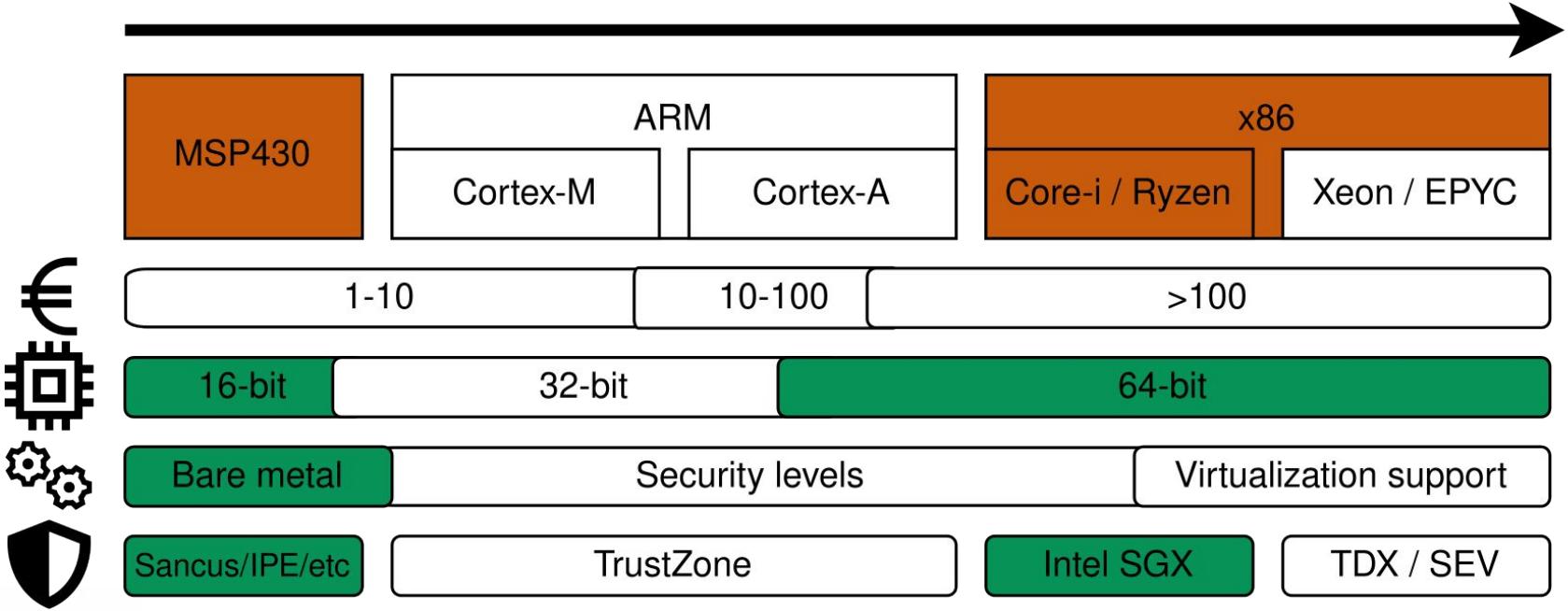
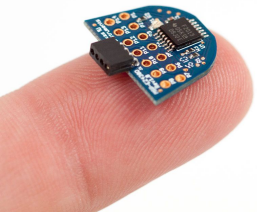
# **Wait a Cycle**: Eroding Cryptographic Trust in Low-End TEEs via Timing Side Channels

**Ruben Van Dijck, Marton Bognar, Jo Van Bulck**
*DistriNet, KU Leuven, Belgium*
**@ SysTEX (July 4th, 2025)**

DistriNet
KU LEUVEN

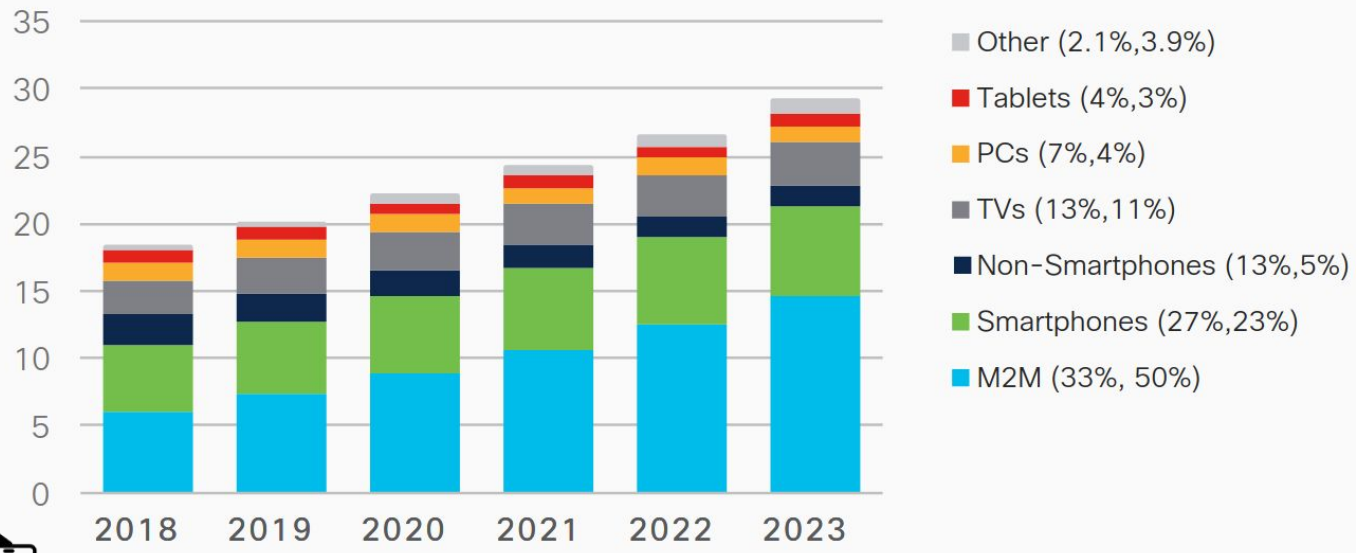
# Computing spectrum: “Low-end” vs. “high-end”



