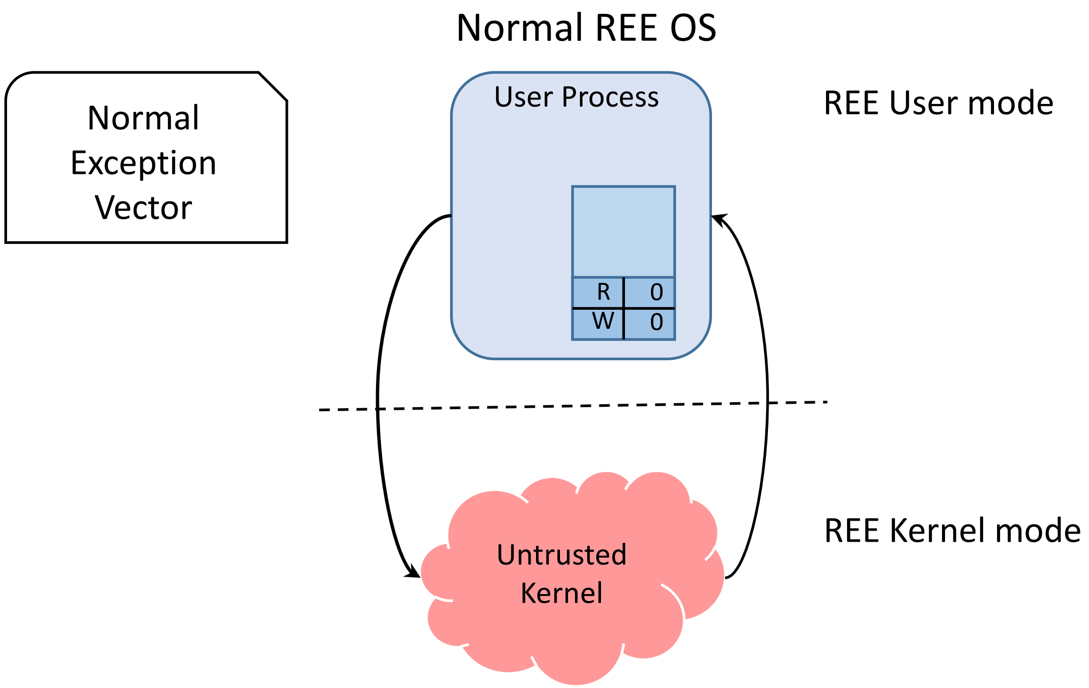
CySecLab

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

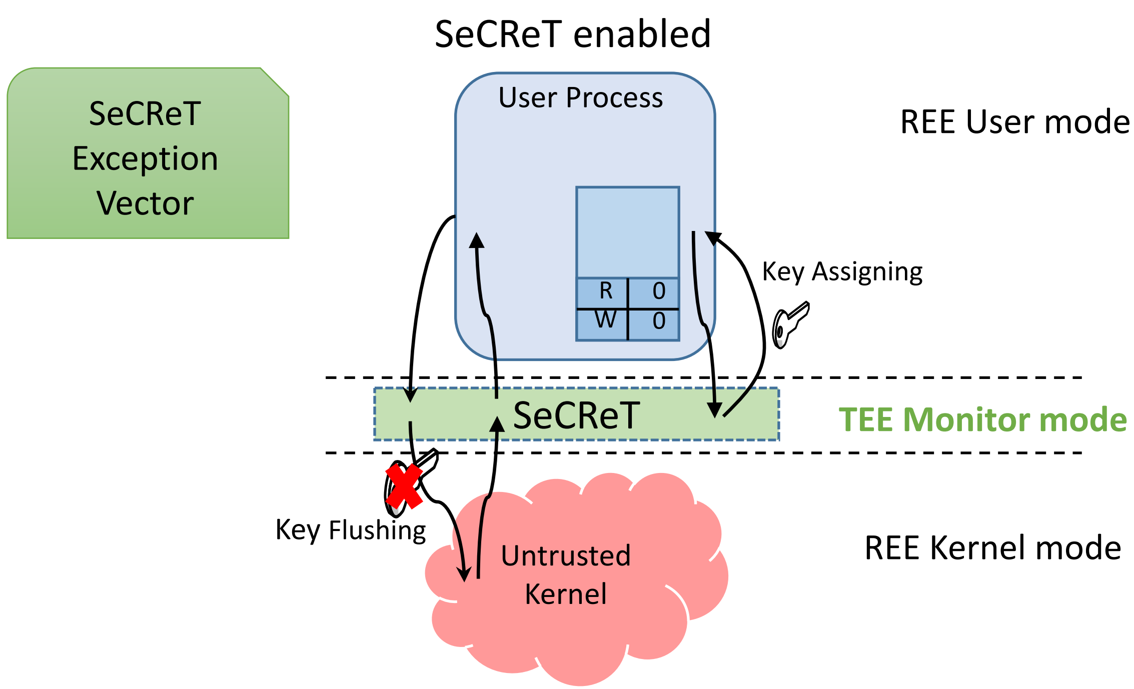
Interposition with Mode Switches

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



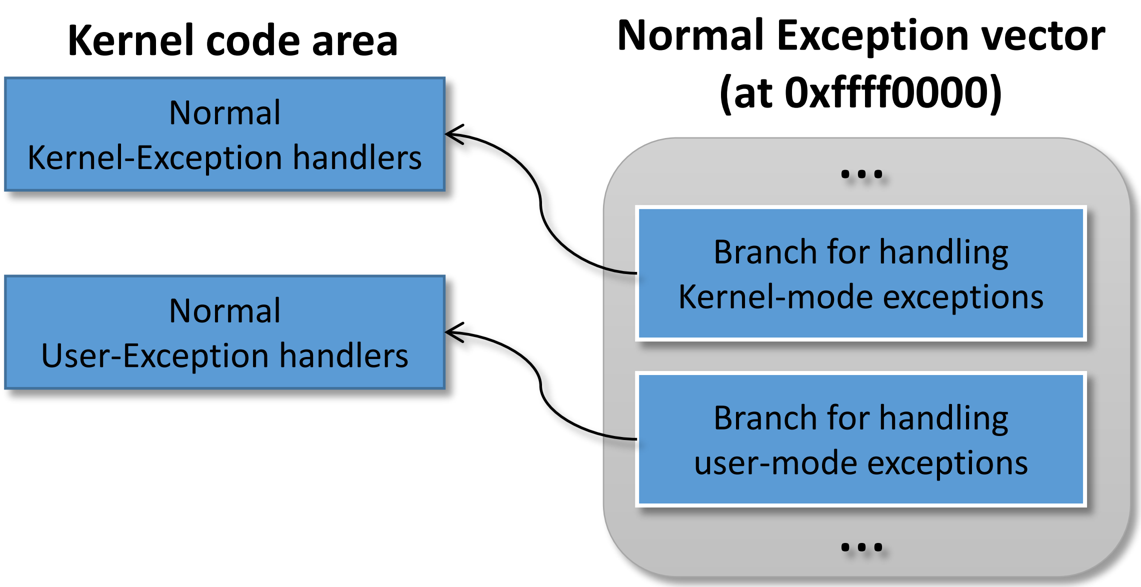
Interposition with Mode Switches

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



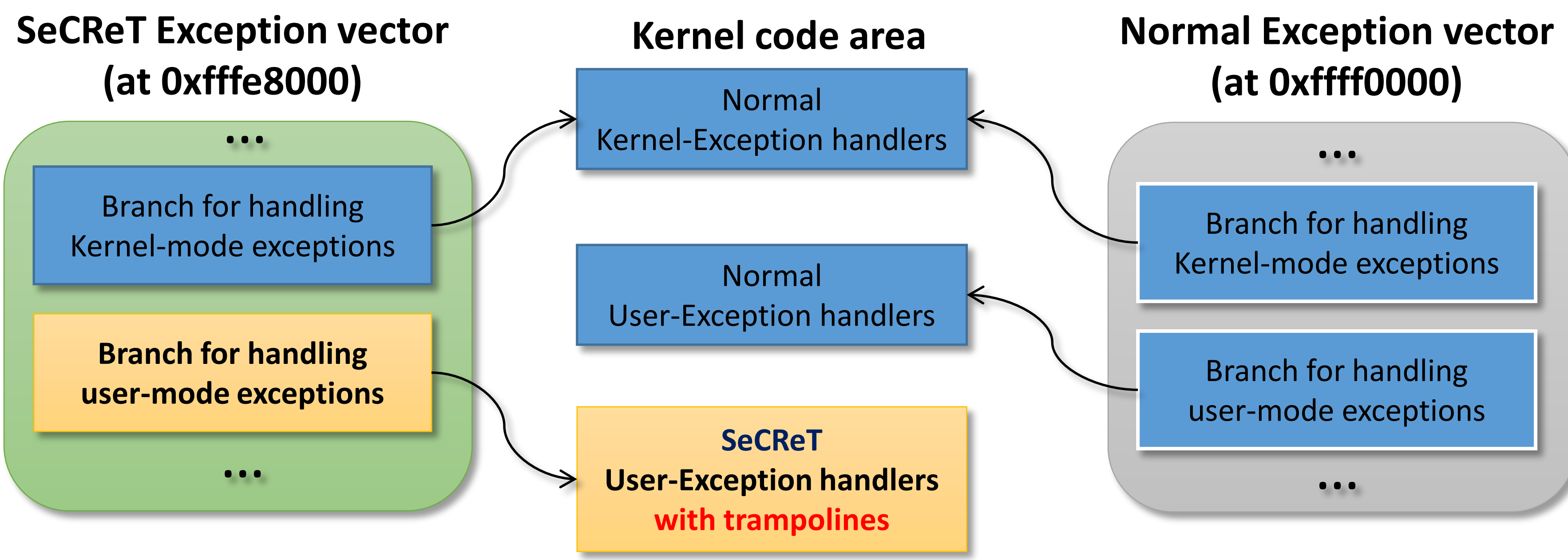
Interposition with Mode Switches

- SeCReT_EXV: New exception vector for SeCReT
  - Trampoline code is inserted to the starting point of
    - ✓ Handler code for user mode exceptions (User → Kernel)
    - ✓ Switch-to-user code (Kernel → User)



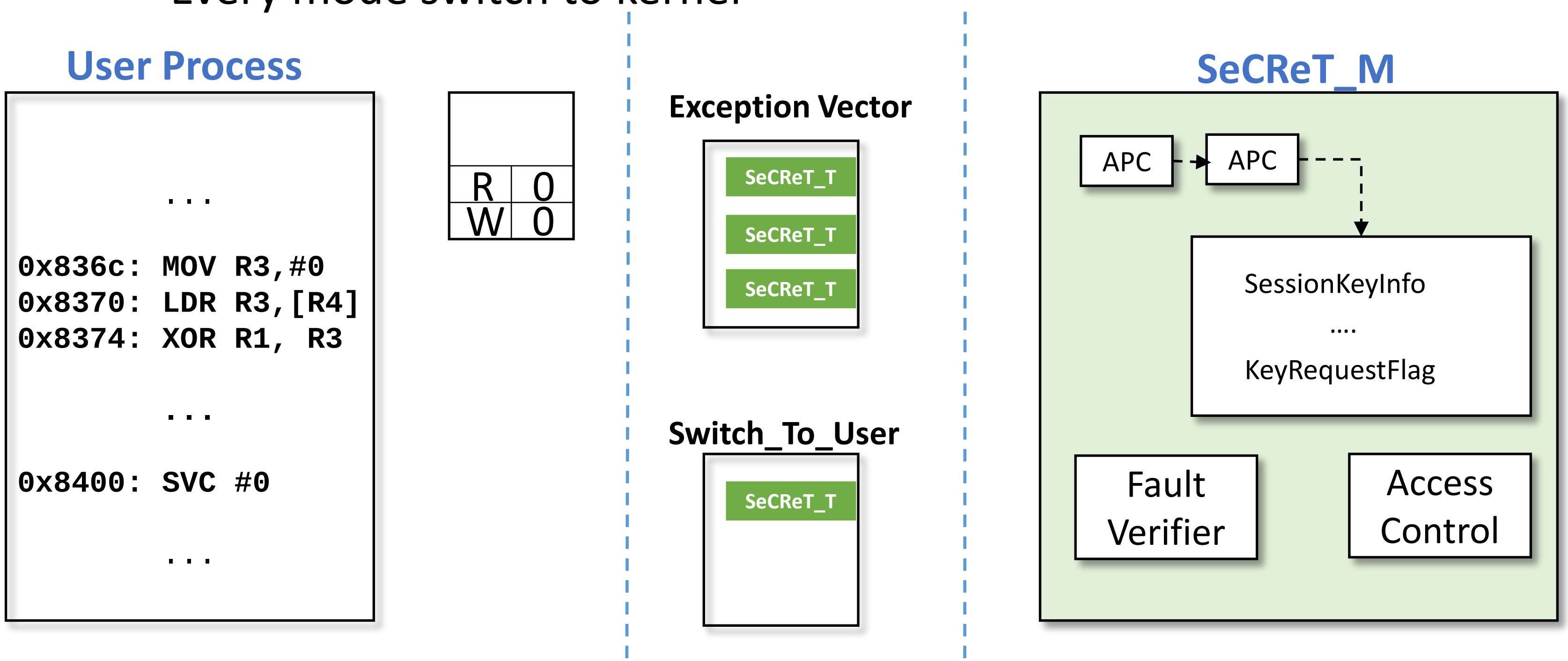
Interposition with Mode Switches

- SeCReT_EXV: New exception vector for SeCReT
  - Trampoline code is inserted to the starting point of
    - ✓ Handler code for user mode exceptions (User → Kernel)
    - ✓ Switch-to-user code (Kernel → User)



