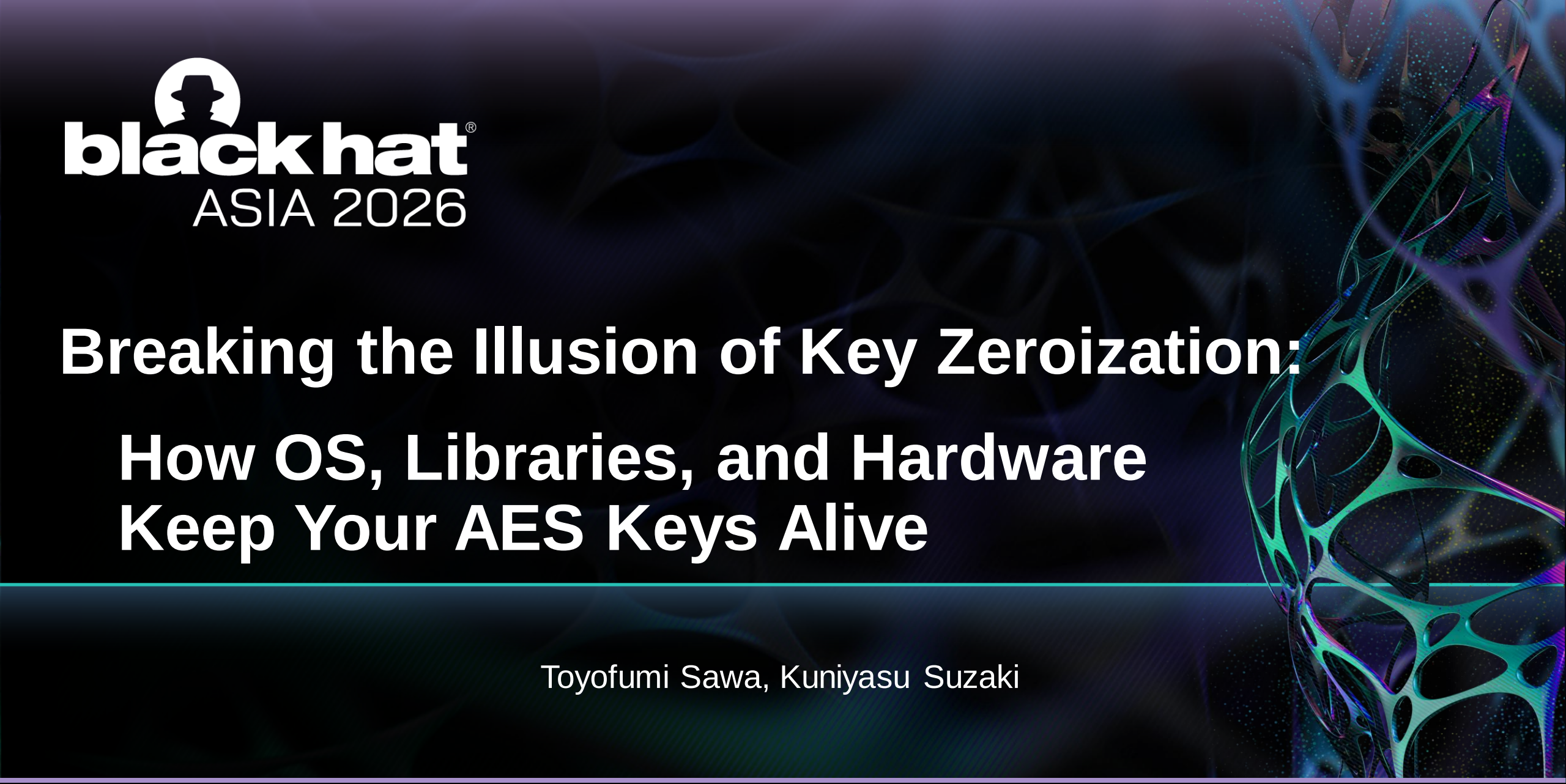


Breaking the Illusion of Key Zeroization:

How OS, Libraries, and Hardware Keep Your AES Keys Alive

Toyofumi Sawa, Kuniyasu Suzaki

Toyofumi Sawa

Ph.D. Student, IISEC

Kuniyasu Suzaki

Professor, IISEC

INSTITUTE of INFORMATION SECURITY

Everything everywhere all encrypted

At Rest

In Transit

In Use

Everything everywhere all encrypted

At Rest

Data in storage is encrypted.

(Windows BitLocker, Mac FileVault, Linux LUKS)

In Transit

In Use

Everything everywhere all encrypted

At Rest

Data in storage is encrypted.

(Windows BitLocker, Mac FileVault, Linux LUKS)

In Transit

Data on network is encrypted.

(TLS, SSH, Signal, Telegram)

In Use

Everything everywhere all encrypted

At Rest

Data in storage is encrypted.

(Windows BitLocker, Mac FileVault, Linux LUKS)

In Transit

Data on network is encrypted.

(TLS, SSH, Signal, Telegram)

In Use

Data in memory is encrypted?

(Confidential Computing, Intel SGX, AMD SEV)

Zeroization (key term)

- Zeroization: overwriting keys in memory.
  - Keys must be zeroized after use.

Zeroization (key term)

- Zeroization: overwriting keys in memory.
  - Keys must be zeroized after use.
- Ransomware without zeroization is NOT threat.

Why zeroization is important?

Zeroization is required from 3 perspectives

defense-in-depth

Residual keys expand the attack surface. Reliable zeroization reduces the chance of successful exploitation.

