Analysis of Encryption Key Zeroization from System-wide Perspective

Toyofumi Sawa, Kuniyasu Suzaki
Institute of Information Security
Kanagawa, Japan

Zeroization

- Zeroization: overwriting the key with zeros so that it cannot be recovered.
- Cryptographic keys must be erased after use to prevent leakage.
- Zeroization of secret data is a long-standing standard practice (ISO/IEC 19790, FIPS 140-3).
- Zeroization is not just an application-level action.
- We aim to verify zeroization empirically on actual hardware.

Our Goal & RQs

- Systematically study key zeroization failures across user space, kernel, and hardware.

- Research Questions

  RQ1: When a process terminates normally, are the cryptographic keys in user space zeroized?

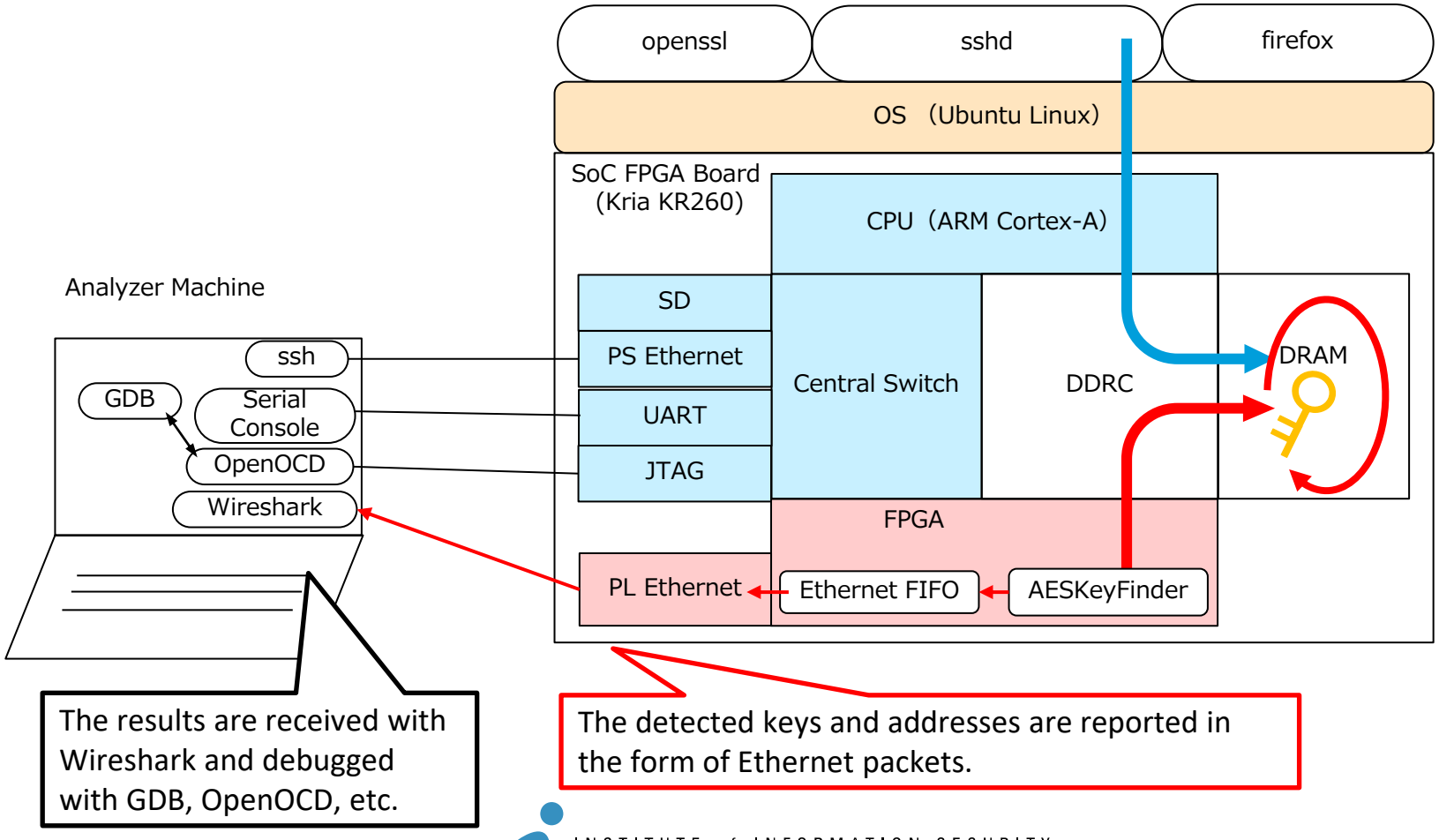
  RQ2: When a process terminates abnormally, are the cryptographic keys in user space zeroized?

