black hat ASIA 2024

APRIL 18-19, 2024
BRIEFINGS

Faults In Our Bus: Novel Bus Fault Attacks to Break ARM TrustZone

Nimish Mishra, Anirban Chakraborty, Debdeep Mukhopadhyay
Indian Institute of Technology Kharagpur, India

disarmament

Who are we?

Nimish Mishra

Anirban Chakraborty

Debdeep Mukhopadhyay

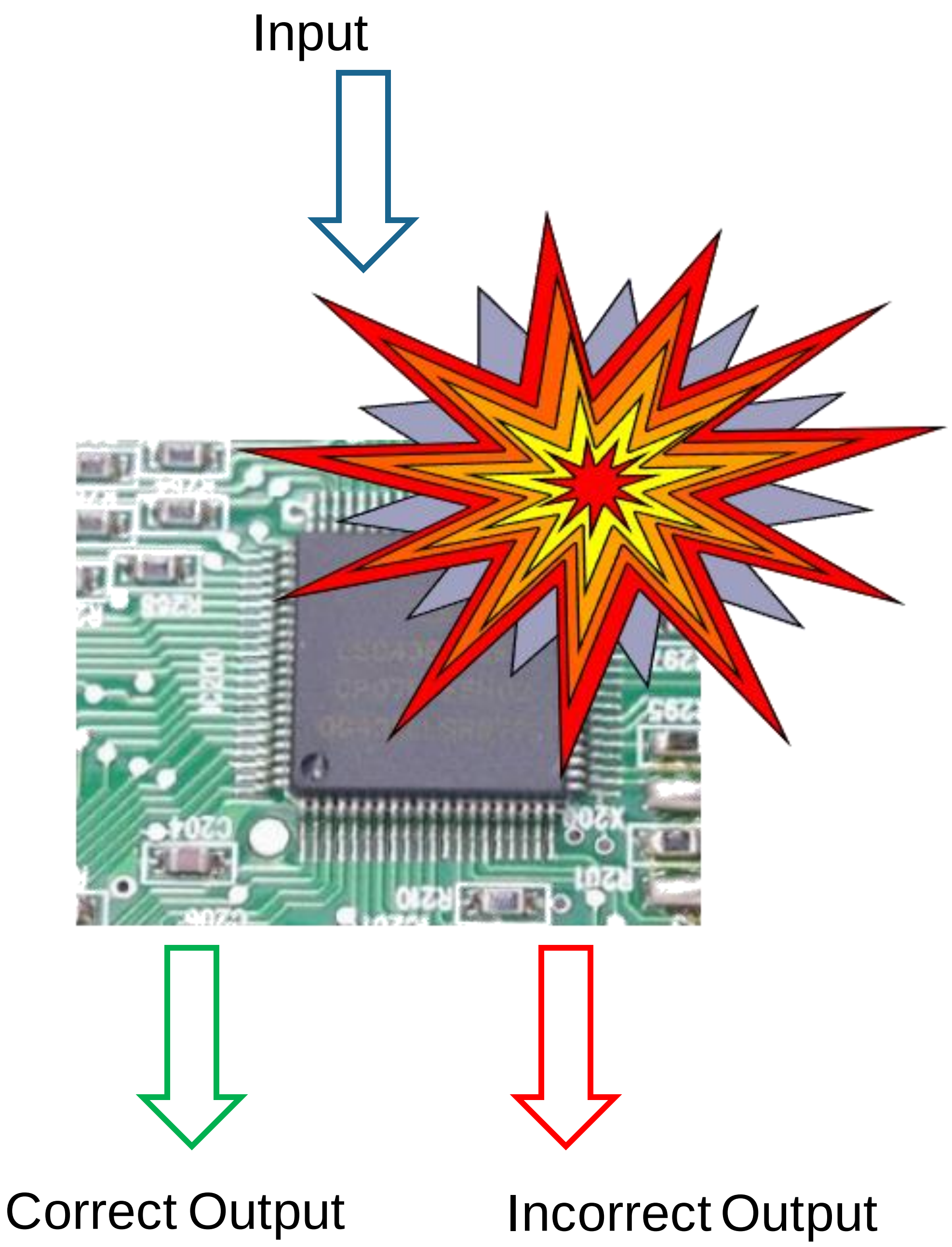
Indian Institute of Technology Kharagpur India

Outline

1. What are Faults?
2. Traditional Fault Points on Embedded Systems and SoCs
3. A (new) Fault Point on SoCs
4. OP-TEE?
5. End-to-end Attack
   - Load (adversarial) Trusted Application through Faults
   - Redirect communication for other Trusted Applications
   - Decrypt (redirected) communication
6. Impact

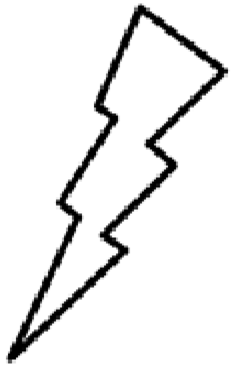
What are Faults?

- Actively perturb data or control-flow of a system and gain information about the secret through faulty system response

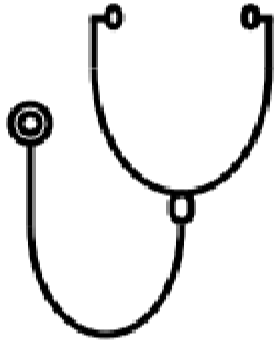


- Fault causes error and error can be exploited to leak secret information

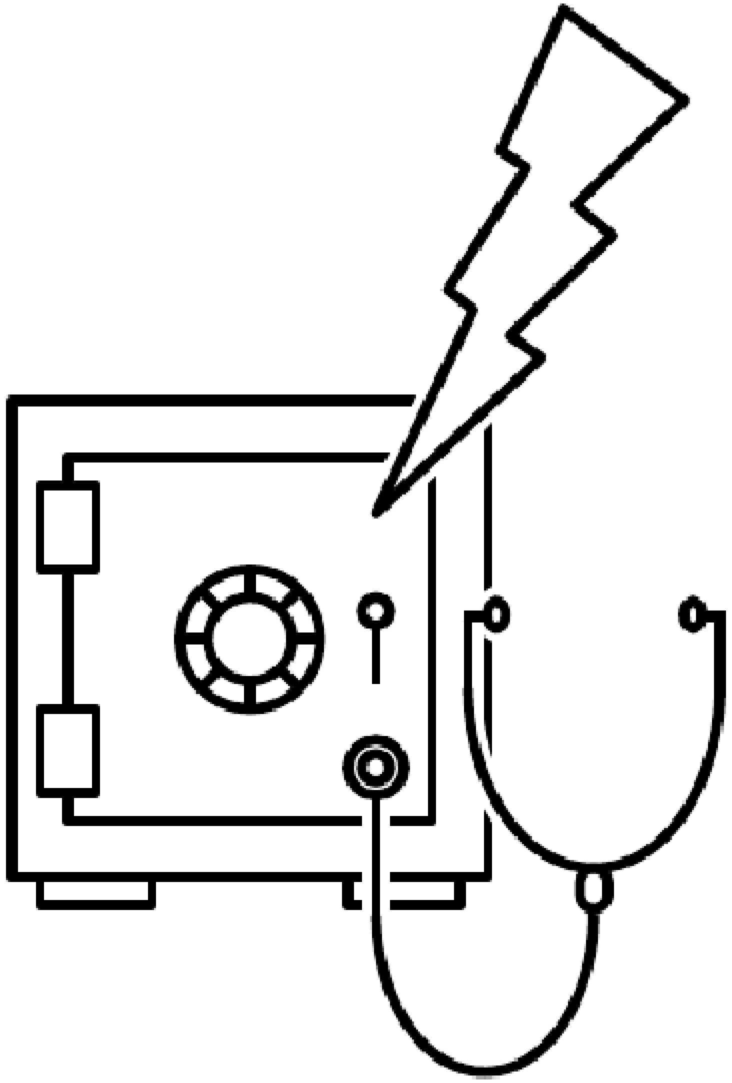
- Fault attack sometimes combined with side channel can lead to stronger attacks



Fault Injection

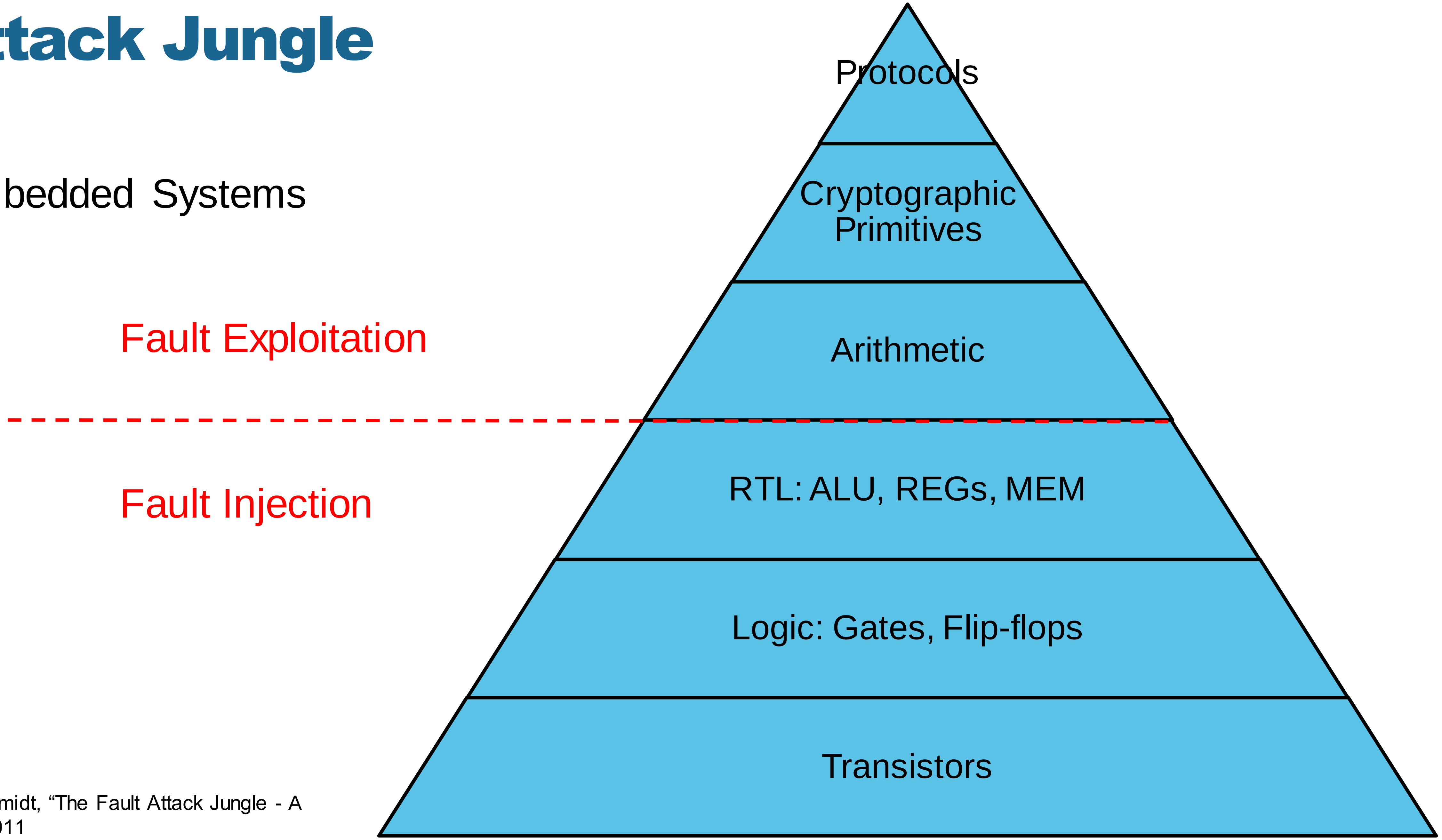


Side Channel Observation



The Fault Attack Jungle

Fault Attack on Embedded Systems



I. Verbauwhede, D. Karaklajid, and J.-M. Schmidt, "The Fault Attack Jungle - A Classification Model to Guide You", FDTC, 2011

Fault Attack Vectors

Fig: Electromagnetic Fault Injection (EMFI) Probe

- **WHAT:** Strategically modify execution environment of a system
- **HOW:** Through changes in external operational conditions

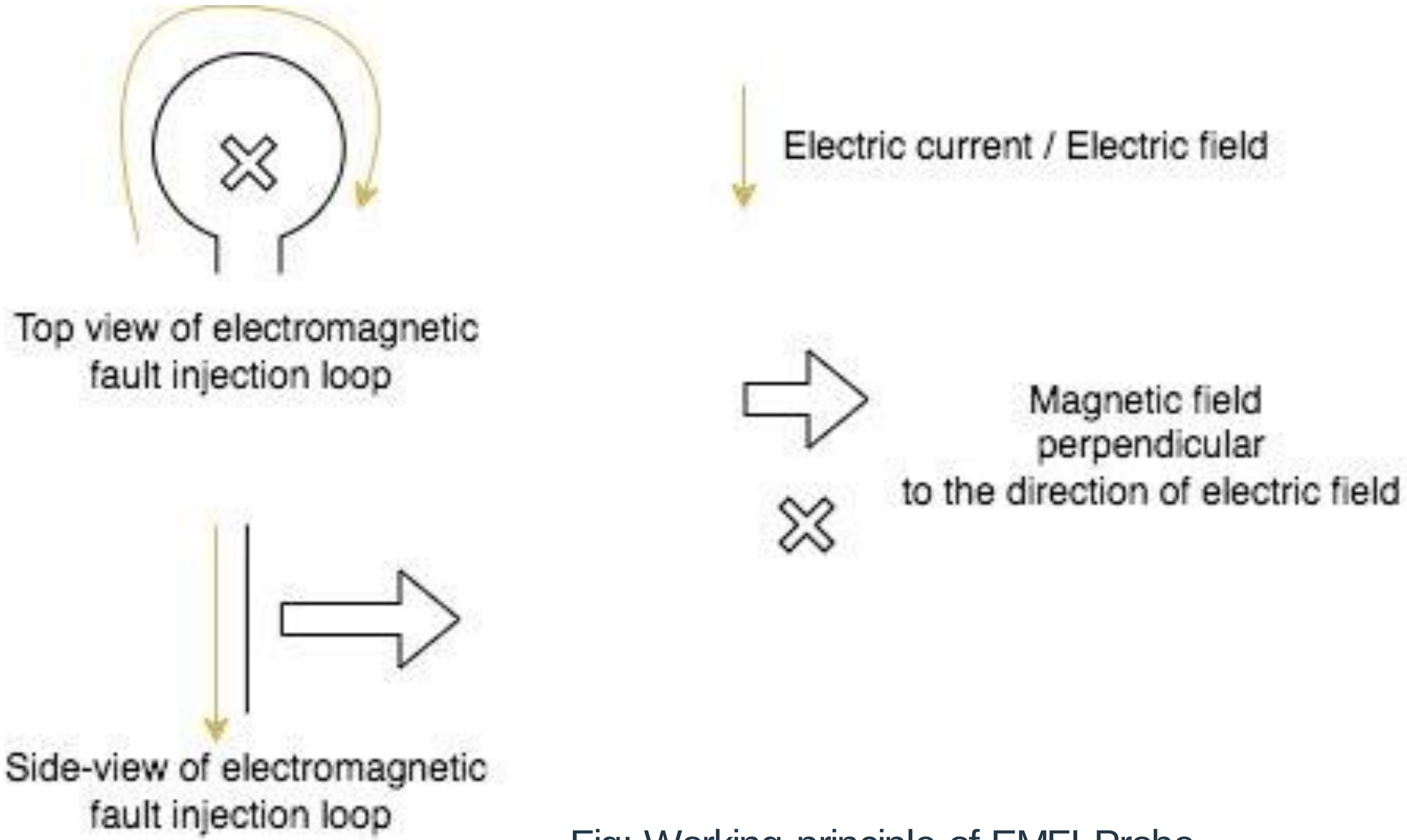


Fig: Working principle of EMFI Probe

FI Attack Vectors

- **WHAT:** Strategically modify execution environment of a system
- **HOW:** Through changes in external operational conditions
- **WHY:** Bias software execution to adversarial advantage