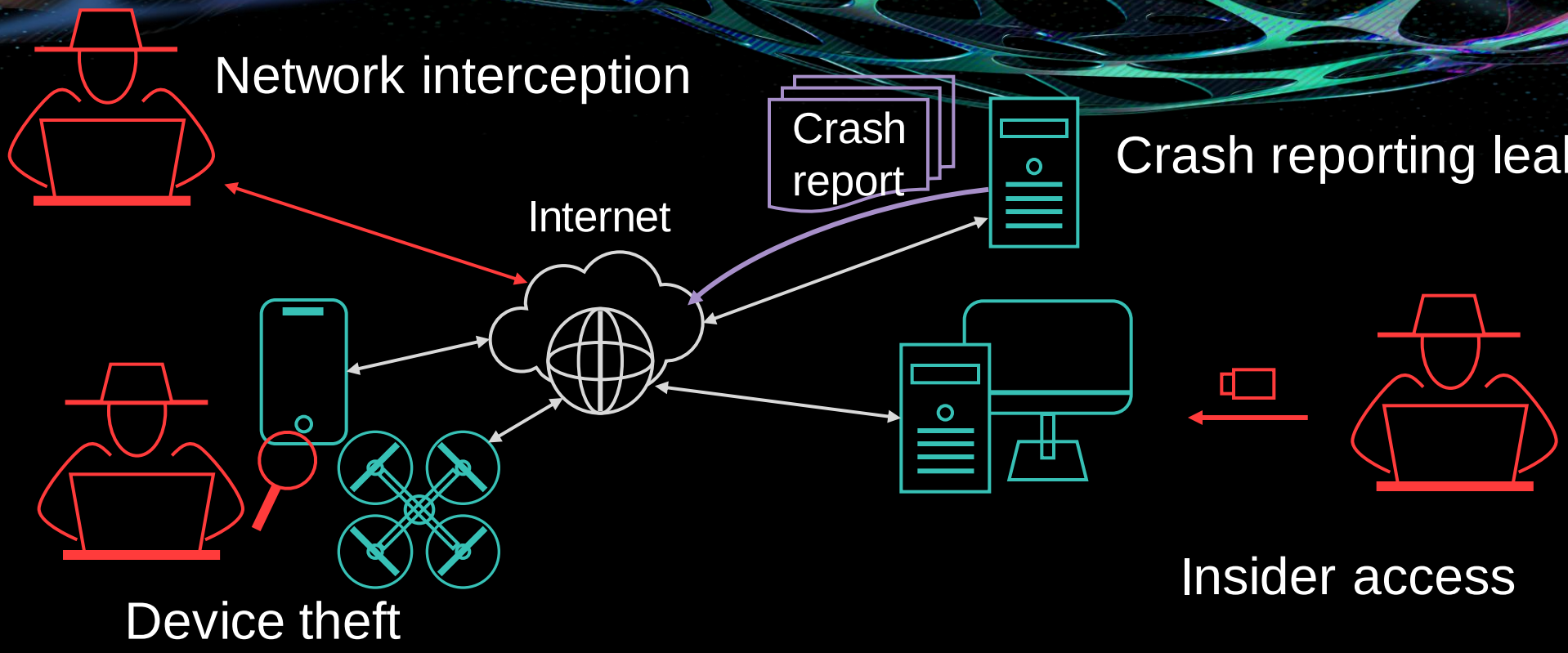
forward secrecy

If key lifetime is unknown, past communications may be exposed after system compromise.

certification

Standards require zeroization of Sensitive Security Parameters (e.g. FIPS 140-3).

Potential key exposure scenarios



We focus on key zeroization as a post-compromise defense for maintaining forward secrecy.

Think about a real-world compromise scenario

The system is compromised, and data is exfiltrated.

The data is safe because it is encrypted ☺
What about the keys? Did they exist in the system during the attack?

For executives:

Can we assure our customers and regulators that the data is safe?

For engineers:

Can we explain how keys are zeroized in our system?

Do we assume that the keys simply disappear ?

OS and API seem to support zeroization

- free() clears memory
- process exit removes secrets
- reboot erases everything

- keys are not copied elsewhere
- the OS takes care of it

Do we assume that the keys simply disappear ?

OS and API seem to support zeroization

- free() clears memory
- process exit removes secrets
- reboot erases everything

- keys are not copied elsewhere
- the OS takes care of it

THIS IS THE ILLUSION OF ZEROIZATION

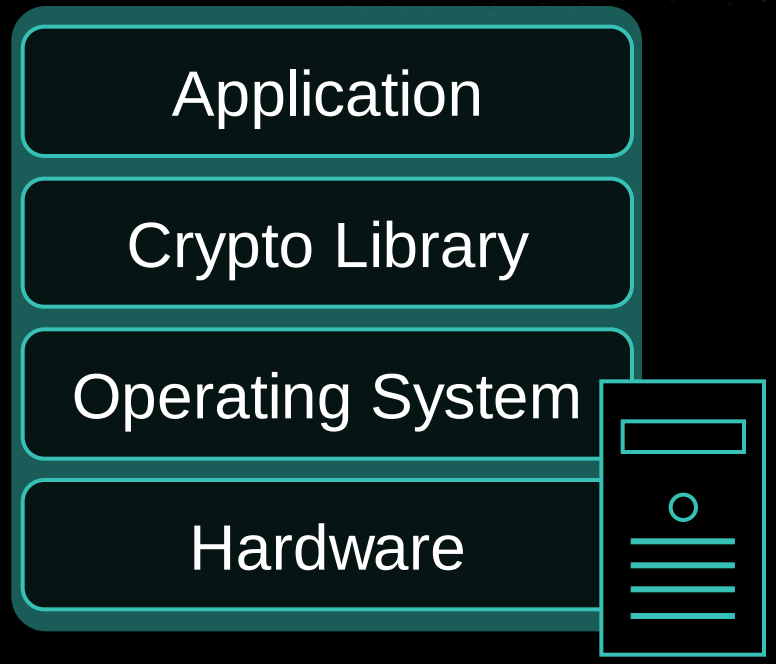
Breaking this illusion requires understanding the entire system stack.

compliance • cryptography • applications • libraries • OS • hardware

Why a system-wide perspective is necessary?