  RQ3: Are cryptographic keys duplicated into memory regions that the process cannot read or write?

  RQ4: Do residual keys persist after a system reboot?

Investigation Method

- FPGA periodically scans physical memory and reports detected AES keys; CPU runs target apps normally.



Periodic-AESKeyFinder Design

- FPGA-implemented AESKeyFinder
  - AESKeyFinder is a forensic tool for AES keys, and is known for its cold boot attack paper.
  - Small Modification: Partial key schedule matching

```
1:  procedure P-AESKeyFinder(ReadAddr, ReadRange)
2:      initialize ReadQueue[0 : 15]
3:      CandAddr ← ReadAddr
4:      while true do
5:          for i = 0 to 14 do
6:              ReadQueue[i] ← ReadQueue[i + 1]
7:          end for
8:          ReadQueue[15] ← Read128bitViaAXI(CandAddr)
9:          CandKey ← ReadQueue[0]
10:         KeyScheCand[10 : 1] ← AESKeySchedule(CandKey)
11:         if KeyScheCand[10 : 1] ∩ ReadQueue[15 : 1] ≠ ∅ then
12:             output(CandKey, CandAddr)
13:         end if
14:         if CandAddr < ReadRange then
15:             CandAddr ← CandAddr + 1
16:         else
17:             CandAddr ← ReadAddr
18:         end if
19:     end while
20: end procedure
```

Full key schedule

```
1  845e74b00: 0fc3b56c309d162d 55cbdd1966fd3458
2  845e74b10: eba2bc81e46109ed d89455c08d5f88d9
3  845e74b20: b5a297925e002b13 e0694a8b38fd1f4b
4  845e74b30: d643455363e1d2c1 0ed71093eebe5a18
5  845e74b40: 69099950bf4adc03 8960d3db87b7c348
6  845e74b50: 6fe4952506ed0c75 613385b6e853566d
7  845e74b60: 27e65ae74802cfc2 ae86893ccfb50c8a
8  845e74b70: 8400d1c2a3e68b25 e53354744bb5dd48
9  845e74b80: b53f9947313f4885 1bb9107bfe8a440f
10 845e74b90: a5af871310901e54 409cd3675b25c31c
11 845e74ba0: 303647179599c004 2b8f576c6b13840b
12 845e74bb0: 0000000000000000 0000000000000000
13 845e74bc0: 0000000000000000 0000000000000000
```

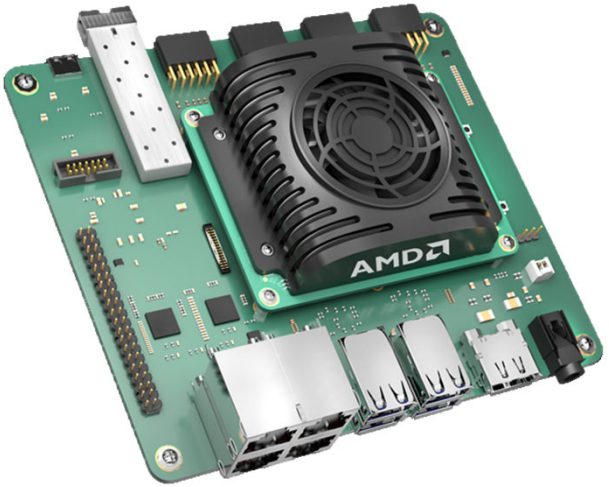
Partial key schedule

```
1  8033e9d90: 0fc3b56c309d162d 55cbdd1966fd3458
2  8033e9da0: eba2bc81e46109ed d89455c08d5f88d9
3  8033e9db0: 05f006f2f39d44f3 2bf60ccac89f43a6
4  8033e9dc0: 17f6ea6224e7872e cad13992c8c8f466
5  8033e9dd0: 27e65ae74802cfc2 ae86893ccfb50c8a
6  8033e9de0: 8400d1c2a3e68b25 e53354744bb5dd48
7  8033e9df0: b53f9947313f4885 1bb9107bfe8a440f
8  8033e9e00: a5af871310901e54 409cd3675b25c31c
9  8033e9e10: 05f006f2f39d44f3 2cf60ccac89f43a6
10 8033e9e20: 05f006f2f39d44f3 2df60ccac89f43a6
11 8033e9e30: d823a4ef35a44bdc 99101d42a81c5753
12 8033e9e40: d3e21b5769b7e96a 89ab4b04e6efd5ae
13 8033e9e50: 0000000000000000 0000000000000000
```