Access Control to the Key

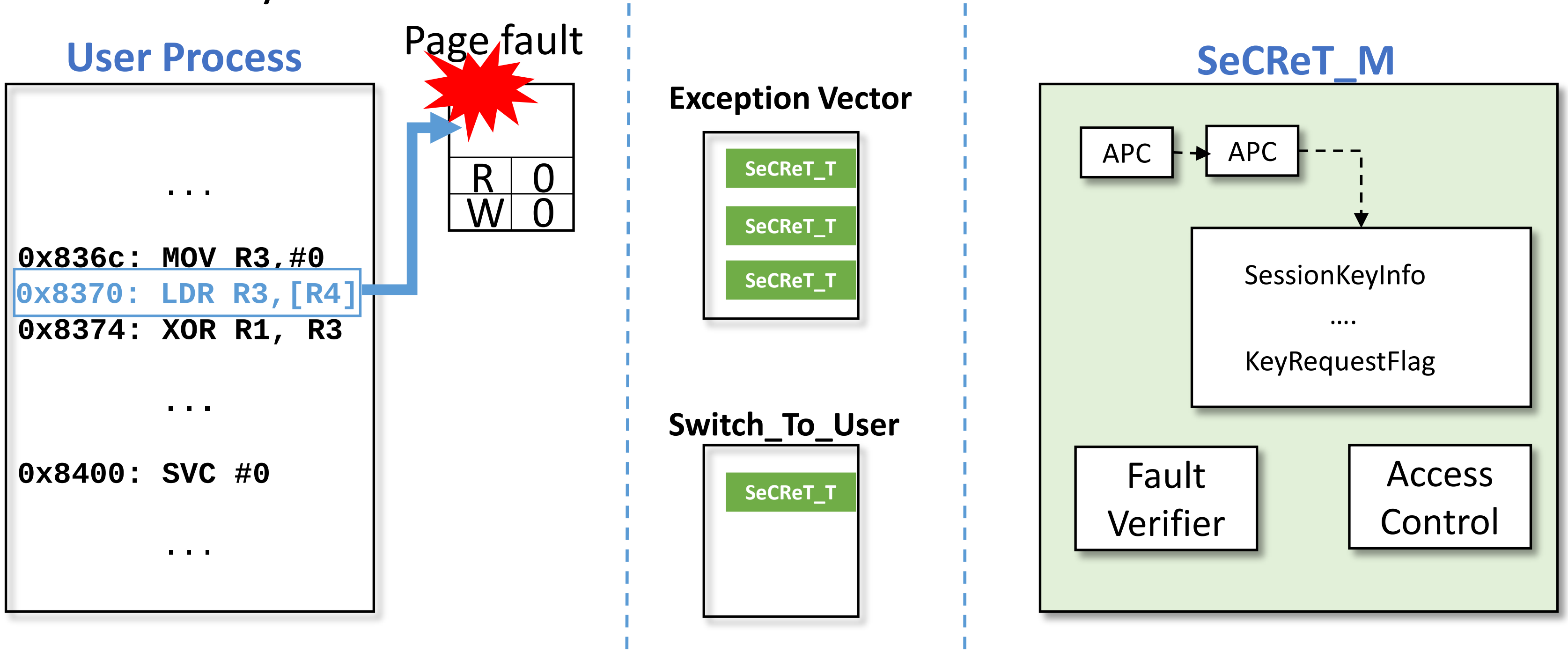
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

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

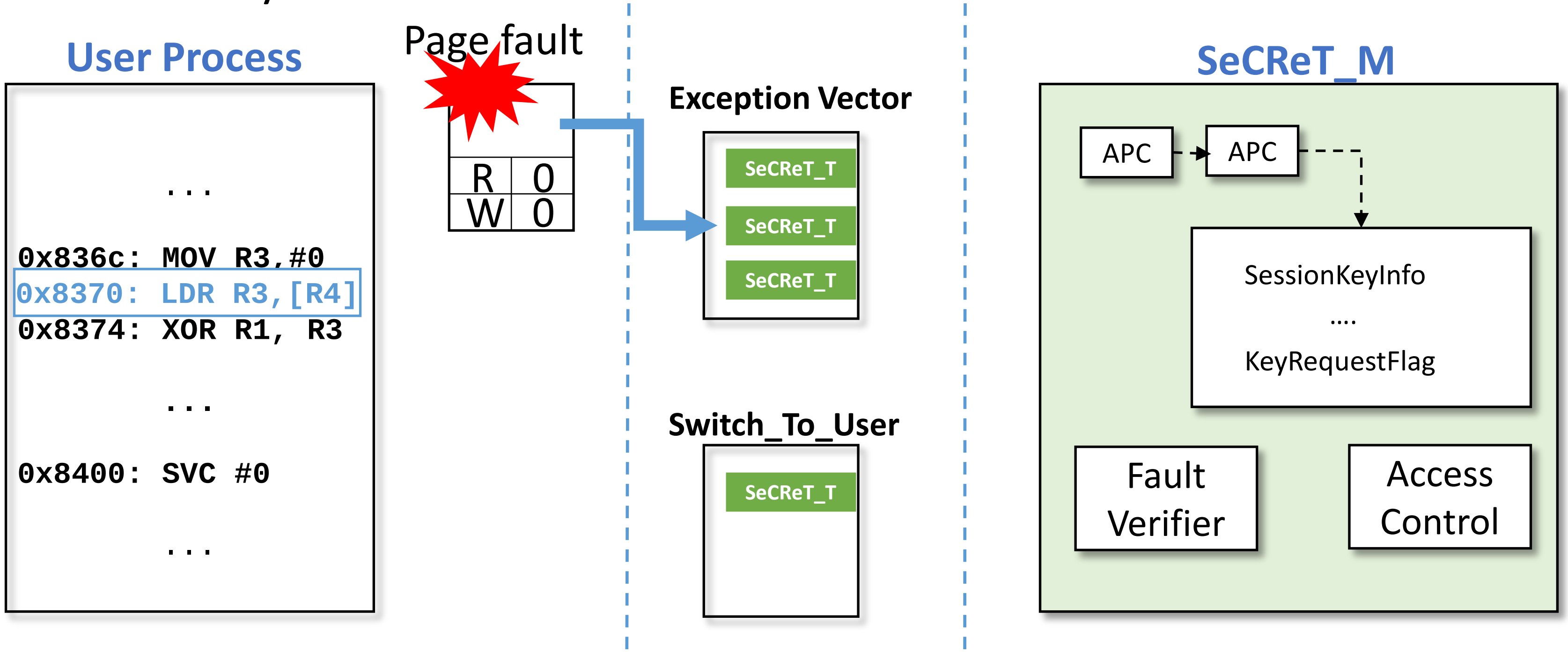


Control-flow for the access control to the session key

KAIST CySecLab

Access Control to the Key

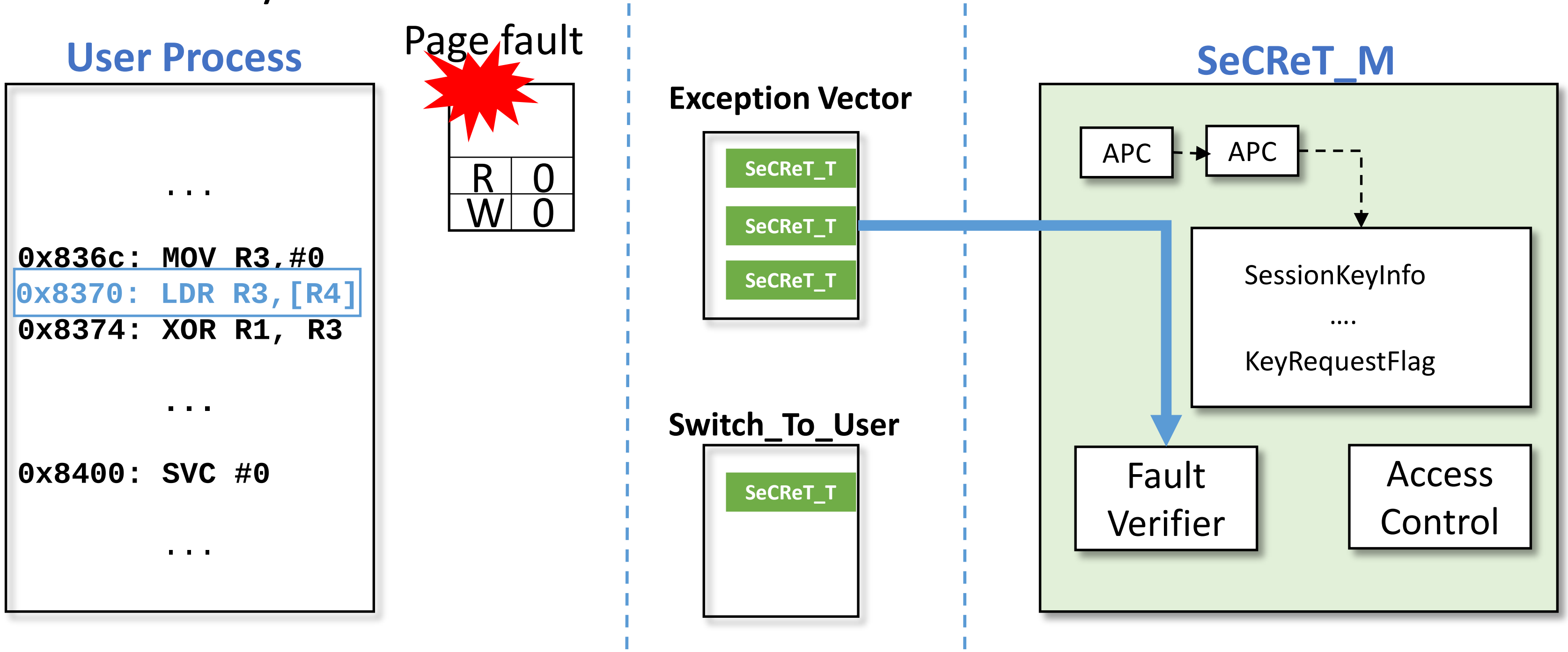
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