Key lifetime is influenced by interactions across multiple system layers

- Zeroization is usually implemented in applications or libraries
- OS holds (copies) App's state



Zeroization can't be examined from user space only!

Research questions:
Understanding zeroization failures across layers

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**?

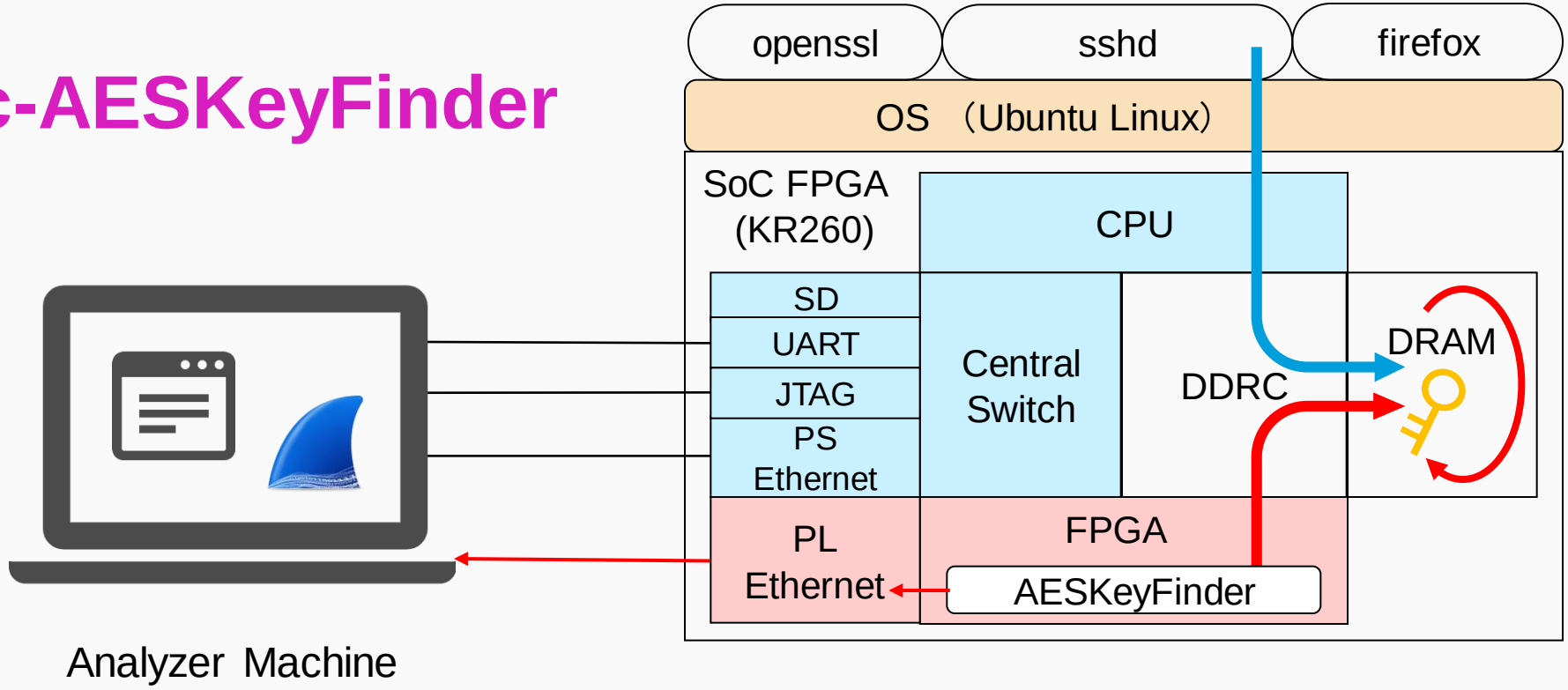
Investigation method

We continuously scan physical memory using an FPGA.

This allows us to observe keys beyond OS control.

Our tool:

Periodic-AESKeyFinder



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

Full key schedule

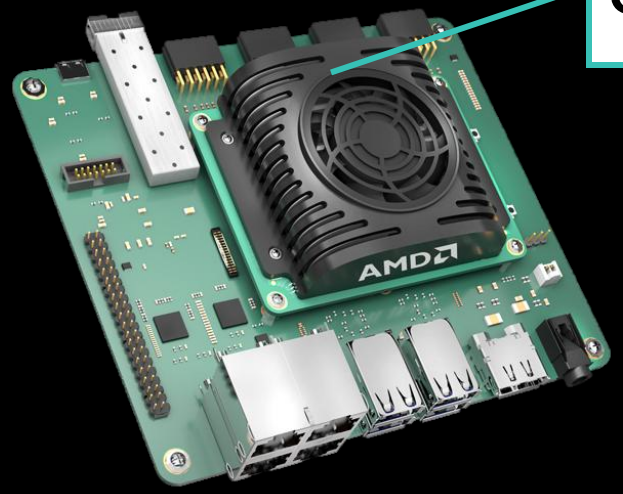
1	845e74b00:	0fc3b56c309d162d 55cbdd1966fd3458
2	845e74b10:	eba2bc81e46109ed d89455c08d5f88d9
3	845e74b20:	b5a297925e002b13 e0694a8b38fd1f4b
4	845e74b30:	d643455363e1d2c1 0ed71093eebe5a18
5	845e74b40:	69099950bf4adc03 8960d3db87b7c348
6	845e74b50:	6fe4952506ed0c75 613385b6e853566d
7	845e74b60:	27e65ae74802cfc2 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