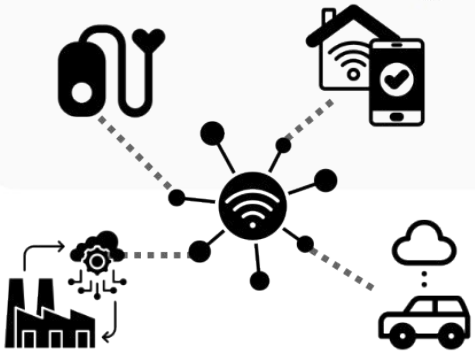



# Context: Growth of the Internet of Things (IoT)

10% CAGR
2018–2023




* Figures (n) refer to 2018, 2023 device share



# Side-channel threats?

“**Compared to higher-end** MMU-based systems, Sancus can be considered **less susceptible to [side-channel] threats** considering the **elementary design** of its security extensions, as well as the underlying processor.”

# Side-channel threats?

"**Compared to higher-end** MMU-based systems, Sancus can be considered **less susceptible to [side-channel] threats** considering the **elementary design** of its security extensions, as well as the underlying processor."

"Given the kind of **small microprocessors** that we target, many side-channels such as cache timing attacks or page fault channels are **not applicable**."

# Focus: timing attacks

- Start-to-end timing: `memcmp` on VRASED (remote attestation)

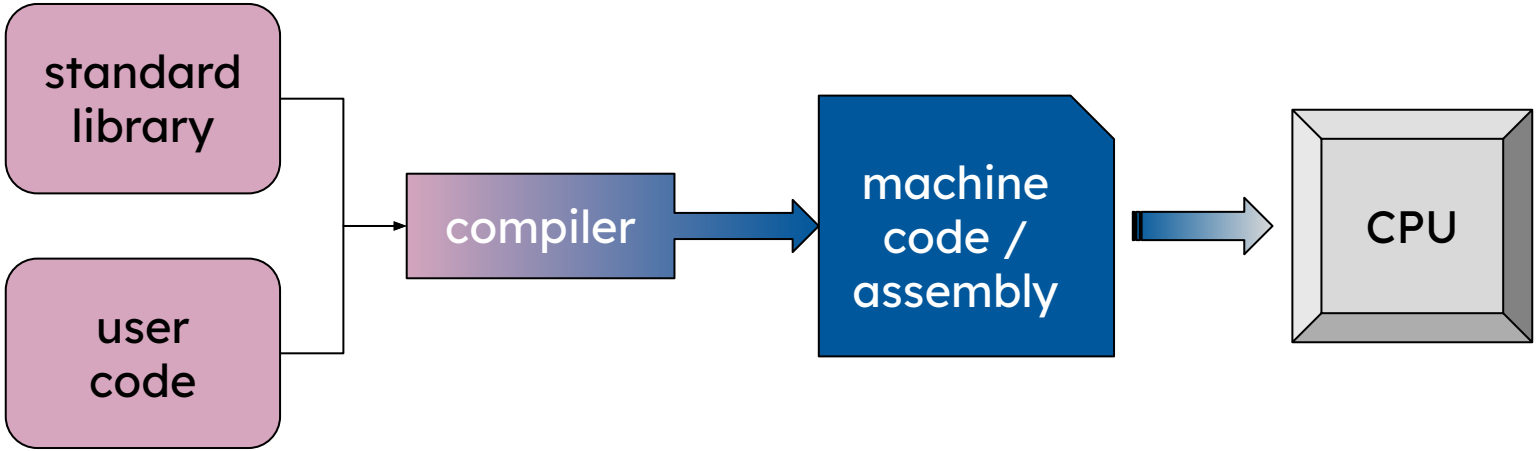
TABLE III. Execution time of $\text{VRASED}_{\text{A}}$ for authentication guesses.

| VRF_AUTH[32] | Execution time (cycles) |
|---|---|
| {0x1} | 210,641 |
| {0x0} | 210,641 |
| {0x59} | 210,654 |
| {0x59, 0x75} | 210,654 |
| {0x59, 0x76} | 210,667 |