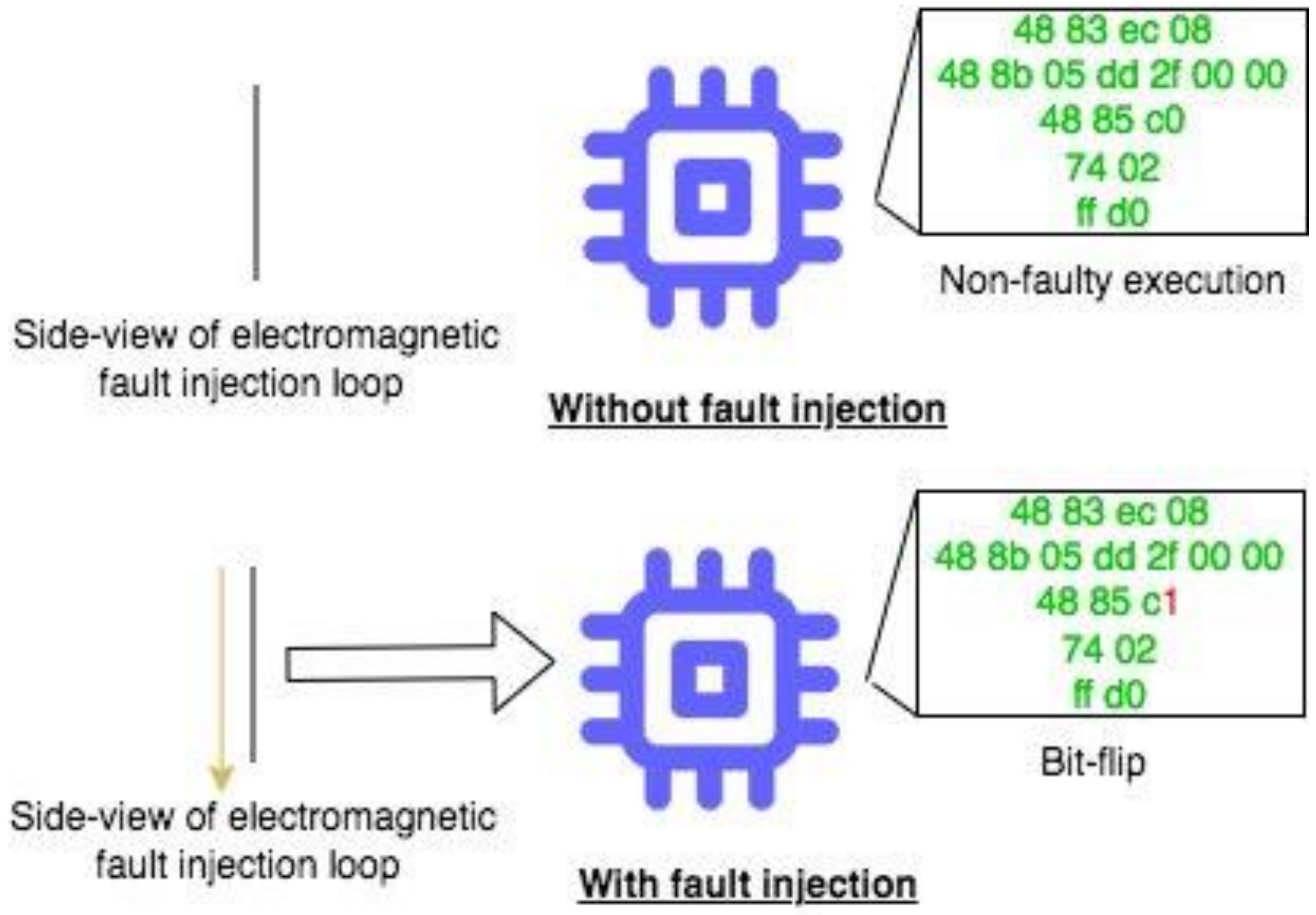


Fig: Representative Fault Attack to introduce a bit-flip

Fault Models

Granularity

1. Single bit
2. Multiple bits
3. Byte or Word

Fault Models

Granularity

1. Single bit
2. Multiple bits
3. Byte or Word

Fault-type

1. Stuck-at (zero or one)
2. Bit flip
3. Random

Fault Models

Granularity

1. Single bit
2. Multiple bits
3. Byte or Word

Fault-type

1. Stuck-at (zero or one)
2. Bit flip
3. Random

Attacker Control

1. Precise
2. Loose
3. None

Fault Models

Granularity

1. Single bit
2. Multiple bits
3. Byte or Word

Fault-type

1. Stuck-at (zero or one)
2. Bit flip
3. Random

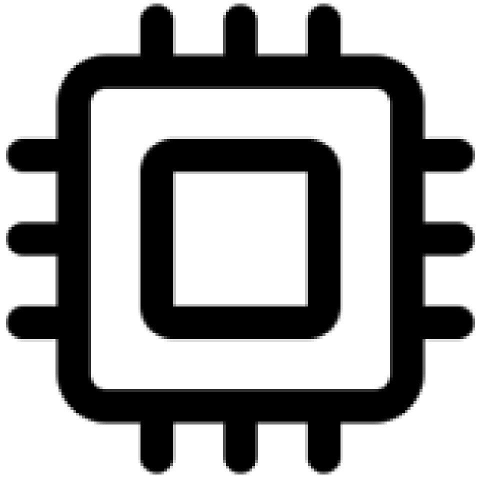
Attacker Control

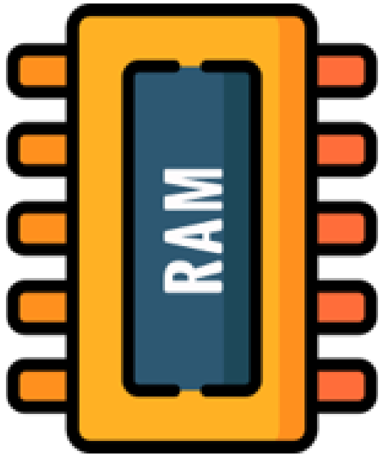
1. Precise
2. Loose
3. None

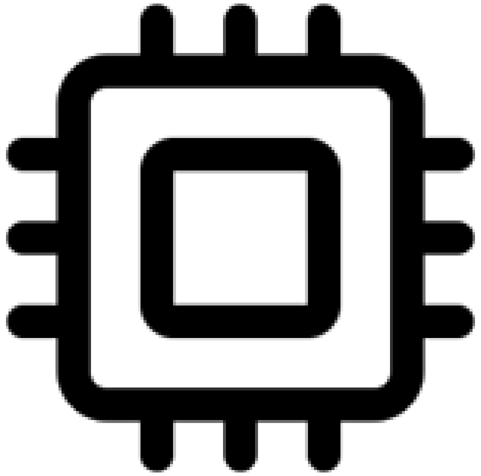
Duration of the fault

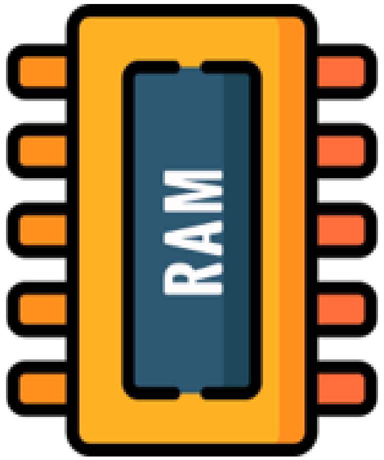
1. Transient
2. Permanent
3. Persistent

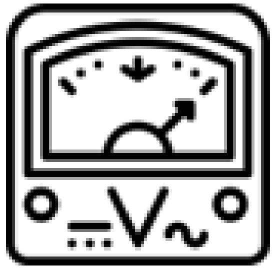
Traditional Fault Points




RAM

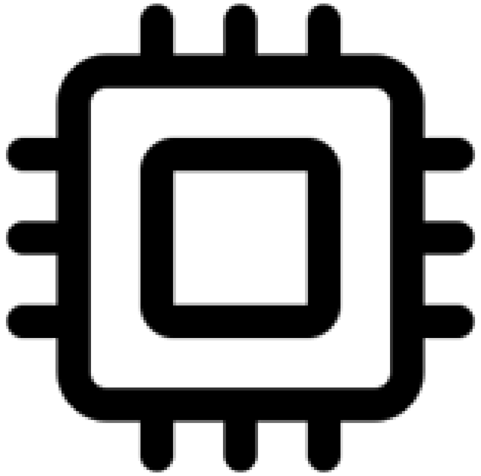


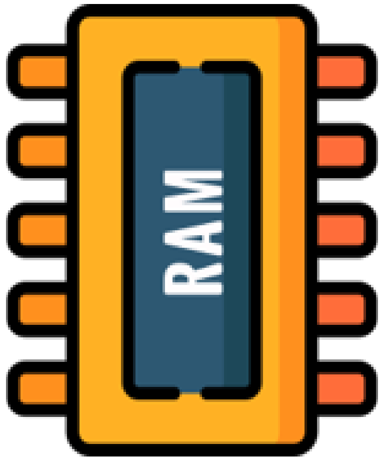


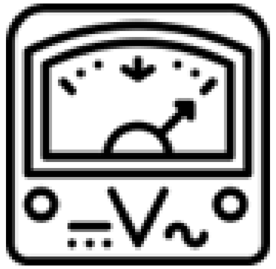


External interface (voltage/clock glitch)

Dynamic Frequency and Voltage Scaling (DVFS)

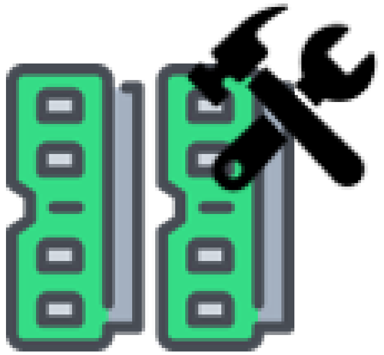




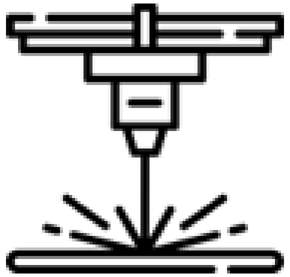


External interface (voltage/clock glitch)

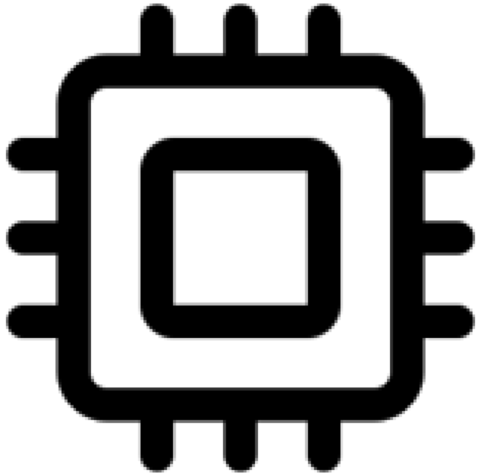
Dynamic Frequency and Voltage Scaling (DVFS)

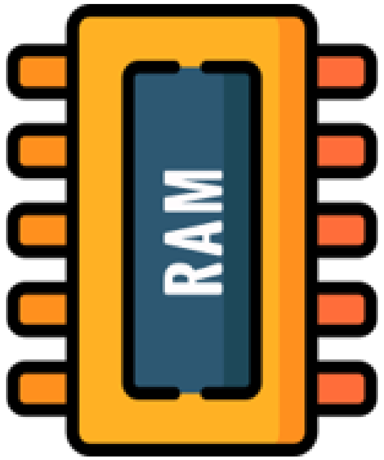


Rowhammer



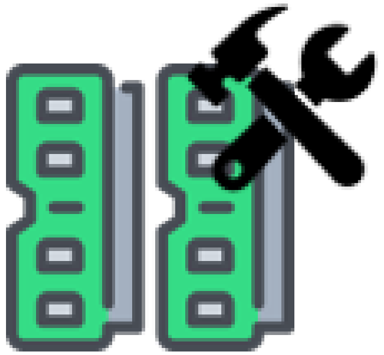
Laser/EM Fault injection



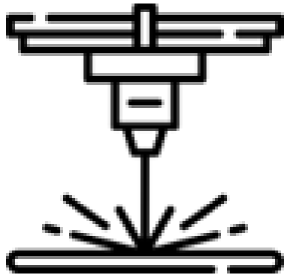


No external interface (in SoCs; ex RPi)

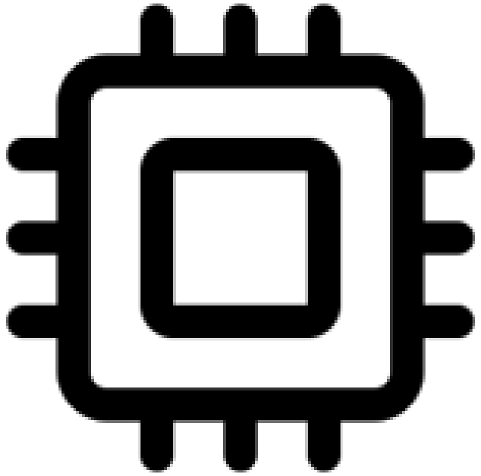
Privileged



Rowhammer

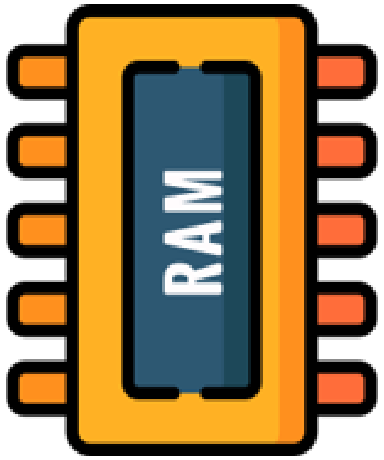


Laser/EM Fault injection



No external interface
(in SoCs; ex RPi)

Privileged



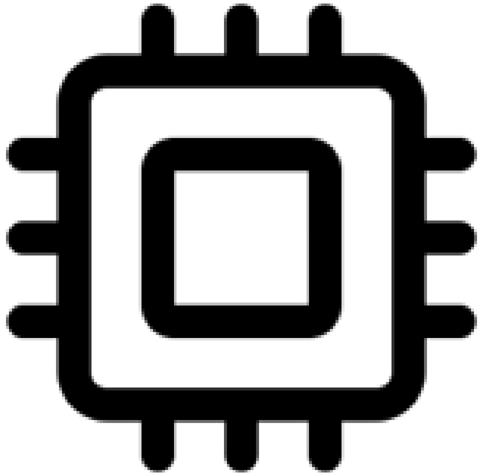
ECC checks

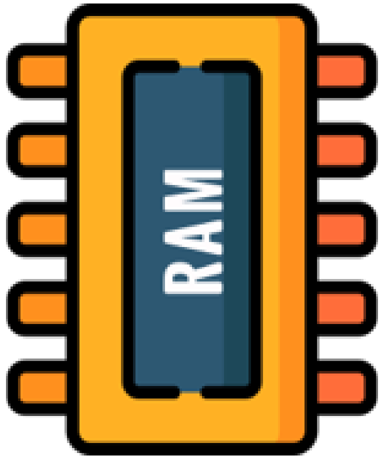
Casings
(requires invasive
depackaging)

Are there other **architectural aspects** which can be **used for faults**,

for which **no known defences** are deployed yet?

A (new) Fault Point on SoCs



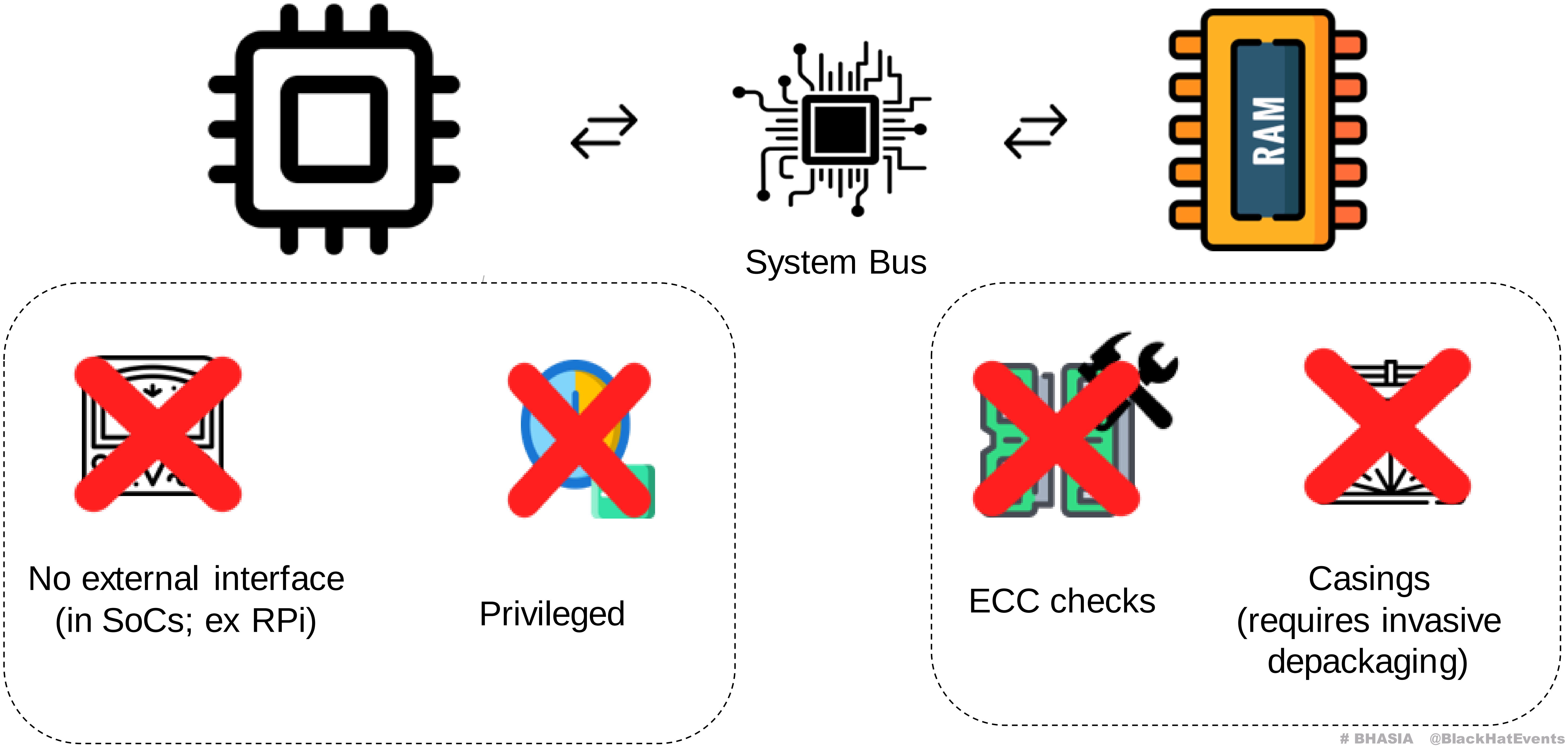


No external interface
(in SoCs; ex RPi)

Privileged

ECC checks

Casings
(requires invasive
depackaging)


System Bus
RAM
No external interface
(in SoCs; ex RPi)
Privileged
ECC checks
Casings
(requires invasive
depackaging)

- Uncased and exposed

- Involved mainly with **load/store** instructions

- Prior works
  - Simulation of bus faults
  - External voltage glitches on PlayStation consoles to **skip** memory cycles