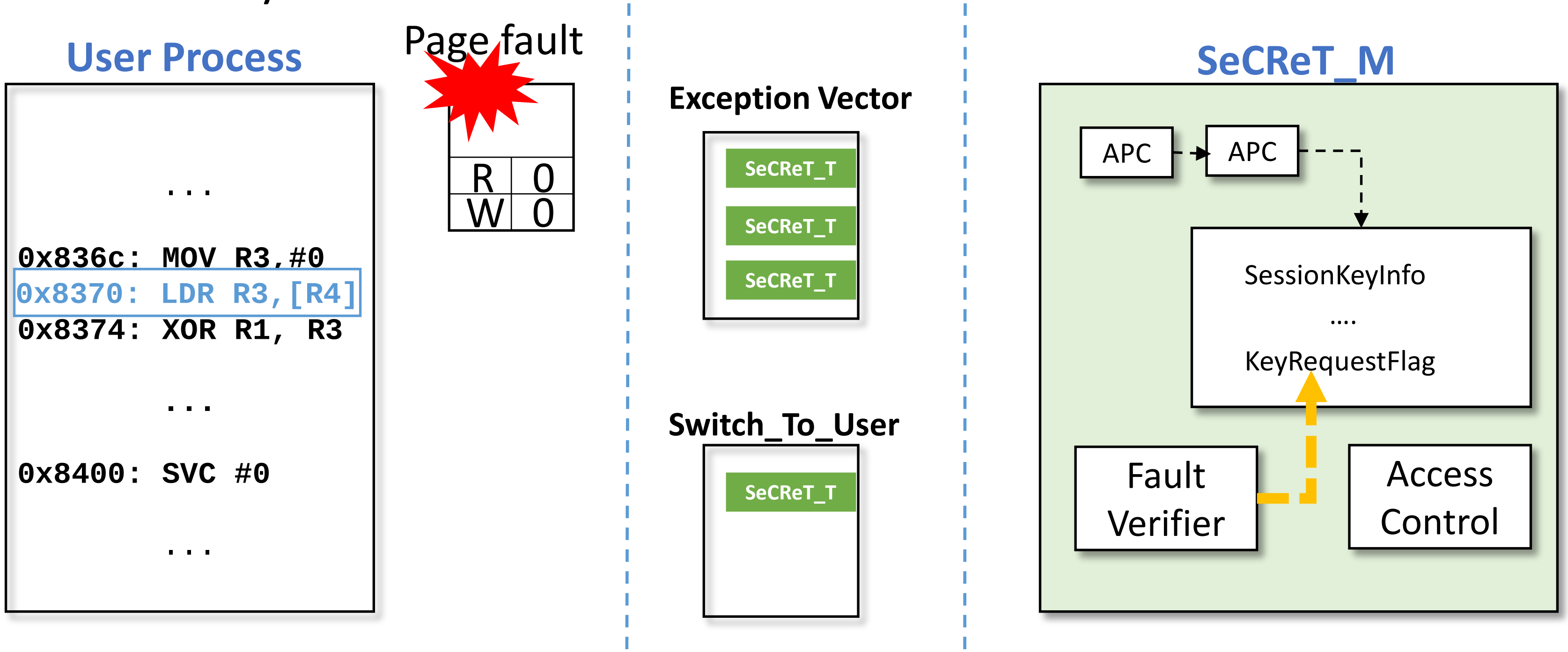
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

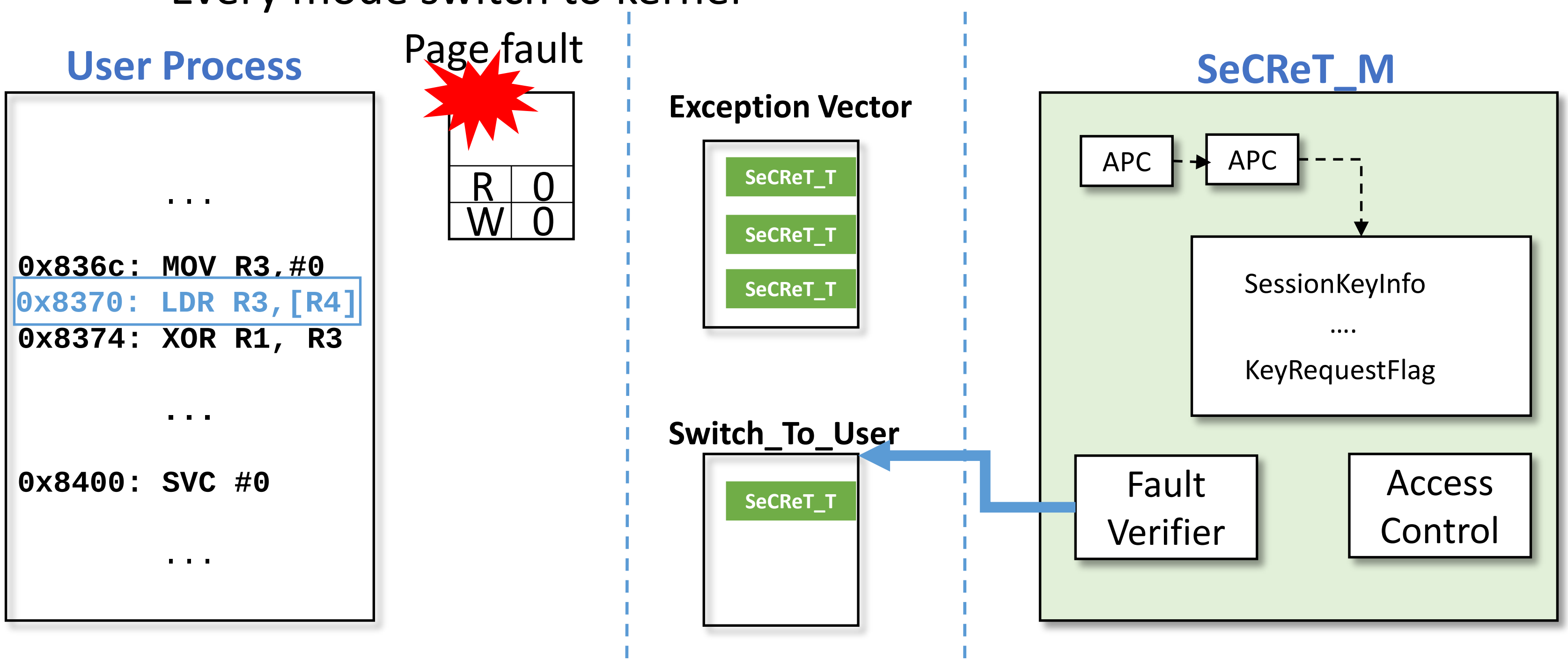
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

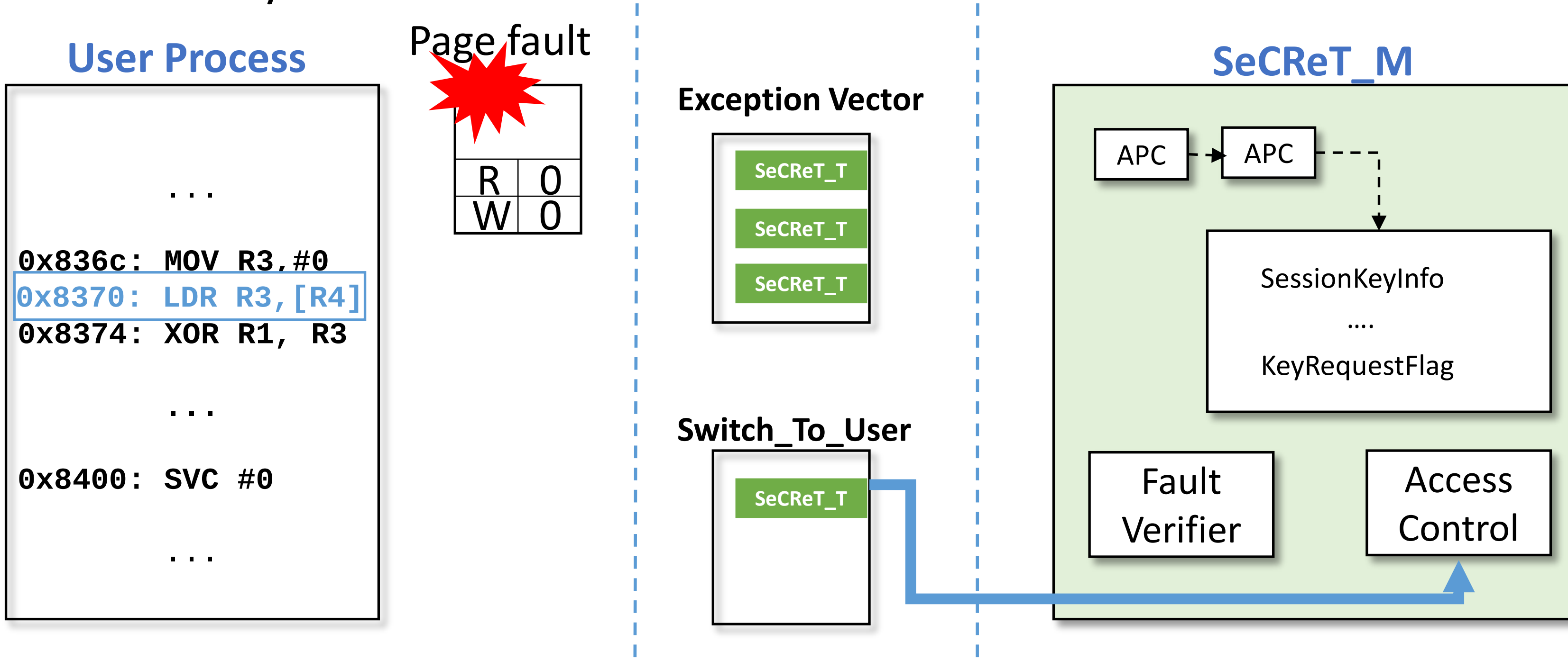
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

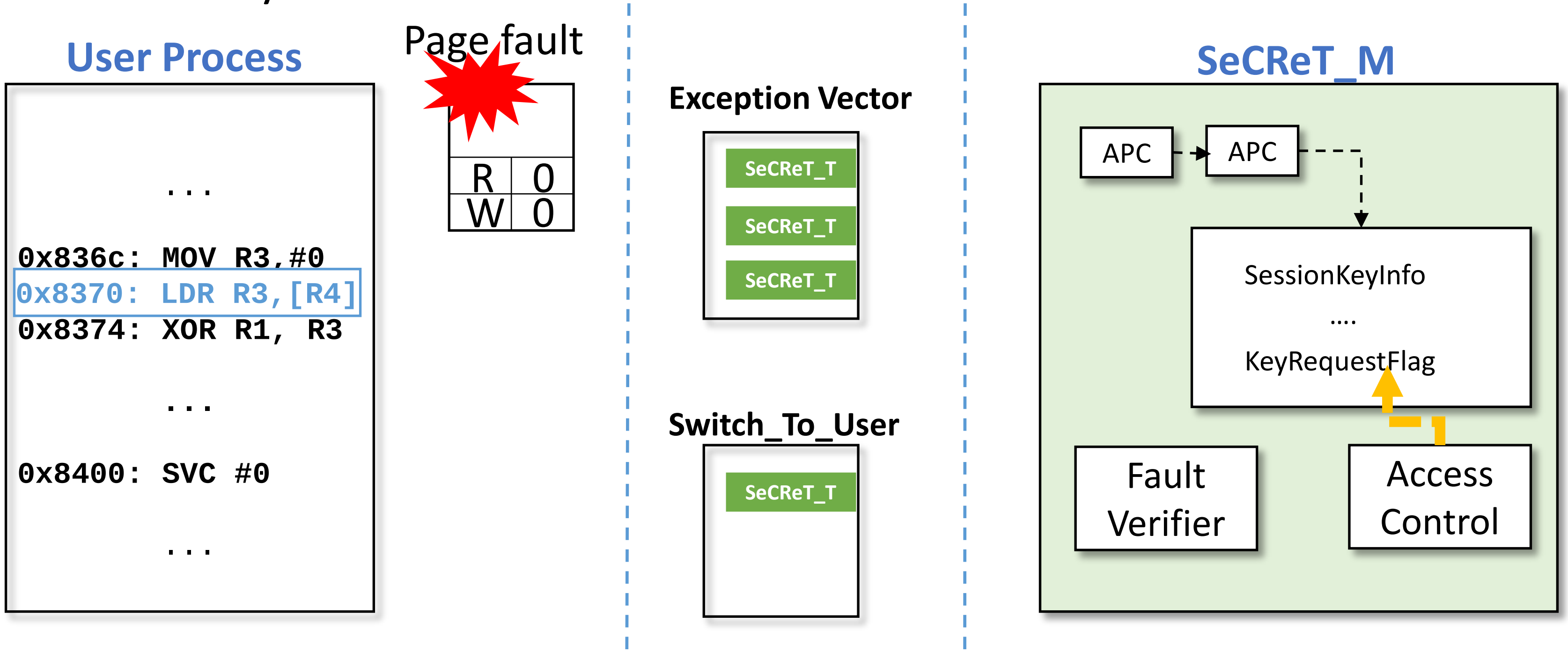
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



