MASTER IN

CYBER–SECURITY

UNLOCK YOUR POTENTIAL

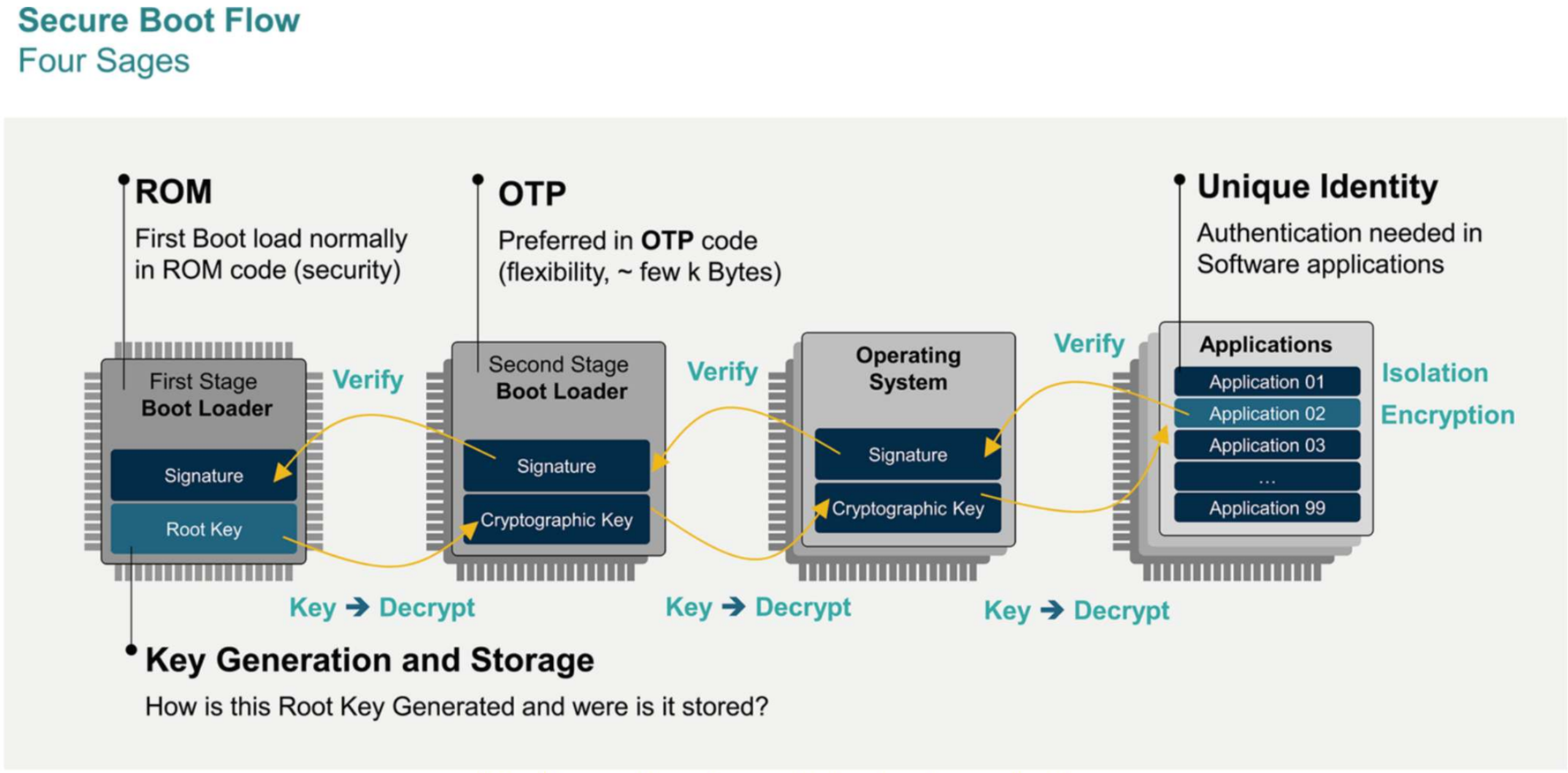
Fault Injection vulnerabilities and countermeasures