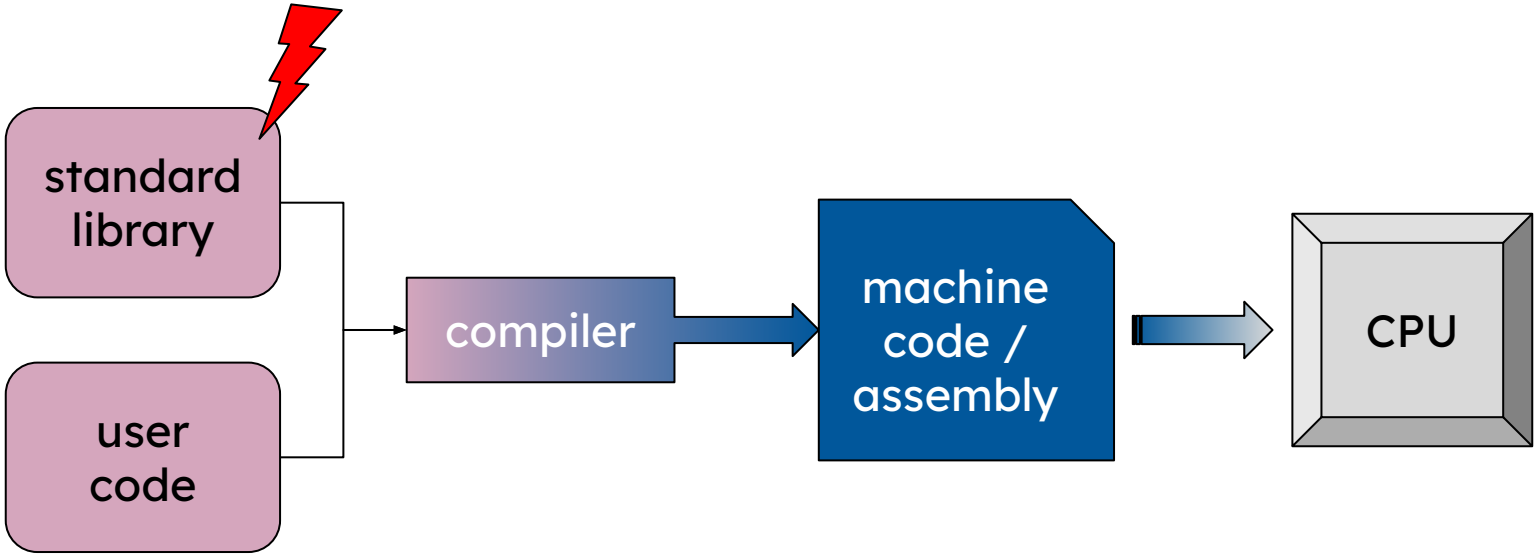
- More elaborate threats: interrupt latency, other contention attacks
- Precise timers!

*Marton Bognar, Jo Van Bulck, Frank Piessens, "Mind the Gap: Studying the Insecurity of Provably Secure Embedded Trusted Execution Architectures", S&P '22.*