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

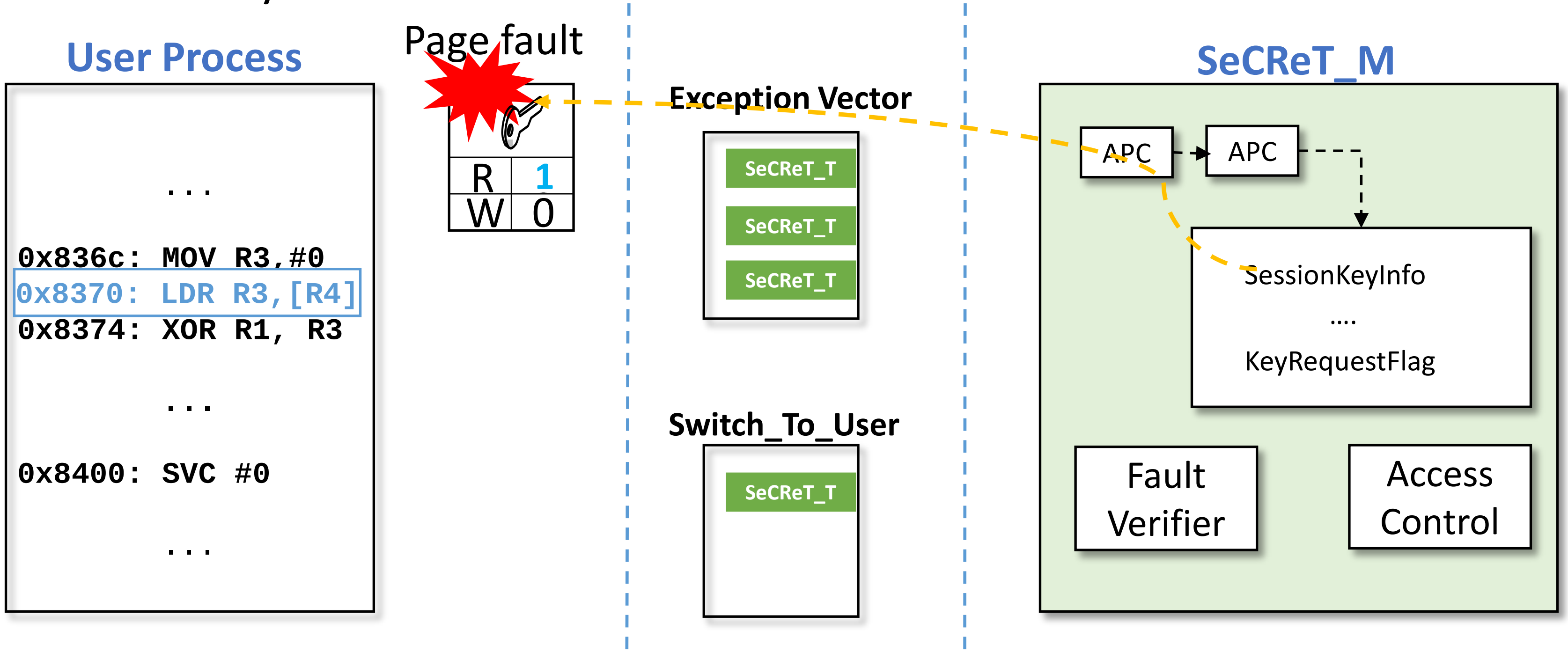


Control-flow for the access control to the session key

KAIST CySecLab

Access Control to the Key

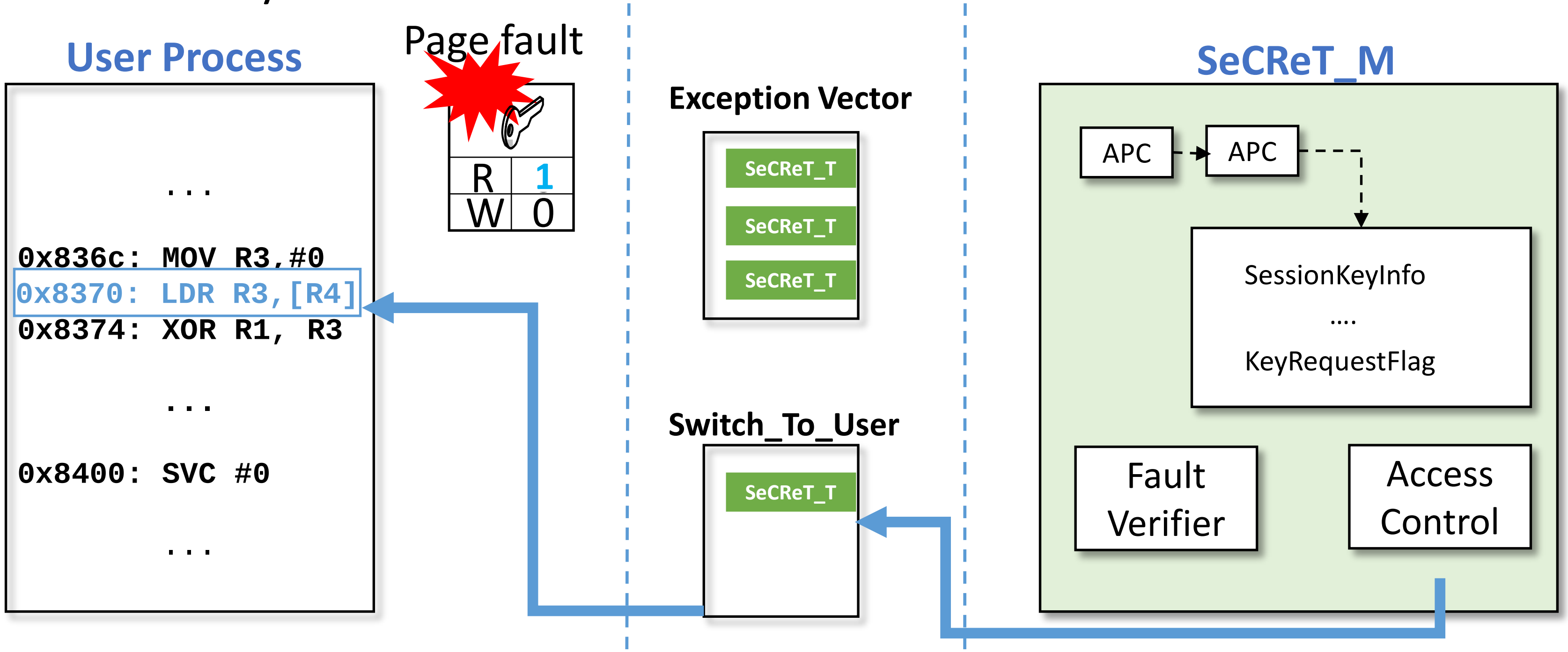
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

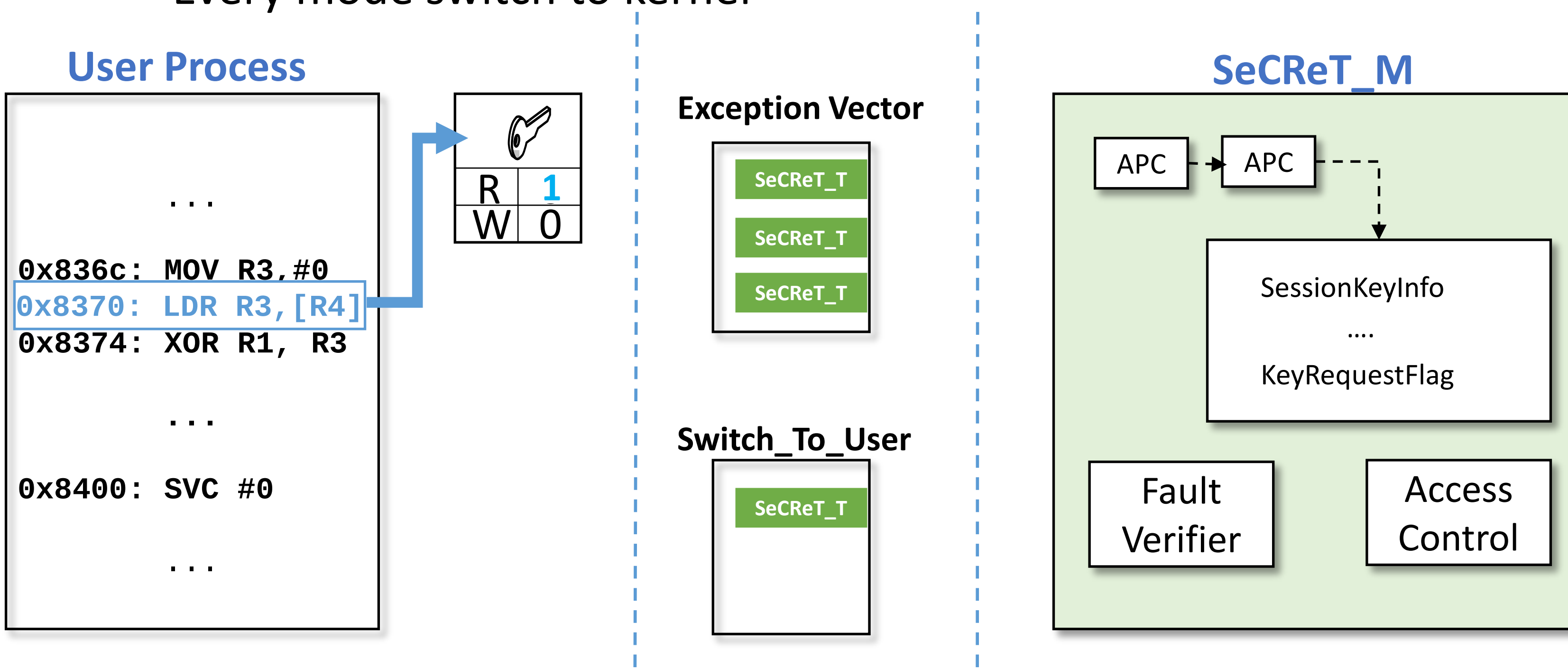
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