[17] J. Alex Halderman et al. “Lest We Remember: Cold Boot Attacks on Encryption Keys”. In: 17th USENIX Security Symposium (USENIX Security 08)

Experimental Setup

- FPGA + Machine



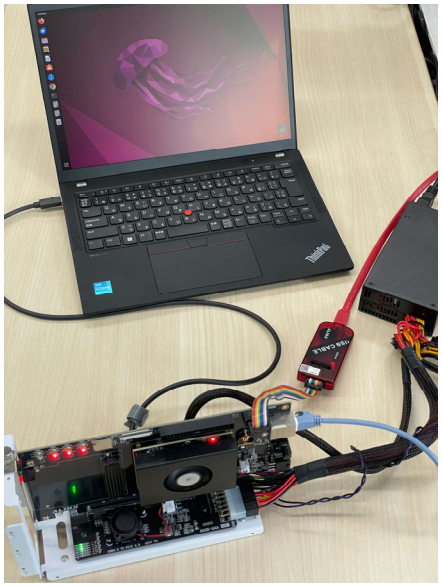


TABLE 1: Target Platform

Environment	SoC	Intel Desktop	AMD Desktop	Laptop
FPGA Board	Xilinx KR260	ALINX AXKU062	ALINX AXKU062	ALINX AXKU3
CPU	Cortex-A53	Core i5-14500	Ryzen 5 8500G	Core i3-1315U
Memory	4 GiB DDR4	32 GiB DDR5	32 GiB DDR5	8 GiB LPDDR5
Motherboard	—	ASUS PRIME B760M-A	ASUS PRIME B650M-A II-CSM	ThinkPad X13 (Gen 4)
BIOS ver.	U-Boot 2022.01	1002	1807	N3OET35W
FPGA Connection	On-chip AXI bus (128 bit width, up to 333 MHz)	PCIe 3.0 ×4	PCIe 3.0 ×4	Thunderbolt 4 (PCIe Tunneling)

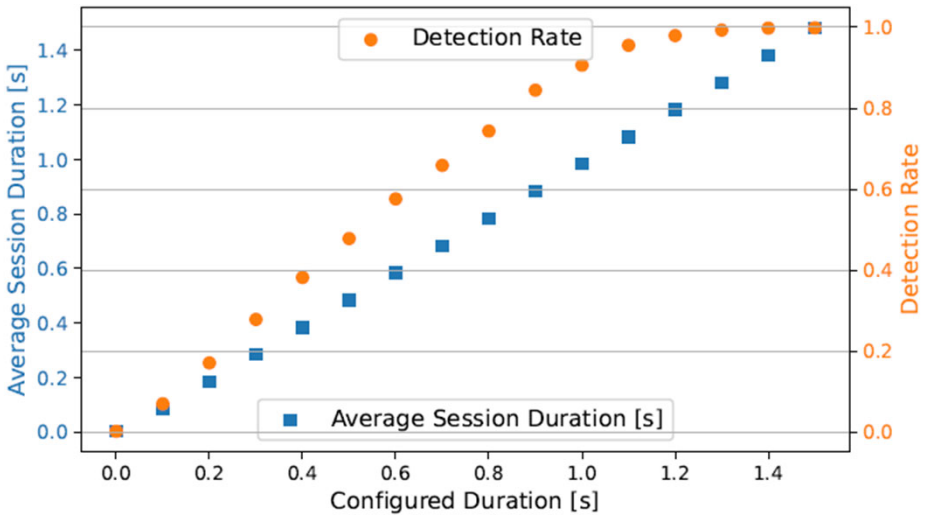
Evaluation

- Original vs. Periodic-AESKeyFinder

	AES-128			AES-256		
		Periodic-AESKeyFinder			Periodic-AESKeyFinder	
Environment	Original	Full	Partial	Original	Full	Partial
SoC (Ubuntu)	169	655	3	35	93	15
Intel Desktop (Ubuntu)	105	164	0	142	177	0
Intel Desktop (Windows)	22	22	0	105	92	3