UNIVERSIDAD DE MURCIA

Facultad de Informática

Secure Boot

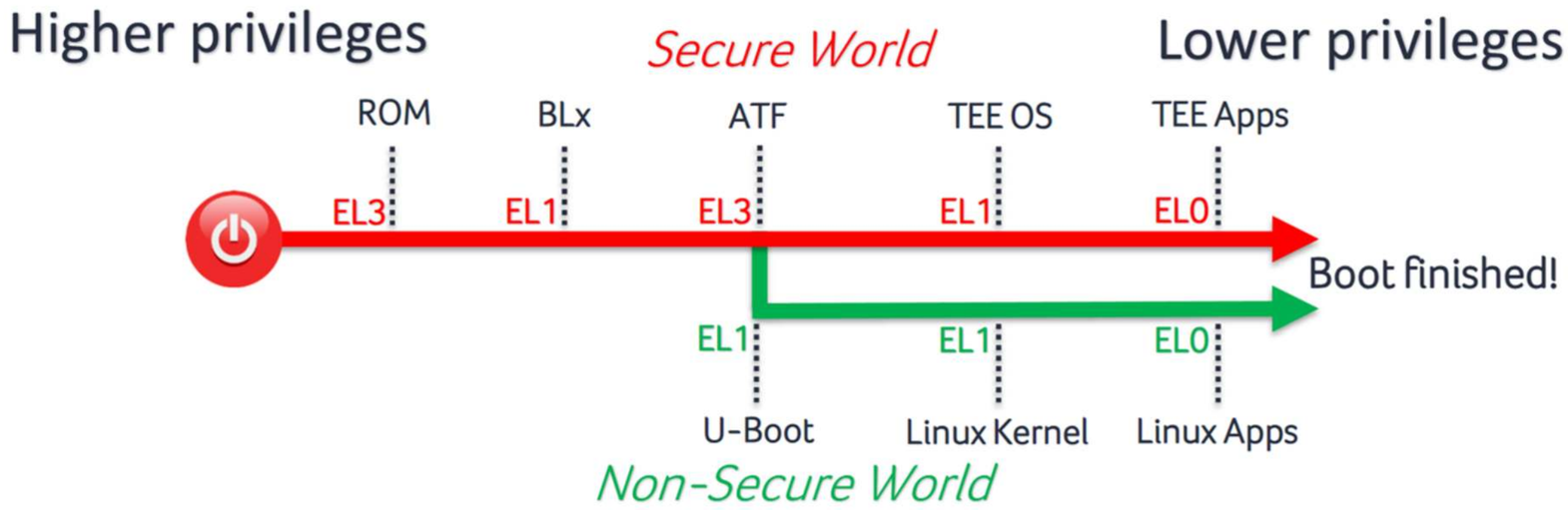
The target code will be a simplified bootloader operating during a secure boot process



Source https://www.pufsecurity. com/technology/secure-boot/

Secure Boot

The target code will be a simplified bootloader operating during a secure boot process



Source

Example Bootloader Code

```
void bl1_entry() {
        MD5_CTX ctx;
        unsigned char img_hash[16];
        unsigned char otp_img_hash[16];
        bool is_sec_boot_en;

        void* img_base = IMG_LOAD_ADDR;
        size_t img_size;

        otp_init();

        otp_is_sec_boot_enabled(&is_sec_boot_en);

        if (!is_sec_boot_en) {
                goto do_boot;
        }

        serial_puts("Start Secure Boot...\n");

        otp_get_img_hash(otp_img_hash);

        flash_load_img(img_base, &img_size);

        MD5_Init(&ctx);
        MD5_Update(&ctx, img_base, img_size);
        MD5_Final(img_hash, &ctx);

        if(is_sec_boot_en && memcmp(img_hash, otp_img_hash, sizeof(otp_img_hash))) {
                serial_puts("Auth failed!\n");
                __SET_SIM_FAILED();
        }

do_boot:
        serial_puts("Boot next stage\n");

        boot_next_stage();
}
```

Fault Injection Reminder

- Intentionally cause errors in a system in order to compromise the security of the system
  - Modifications on the contents of memory, registers or executed instructions
- The attack may result in glitches following different models, we will consider two:
  - A fault/glitch causes the micro-controller to skip an instruction

```
beq 1008f8 ⟶ nop
```

  - Flipping a bit in the encoding of a specific instruction

```
cmp r3, #0 ⟶ cmp r3, \#1
```

NOTE: The instruction code can be changed too!

Software Countermeasures

MASTER IN CYBER–SECURITY | UNLOCK YOUR POTENTIAL

1. Redundant operations
2. Defensive coding
3. Timing countermeasures
4. Fault detection tests
5. Infective computing
6. Control flow integrity

UMU

Redundant Operations

The main purpose of performing redundant operations (e.g., double check signature) is to **force** an attacker to successfully perform **multiple successive glitches**

Defensive coding

Non-trivial constants:

Using “magic numbers” for constants, enums… (good mix of 1s and 0s, high hamming distances…) and “closed-on-fail” principles

Initialize to error values:

The variables before a check is initially set to an error value, avoiding some simple attacks by e.g. jumping instructions.