# Embedded systems



# Embedded systems: timing leakage



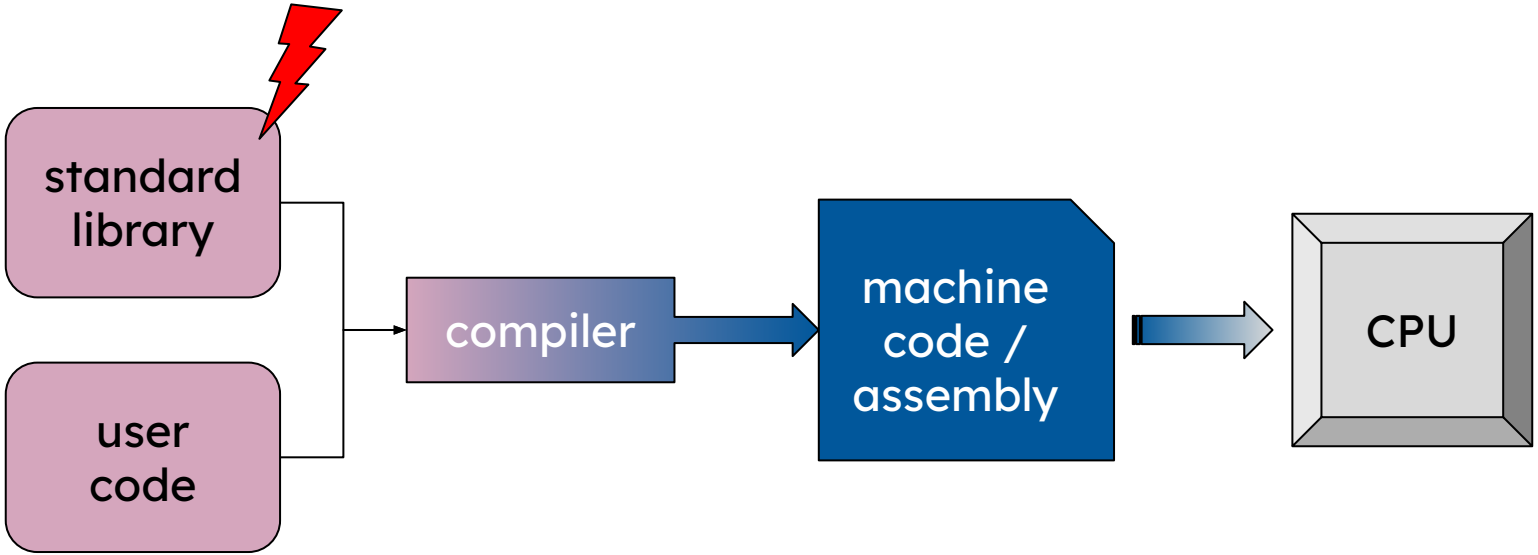
# Standard library functions

- Authentic Execution C++ library
- VatiCAN authenticated automotive bus protocol

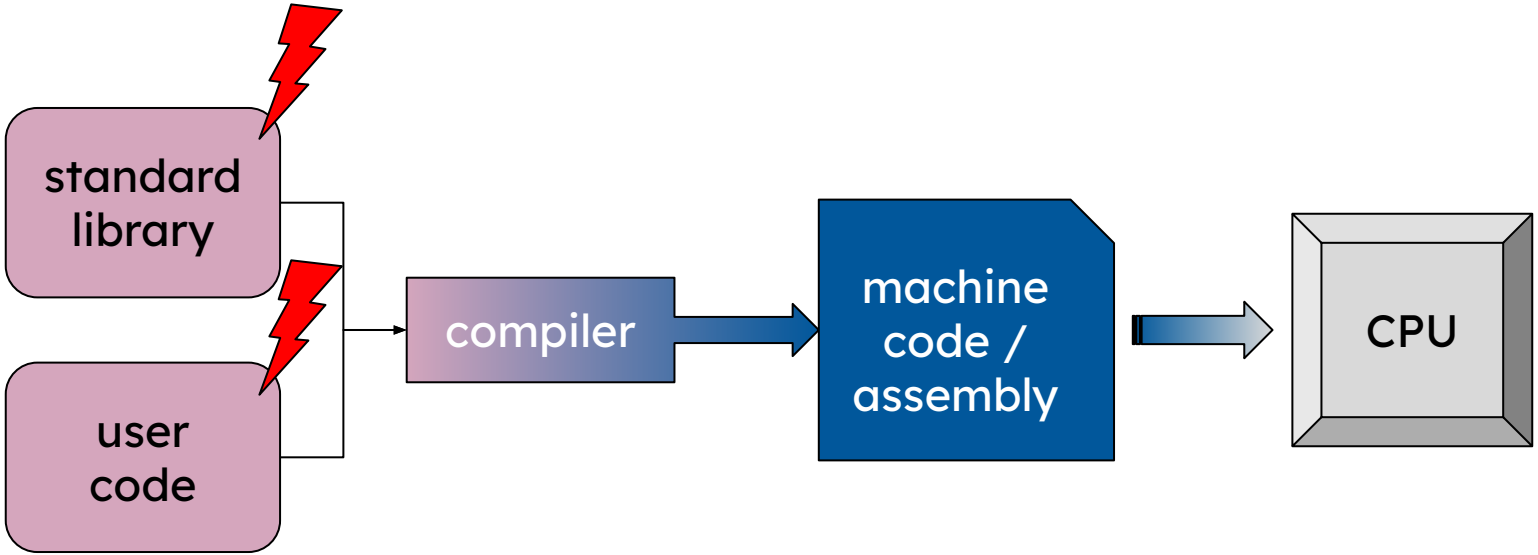
```
if (!std::equal(std::begin(tag), std::end(tag), expectedTag))
    return BAD_TAG;
```

```
if (memcmp(ch->MAC, _tempBuffer, 8) == 0) {
    res = MAC_OK;
    ch->RemoteCounter++;
} else {
    res = MAC_WRONG;
}
```