- Detection Rate
  - Experiment with short-lived TLS sessions

$$\text{key detection rate} \cong \frac{key\ lifetime}{scan\ cycle}$$



Demonstration of Zeroization Failures

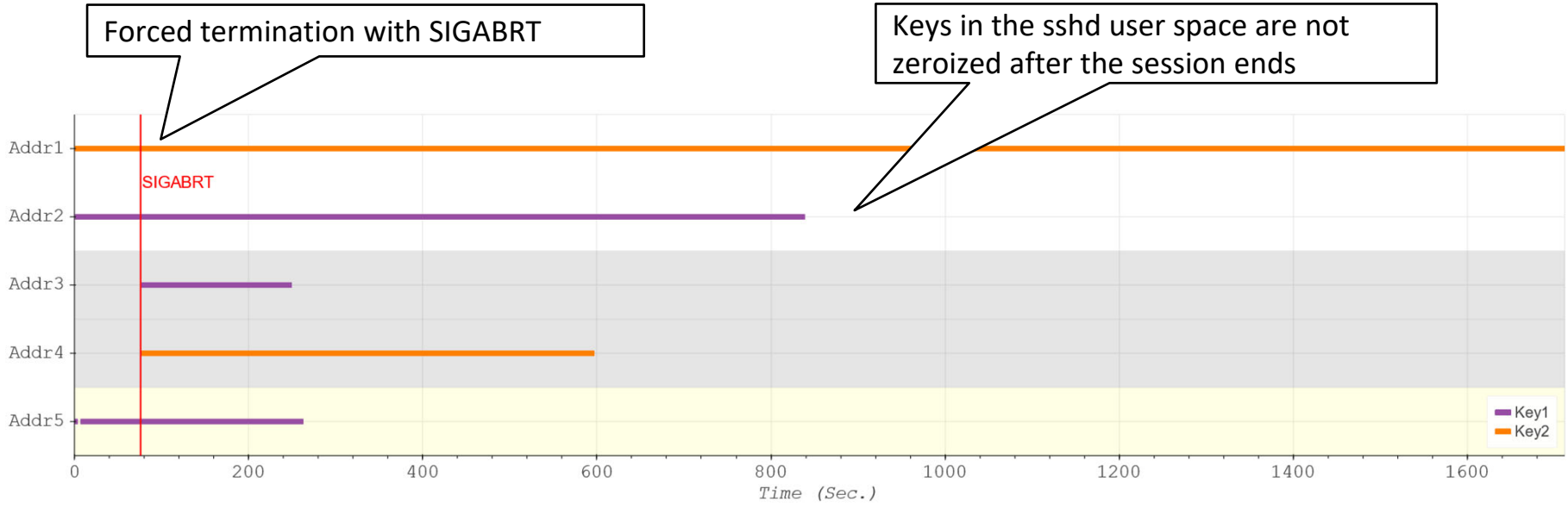
- Experiment with multiple combinations of apps, libraries, operating systems, and architectures.

RQ1 RQ2 RQ3

Project	Library	OS	Exit	SIGKILL	SIG-term	SIG-core	Context Switch	Core dump
OpenSSH	OpenSSL	Ubuntu 22.04 (AArch64)	✓	✗	✗	✗	✗	✗
OpenSSH	OpenSSL	Ubuntu 22.04 (x86-64)	✓	✗	✗	✗	✓	✗
OpenSSH	LibreSSL	Windows 11 (x86-64)	✓	✗	-	-	✓	-
OpenSSL	OpenSSL	Ubuntu 22.04 (AArch64)	✓	✗	✗	✗	✗	✗
OpenSSL	OpenSSL	Ubuntu 22.04 (x86-64)	✓	✗	✗	✗	✓	✗
Firefox	LibNSS	Ubuntu 22.04 (AArch64)	✓	✗	✗	✗	✗	✗
Firefox	LibNSS	Ubuntu 22.04 (x86-64)	✓	✗	✗	✗	✓	✗
Chromium	BoringSSL	Ubuntu 22.04 (AArch64)	✓	✗	✗	✗	✓	-
Chromium	BoringSSL	Ubuntu 22.04 (x86-64)	✓	✗	✗	✗	✓	-
Edge	Schannel	Windows 11 (x86-64)	✓	✗	✗	-	✓	-
7-Zip	Original	Windows 11 (x86-64)	✗	✗	✗	-	✓	-