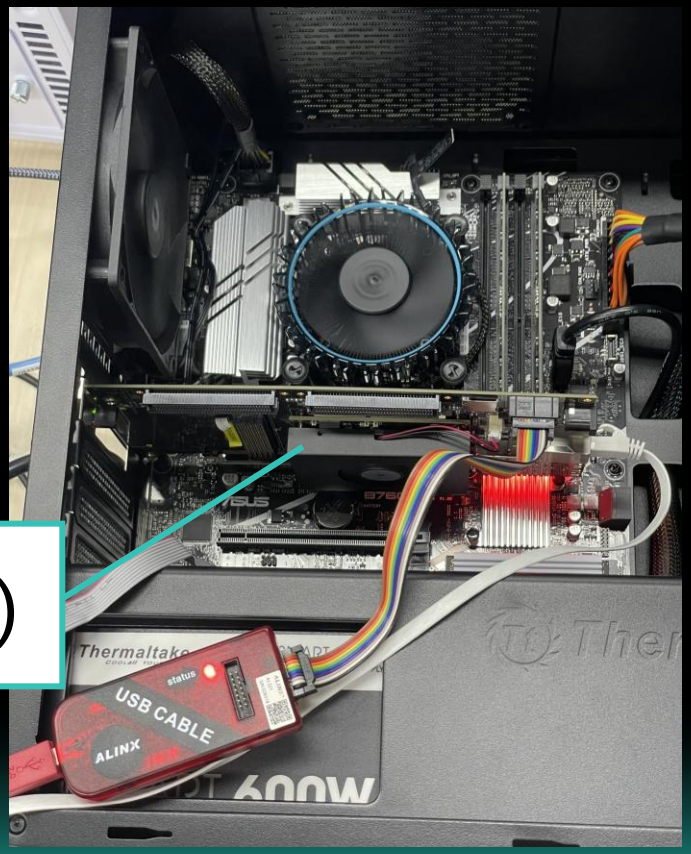
1	8033e9d90:	0fc3b56c309d162d 55cbdd1966fd3458
2	8033e9da0:	eba2bc81e46109ed d89455c08d5f88d9
3	8033e9db0:	05f006f2f39d44f3 2bf60ccac89f43a6
4	8033e9dc0:	17f6ea6224e7872e cad13992c8c8f466
5	8033e9dd0:	27e65ae74802cfc2 ae86893ccfb50c8a
6	8033e9de0:	8400d1c2a3e68b25 e53354744bb5dd48
7	8033e9df0:	b53f9947313f4885 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

Experimental setups

SoC FPGA



Intel Desktop (PCIe)



Laptop (Thunderbolt)

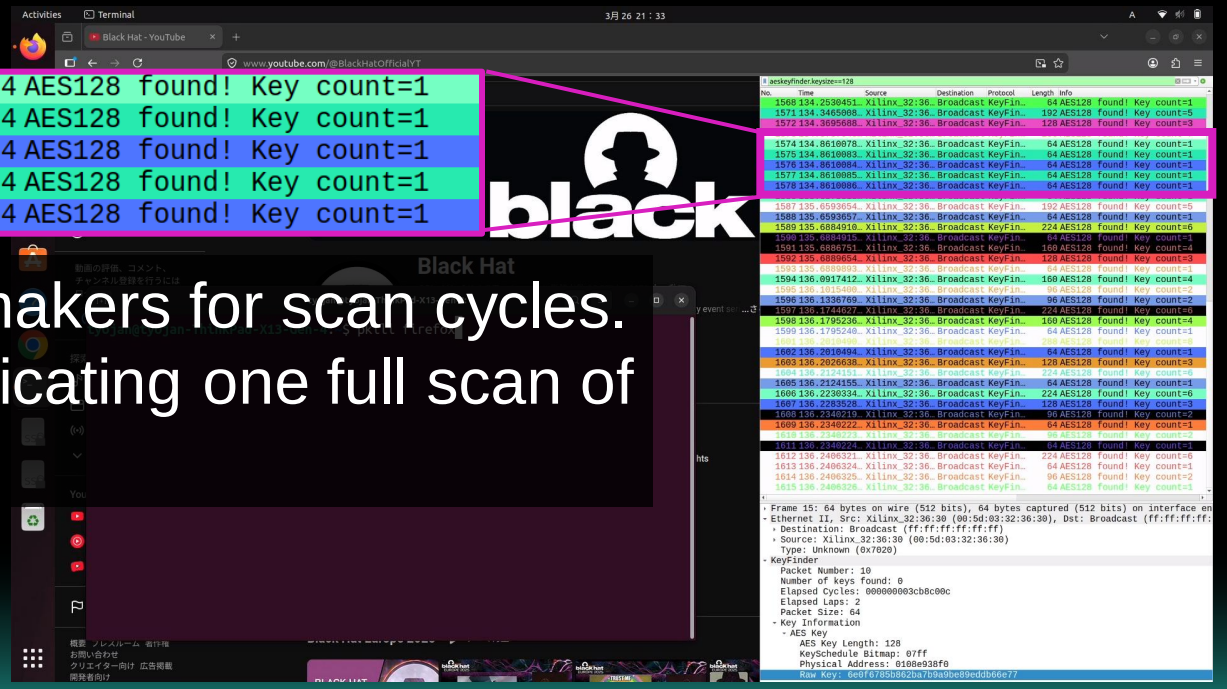
FPGA

Demonstration

We test whether AES keys remain in memory after three actions.