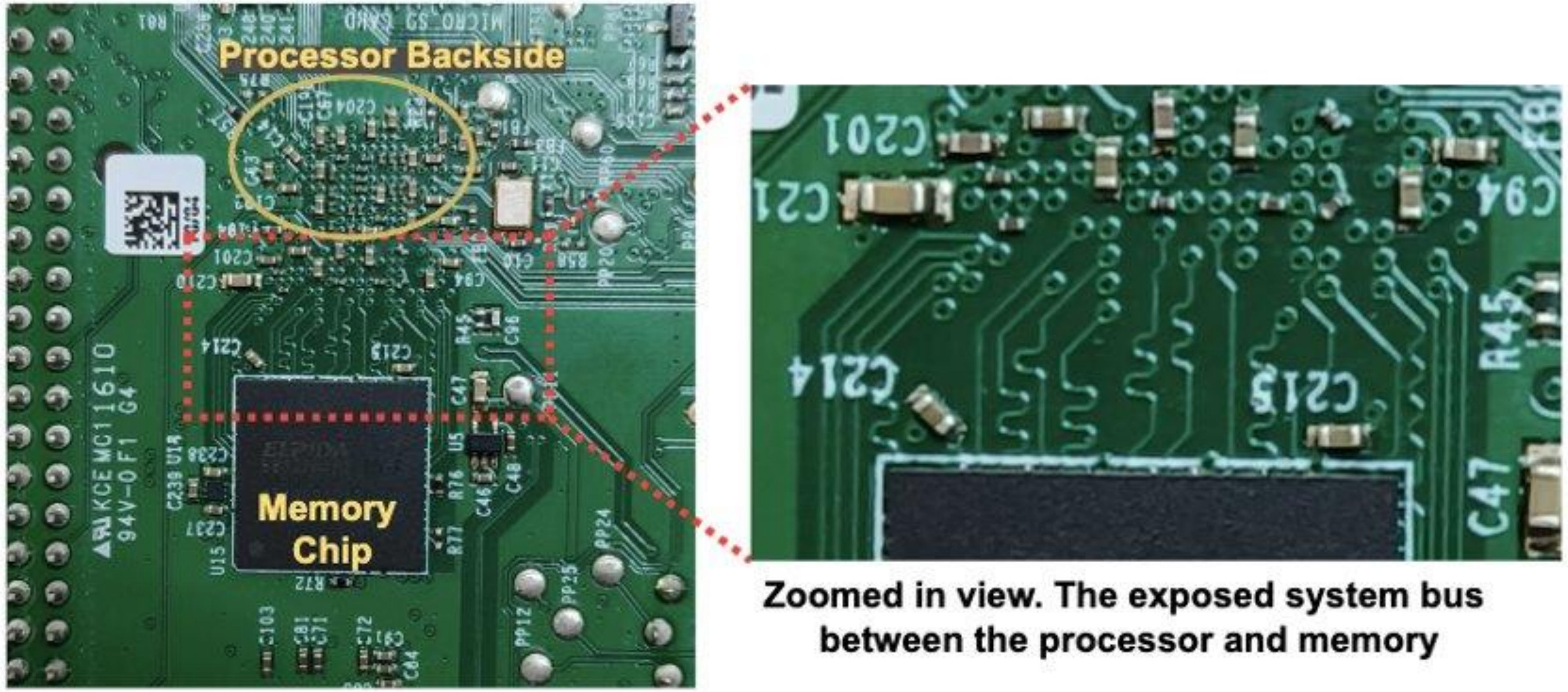


Fig: Exposed bus connections in RPi3

FI on System Bus: Attack Principle

load dest_reg, [mem_addr]

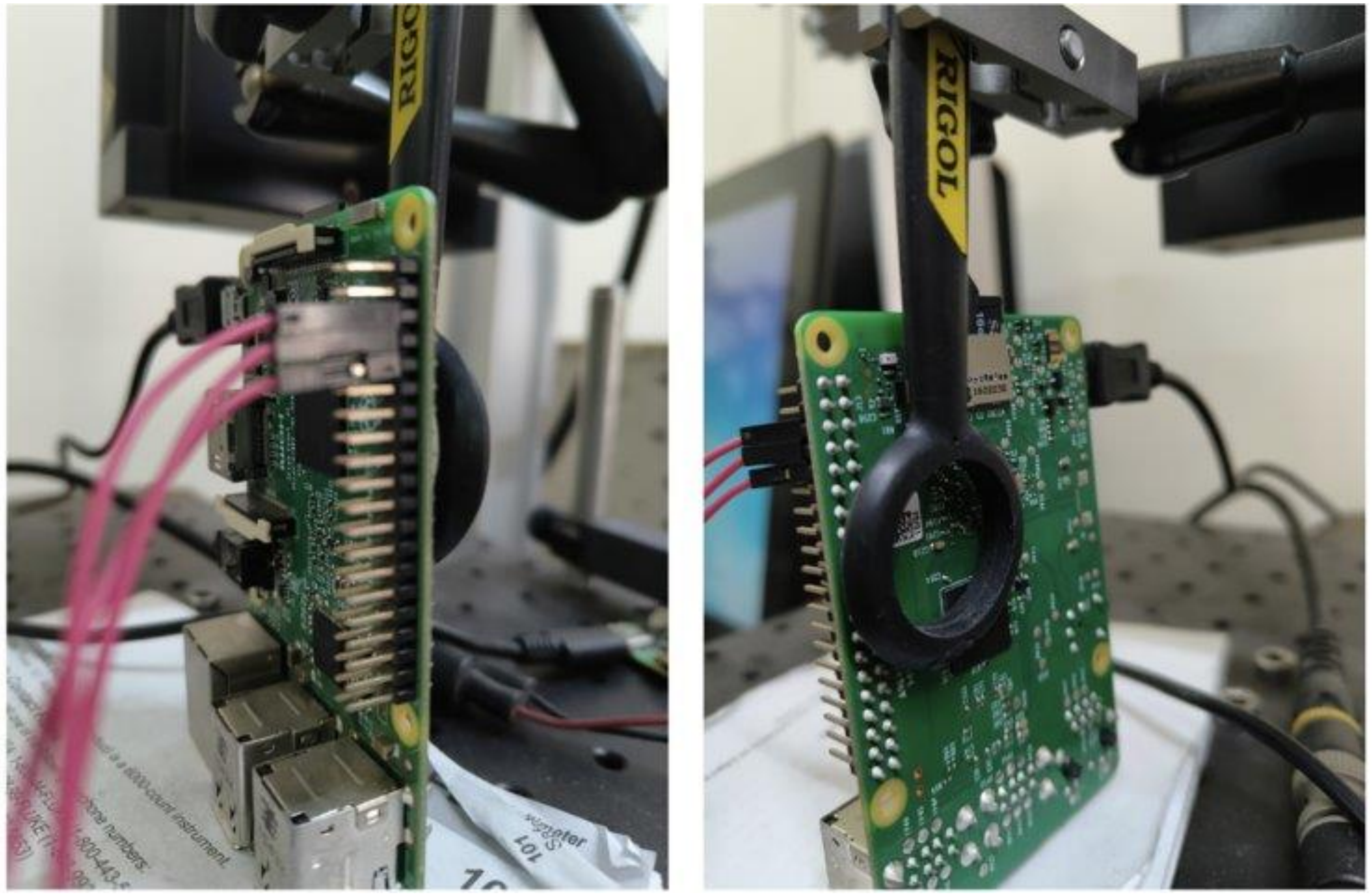
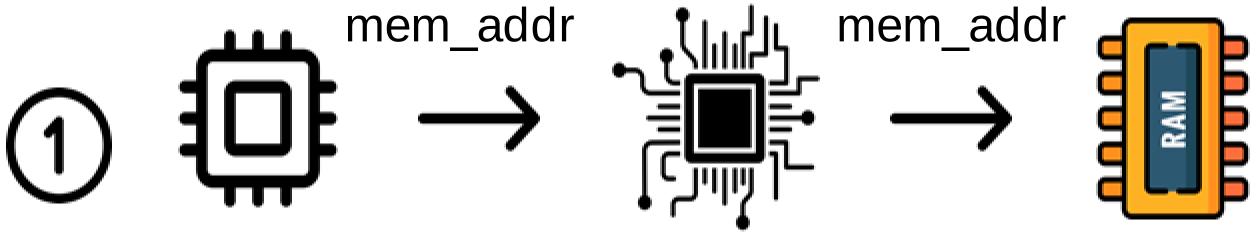


Fig: Electromagnetic Fault Injection probe positioned over the exposed system bus on a RPi3

FI on System Bus: Attack Principle



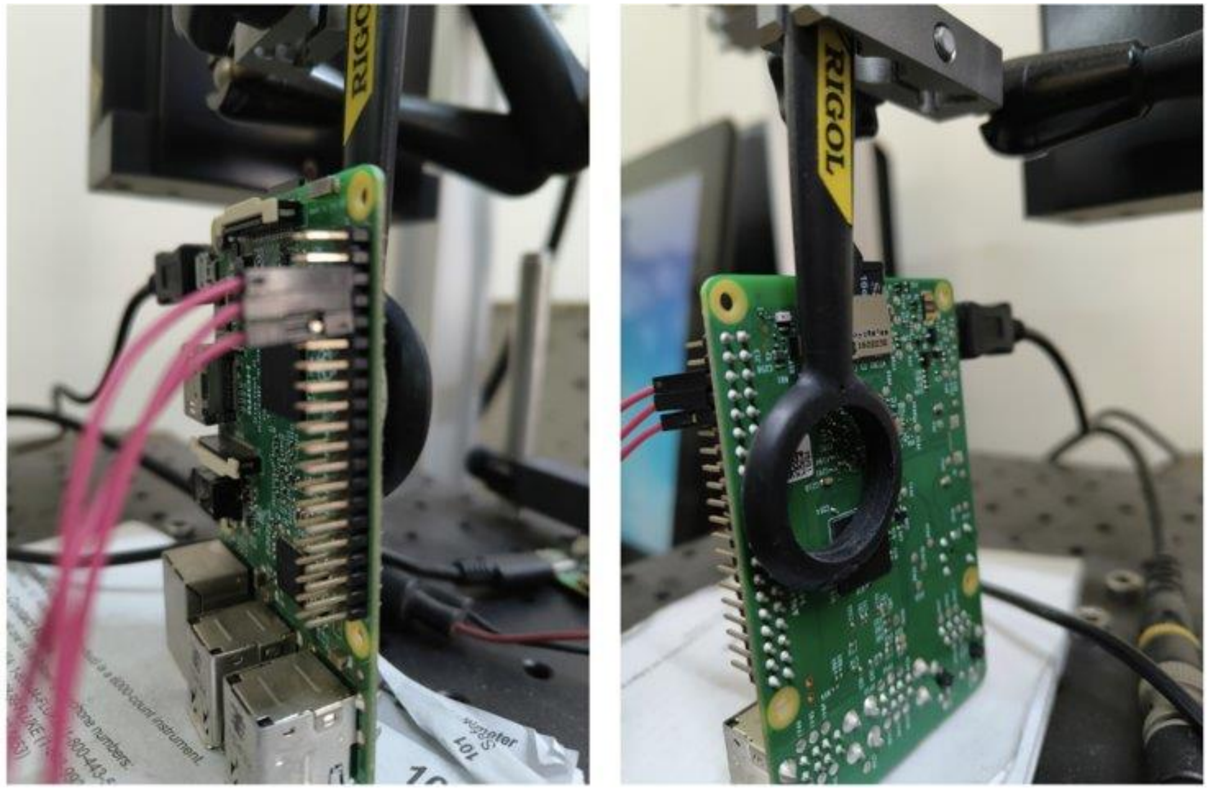
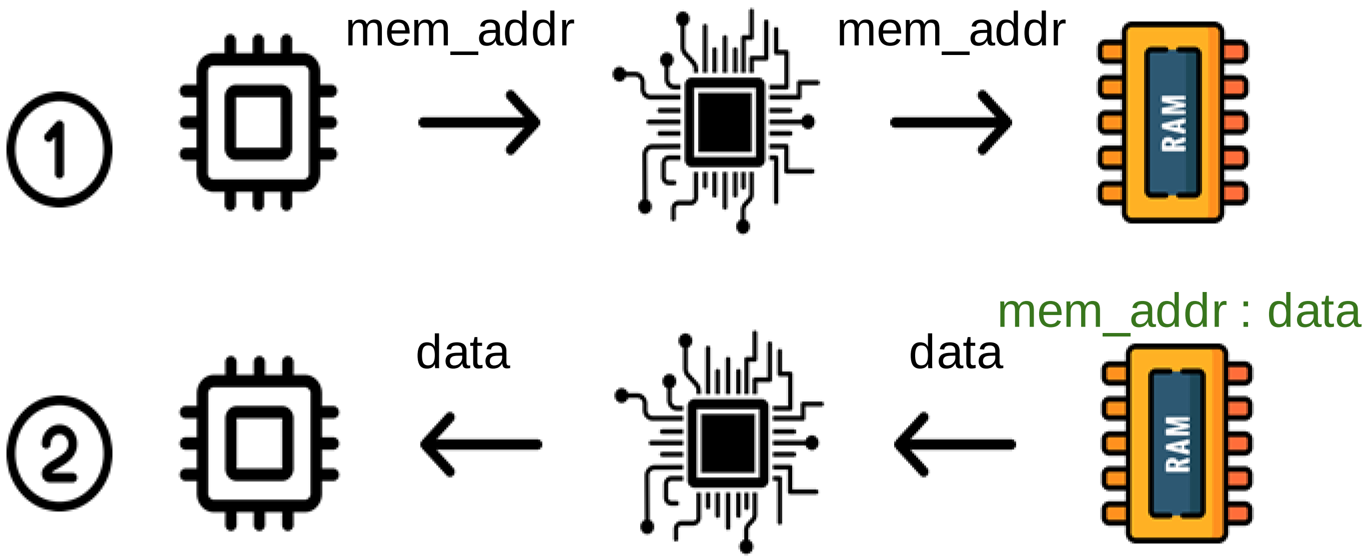


Fig: Electromagnetic Fault Injection probe positioned over the exposed system bus on a RPi3

load dest_reg, [mem_addr]

FI on System Bus: Attack Principle



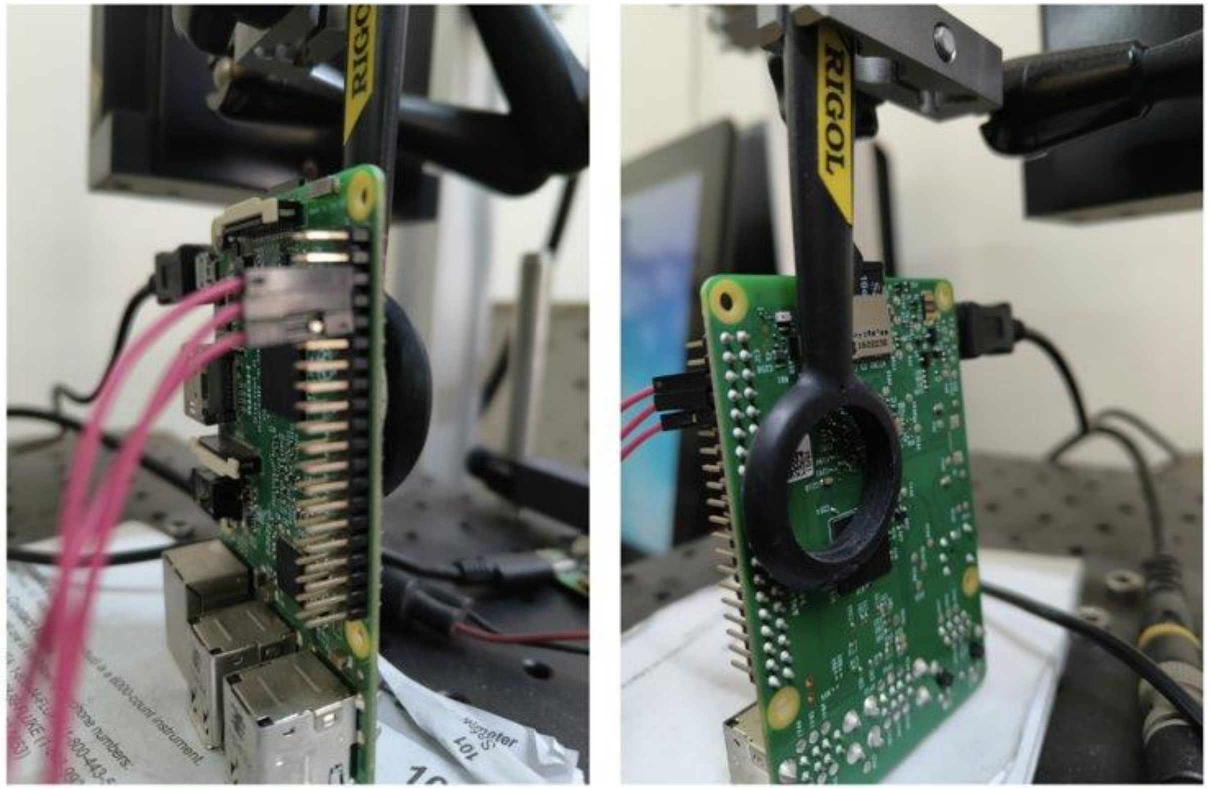
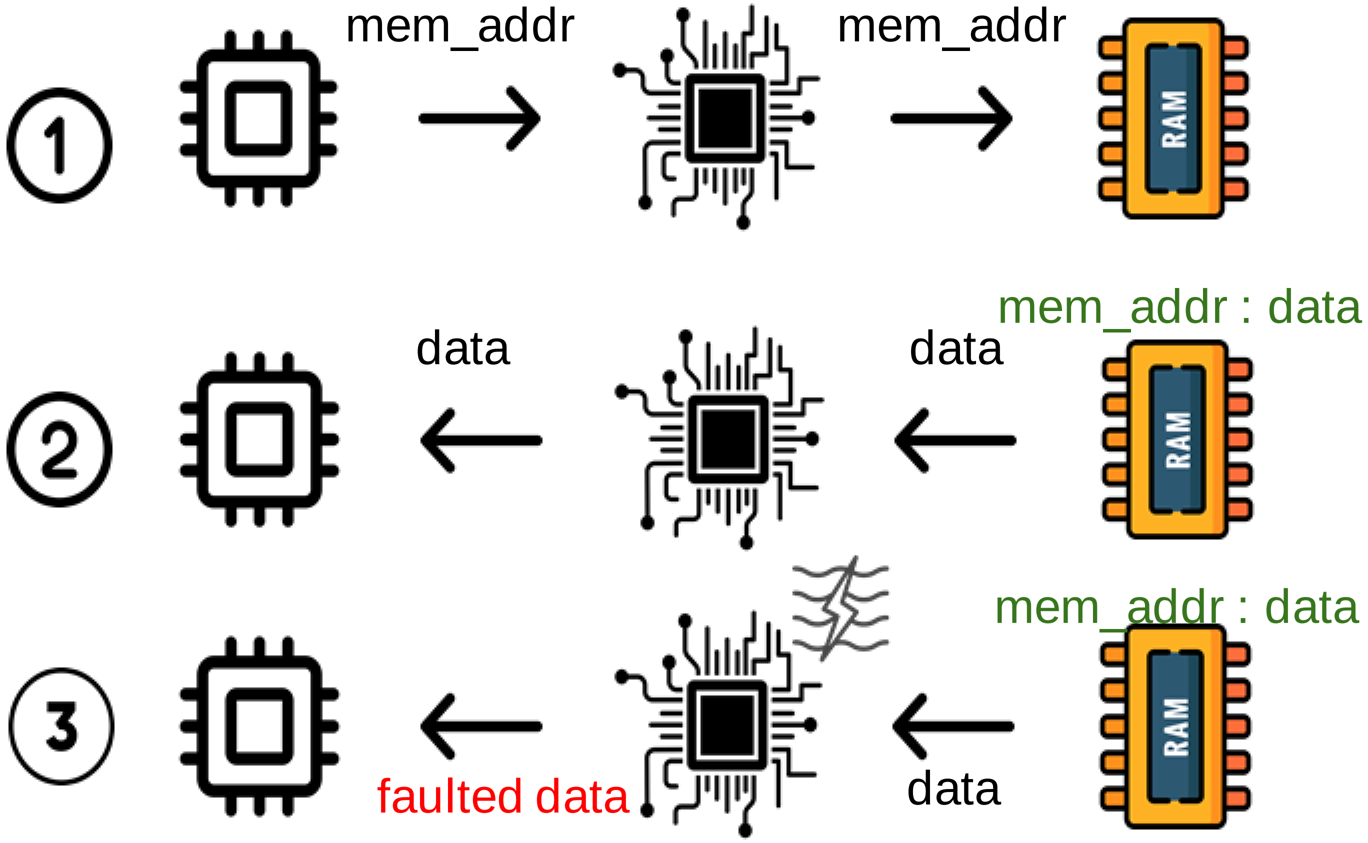