Error by default:

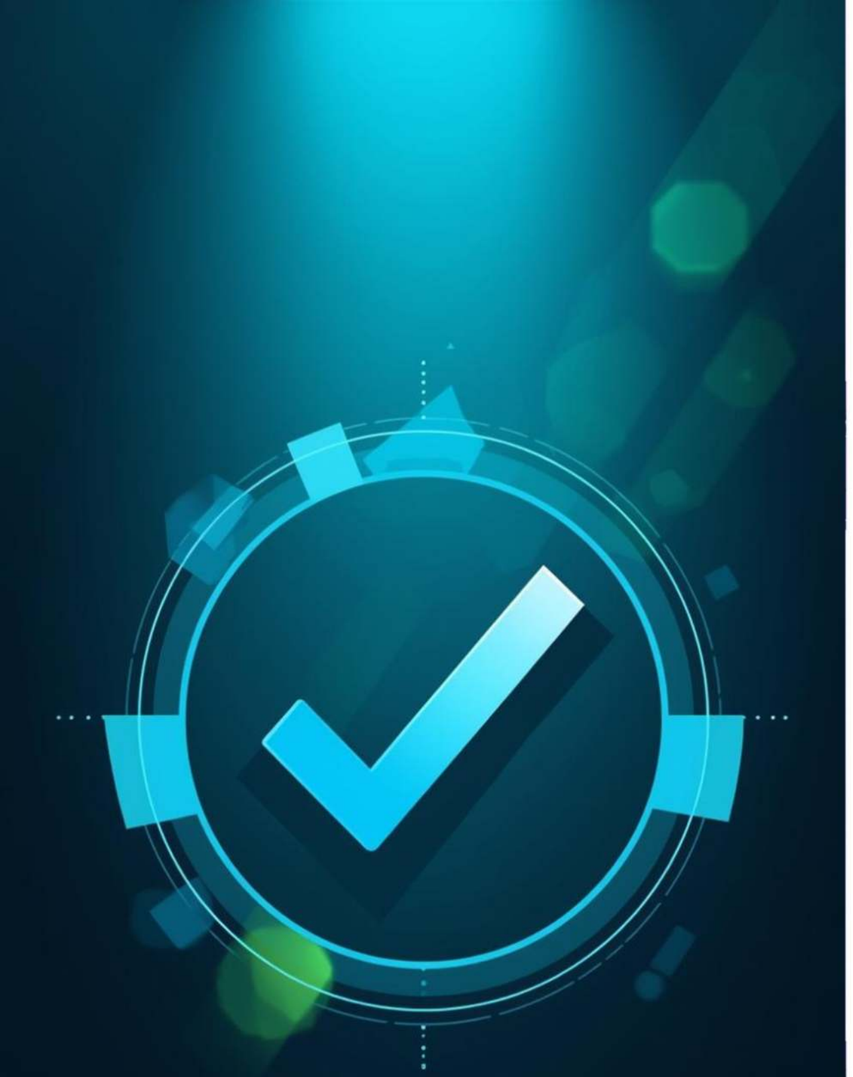
If an unexpected event happens, then a fault can be assumed and an error notified (default switch branch, “unreachable” branch…)

MASTER IN CYBER–SECURITY | UNLOCK YOUR POTENTIAL

Timing Countermeasures

Not a direct countermeasure to the fault injection effect.
Fault attacks are **timing critical**, so introducing randomized access, random delays… makes it more difficult to target the glitch

UMU



MASTER IN CYBER–SECURITY | UNLOCK YOUR POTENTIAL

Fault Detection Test

It attempts to detect fault injections by **verifying that some arithmetic invariants are respected**, and branch to return an error instead of the numerical result of the algorithm in case of invariant violation

UMU

Advanced Measures

Infective computing

Uses arithmetic invariants to compute a neutral element that is used to infect the result of the algorithm before returning it to make it unusable by the attacker

Control flow integrity:

With limited applicability, introduces checks that the correct execution flow of the program has occurred, e.g. through CRCs

UNIVERSIDAD DE MURCIA

Facultad de Informática

COOPERATIVE EXERCISE

GROUPING

Divided into 3 groups of 4 people. One person in each group will be the *expert* in one of the previous countermeasures (1,2,4 or 5)