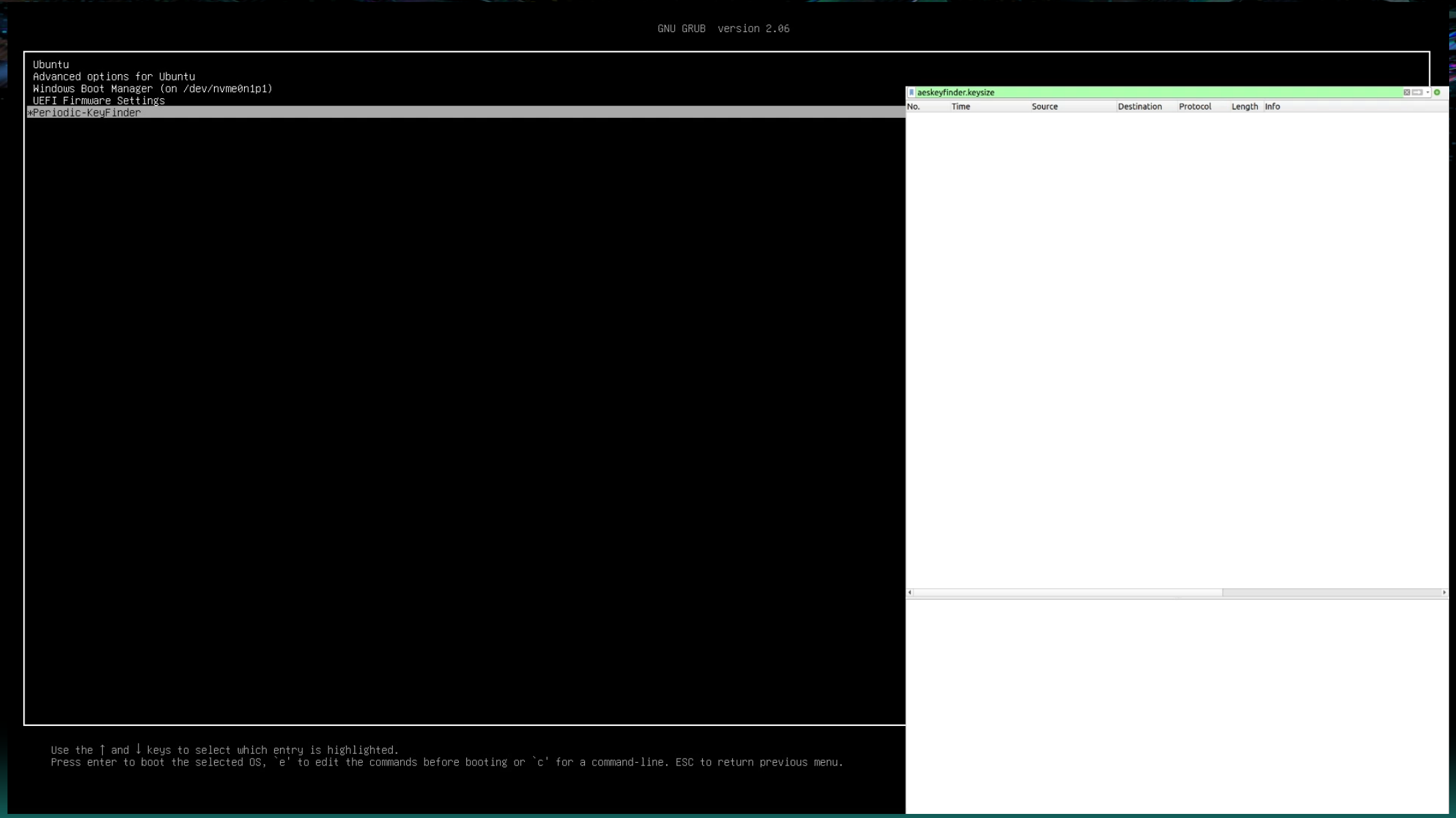
RQ1: Normal termination: Exit the web browser with close button
RQ2: Abnormal termination: Fore terminate the browser (pkill)
RQ4: Reboot experiment: Reboot the PC

```
11 3.842486579  Xilinx_32:36… Broadcast KeyFin…  64 AES128 found! Key count=1
12 3.842487168  Xilinx_32:36… Broadcast KeyFin…  64 AES128 found! Key count=1
13 3.842487255  Xilinx_32:36… Broadcast KeyFin…  64 AES128 found! Key count=1
14 3.842487335  Xilinx_32:36… Broadcast KeyFin…  64 AES128 found! Key count=1
15 3.842487416  Xilinx_32:36… Broadcast KeyFin…  64 AES128 found! Key count=1
```



Use the Ocean-colored keys as a makers for scan cycles. They appear every ~2 seconds, indicating one full scan of 8 GB memory.