Fig: Electromagnetic Fault Injection probe positioned over the exposed system bus on a RPi3

load dest_reg, [mem_addr]

FI on System Bus: Attack Principle



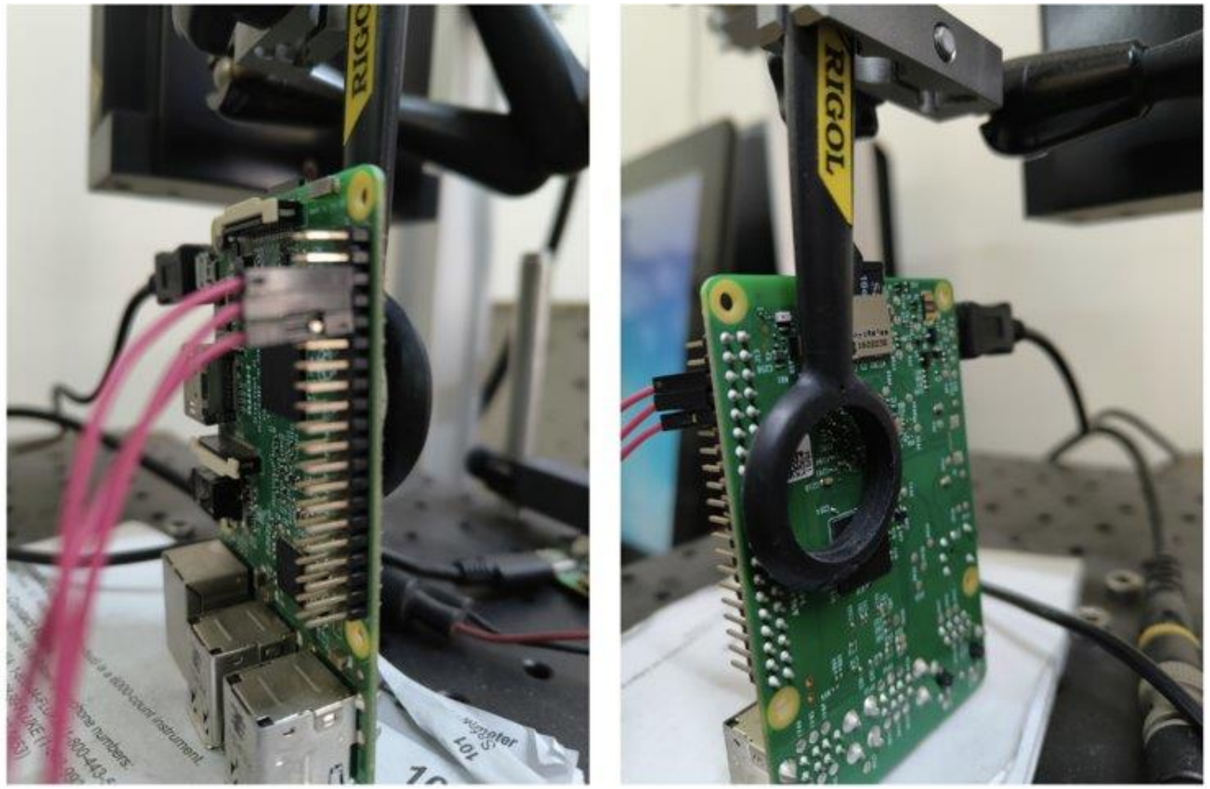


Fig: Electromagnetic Fault Injection probe positioned over the exposed system bus on a RPi3

load dest_reg, [mem_addr]

FI on System Bus: Success Rates

```
load   dest_reg, [mem_addr]
```

FI on System Bus: Success Rates

```
load dest_reg, [mem_addr]
```

Data Bus Faults

- Result in **incorrect data**
- Success rate breakdown
  - **No fault**: 38%
  - **Fault to 0x0:** 35%
  - **Other cases**: 27%

FI on System Bus: Success Rates

load dest_reg, [mem_addr]

Data Bus Faults

- Result in **incorrect data**
- Success rate breakdown
  - **No fault**: 38%
  - **Fault to 0x0:** 35%
  - **Other cases**: 27%

Address Bus Faults

- Result in **SEGFAULT**
- Success rate breakdown
  - **SEGFAULT**: 31%
  - **Other cases**: 69%

FI on System Bus: Success Rates

load dest_reg, [mem_addr]

Data Bus Faults

- Result in **incorrect data**
- Success rate breakdown
  - **No fault**: 38%
  - **Fault to 0x0:** 35%
  - **Other cases**: 27%

Register sweeping
(cleans the value of a **load**)

Address Bus Faults

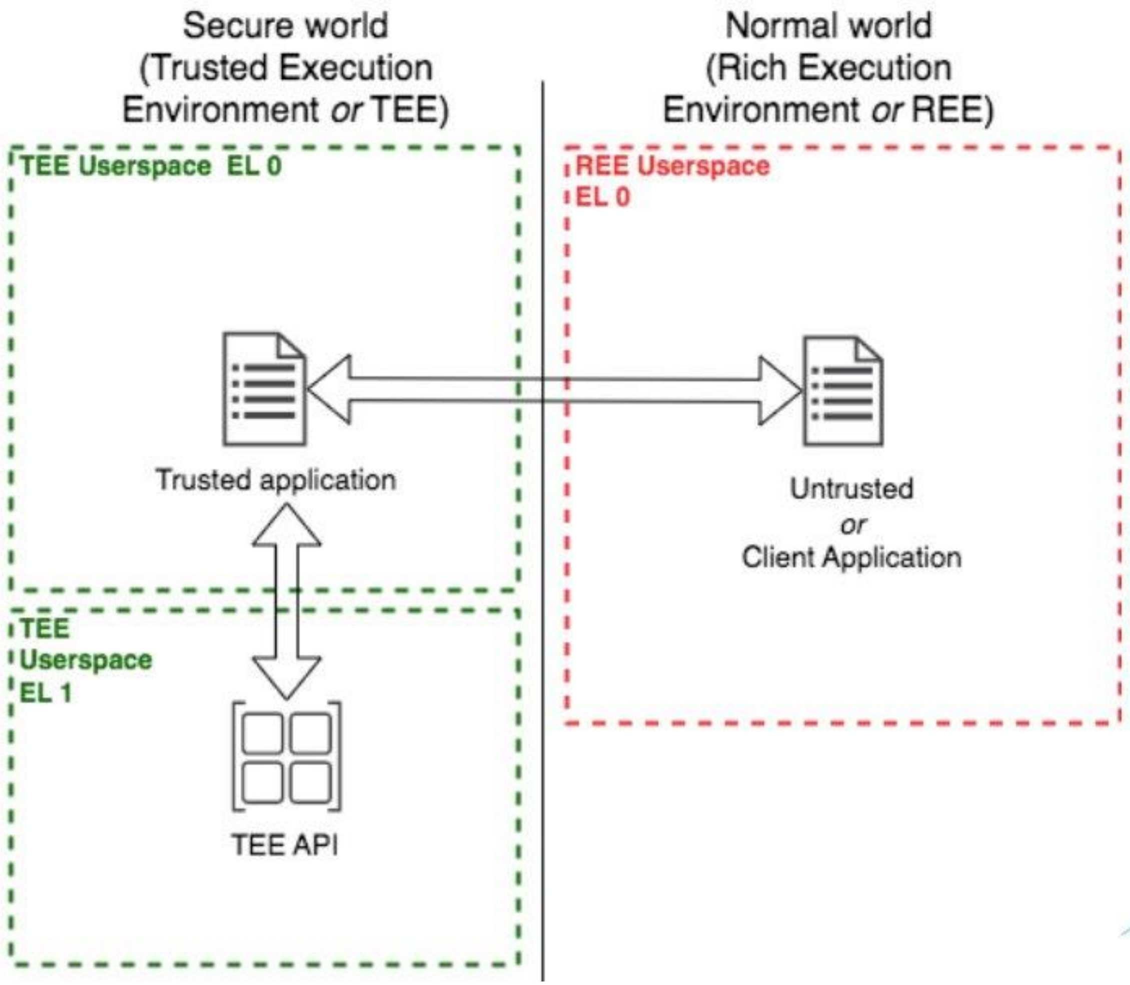
- Result in **SEGFAULT**
- Success rate breakdown
  - **SEGFAULT**: 31%
  - **Other cases**: 69%

Implication: Register sweeping to mount an end-to-end attack on Open Portable Trusted Execution Environment (OP-TEE)

OP-TEE?

"Trusted" Execution Environment

- **WHAT:** An attempt to **disentangle** critical applications from generic software (including kernel)
- **HOW: (**Hardware backed) isolation of system resources
- **OP-TEE:** Implementation of **GlobalPlatformAPI** specification for ARM TZ
  - Maintained by the **Trusted Firmware,** with members like Google, ARM, Linaro, NXP, STMicroelectronics
  - Deployed in commercial platforms like Apertis, iWave, and so on



"Trusted" Execution Environment

- **Two main divisions**

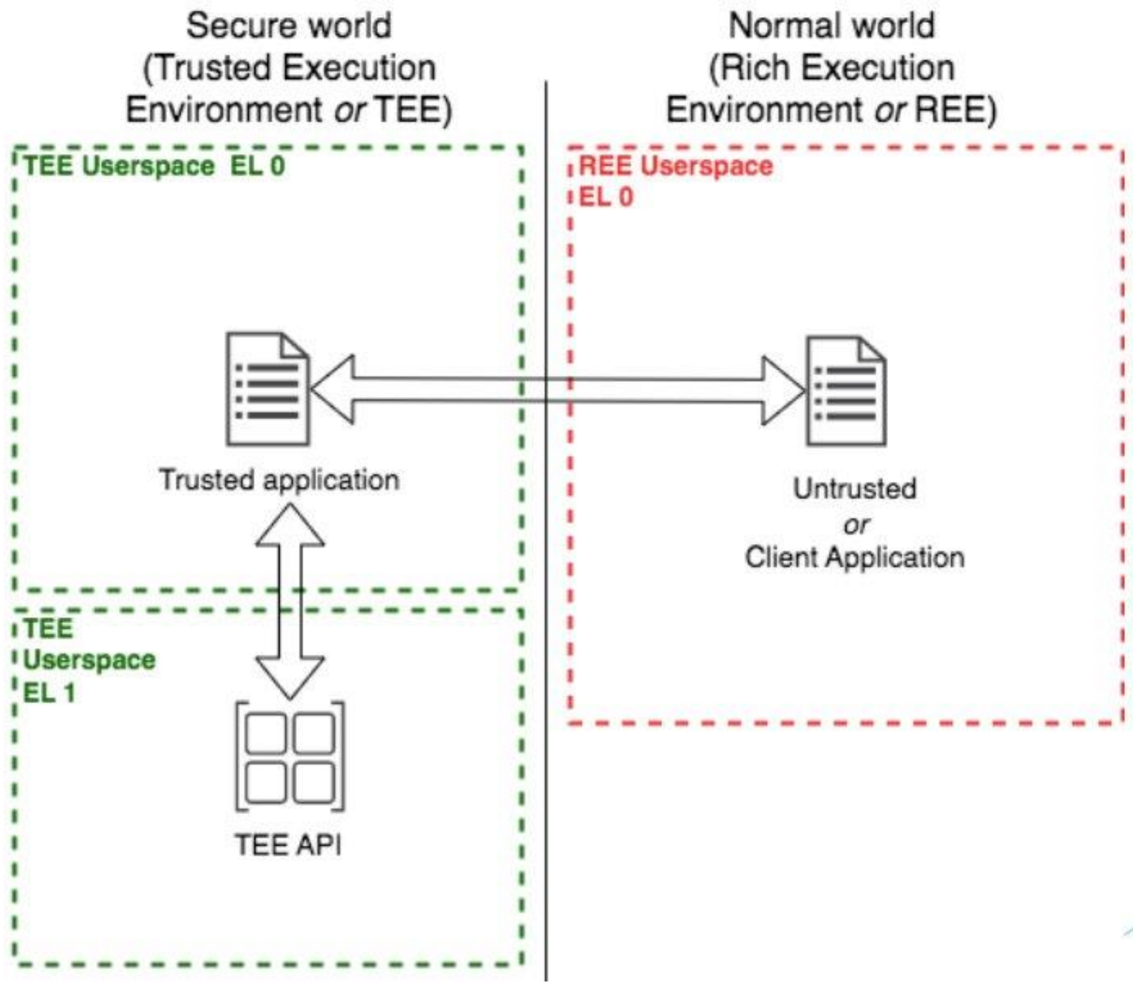
1. **TEE or Trusted Execution Environment**

Execution context where all the security critical operations reside. TEE has its own

a) **secure/encrypted memory storage,**

b) **secure I/O peripherals,**

c) **secure context switching**



"Trusted" Execution Environment

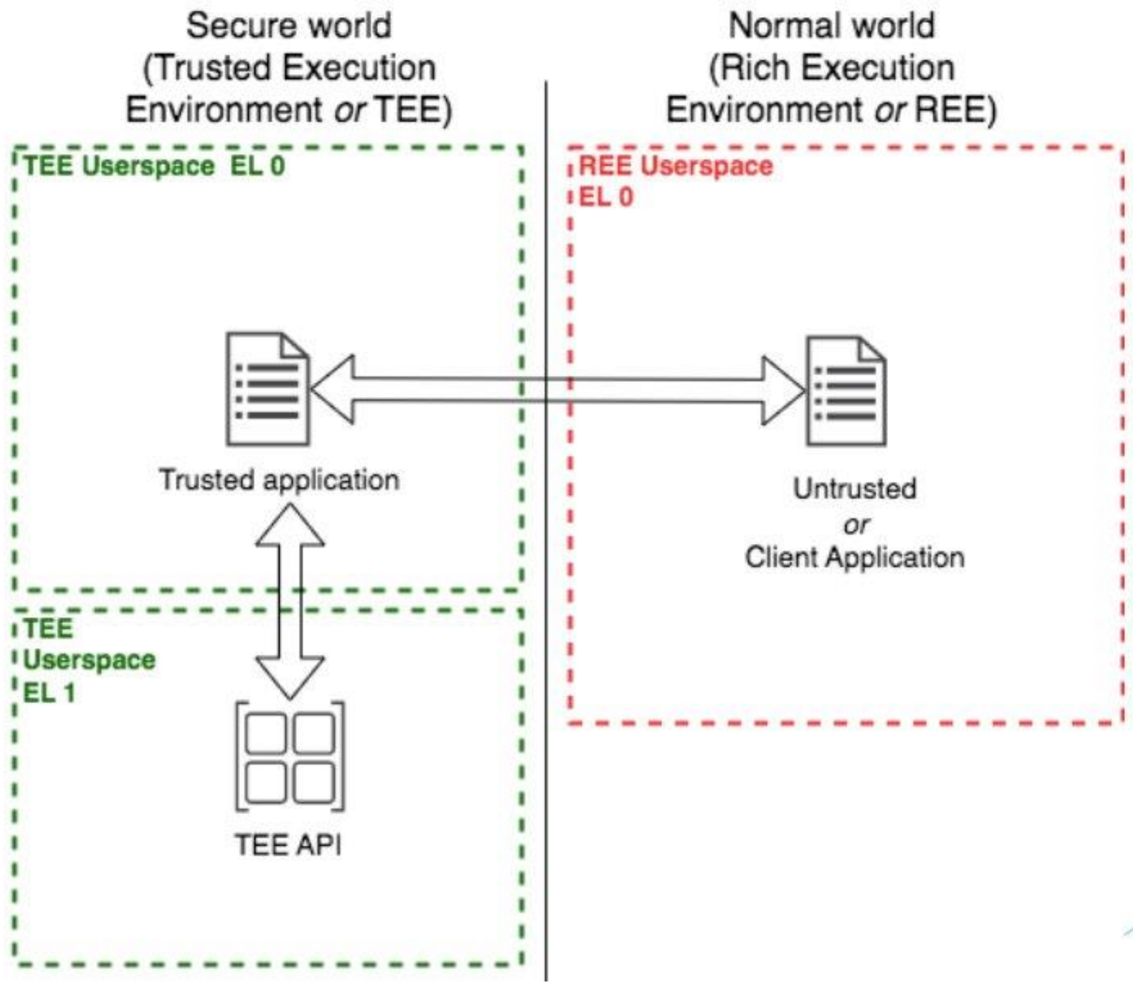
- **Two main divisions**

1. **TEE or Trusted Execution Environment**

Execution context where all the security critical operations reside. TEE has its own
a) **secure/encrypted memory storage,**
b) **secure I/O peripherals,**
c) **secure context switching**

2. **REE or Rich Execution Environment**

Execution context where rest of the things run. REE invokes the services of TEE when required.