EXPERTISE

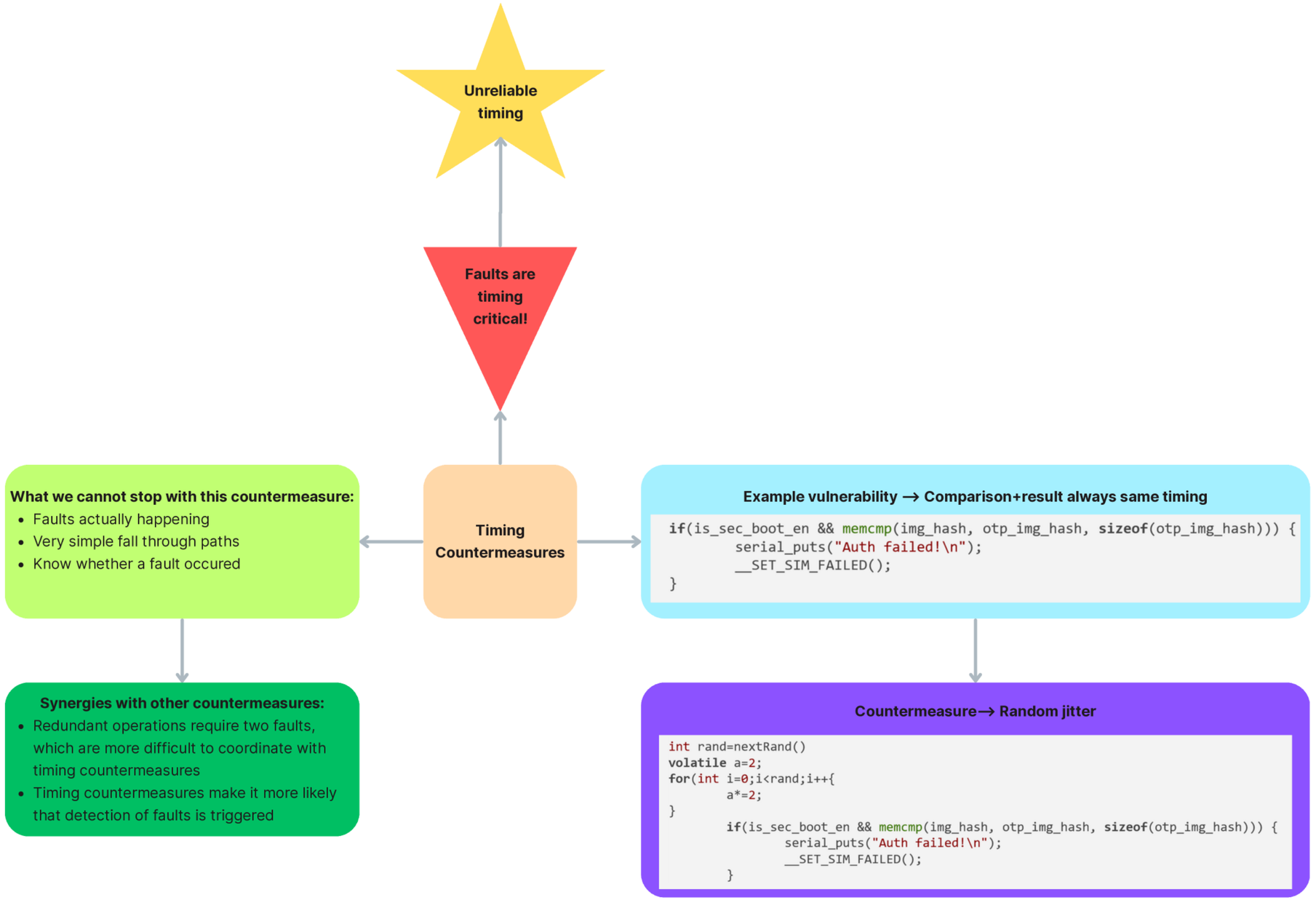
Experts of the same topic will group together to research and gain a good understanding of each topic

MENTAL MAPS

Each group of expert will create a mental map about their assigned countermeasure

RE-GROUPING

All students go back to their original groups, and explain their topic of expertise, bringing everything together


Unreliable timing
Faults are timing critical!
Timing Countermeasures
What we cannot stop with this countermeasure:
• Faults actually happening
• Very simple fall through paths
• Know whether a fault occured
Synergies with other countermeasures:
• Redundant operations require two faults, which are more difficult to coordinate with timing countermeasures
• Timing countermeasures make it more likely that detection of faults is triggered
Example vulnerability → Comparison+result always same timing
if(is_sec_boot_en && memcmp(img_hash, otp_img_hash, sizeof(otp_img_hash))) {
serial_puts("Auth failed!\n");
__SET_SIM_FAILED();
}
Countermeasure→ Random jitter
int rand=nextRand()
volatile a=2;
for(int i=0;i<rand;i++{
a*=2;
}
if(is_sec_boot_en && memcmp(img_hash, otp_img_hash, sizeof(otp_img_hash))) {
serial_puts("Auth failed!\n");
__SET_SIM_FAILED();
}