# Embedded systems: timing leakage



# Embedded systems: timing leakage

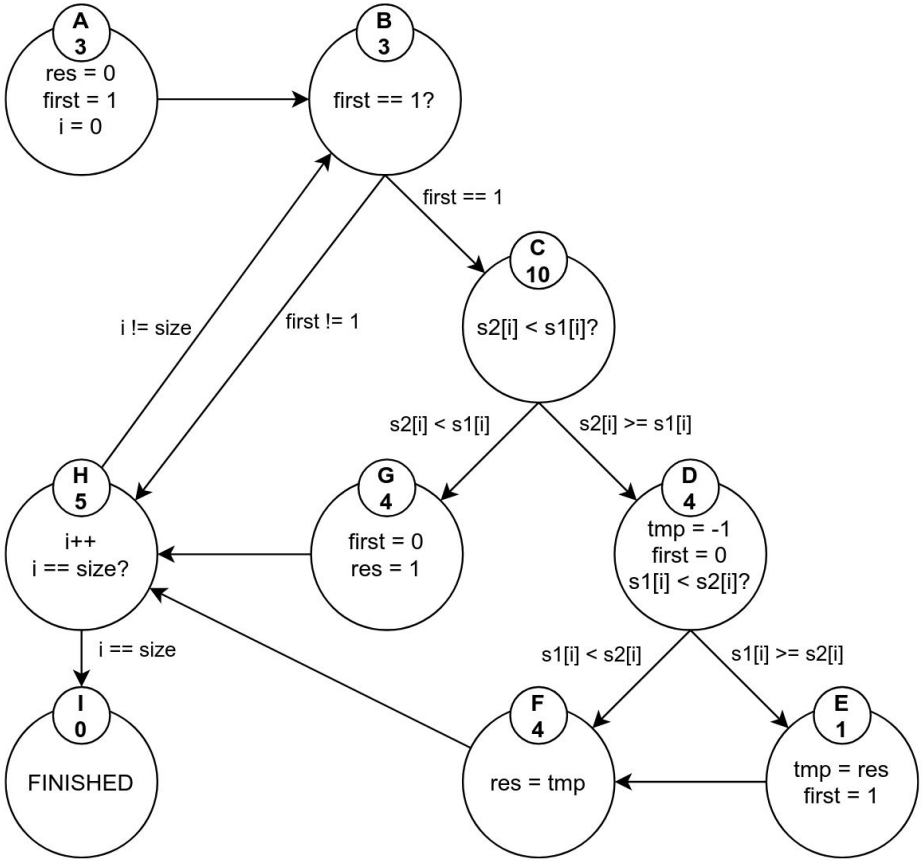


# User code

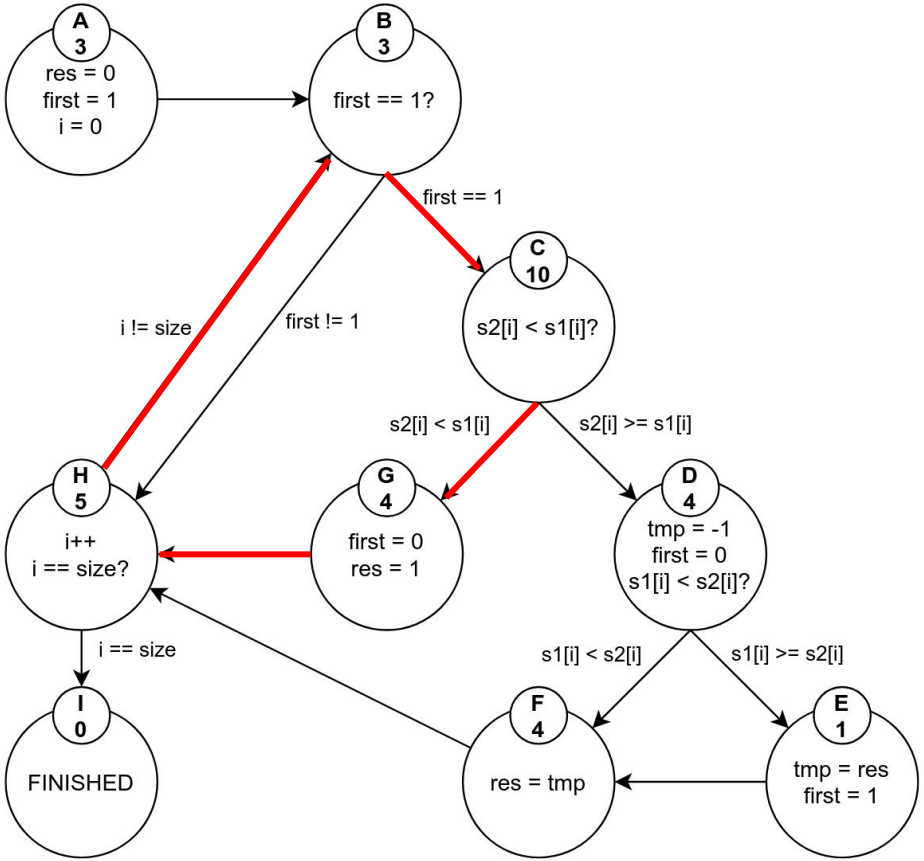
- In newer VRASED-based systems: secure_memcmp

```
__attribute__ ((section (".do_mac.body"))) int secure_memcmp(const uint8_t* s1, const uint8_t* s2, int size) {
    int res = 0;
    int first = 1;
    for(int i = 0; i < size; i++) {
      if (first == 1 && s1[i] > s2[i]) {
        res = 1;
        first = 0;
      }
      else if (first == 1 && s1[i] < s2[i]) {
        res = -1;
        first = 0;
      }
    }
    return res;
}
```