Demonstration



Demonstration of zeroization failures

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

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)	✗	✗	✗	-	✓	-

Demonstration of zeroization failures

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

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)	✗	✗	✗	-	✓	-

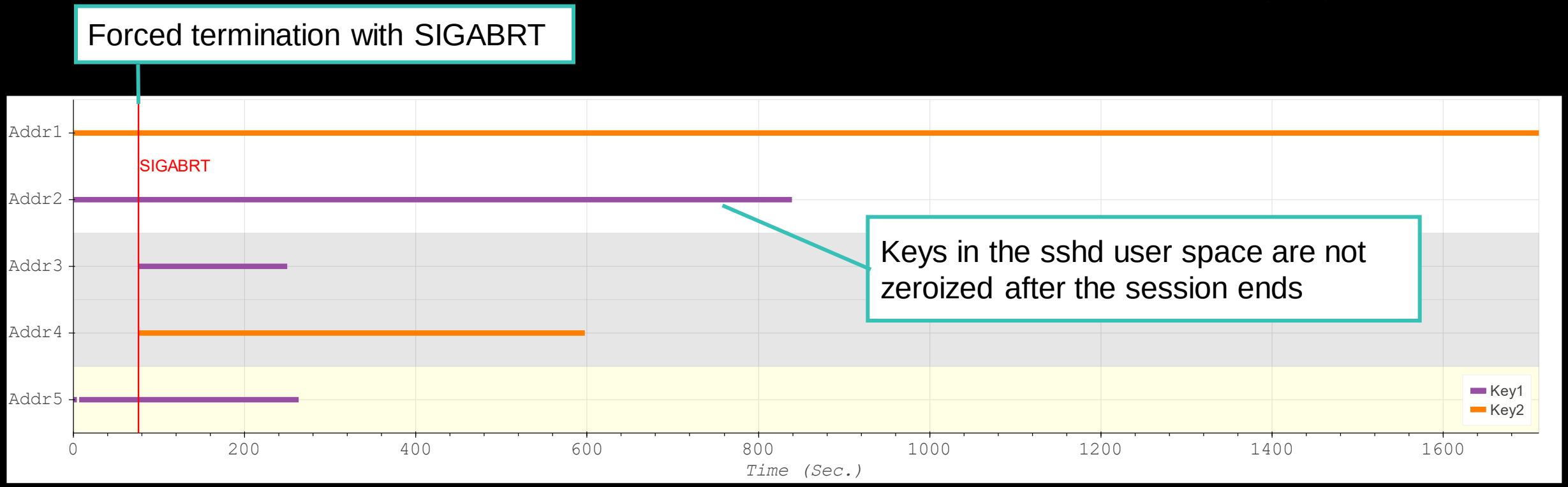
Demonstration of zeroization failures

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

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)	✗	✗	✗	-	✓	-

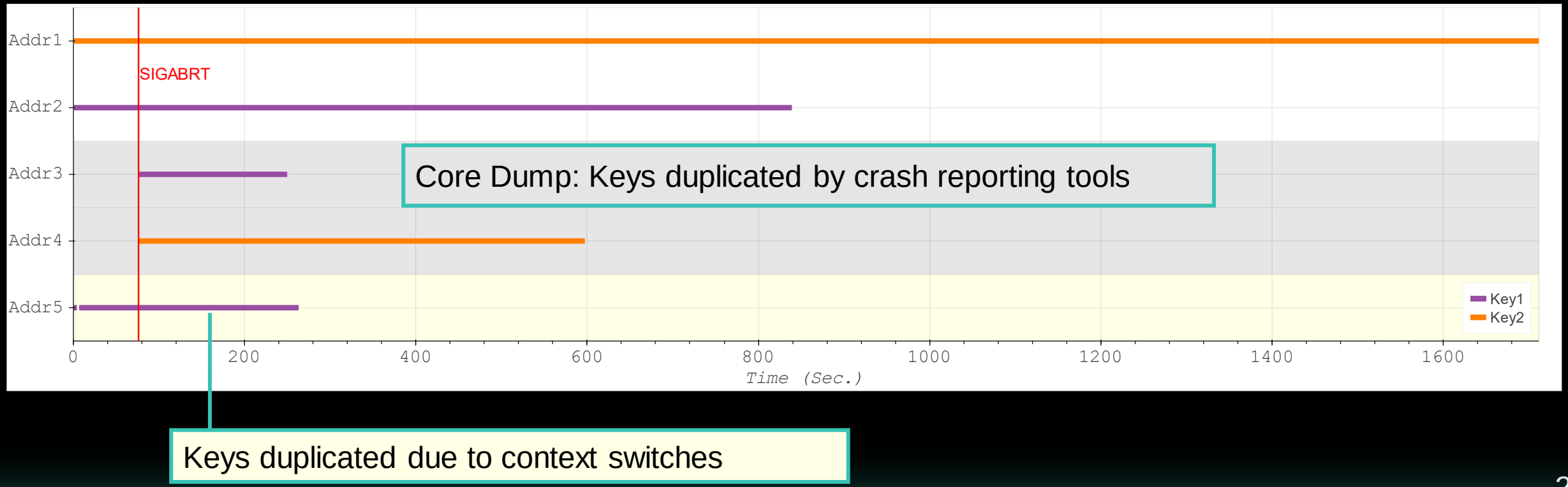
Zeroization upon abnormal app. termination (RQ2)