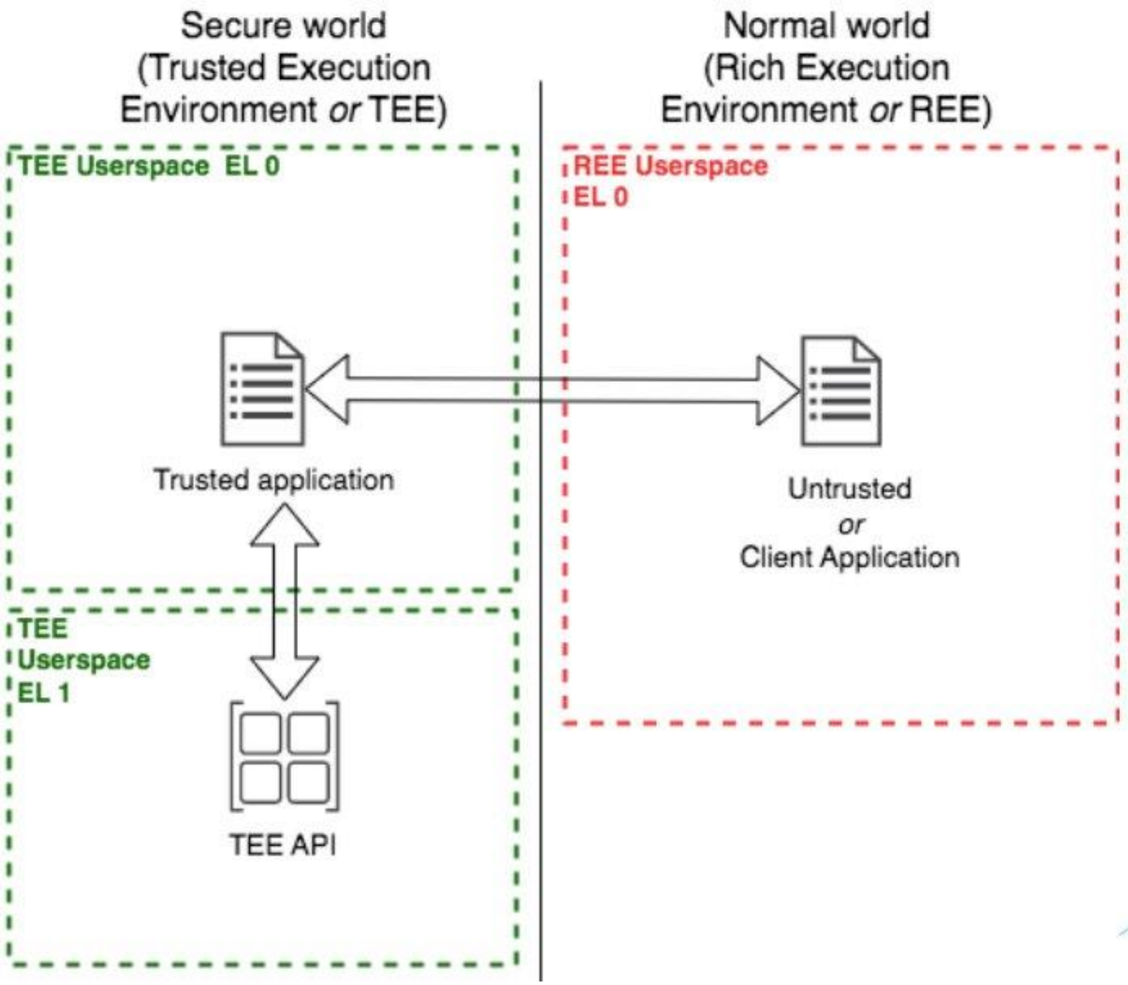
"Trusted" Execution Environment

- **Two main divisions**

1. **TEE or Trusted Execution Environment**

2. **REE or Rich Execution Environment**

Note: All Trusted Applications (TAs) running in the TEE are checked for integrity, implying no adversary having complete control over REE can execute arbitrary TEE code.



"Trusted" Execution Environment

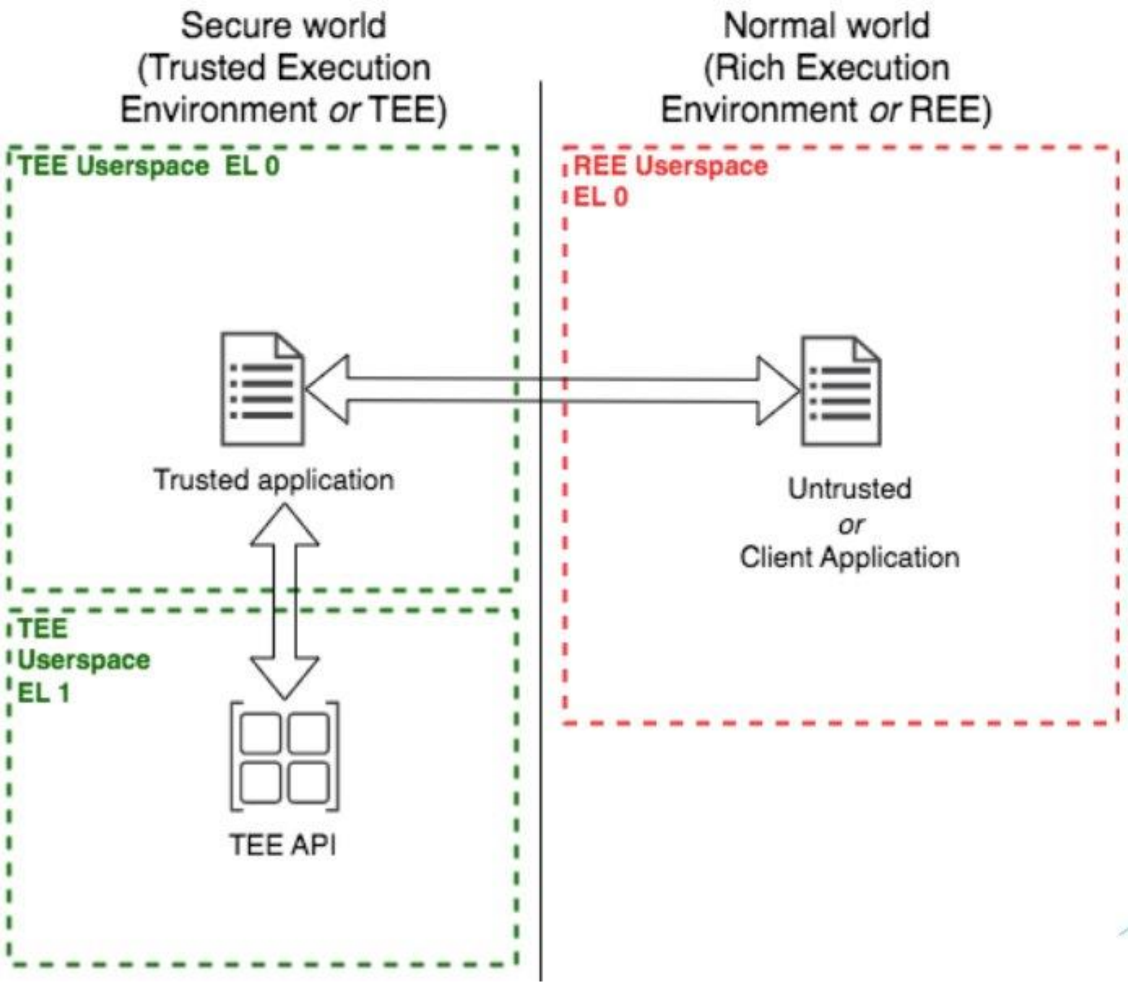
- Two main divisions

1. TEE or Trusted Execution Environment

2. REE or Rich Execution Environment

ADVERSARIAL GOAL !

Note: All Trusted Applications (TAs) running in the TEE are checked for integrity, implying no adversary having complete control over REE can execute arbitrary TEE code.



Adversarial Goals

- **Goal 1 :** Entire attack must be **online** (without taking the device offline)

Adversarial Goals

- **Goal 1 :** Entire attack must be **online** (without taking the device offline)
  - **Challenge 1** : Secure Boot cannot be attacked (requires taking the device offline)

(Our) **Solution**: Attack the loading of Trusted Applications in the TEE

Adversarial Goals

- **Goal 1 :** Entire attack must be **online** (without taking the device offline)
  - **Challenge 1** : Secure Boot cannot be attacked (requires taking the device offline)

    (Our) **Solution**: Attack the loading of Trusted Applications in the TEE
  - **Challenge 2 :** Cannot use **code-based** triggers (requires code modifications to the OP-TEE kernel)

    (Our) **Solution**: Construct a combined adversary (side-channel analysis + fault injection)

Adversarial Goals

Goal 2 : The attack must be non-invasive

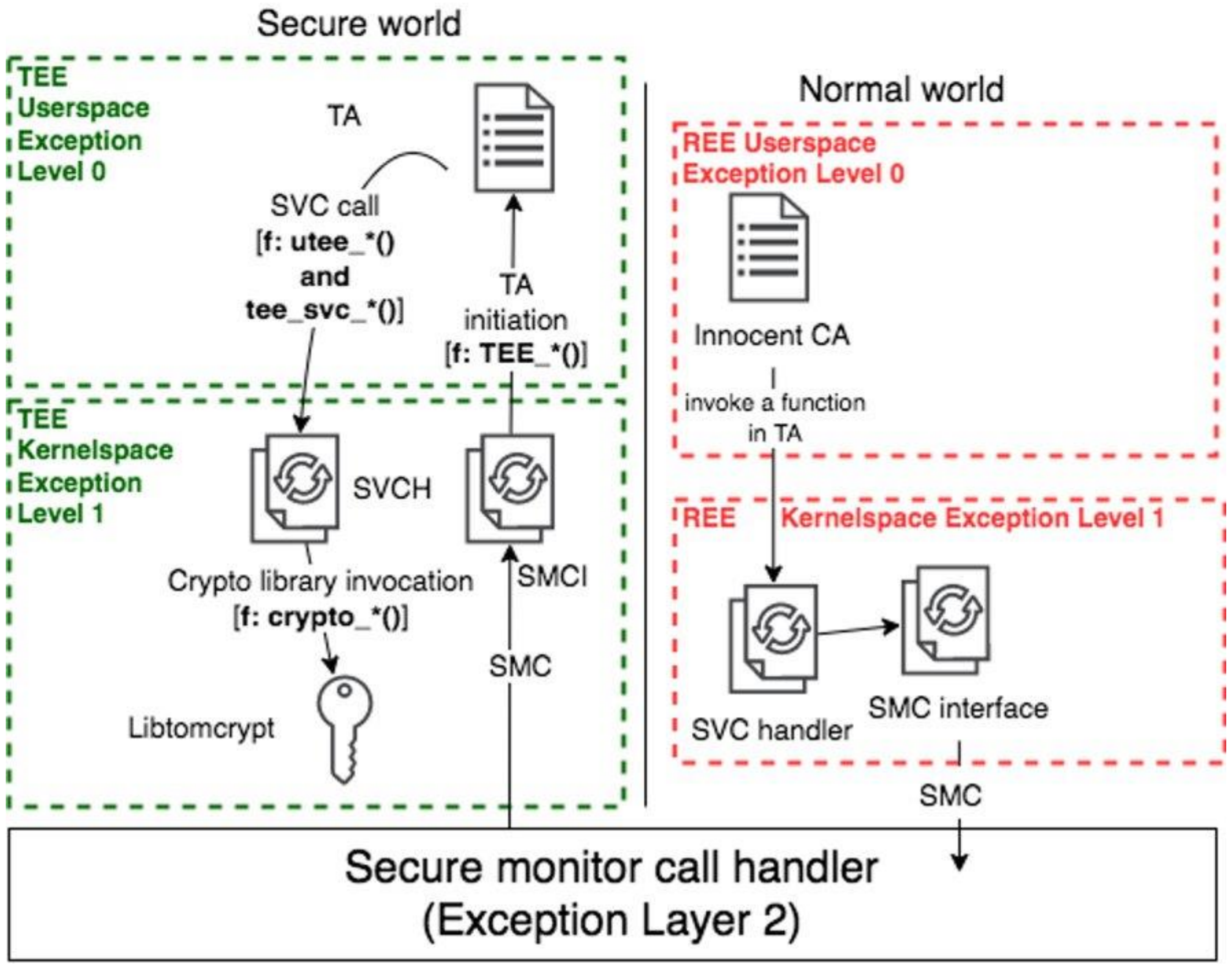
Adversarial Goals

Goal 2 : The attack must be non-invasive

- **Challenge 3** : Cannot inject processor faults (requires depackaging). Trivial attacks like instruction skips cannot work

(Our) **Solution**: Work with a new fault model (register sweeping) on the system-bus (requires no invasive alterations to the target device)

Fault Attack Target

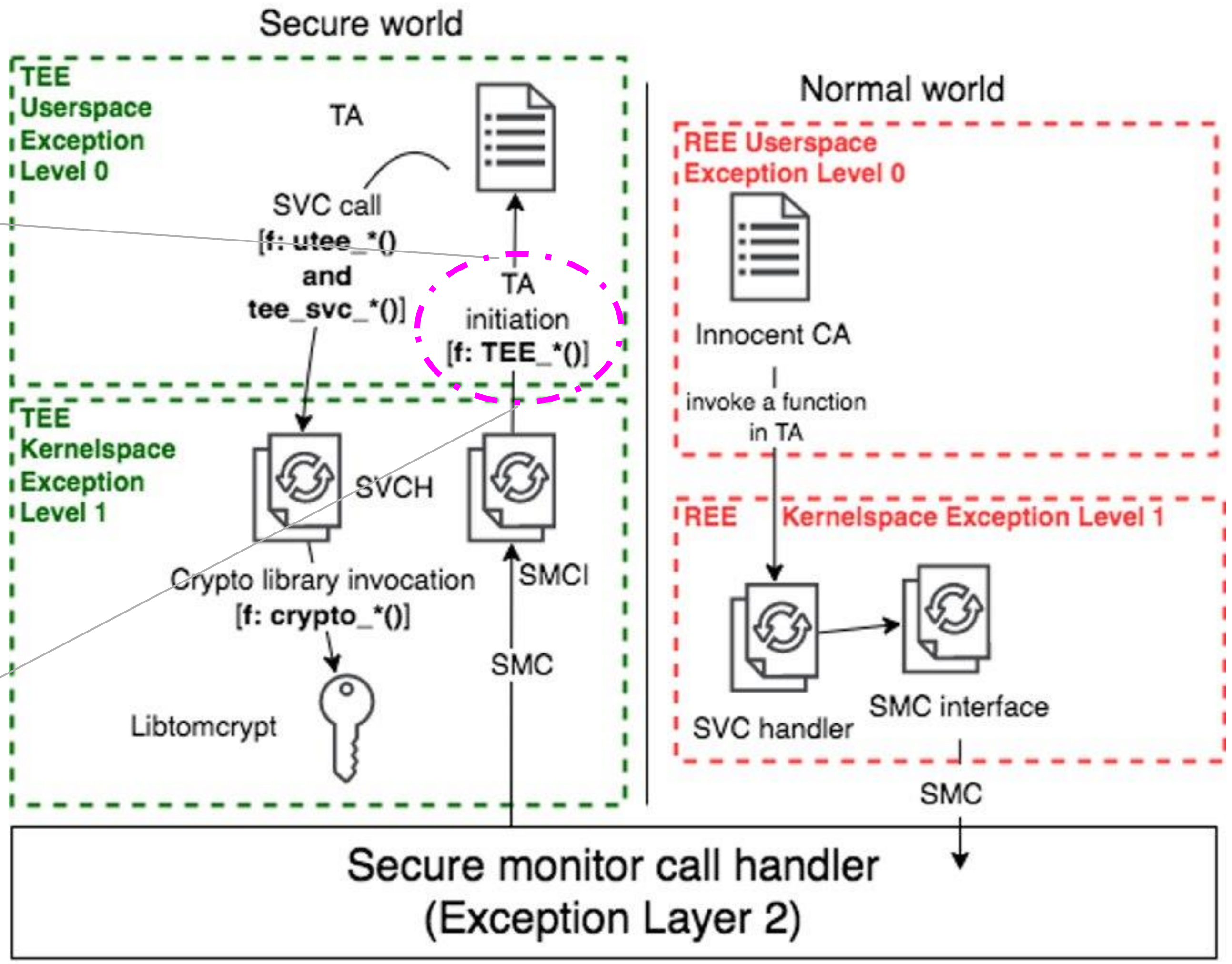


Fault Attack Target

```
#define TEE_SUCCESS 0x00000000
#define TEE_ERROR_SECURITY 0xFFFF000F

TEE_Result verify_signature(char* ta_binary, uint8_t* signature){
    if(/*signature is valid*/)
        return TEE_SUCCESS;
    return TEE_ERROR_SECURITY;
}

// load a TA referenced by a CA
void load_TA(...){
    // some code here
    TEE_Result res = verify_signature(...)
    if(res != TEE_SUCCESS)
        // abort execution
    // some more code here
}
```

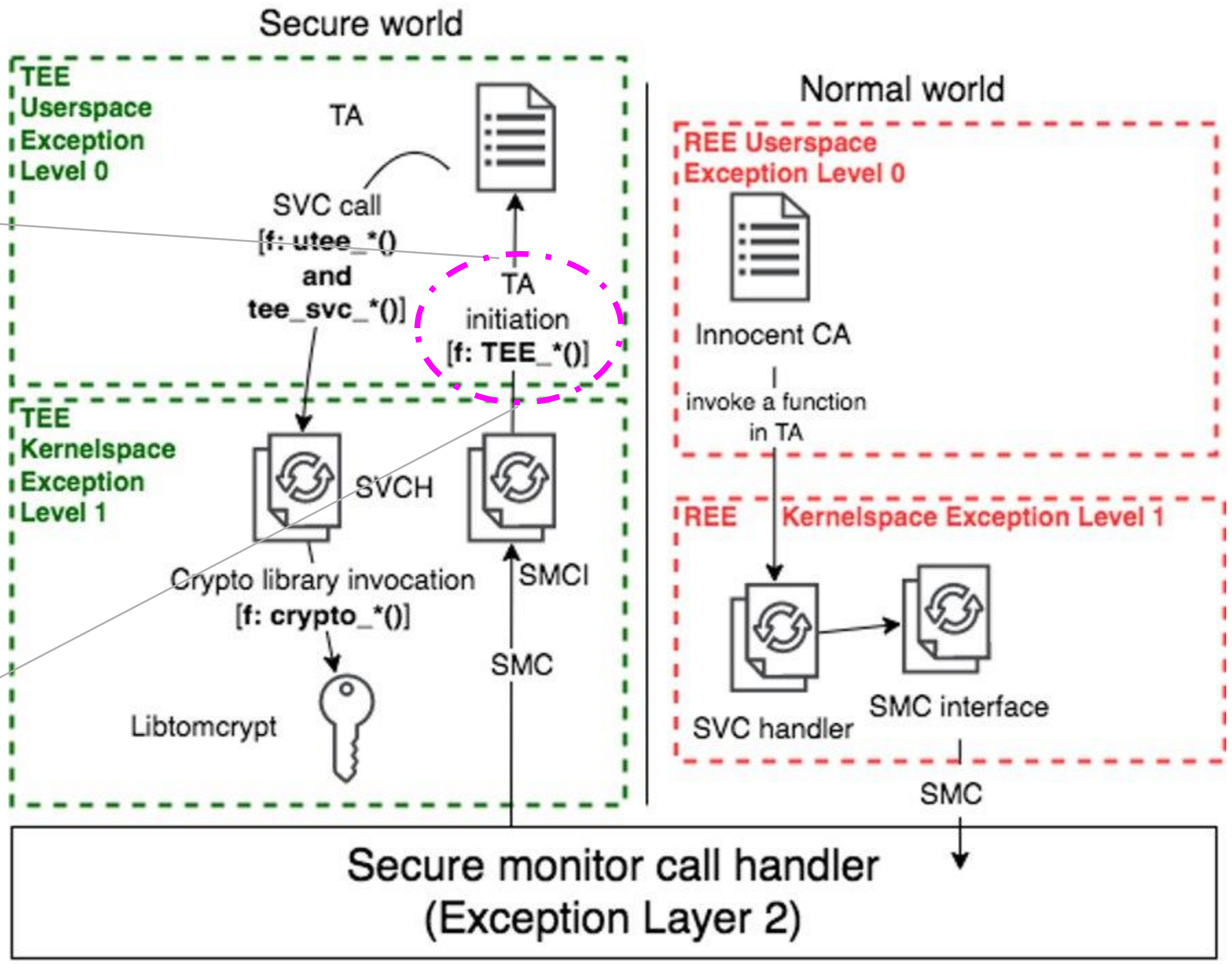


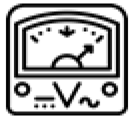
Fault Attack Target

```
#define TEE_SUCCESS 0x00000000
#define TEE_ERROR_SECURITY 0xFFFF000F

TEE_Result verify_signature(char* ta_binary, uint8_t* signature){
    if(/* signature is valid */)
        return TEE_SUCCESS;
    return TEE_ERROR_SECURITY;
}

// load a TA referenced by a CA
void load_TA(...){
    // some code here
    TEE_Result res = verify_signature(...)
    if(res != TEE_SUCCESS)
        // abort execution
    // some more code here
}
```