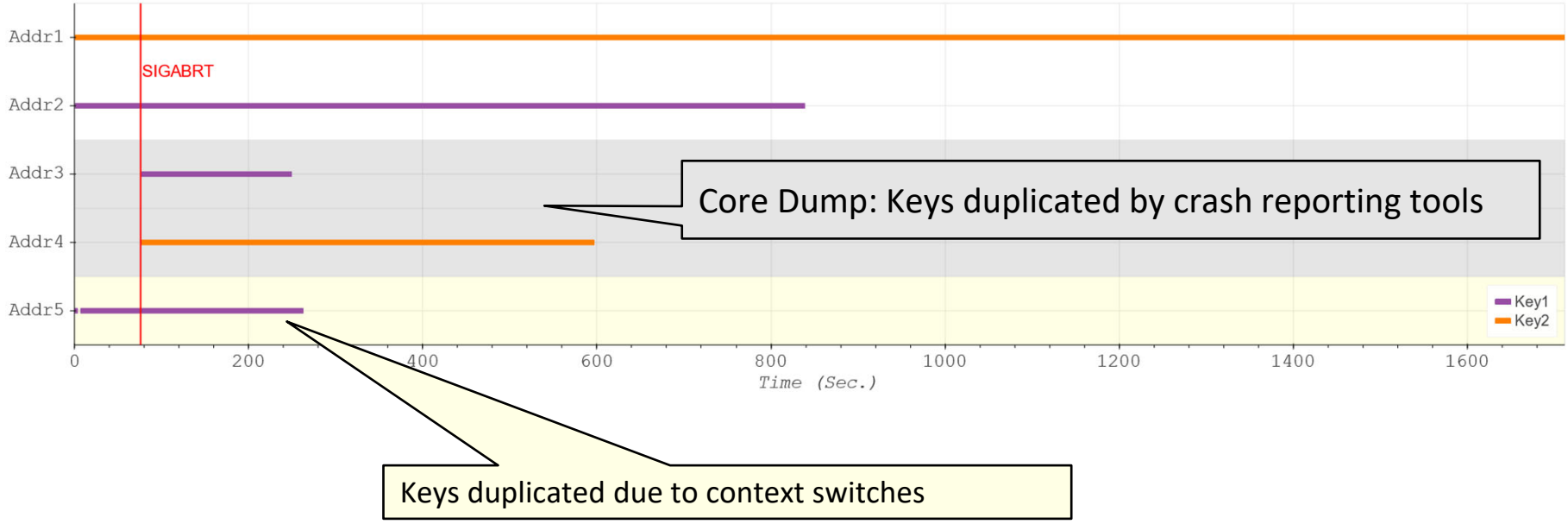
Zeroization upon Abnormal App. Termination (RQ2)

- Key appearance timeline when sending SIGABRT to sshd during an SSH session.
- OpenSSH uses two AES session keys per connection.



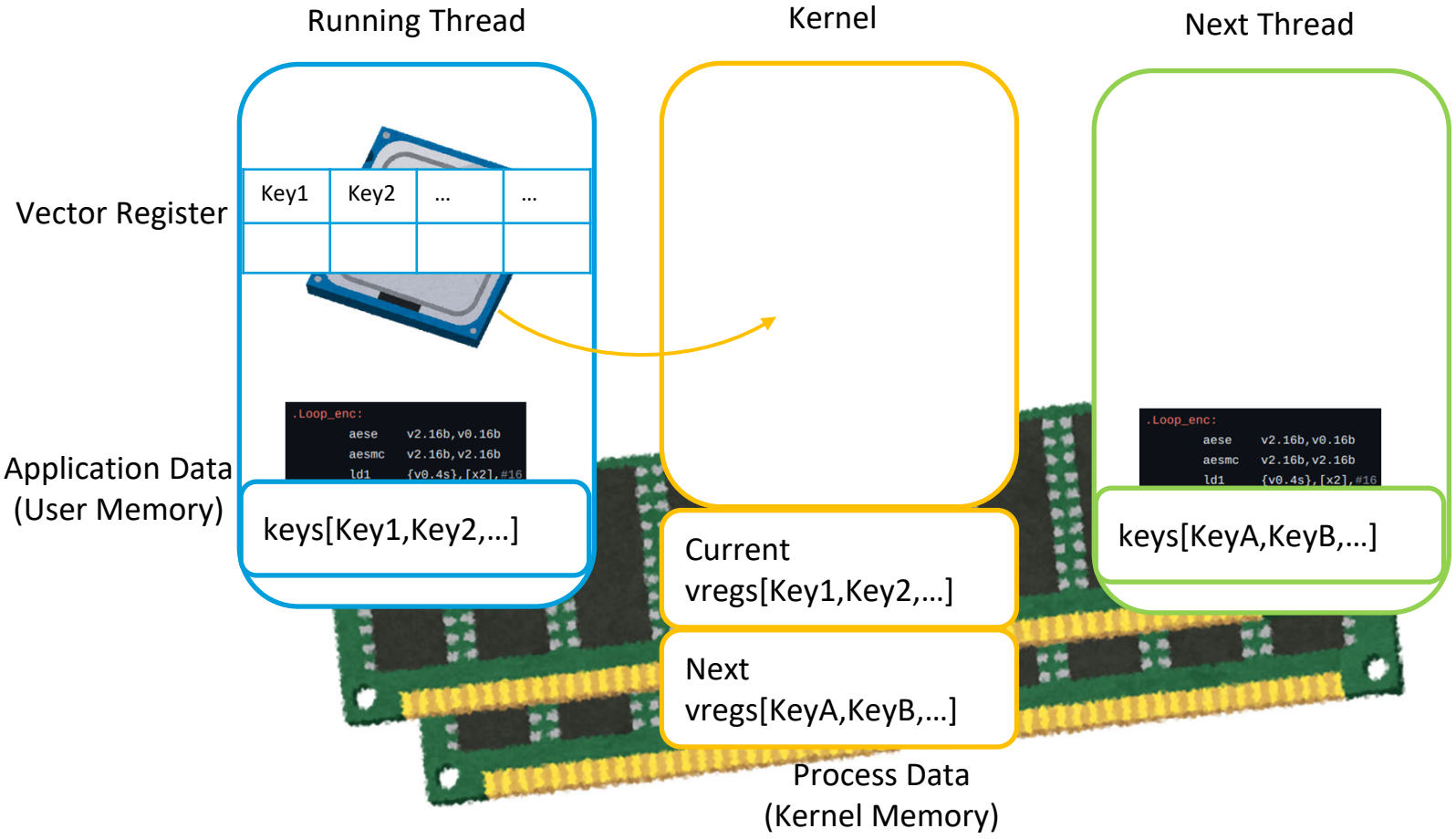
Duplication of Detected Cryptographic Keys (RQ3)