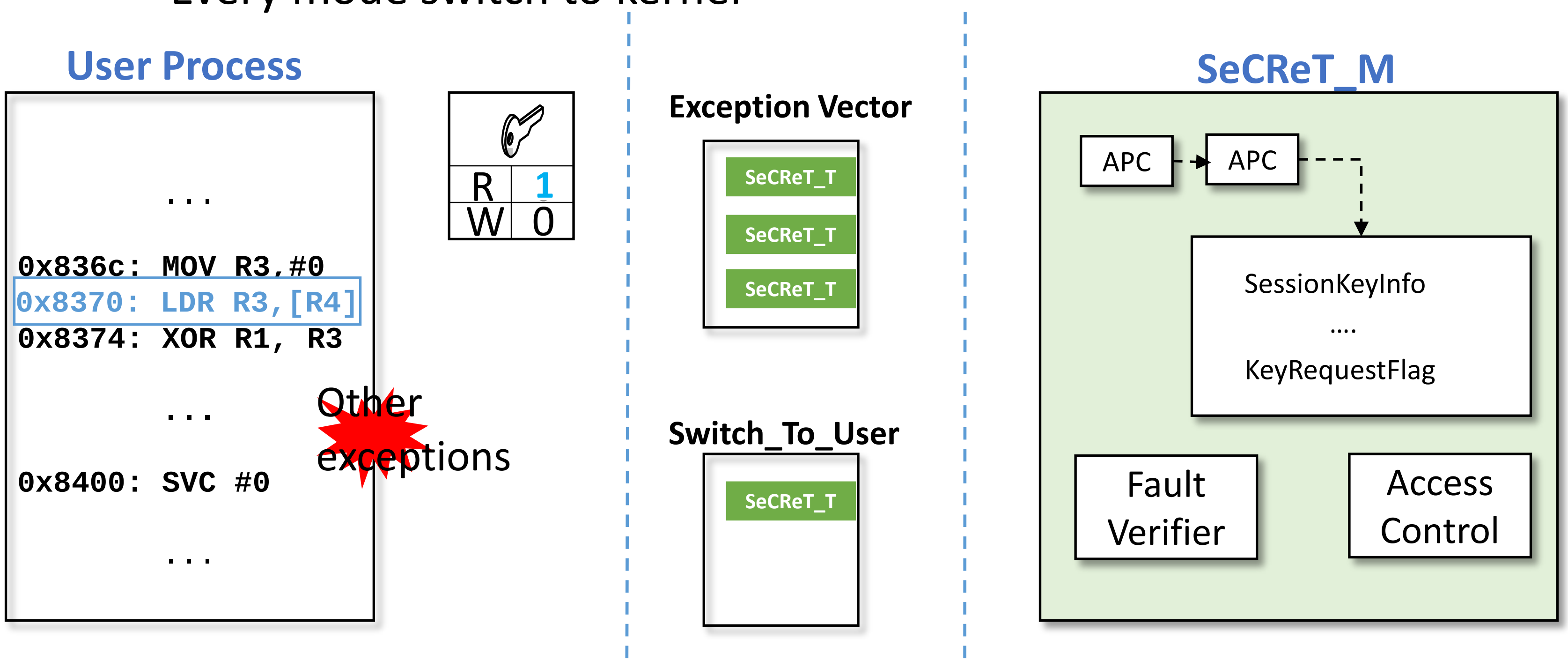
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

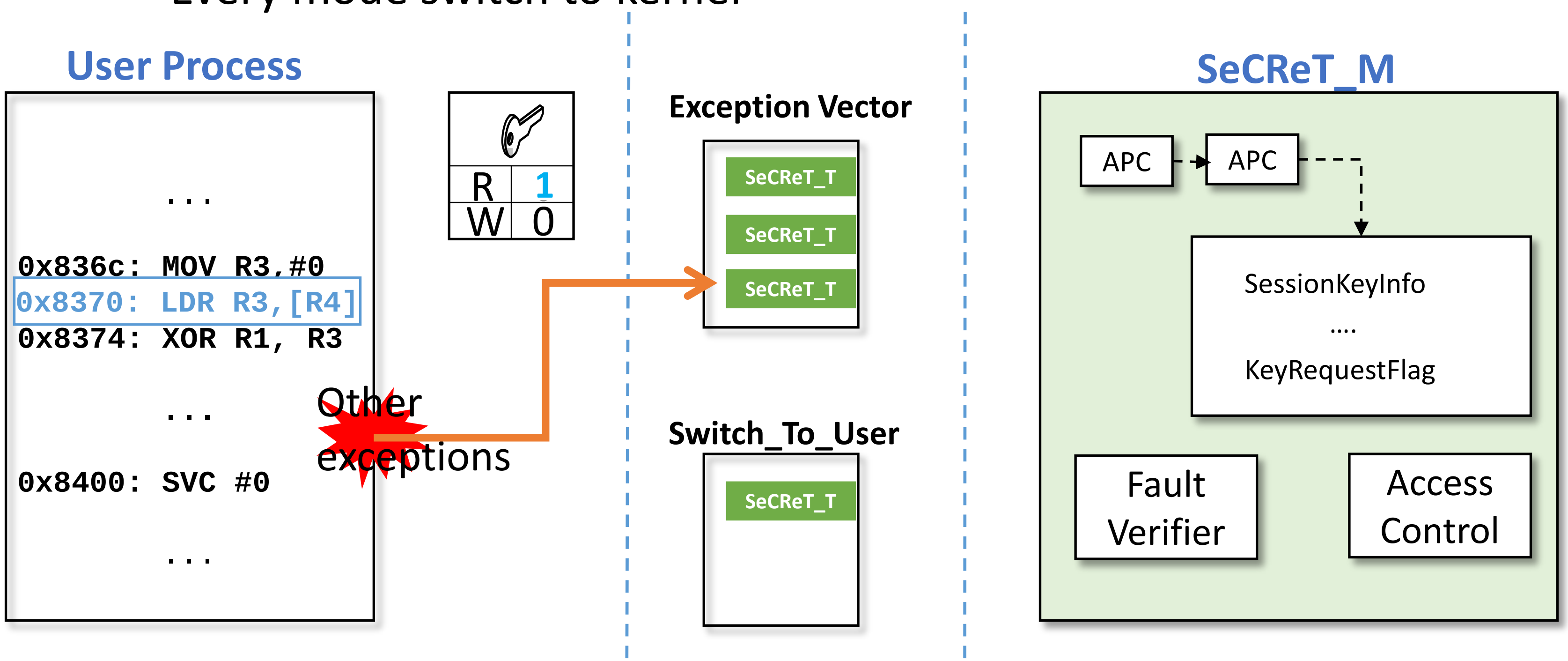
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

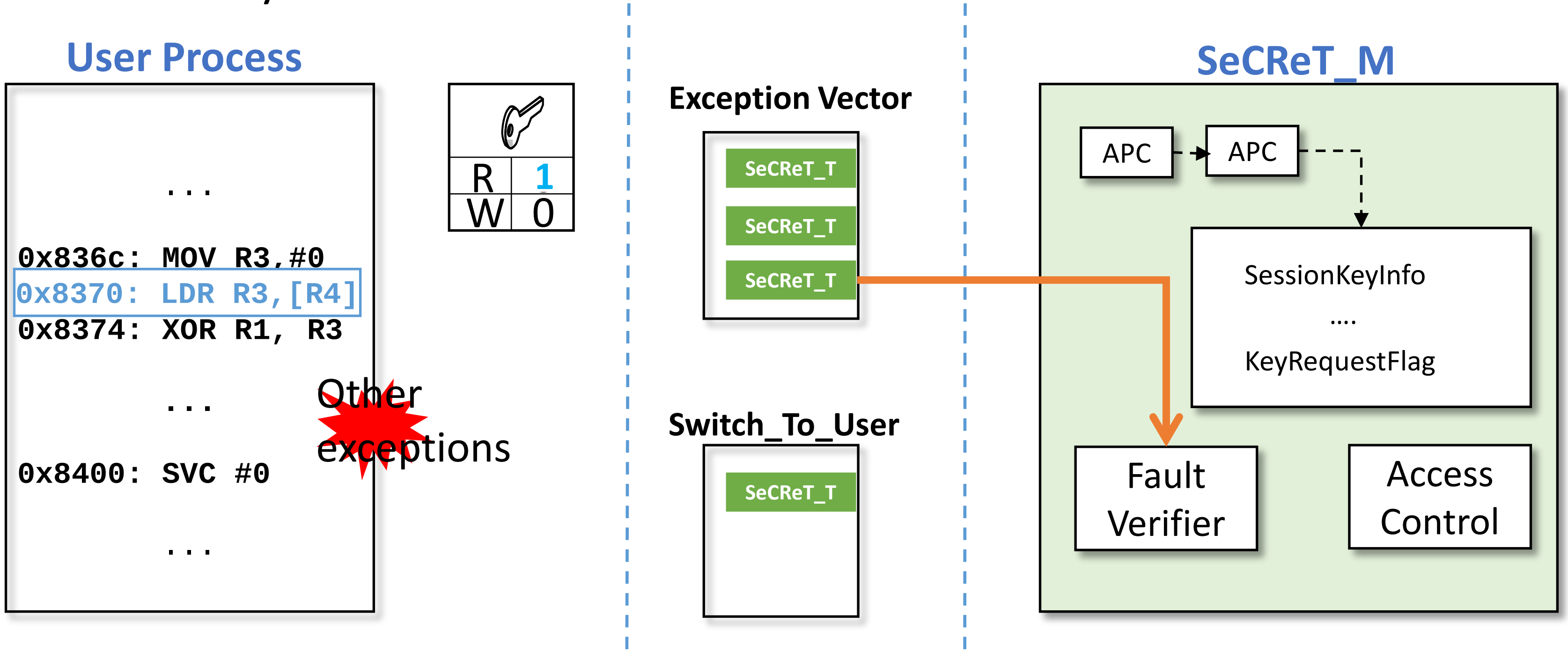
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



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



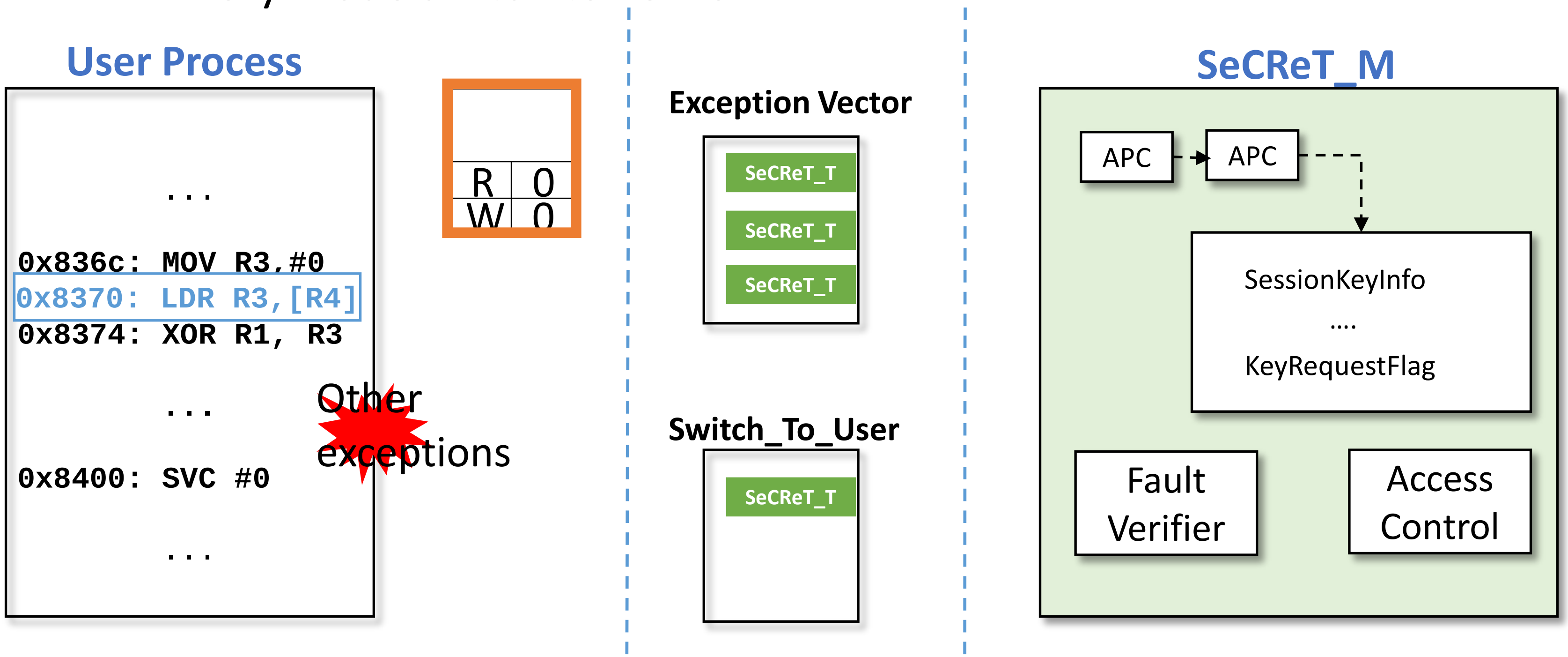
Control-flow for the access control to the session key