External glitch


DVFS


Rowhammer

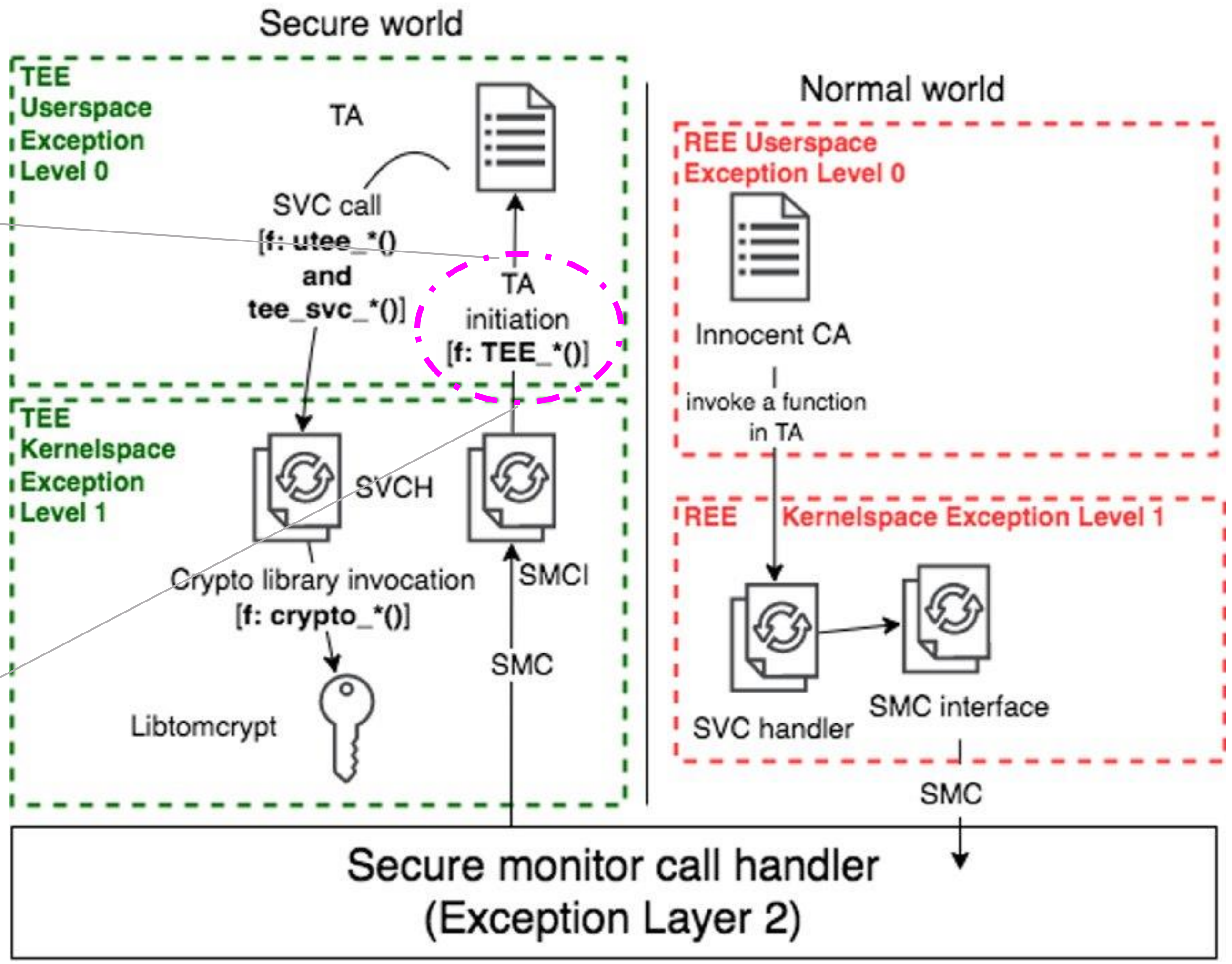

Stealing signing key

Fault Attack Target

```
#define TEE_SUCCESS 0x00000000
#define TEE_ERROR_SECURITY 0xFFFF000F

TEE_Result verify_signature(char* ta_binary, uint8_t* signature){
    if(/* signature is valid */)
        return TEE_SUCCESS;
    return TEE_ERROR_SECURITY;
}

// load a TA referenced by a CA
void load_TA(...){
    // some code here
    TEE_Result res = verify_signature(...)
    if(res != TEE_SUCCESS)
        // abort execution
    // some more code here
}
```



Not Available

Not Available

Protected TA access

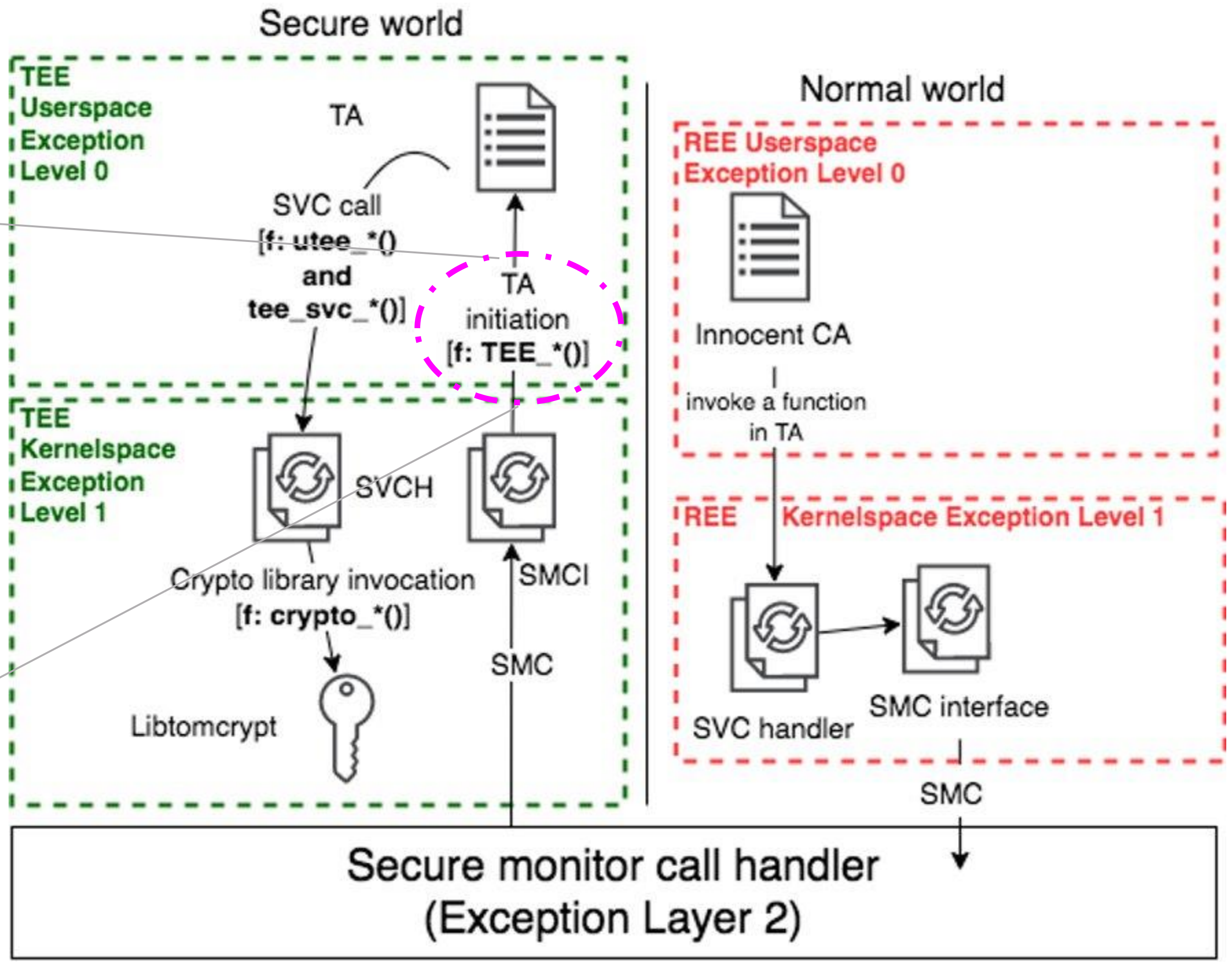
Signing key not stored on device

Fault Attack Target

```
#define TEE_SUCCESS 0x00000000
#define TEE_ERROR_SECURITY 0xFFFF000F

TEE_Result verify_signature(char* ta_binary, uint8_t* signature){
    if(/* signature is valid */)
        return TEE_SUCCESS;
    return TEE_ERROR_SECURITY;
}

// load a TA referenced by a CA
void load_TA(...){
    // some code here
    TEE_Result res = verify_signature(...)
    if(res != TEE_SUCCESS)
        // abort execution
    // some more code here
}
```



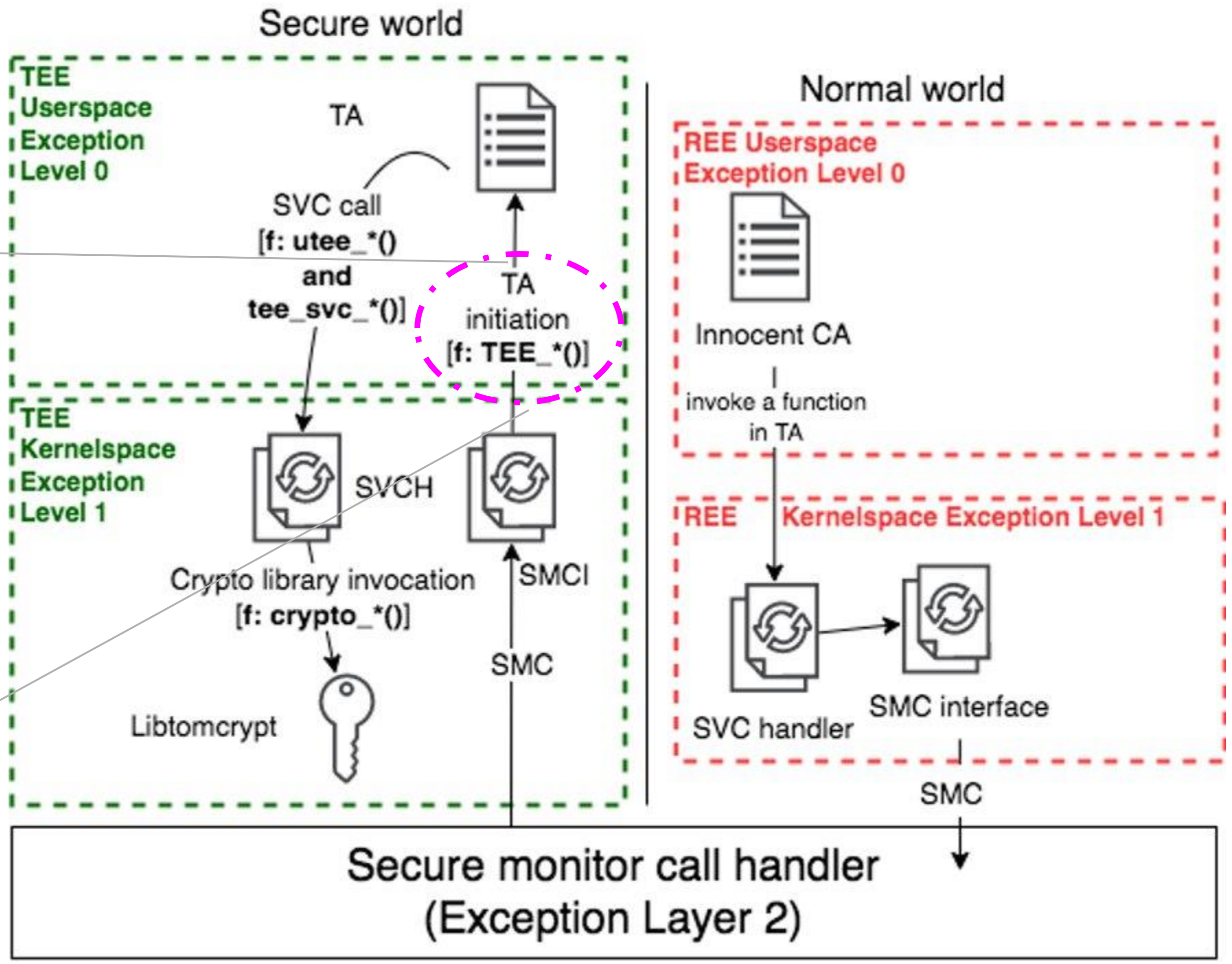
Register Sweeping: Fault the load to 0x0 through data bus faults

Fault Attack Target

```
bl      0 <crypto_acipher_rsassa_verify>
str     w0, [sp, #76]
```

FAULT INJECTION TARGET!

```
ldr     w0, [sp, #76]
cmp     w0, #0x0
b.eq    1e0 <shdr_verify_signature+0x1e0>  // b.none
mov     w0, #0xffff000f                    // #-65521
```



Register Sweeping: Fault the load to 0x0 through data bus faults

Fault Attack Results

- **No Effect** (denoted by a "dot") : No effect of the injected fault
- **Partial Success** : Injected fault changes the value of the load, but not to 0x0
  Or causes SEGFAULT
- **Success** : Faults value of the load to 0x0.

```
[PARTIAL SUCCESS] Register value corrupted to 0x1073c....
Segmentation fault (core dumped)
....Illegal instruction (core dumped)
Illegal instruction (core dumped)
........................................Illegal instructio
........Illegal instruction (core dumped)
...Illegal instruction (core dumped)
................[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
[SUCCESS] Register value corrupted to 0x0
```

End to End Attack

Load (adversarial) Trusted Applications through Faults

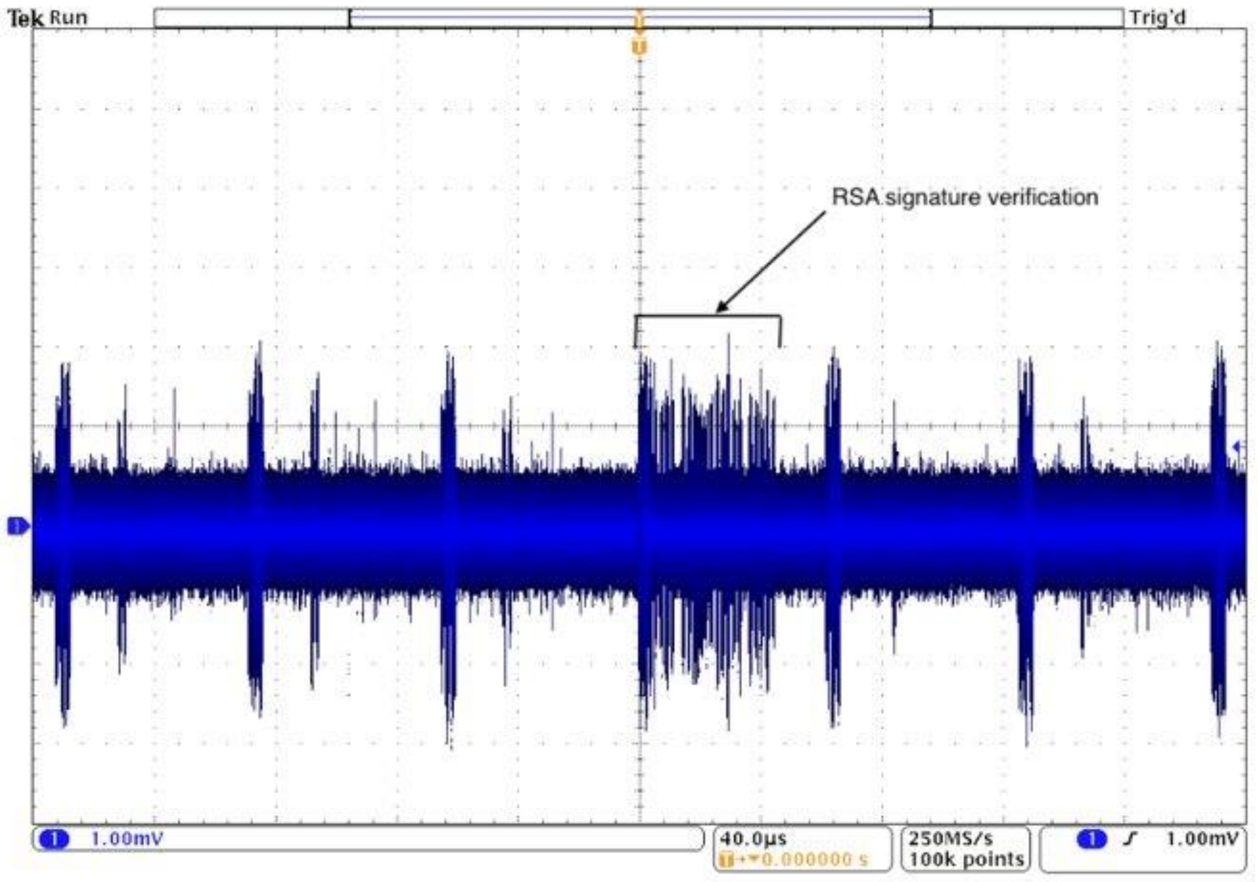
Redirect communication for other Trusted Applications