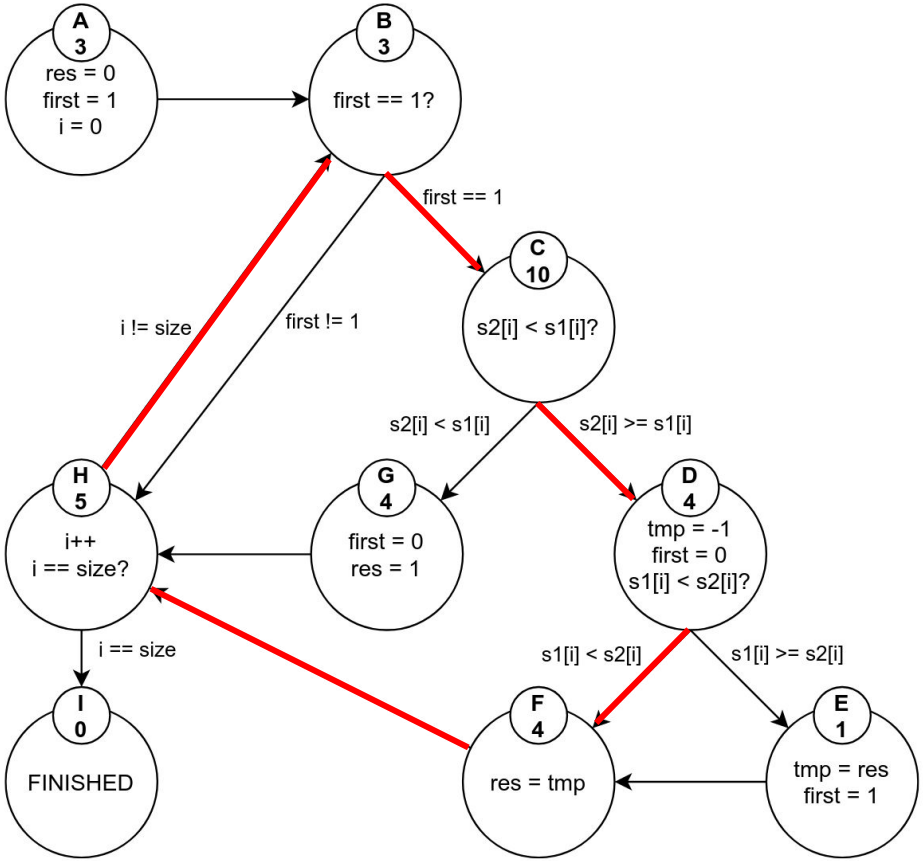
# User code: secure_memcmp



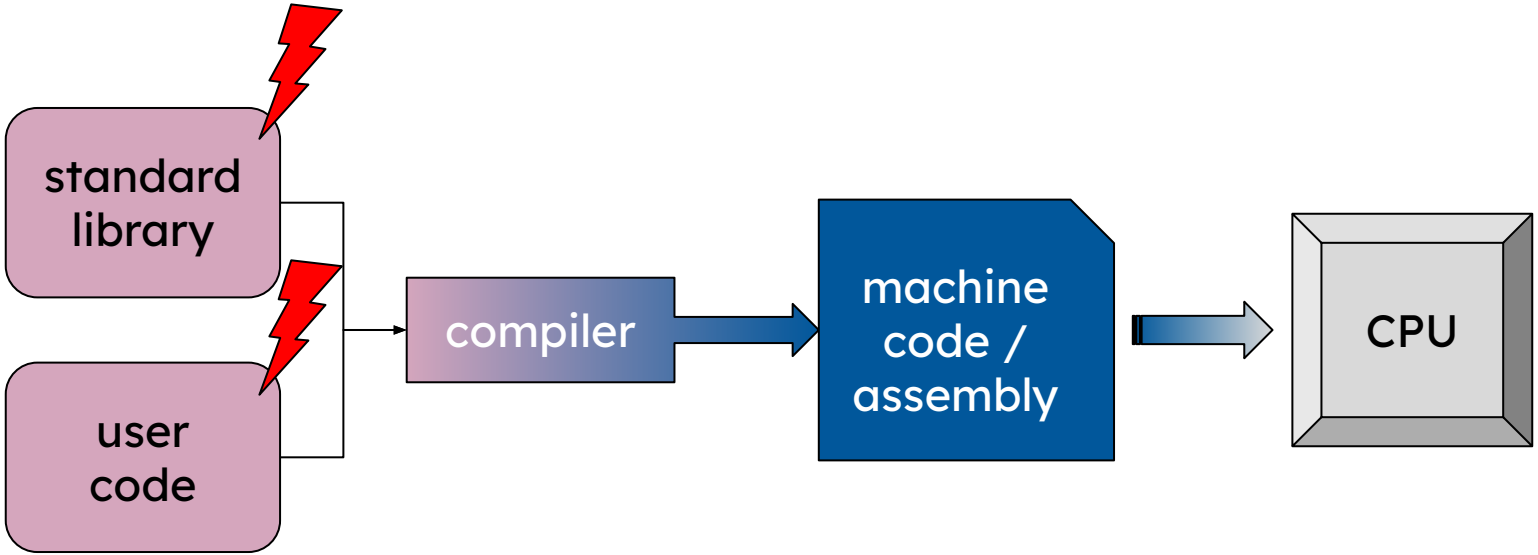
# User code: secure_memcmp



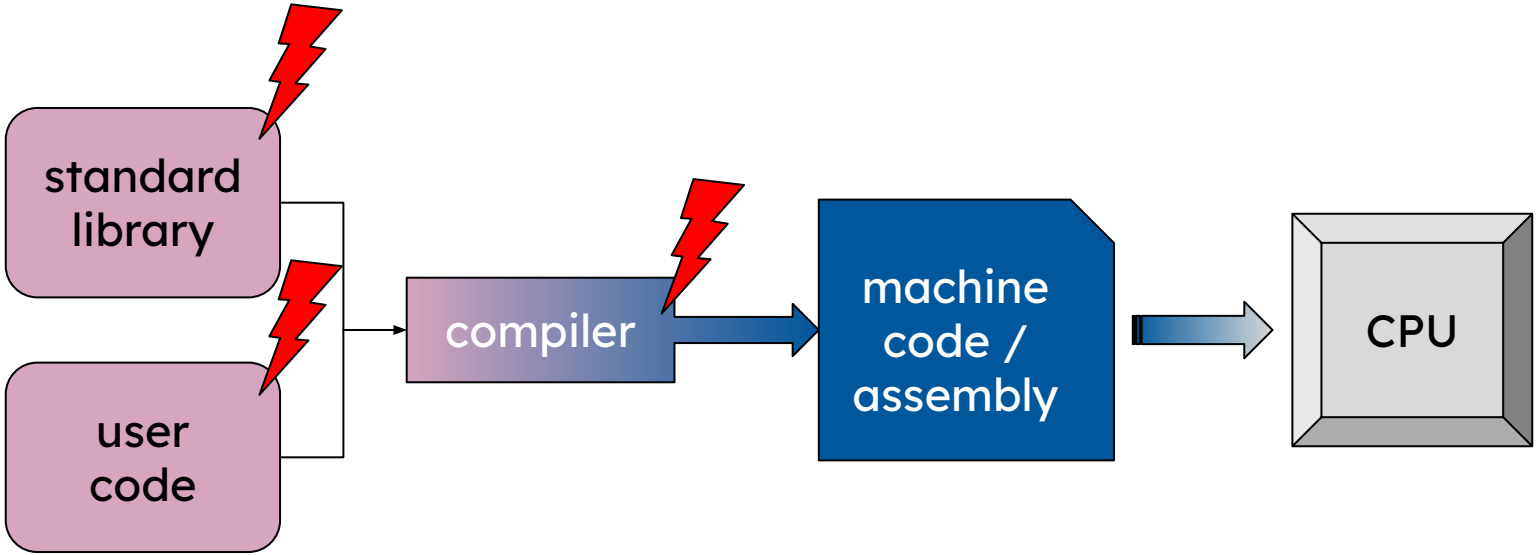
# User code: secure_memcmp



# Embedded systems: timing leakage



# Embedded systems: timing leakage



# What is the problem in this code?

```
/* 2. authenticated connection ? calculate and verify MAC */
if (vatican_mac_create(mac_me.bytes, *id, buf, rv) >= 0)
{
    recv_len = vatican_receive(ican, &id_recv, mac_recv.bytes, /*block=*/1);
    fail = (id_recv != *id + 1) || (recv_len != CAN_PAYLOAD_SIZE) ||
           (mac_me.quad != mac_recv.quad);
}
```

# Compiler-introduced timing leakage

```
void function(uint64_t mac, uint64_t guess) {
    bool match = mac != guess;
    …
}
```

# Compiler-introduced timing leakage

```
void function(uint64_t mac, uint64_t guess) {