KAIST

CySecLab

Access Control to the Key

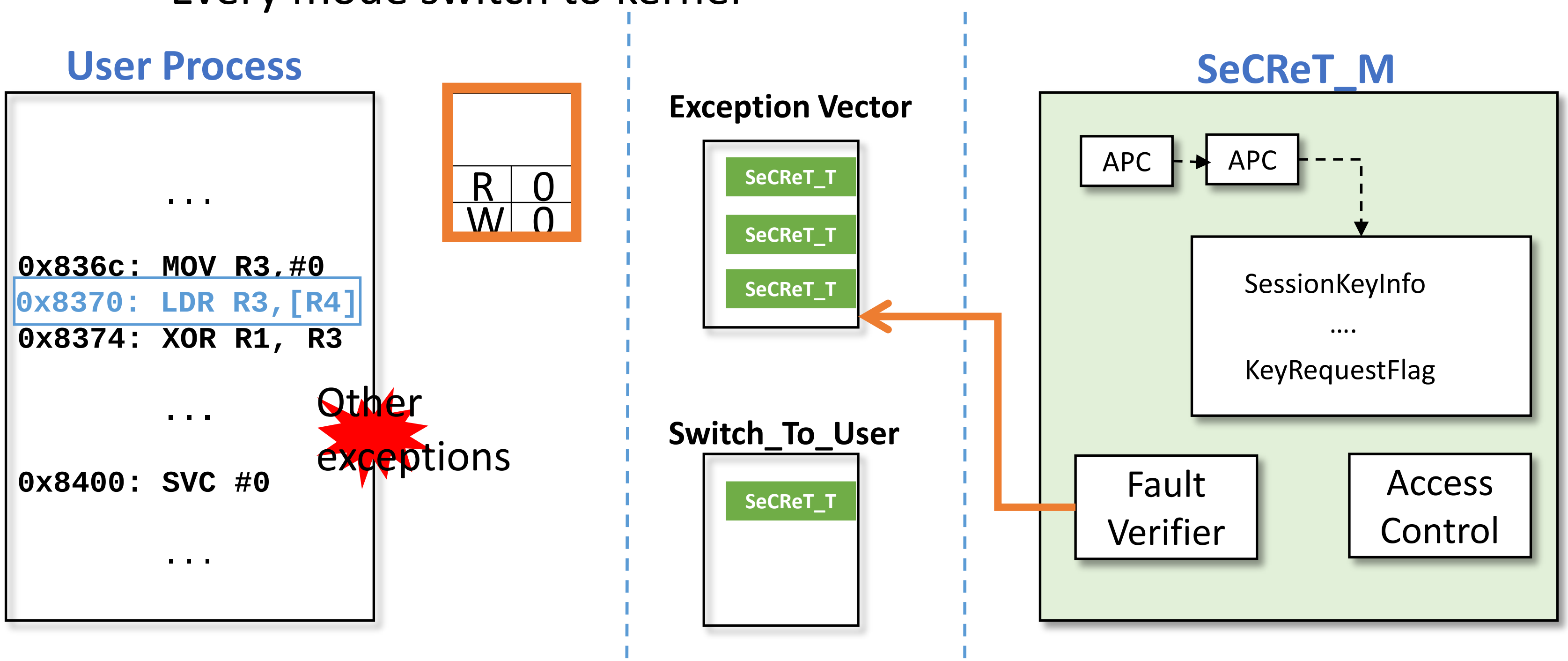
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



Control-flow for the access control to the session key

Access Control to the Key

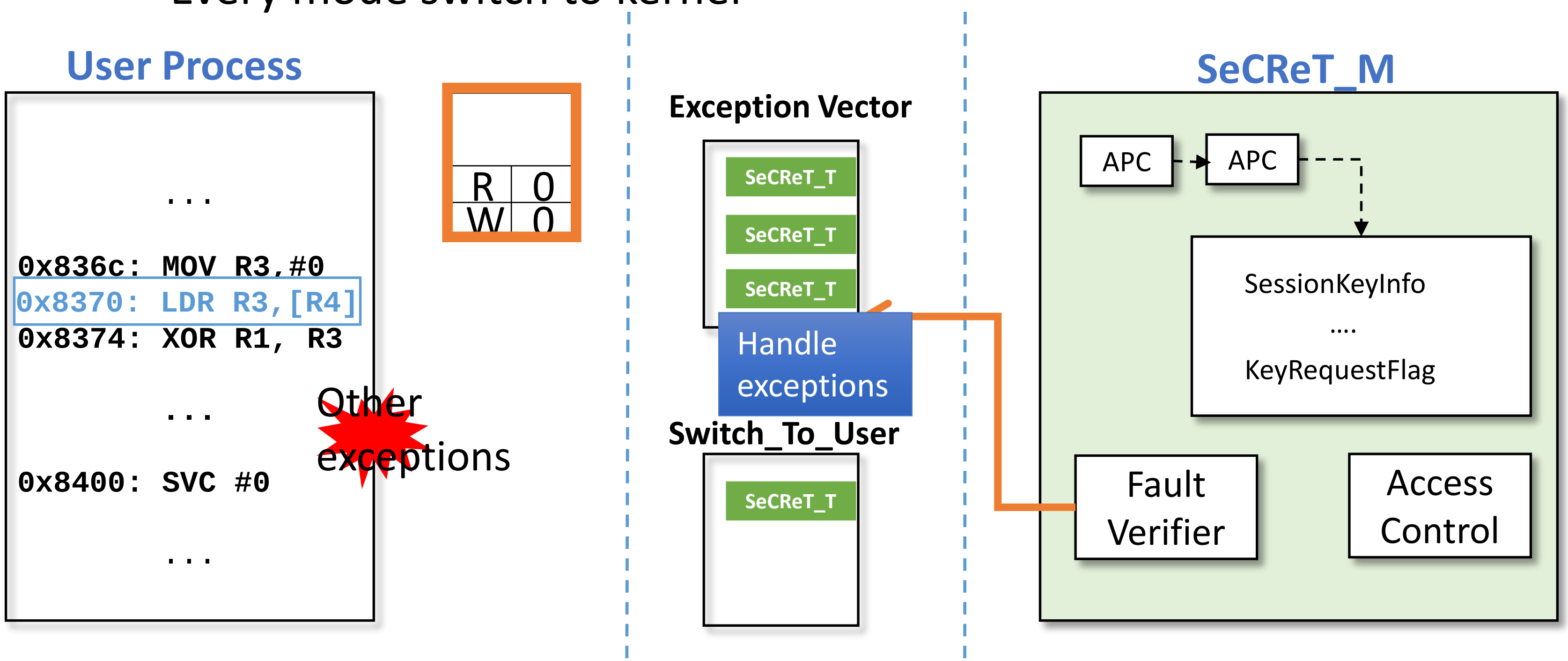
- Key assignment
  - Data abort in the page reserved for key-assignment
  - Hash-check for code area
- Key flush
  - Every mode switch to kernel



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

Coarse-Grained CFI (1/2)

- Attackers can try to exfiltrate the key by
  - Manipulating the process' code area

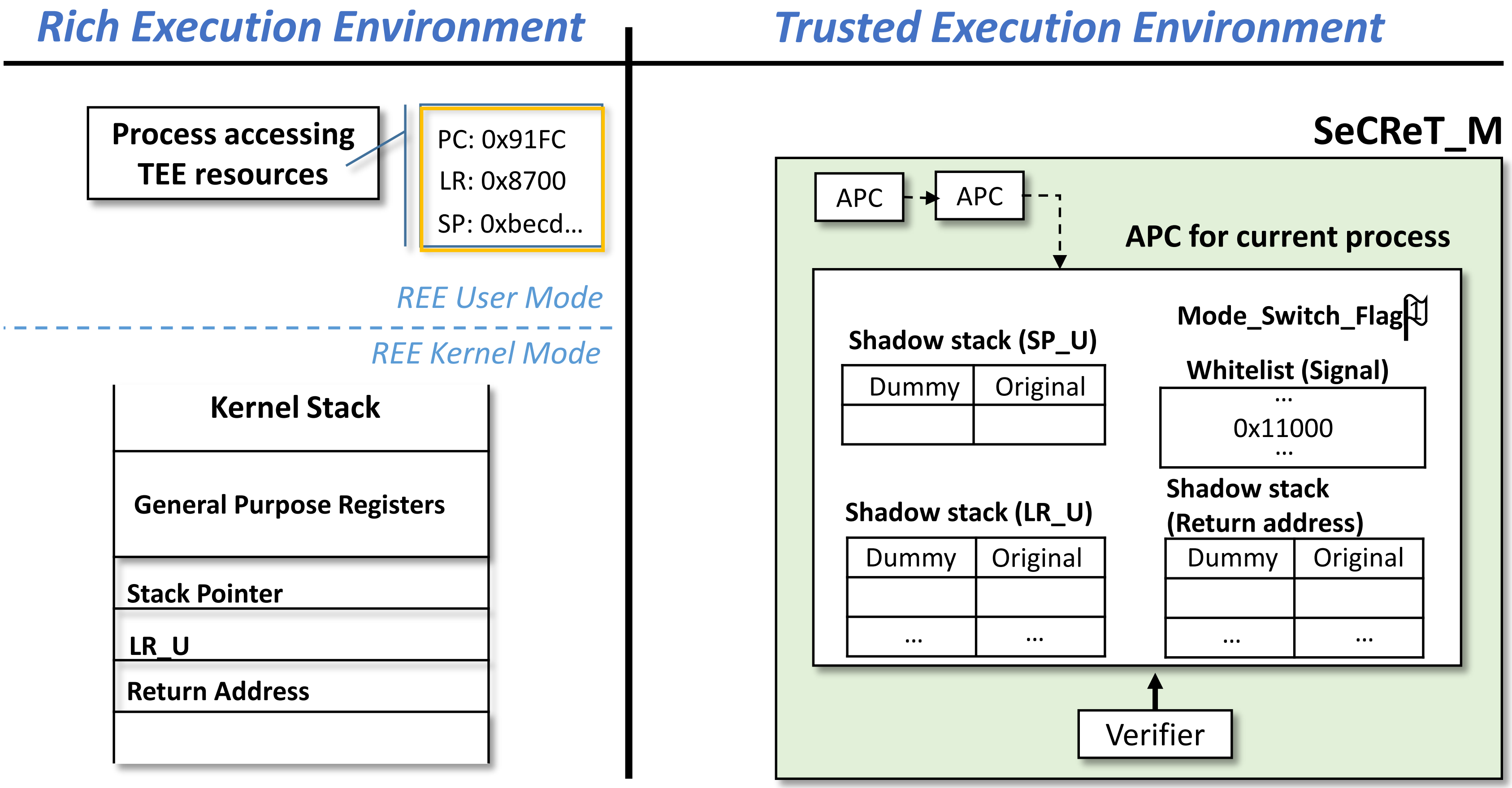
  → **Hash-check for code area**
  - Directly mapping the protected memory area

  → **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)