Decrypt (redirected) communication

End to End Attack

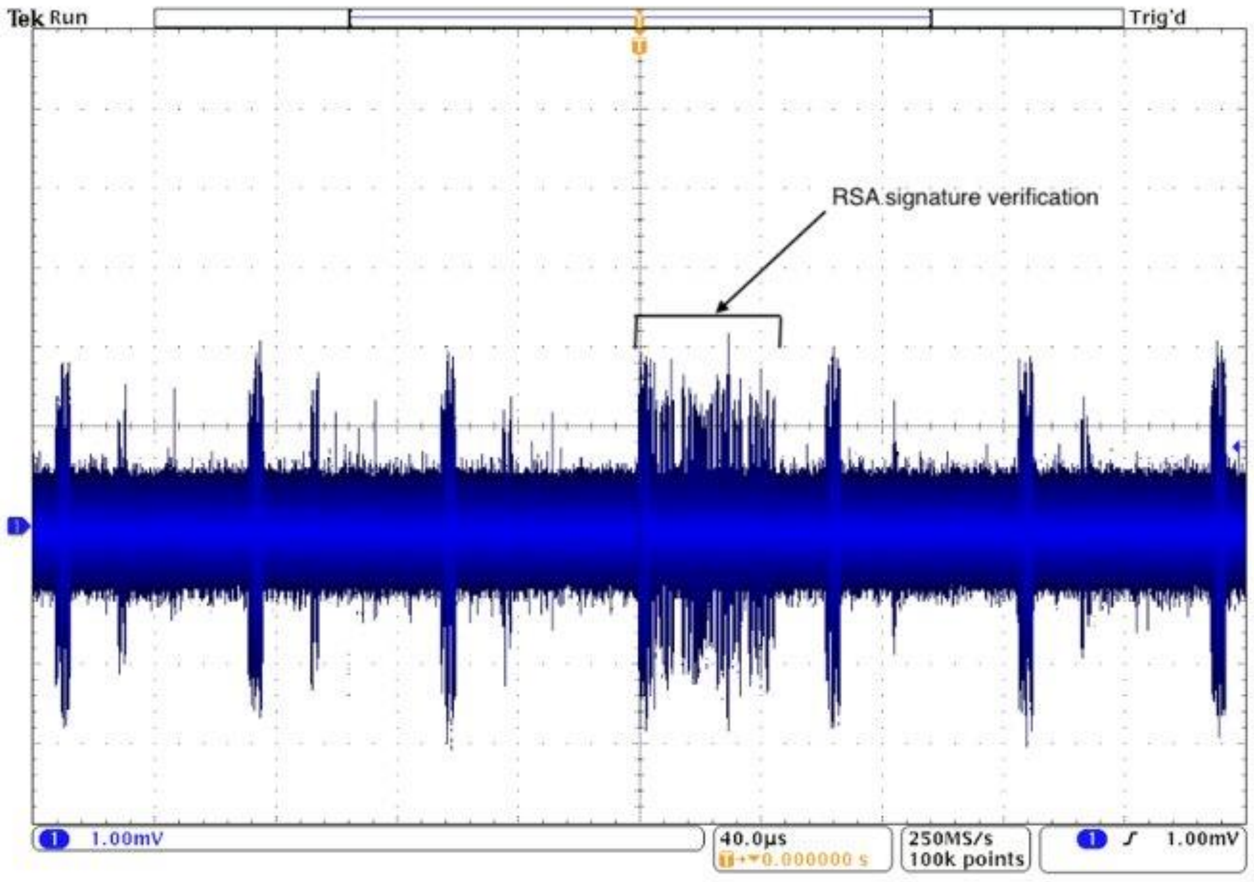
Load (adversarial) Trusted Applications through Faults

Combined Adversary = Power SCA + FI



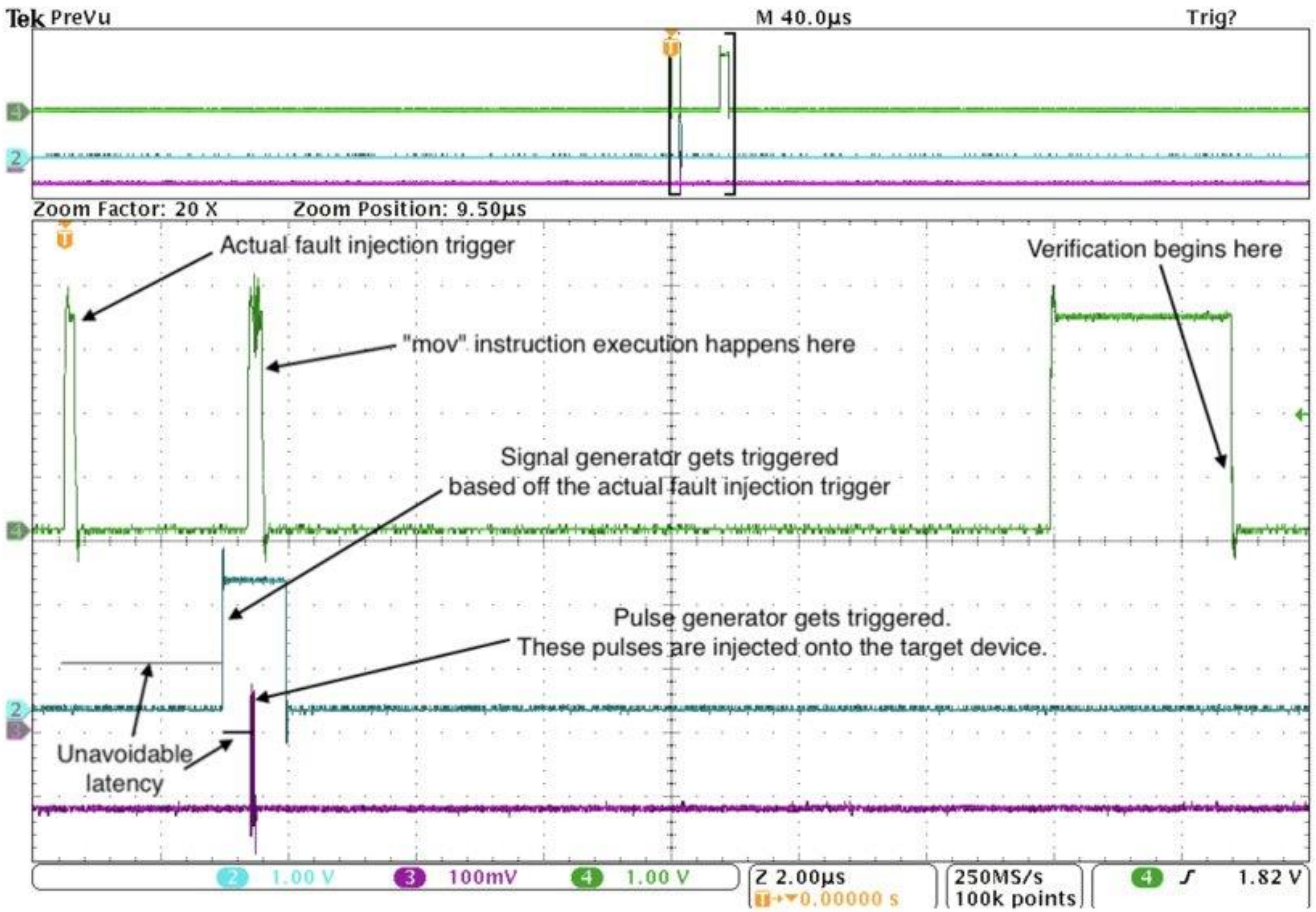
Power side-channel to inform
fault injection in a **non-invasive** way
(no recompilation of OP-TEE necessary)

Combined Adversary = Power SCA + FI



Power side-channel to inform
fault injection in a **non-invasive** way
(no recompilation of OP-TEE necessary)

⊕



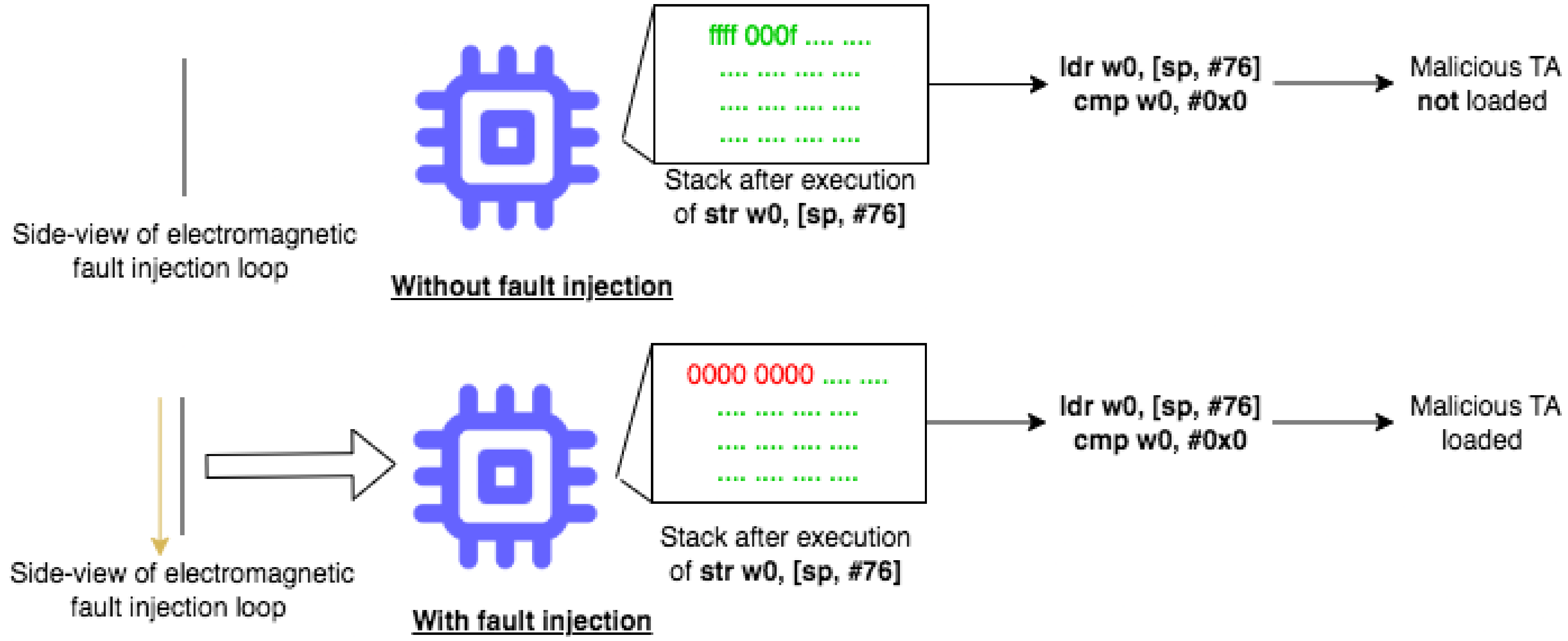
Actual Fault Injection on signature verification

Combined Adversary = Power SCA + FI

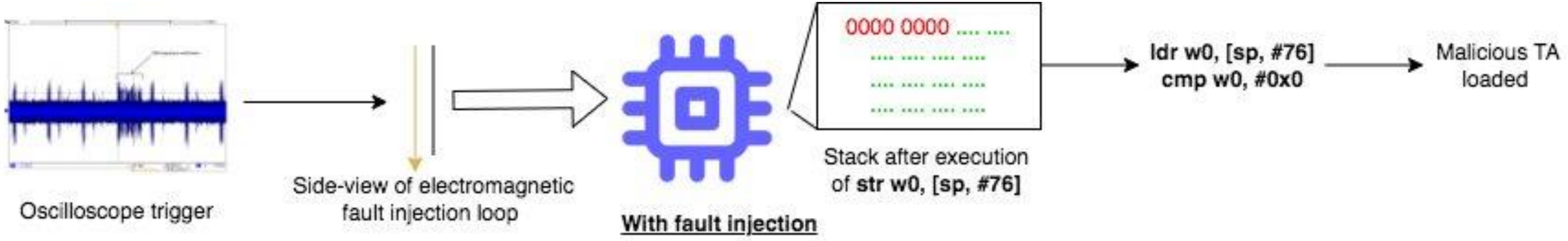
```
bl        0 <crypto_acipher_rsassa_verify>
str       w0, [sp, #76]
```

FAULT INJECTION TARGET!

```
ldr      w0, [sp, #76]
cmp      w0, #0x0
b.eq     1e0 <shdr_verify_signature+0x1e0>  // b.none
mov      w0, #0xffff000f                   // #-65521
```



Combined Adversary = Power SCA + FI



Fallout: **Register sweeping fault attack** loads a **self-signed**, adversarial controlled Trusted Application in the secure world of OP-TEE

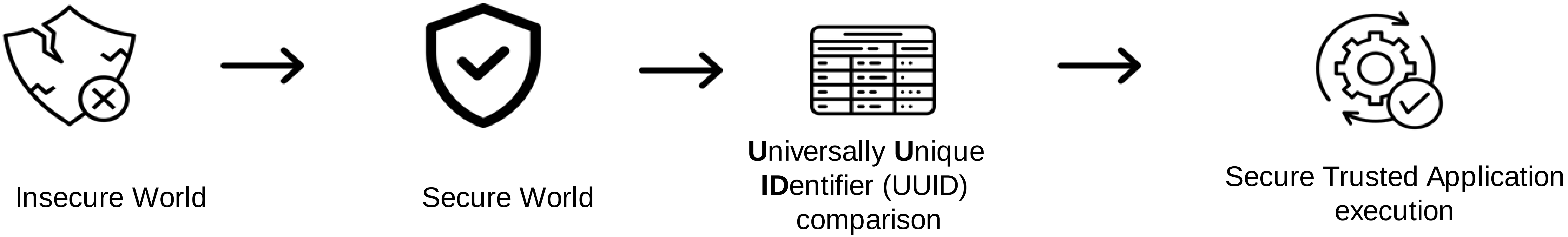
Fallout: Register sweeping fault attack le… …ed, adversarial controlled Trusted Application in t… …of OP-TEE

Privilege Escalation !

End to End Attack

Redirect communication for other Trusted Applications

Communication Redirection



Communication Redirection

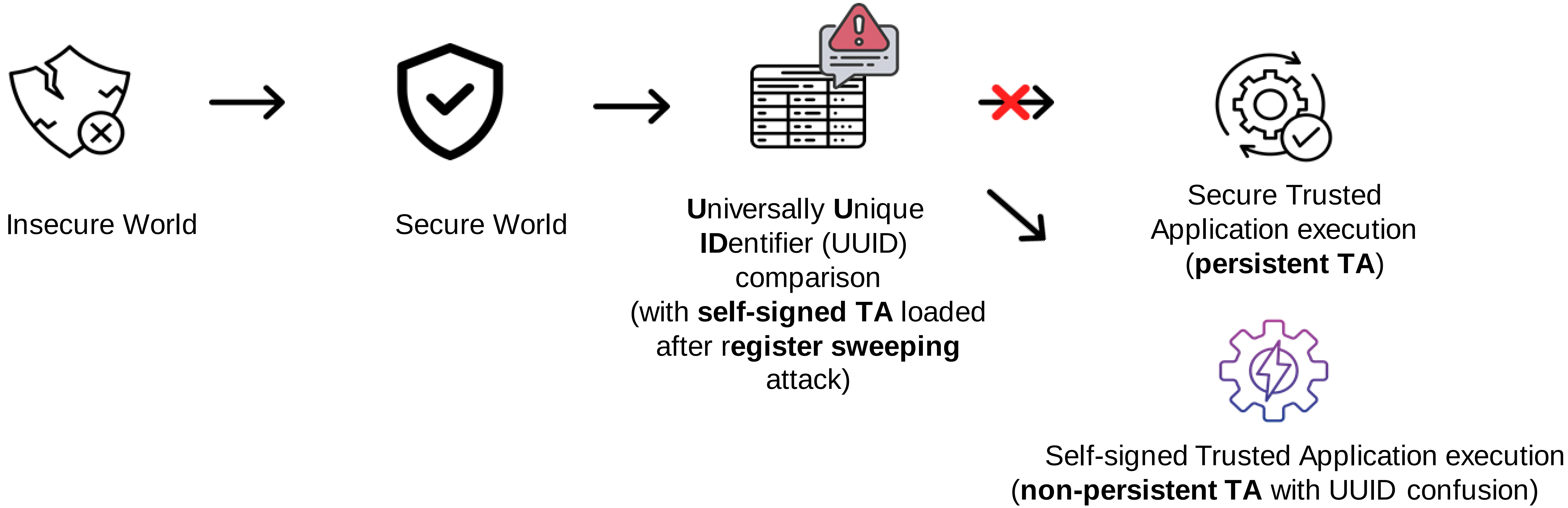
Our Findings: GlobalPlatform API specification (upon which OP-TEE is constructed) **offloads** the responsibility of choosing UUID to **Original Equipment Manufacturer**. It is the responsibility of the OEM to ensure **no two Trusted Applications (TA) share same UUID**.

Communication Redirection

Our Findings: GlobalPlatform API specification (upon which OP-TEE is constructed) **offloads** the responsibility of choosing UUID to **Original Equipment Manufacturer**. It is the responsibility of the OEM to ensure **no two Trusted Applications (TA) share same UUID**.