- Key appearance timeline when sending SIGABRT to sshd during an SSH session



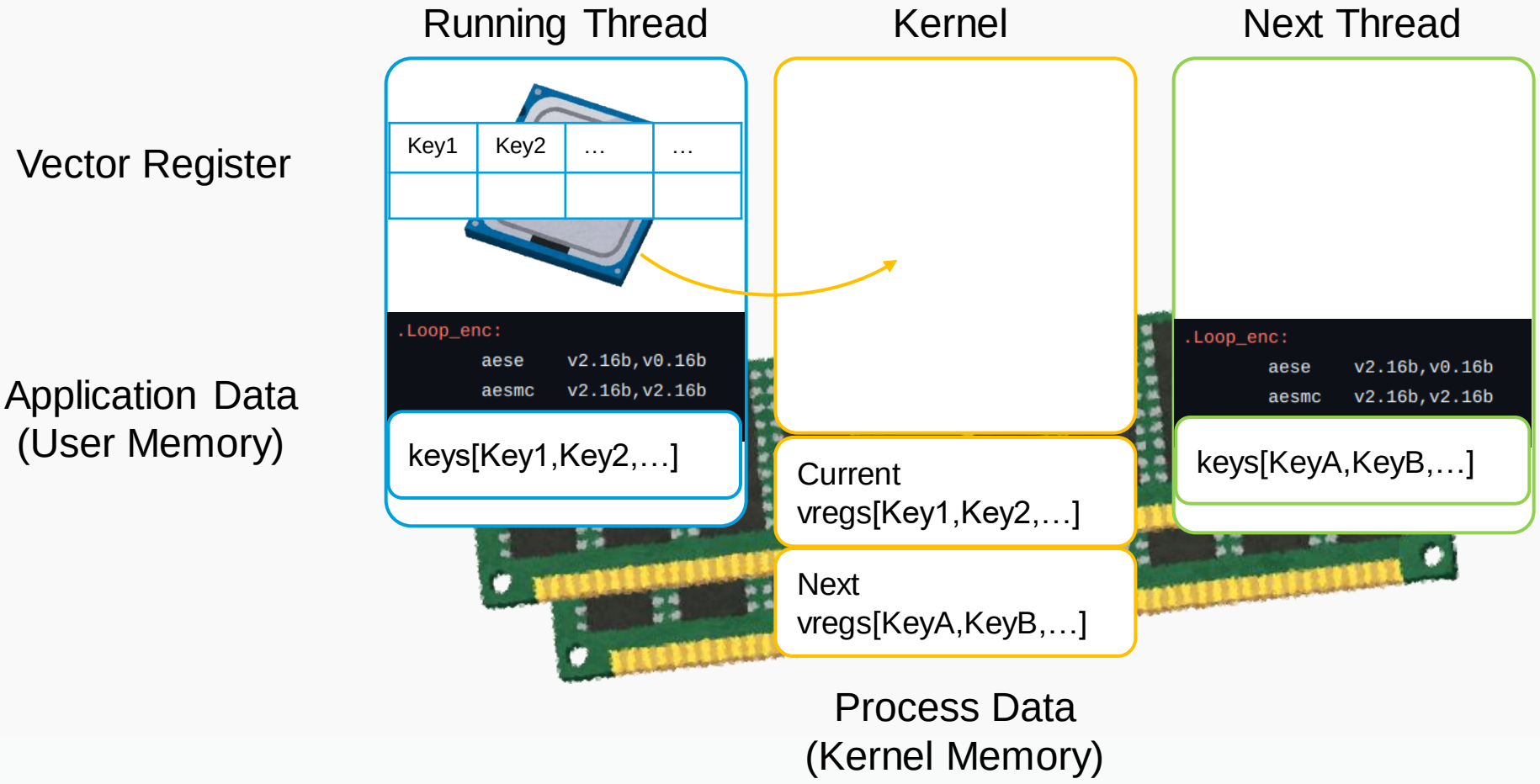
Duplication of detected cryptographic keys (RQ3)

- Core Dump
- Context Switch



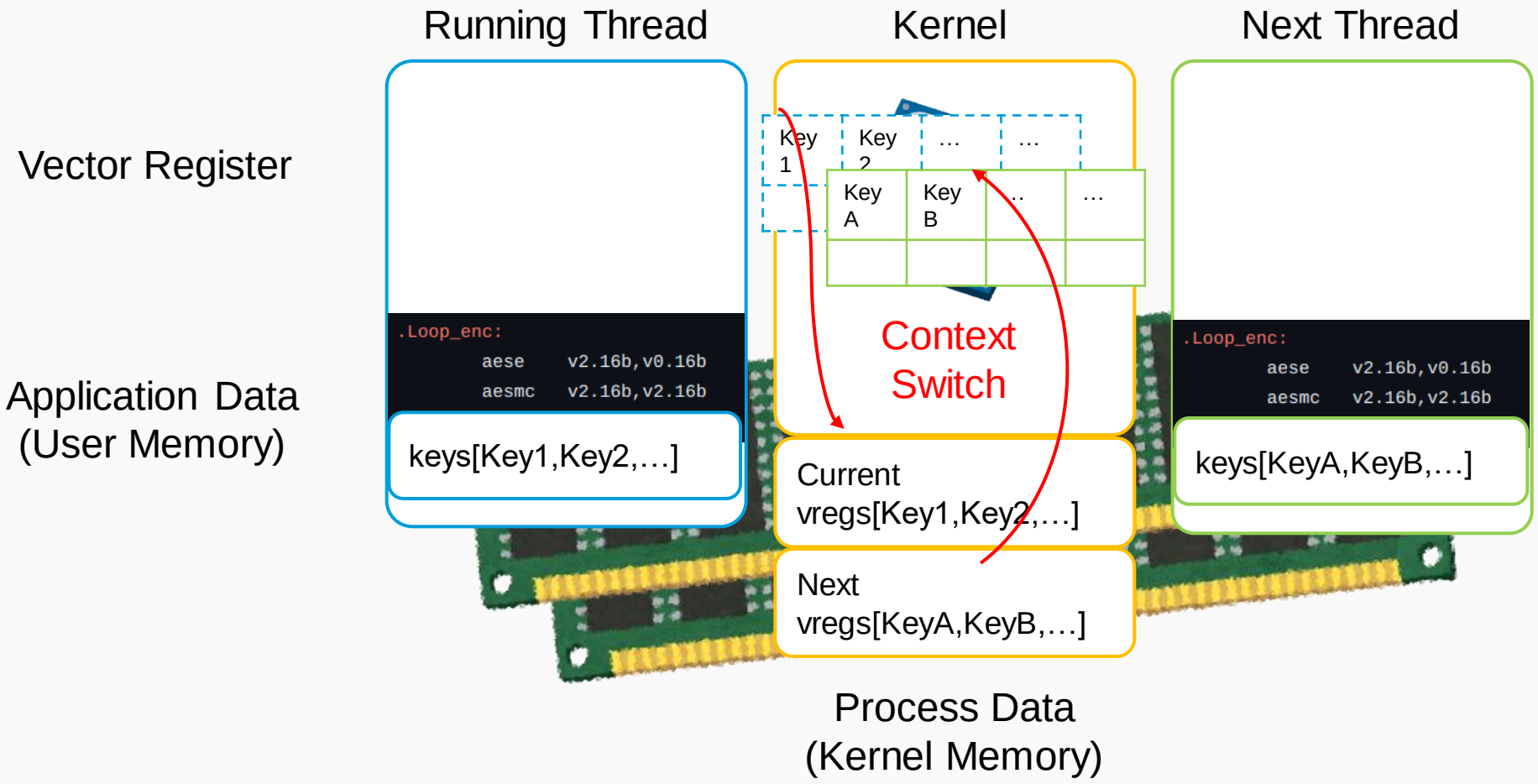
Key duplication caused by vector-register context switching (1/3)