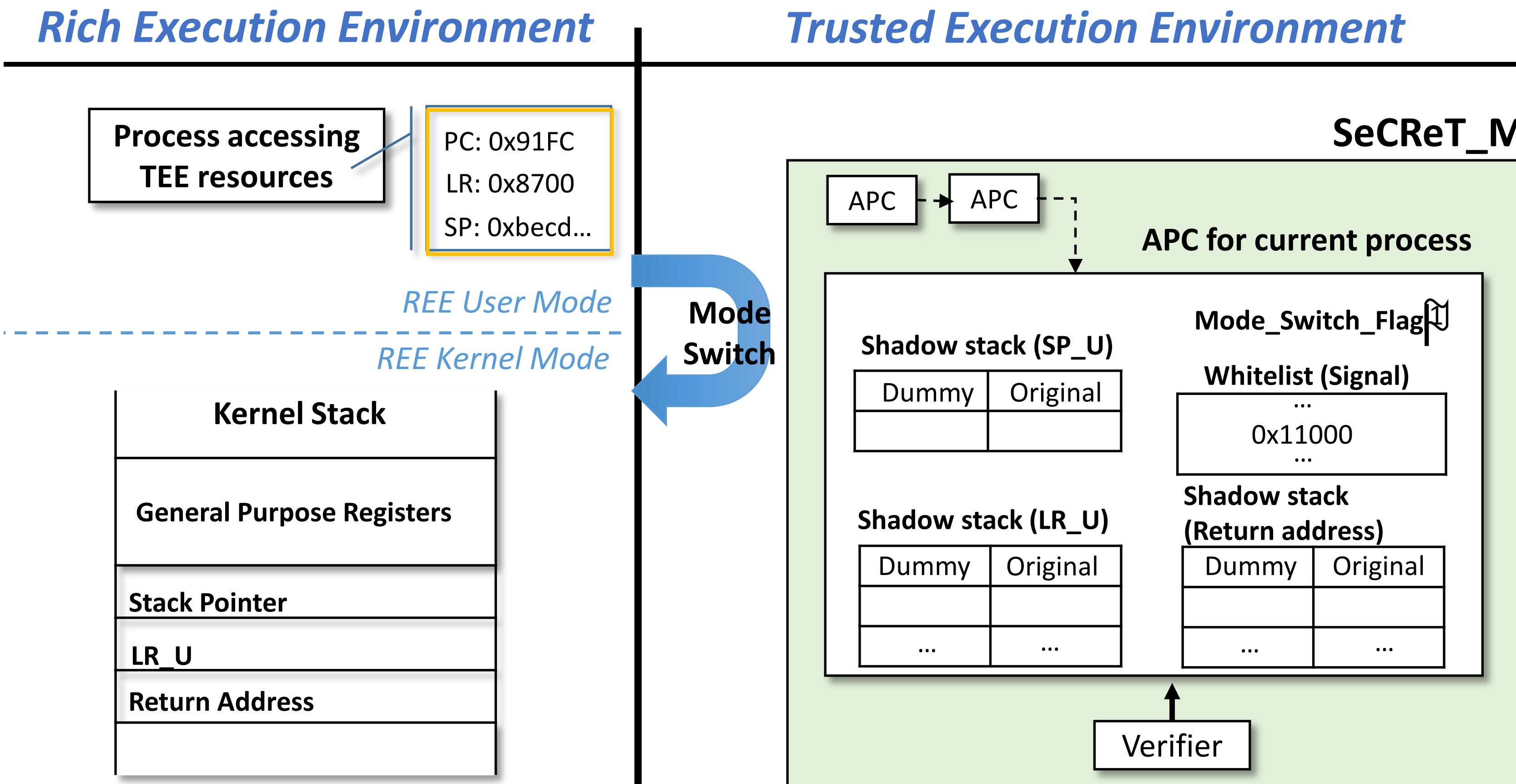
- Protection of user-mode context



Shadow stacks for protection of critical values

Coarse-Grained CFI (2/2)

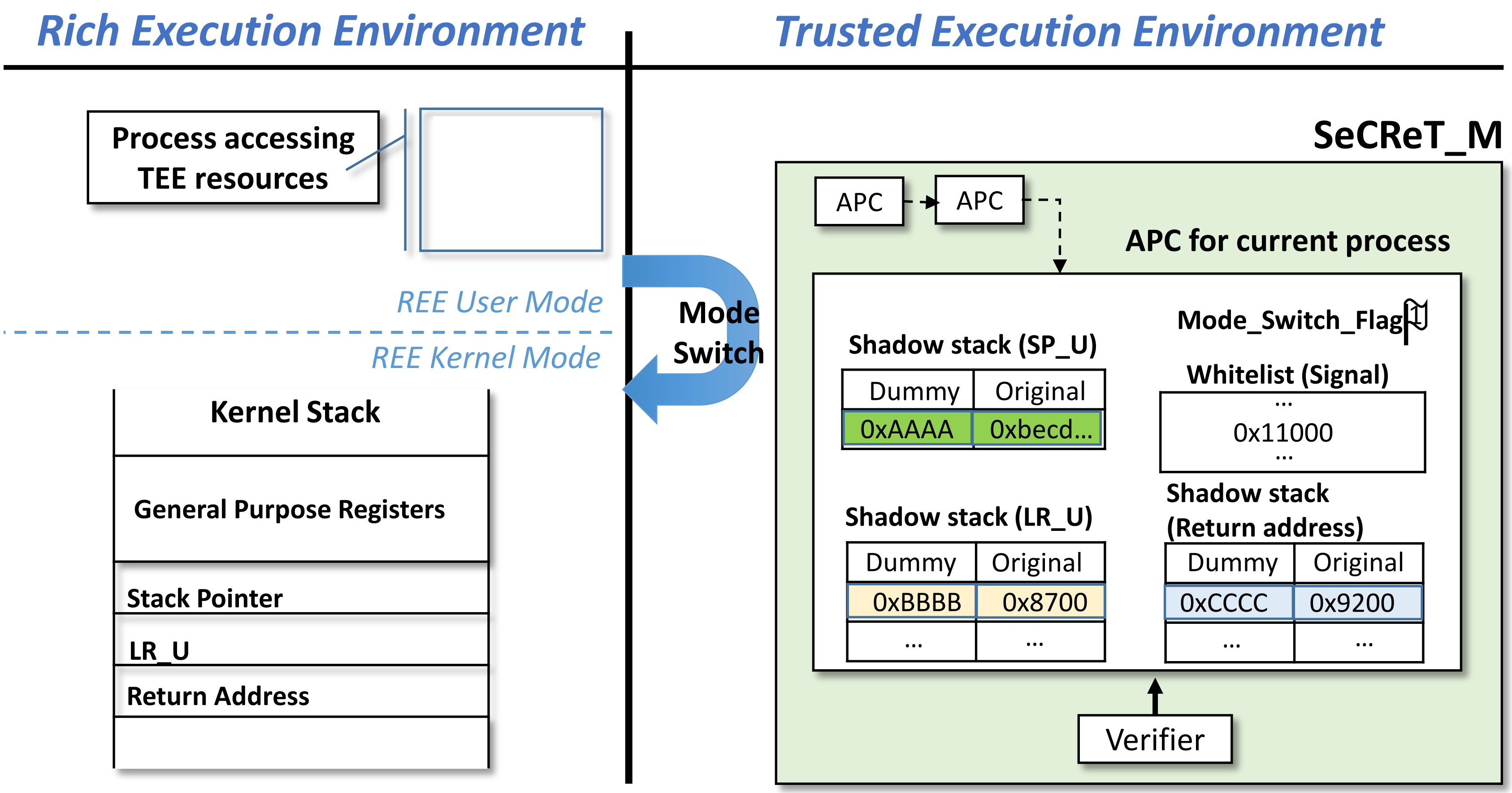
- Protection of user-mode context



Shadow stacks for protection of critical values

Coarse-Grained CFI (2/2)

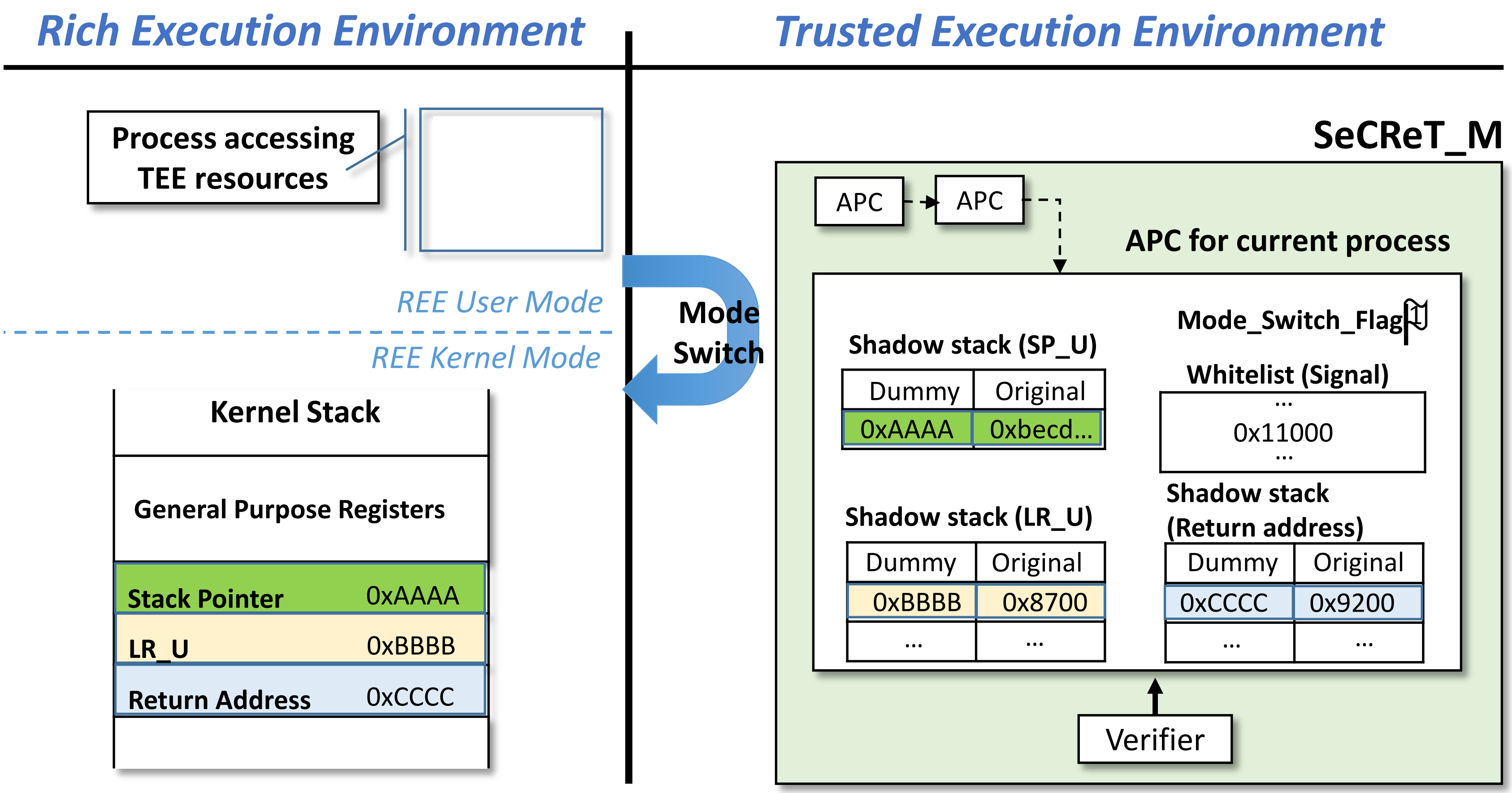
- Protection of user-mode context



Shadow stacks for protection of critical values

Coarse-Grained CFI (2/2)