- Two duplication paths:
  - Core Dump
  - Context Switch



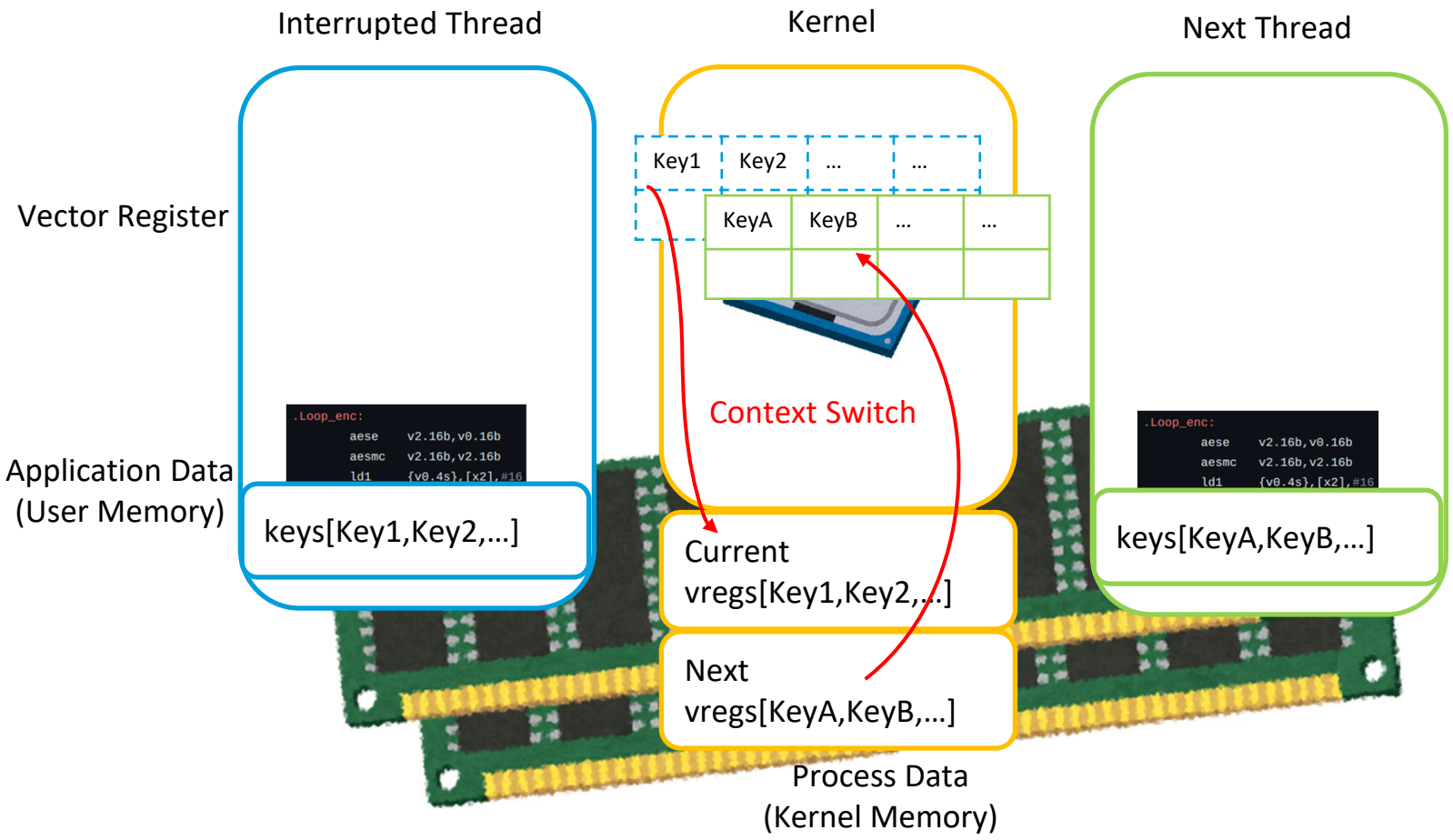
Key duplication caused by vector-register context switching (1/3)