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



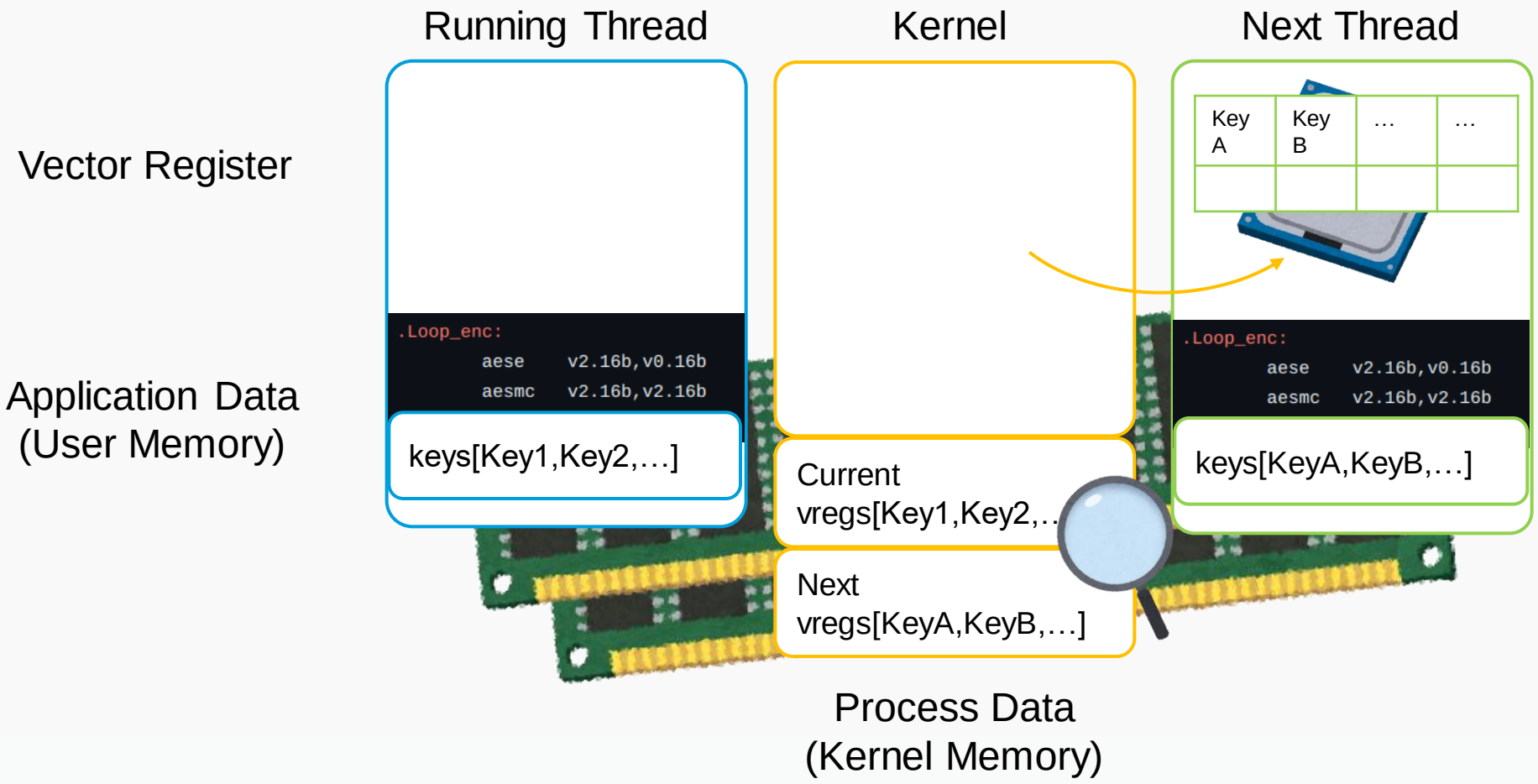
Key duplication caused by vector-register context switching (2/3)

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)	✗	✓	✓

Who is responsible for zeroization?

Key lifetime is influenced by interactions across multiple system layers.

- Missing signal handlers
- Core dumps
- Lazy page zeroization
- Duplication via register
- No zeroization on reboot

Recommendations and mitigations

App. 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.

Recommendations and mitigations

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.

Takeaways

- Zeroization must be verified by observing physical memory.
- Keys persist, duplicate, and survive reboot in real systems.
- Reliable zeroization requires a system-wide approach.

Conclusion

- Zeroization spans compliance, applications, OS, hardware, and cryptography.
- FPGA-based live forensics revealed duplication and persistence across layers and reboots.
- We responsibly disclosed these issues to real-world systems.
- Code & tools available on GitHub

GitHub - Tyojan/Periodic-AESKeyFinder

Thank you for listening!