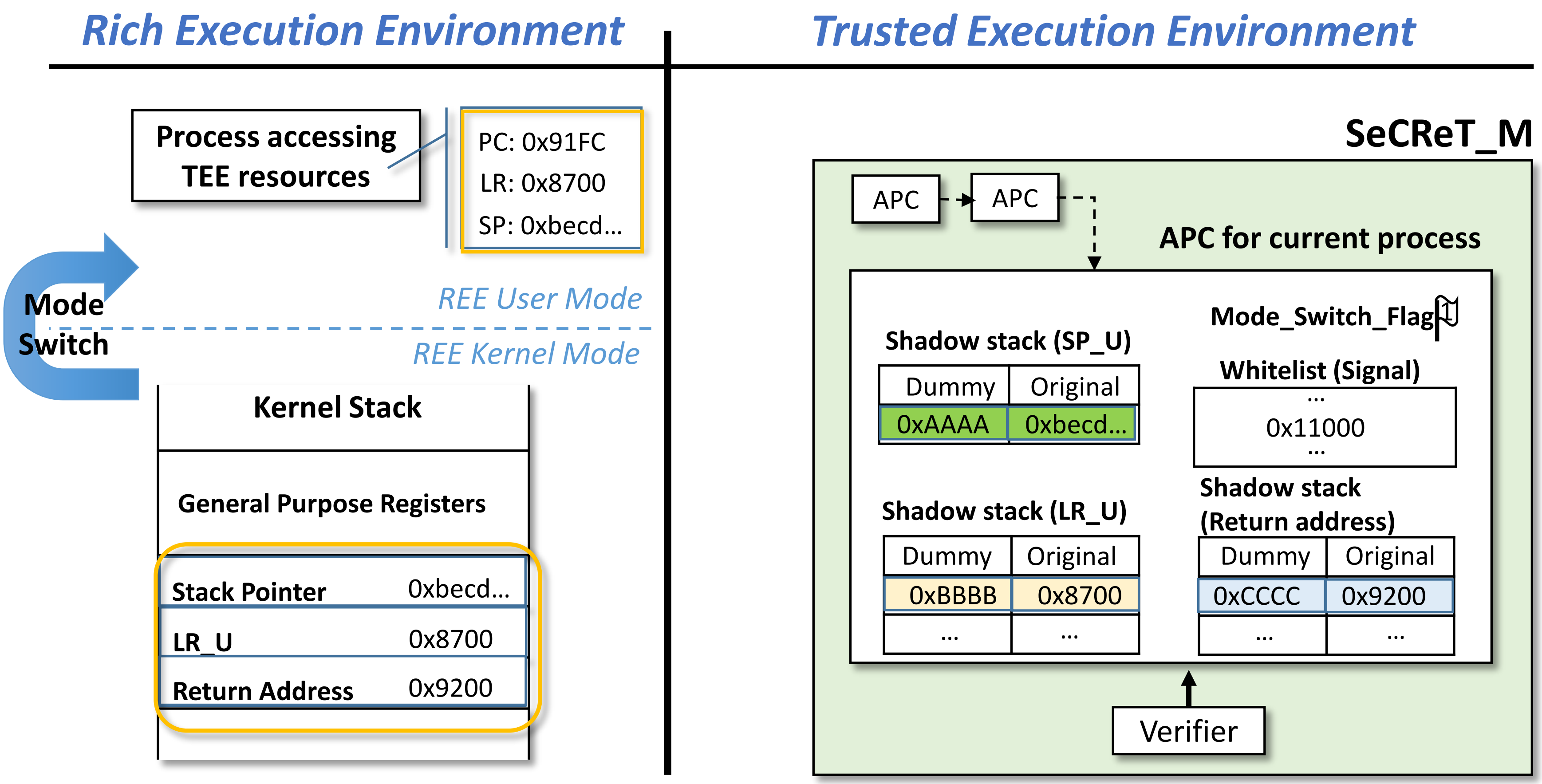
- Protection of user-mode context



Shadow stacks for protection of critical values

Coarse-Grained CFI (2/2)

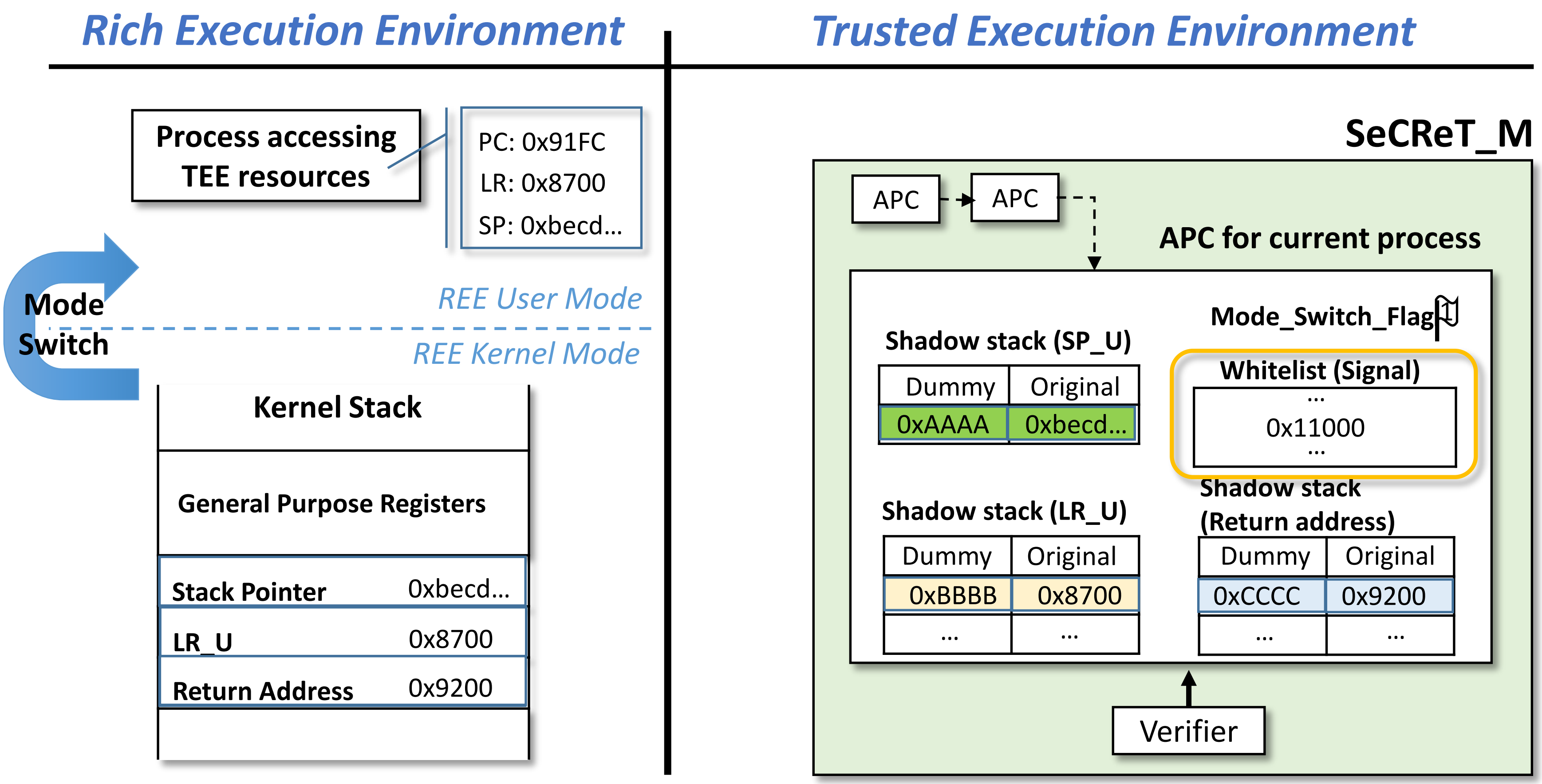
- Protection of user-mode context



Shadow stacks for protection of critical values

Coarse-Grained CFI (2/2)

- Protection of user-mode context



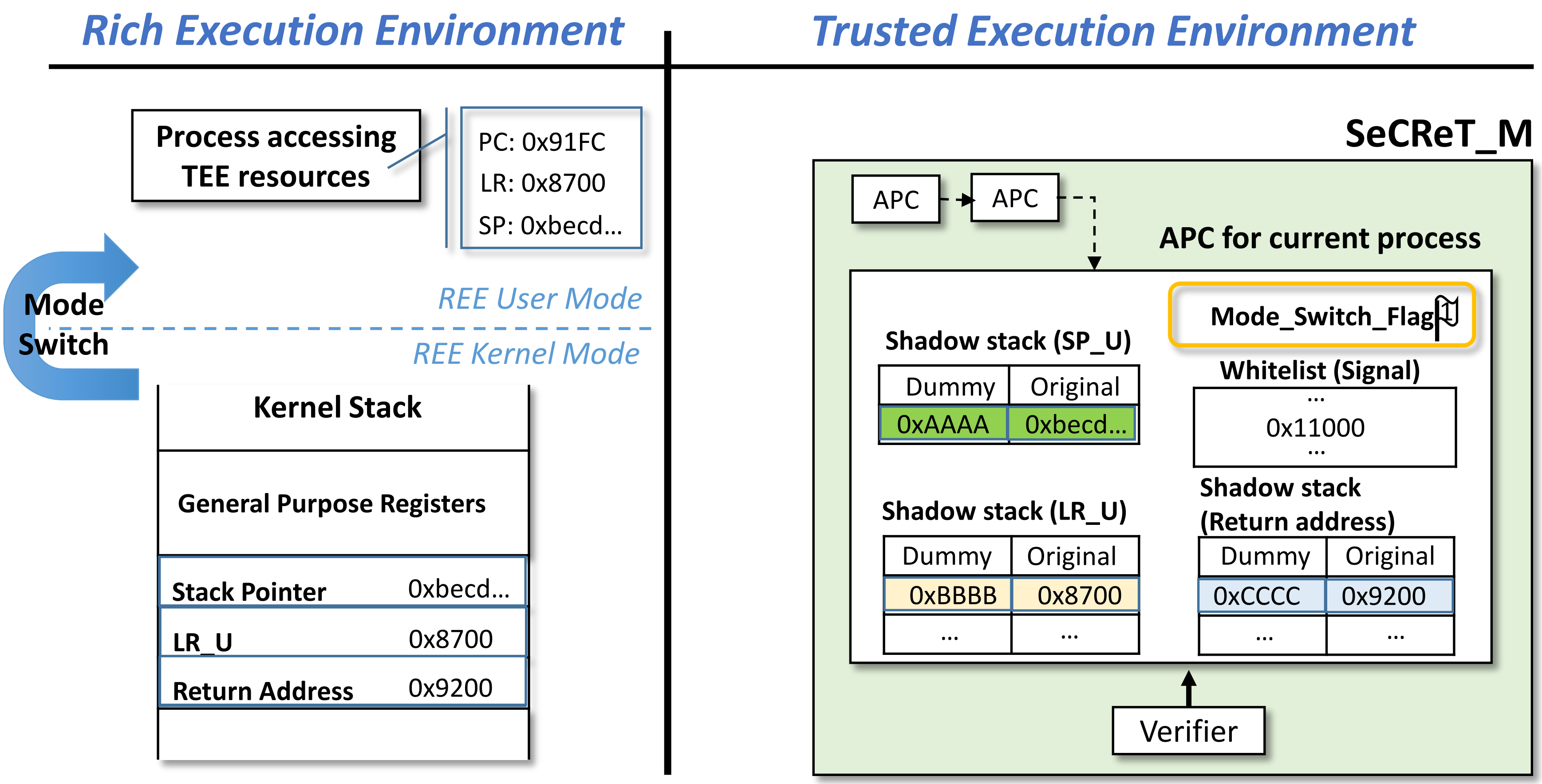
Shadow stacks for protection of critical values

KAIST

CySecLab

Coarse-Grained CFI (2/2)

- Protection of user-mode context



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

Type		Usage in SeCReT
Kernel code	Exception Vector	• SeCReT Trampoline
Kernel code	process execution and termination	• SeCReT Trampoline
Register	Translation Table Base Register (TTBR)	• APC lookup
Register	Data Fault Status Register (DFSR)	• exception verification
Register	Data Fault Address Register (DFAR)	• exception verification
Register	Vector Base Address Register (VBAR)	• Exception vector remapping
Register	System Control Register (SCTLR)	• Exception vector remapping

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

Operation	Linux	SeCReT	Overhead
Null	**0.27**	**1.06**	**3.9259x**
Read	0.33	1.23	3.7273x
Write	0.42	1.57	3.7381x
Open/Close	5.43	8.83	1.6264x
Fork	147.78	174.66	1.1819x
Fork/exec	160.32	189.03	1.1781x

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

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

Test Environment
Linux
SeCReT-enabled Linux
SeCReT-enabled w/ key protection

KAIST CySecLab

Key Access-Control Overhead

- Average latency after running 10 times for each payload

Payload Size (Bytes)	Linux	SeCReT Enabled		SeCReT w/ Key Protection	
	Time	Time	Overhead	Time	Overhead
128	1334.6	1544.5	15.73%	1979.0	48.28%
256	1642.5	1912.1	16.41%	2425.8	47.69%
512	2279.4	2509.8	10.11%	3068.2	34.61%
1024	3650.9	3822.6	4.70%	4516.7	23.71%
2048	340225.7	340244.6	0.01%	341531.4	0.38%
4096	679761.2	679818.7	0.01%	681604.3	0.27%
8192	1693561.2	1693683.6	0.01%	1696639.1	0.18%

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