- Context switch: switching threads running on the CPU
- When an interrupt occurs, the running thread yields the CPU to the kernel.



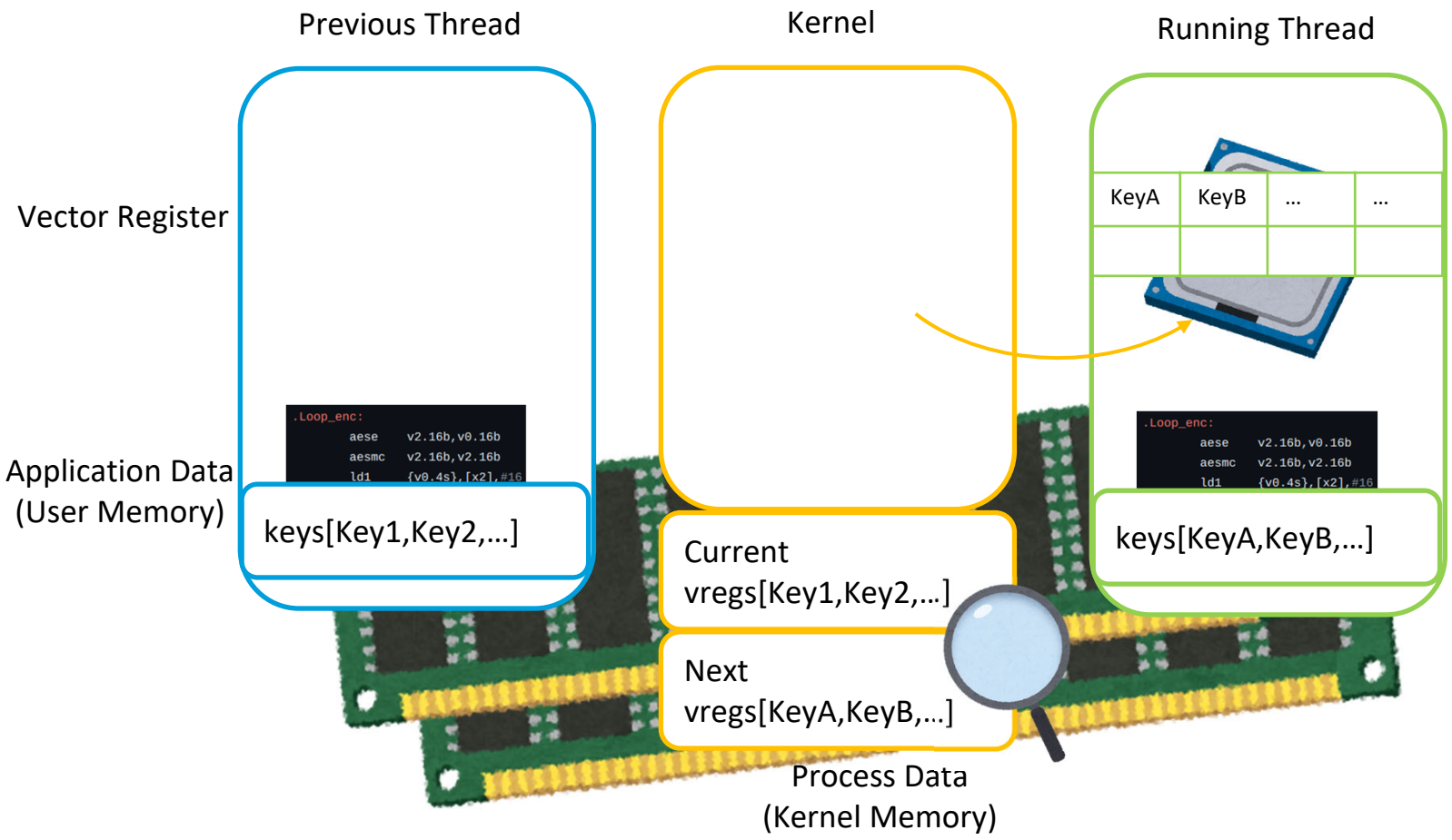
Key duplication caused by vector-register context switching (2/3)

The register contents of the previous process are saved into kernel memory, and the next process's register contents are loaded into the CPU to resume execution.