UUID confusion: Behaviour of the system when **UUID are non-unique is undefined**. Our empirical conclusion is that, when UUIDs are shared, a **non-persistent TA is preferred over persistent TA.**

Communication Redirection



End to End Attack

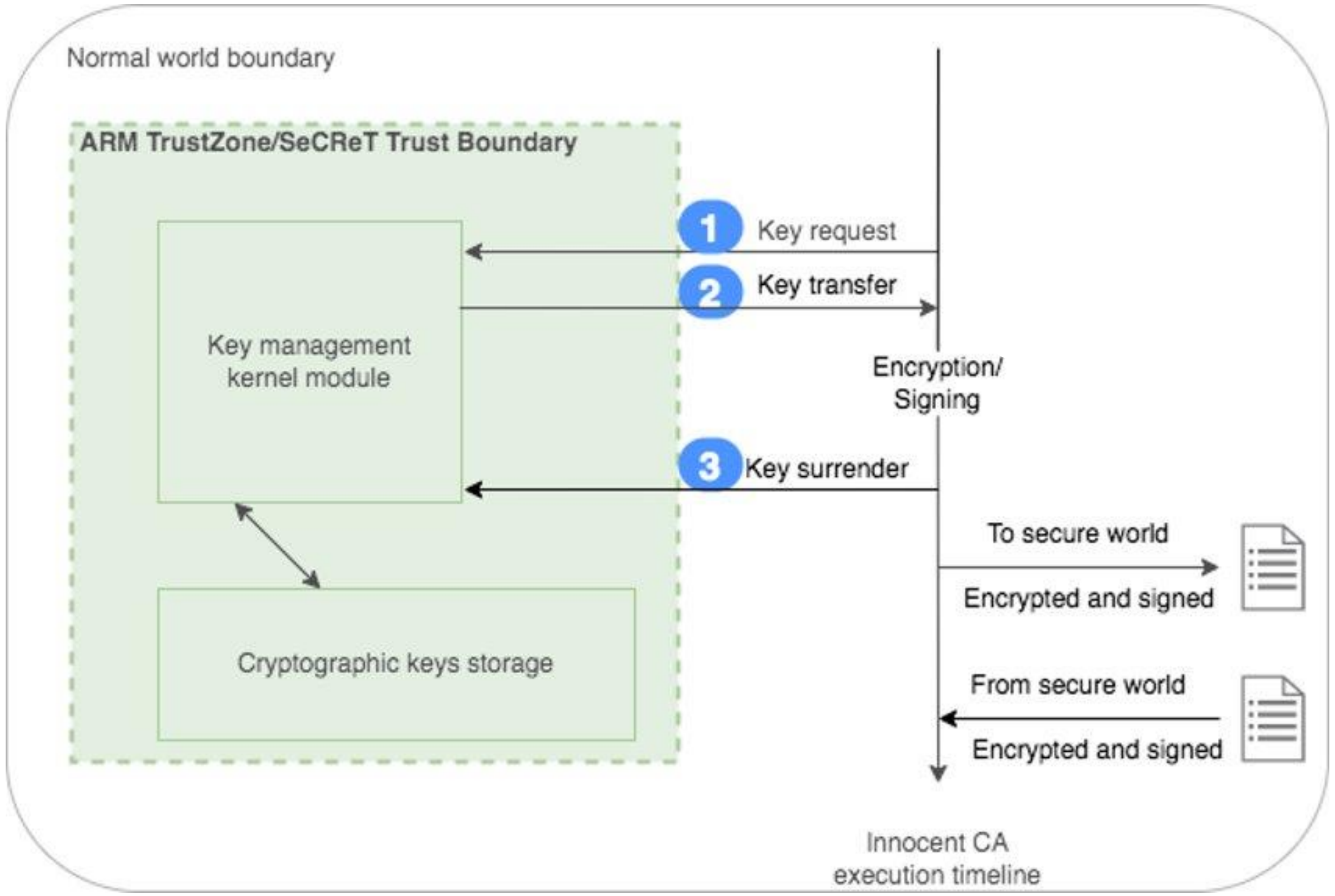
Decrypt (redirected) communication

Decrypt (redirected) communication

Third Party extension: SeCReT

- Symmetric key management
- Blocks SIGTRAP
- Blocks unauthorized read to sensitive data pages

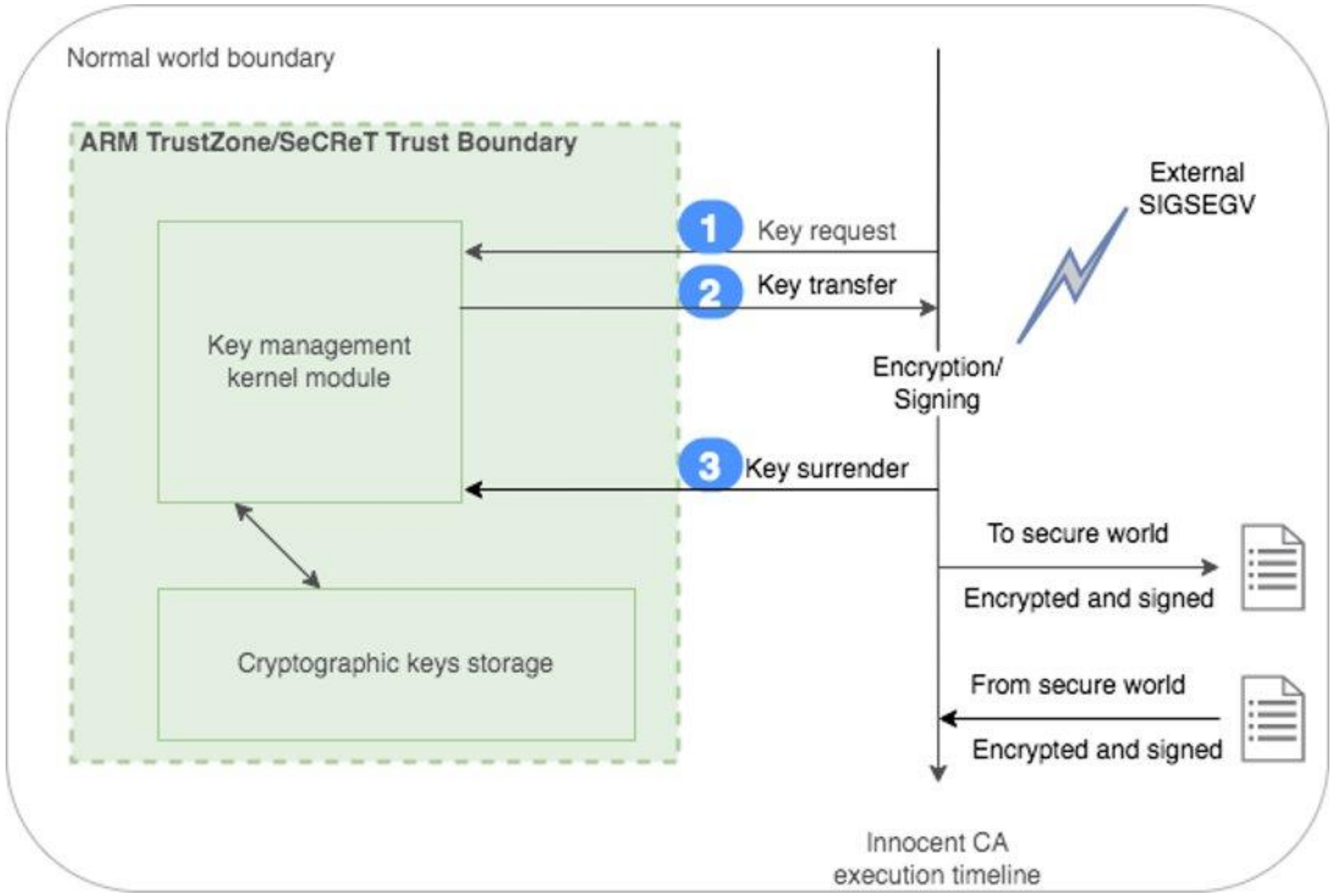
Decrypt (redirected) communication



Third Party extension: SeCReT

- Symmetric key management
- Blocks SIGTRAP
- Blocks unauthorized read to sensitive data pages

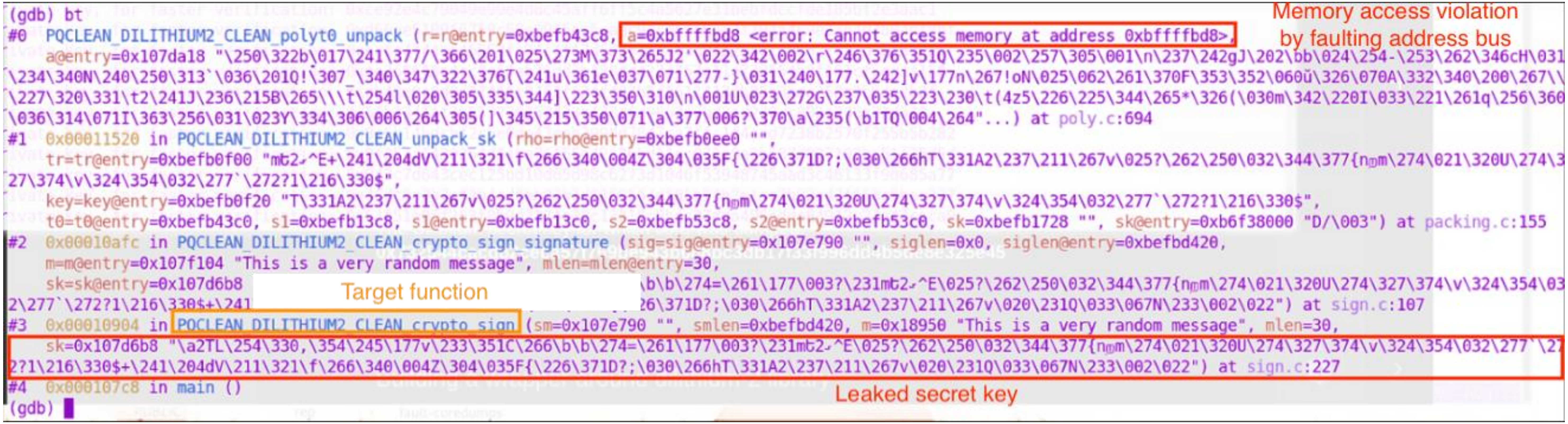
Decrypt (redirected) communication



Third Party extension: SeCReT

- Symmetric key management
- Blocks SIGTRAP
- Blocks unauthorized read to sensitive data pages
- Does not block SIGSEGV. Leaks key through coredumps

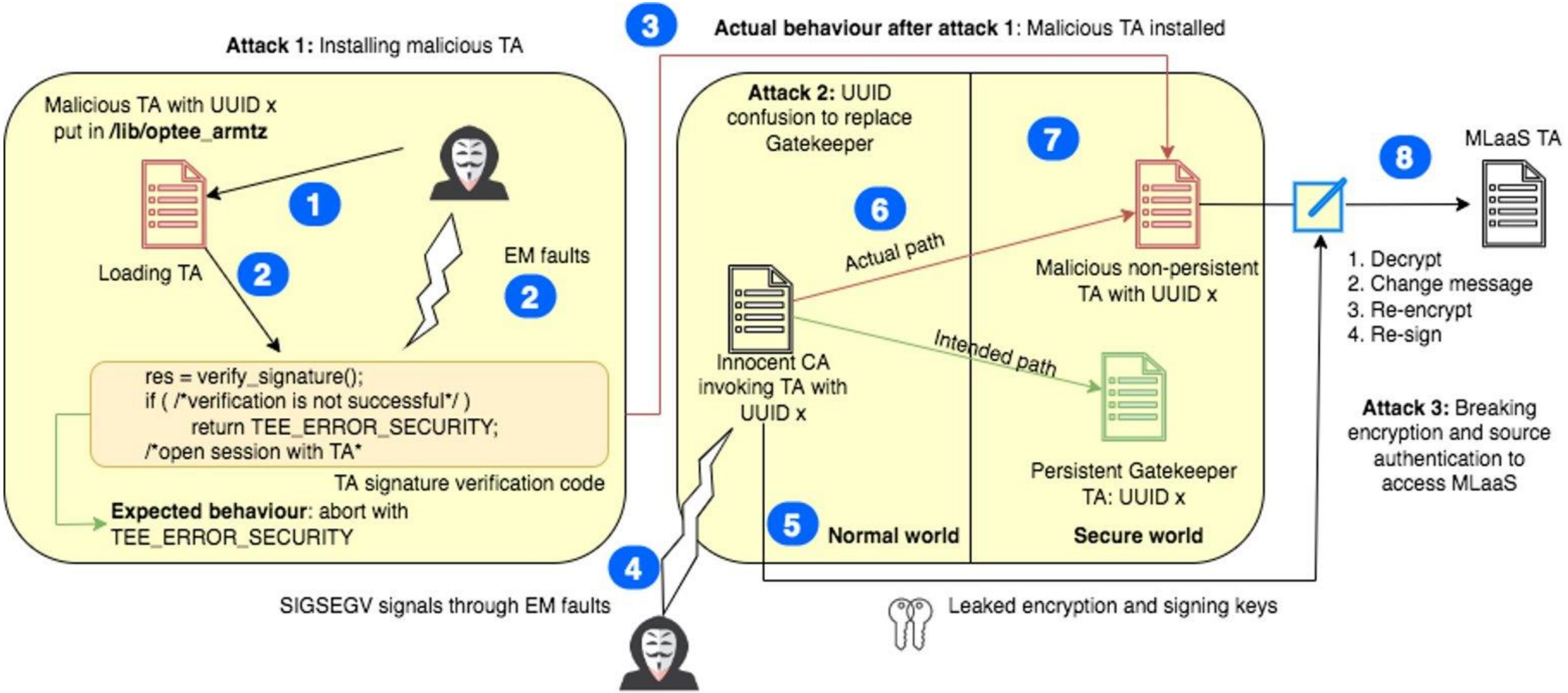
Decrypt (redirected) communication



End to End Attack

Bird's eye-view

End to End Attack



Impact

Responsible Disclosure

- CVE 2022-47549
- Worked together with Linaro to deploy countermeasure in OP-TEE kernel
- **Website:** https://nimishmishra.wixsite.com/disarmament

Countermeasure

```
-        res = crypto_acipher_rsassa_verify(shdr->algo, &key, shdr->hash_size,
-                                           SHDR_GET_HASH(shdr), shdr->hash_size,
-                                           SHDR_GET_SIG(shdr), shdr->sig_size);
+        FTMN_CALL_FUNC(res, &ftmn, FTMN_INCR0,
+                       crypto_acipher_rsassa_verify, shdr->algo, &key,
+                       shdr->hash_size, SHDR_GET_HASH(shdr), shdr->hash_size,
+                       SHDR_GET_SIG(shdr), shdr->sig_size);
+        if (!res) {
+                ftmn_checkpoint(&ftmn, FTMN_INCR0);
+                goto out;
+        }
+        err_incr = 1;
+ err:
+        res = TEE_ERROR_SECURITY;
+        FTMN_SET_CHECK_RES_NOT_ZERO(&ftmn, err_incr * FTMN_INCR0, res);
```

Other Implications

- Re-enable Differential Fault Attack (DFA) on T-table implementation of AES (on SoCs)

- Address Bus Faults to leak **all** shares of Masked PQC implementations (like Kyber)

Observation: All shares encapsulated within a **single** memory structure

Takeaways!

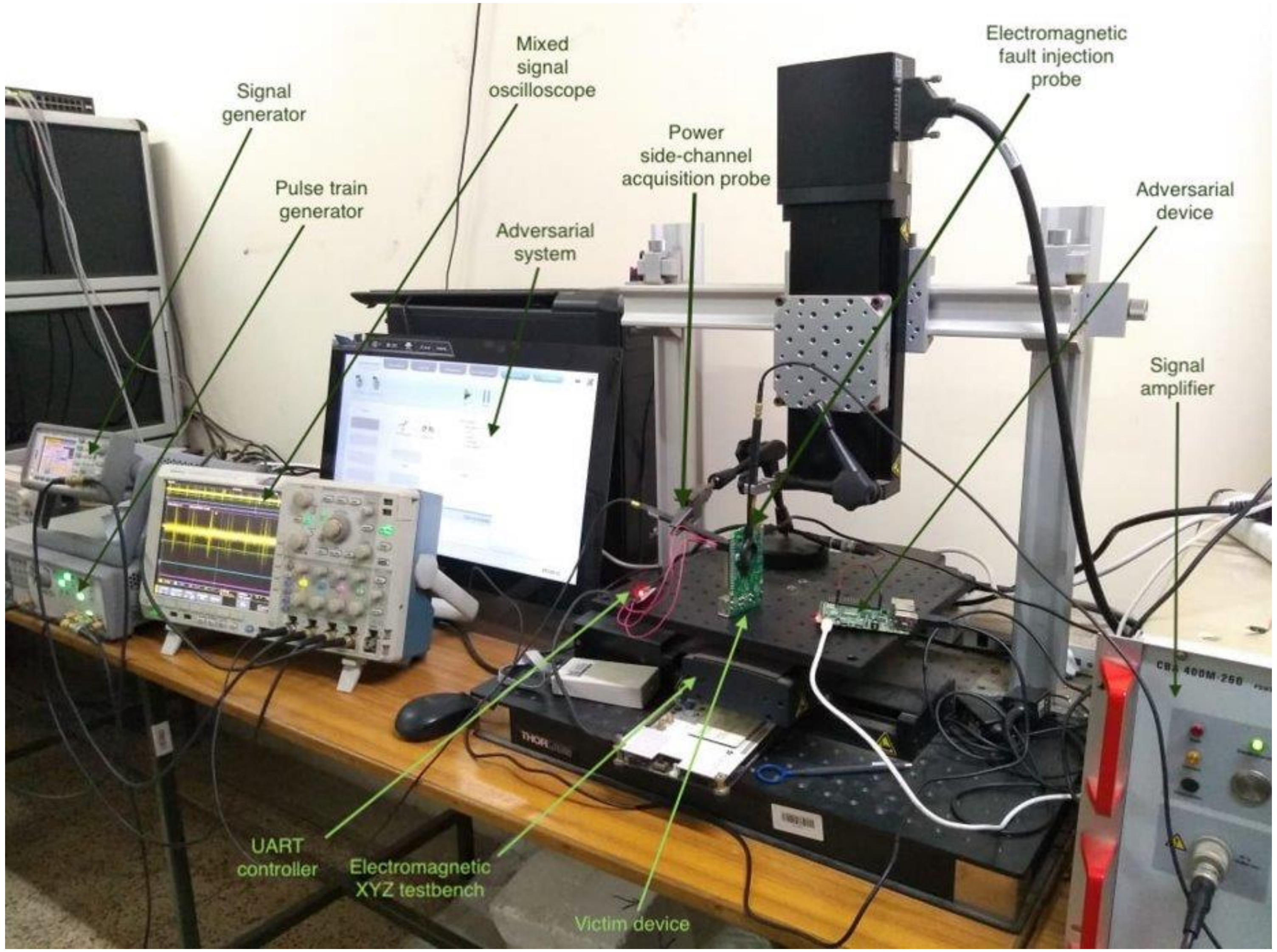
- System + Execution Environment, not *just* the System
- Register sweeping fault model on a (new) architectural aspect – System Bus

  - Implications for other systems?
- Rethinking protocol specifications for embedded systems in light of SCA+FI adversaries

Research @ Secured Embedded Architecture Laboratory, IIT Kgp

(Some) Research Directions

- Power/EM **Side-channel evaluation** of FPGAs/micro-controllers/SoCs
- **Fault** Attacks, Fault Analysis, and design of countermeasures
- Evaluation of **Microarchitectural attack** s cenarios on workstations as well as embedded systems
- Others directions…



black hat ASIA 2024

APRIL 18-19, 2024
BRIEFINGS

WEBSITE

Thank You!