Key duplication caused by vector-register context switching (3/3)

Keys duplicated into kernel space during context switching were observed.



Key Persistence after OS Reboot (RQ4)

- FIPS-compliant Linux zeroizes on reboot.
- Other environments sometimes retain keys.

OS (Environment)	Warm boot	After kernel crash	Reset switch
Linux 5.15.0-1038-Xilinx-zynqmp(SoC)	✗	✗	✗
Linux-5.15.0-130-fips (Intel Desktop)	✓	✓	✓
Linux-6.8.0-50-generic (Intel Desktop)	✗	✓	✓
Linux-6.8.0-50-generic (AMD Desktop)	✓	✓	✓
Linux-6.8.0-50-generic (Laptop)	✗	✓	-
Windows 11 Build 22621 FIPS (Intel Desktop)	✗	✓	✓
Windows 11 Build 22621 (Intel Desktop)	✗	✓	✓

Key Persistence after OS Reboot (RQ4)

- FIPS-compliant Linux zeroizes on reboot.
- Other environments sometimes retain keys.
- PCs zeroize memory after crash or hardware reset.

OS (Environment)	Warm boot	After kernel crash	Reset switch
Linux 5.15.0-1038-Xilinx-zynqmp(SoC)	✗	✗	✗
Linux-5.15.0-130-fips (Intel Desktop)	✓	✓	✓
Linux-6.8.0-50-generic (Intel Desktop)	✗	✓	✓
Linux-6.8.0-50-generic (AMD Desktop)	✓	✓	✓
Linux-6.8.0-50-generic (Laptop)	✗	✓	-
Windows 11 Build 22621 FIPS (Intel Desktop)	✗	✓	✓
Windows 11 Build 22621 (Intel Desktop)	✗	✓	✓

Key Persistence after OS Reboot (RQ4)

- FIPS-compliant Linux zeroizes on reboot.
- Other environments sometimes retain keys.
- PCs zeroize memory after crash or hardware reset.
- Keys remain on the SoC even after resetting

OS (Environment)	Warm boot	After kernel crash	Reset switch
Linux 5.15.0-1038-Xilinx-zynqmp(SoC)	✗	✗	✗
Linux-5.15.0-130-fips (Intel Desktop)	✓	✓	✓
Linux-6.8.0-50-generic (Intel Desktop)	✗	✓	✓
Linux-6.8.0-50-generic (AMD Desktop)	✓	✓	✓
Linux-6.8.0-50-generic (Laptop)	✗	✓	-
Windows 11 Build 22621 FIPS (Intel Desktop)	✗	✓	✓
Windows 11 Build 22621 (Intel Desktop)	✗	✓	✓

Discussion: Causes of Zeroization Failueres

1. Missing signal handlers
   → Abnormal termination persistence

2. Lazy page zeroization
   → Residual windows in memory

3. Duplication via register
   → Vector register contents stored in kernel memory during context switching

4. Core dumps
   → Leakage via crash report

Recommendations and Mitigations

- Application Developers
  - Implement signal-handler zeroization for abnormal termination.
  - Exclude sensitive pages from core dumps.
- Library Developers
  - Zeroize temporary key material stored in registers and temporary buffers after cryptographic operations.
- OS Developers
  - Zeroize the kernel memory that stores saved register contexts when a process exits.
  - Implement zeroization for SIGKILL.
- System Managers
  - Configure the system to not generate core dumps.
  - Configure crash-reporting tools to avoid external transmissions.
- Hardware / Platform Vendors
  - Provide clear documentation on memory zeroization behavior during reboot.
- Certification Testers
  - Understand these system-level zeroization issues and require vendors to document how their platforms behave.

Conclusion

- Zeroization cannot be guaranteed by app code alone; OS/hardware interactions matter.
- FPGA-based live forensics revealed duplication and persistence across layers and reboots.
- Disclosed issues to OpenSSH/OpenSSL/Firefox; fixes landed for dump handling.
- Code Available on GitHub

GitHub - Tyojan/Periodic-AESKeyFinder