    bool match = mac != guess;
    …
}
```

```
cmp.w    6(r1), r12
jne      .L1
cmp.w    r9, r13
jne      .L1
cmp.w    r10, r14
jne      .L1
cmp.w    r11, r15
jne      .L1
```

# Compiler-introduced timing leakage

```
void function(uint64_t mac, uint64_t guess) {
    bool match = mac != guess;
    …
}
```

```
cmp.w    6(r1), r12
jne      .L1
cmp.w    r9, r13
jne      .L1
cmp.w    r10, r14
jne      .L1
cmp.w    r11, r15
jne      .L1
```

| Compiler | Word size | uint16_t | uint32_t | uint64_t |
|---|---|---|---|---|
| MSP430 `gcc` v14.2.0 | 16 | | ✖ | ✖ |
| `sancus-cc` (LLVM v4.0.1) | 16 | | | |
| RISC-V `gcc` v14.2.0 | 32 | | | ✖ |
| MIPS (el) `gcc` v14.2.0 | 32 | | | ✖ |
| x86 MSVC v19 | 32 | | | ✖ |

# Does this matter?

Brute-forcing a 64-bit tag at 1000 cycles/guess (@ 16 MHz):

**36.6 million years**

# Does this matter?

Brute-forcing 4 * 16-bit tags at 1000 cycles/guess:

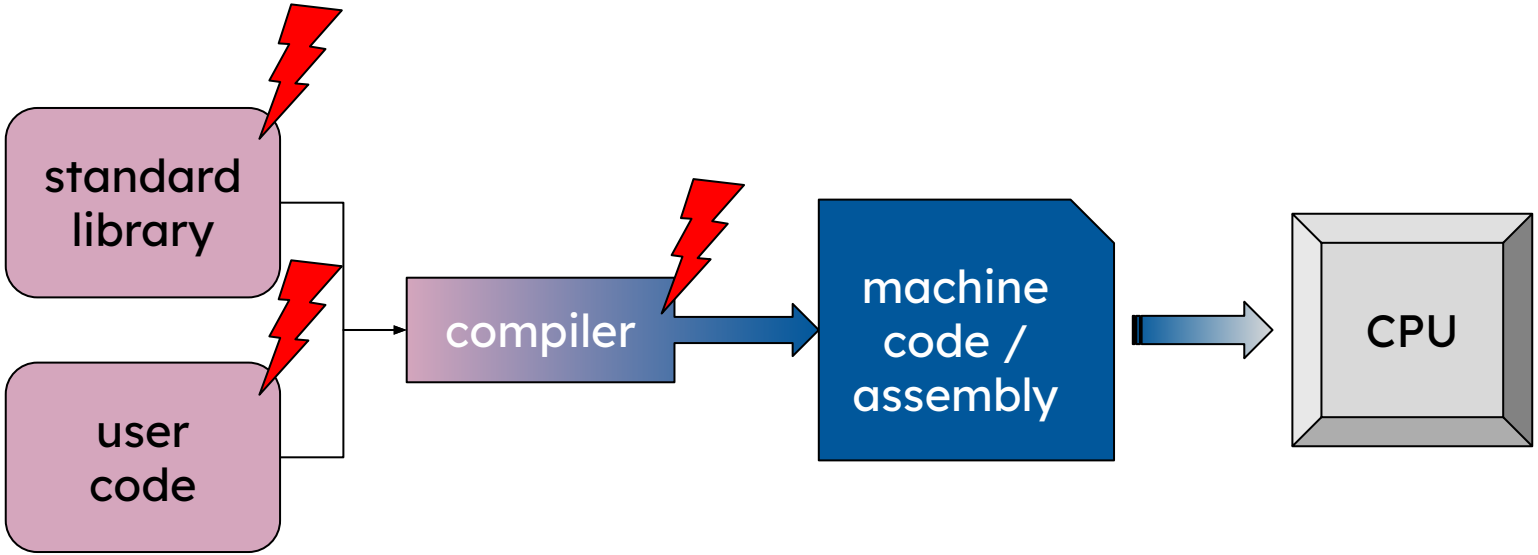
**16.4**

# Does this matter?

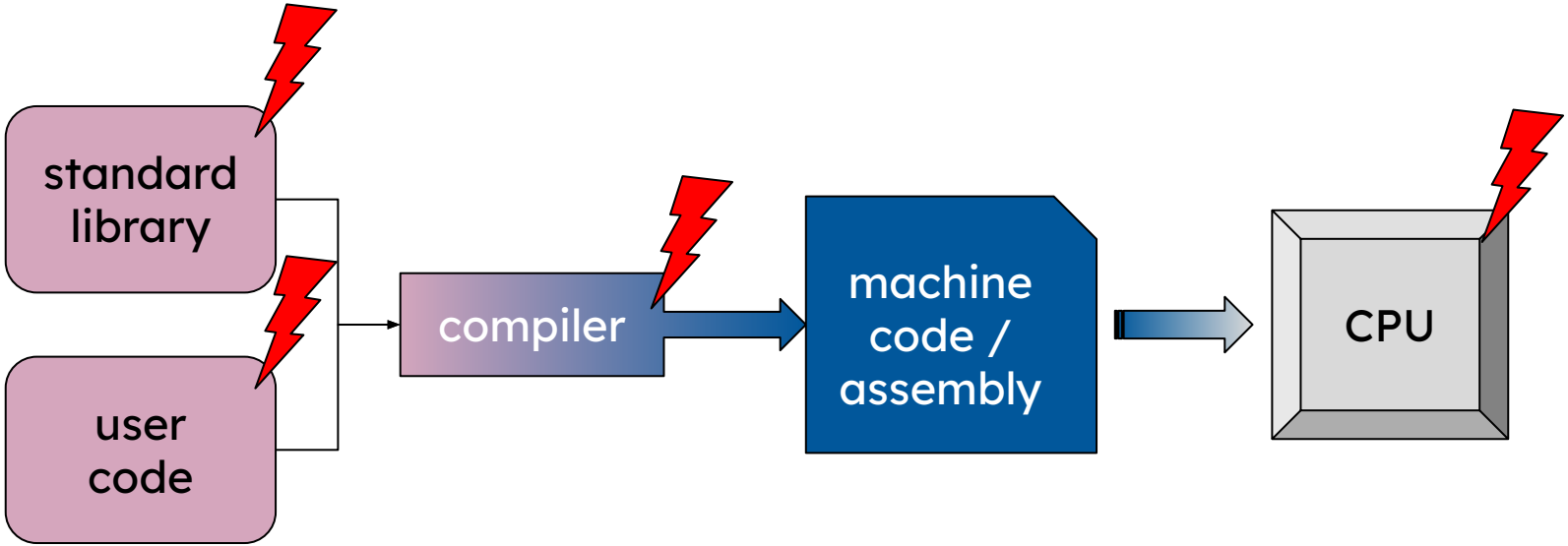
Brute-forcing 4 * 16-bit tags at 1000 cycles/guess:

**16.4 seconds**

# Embedded systems: timing leakage

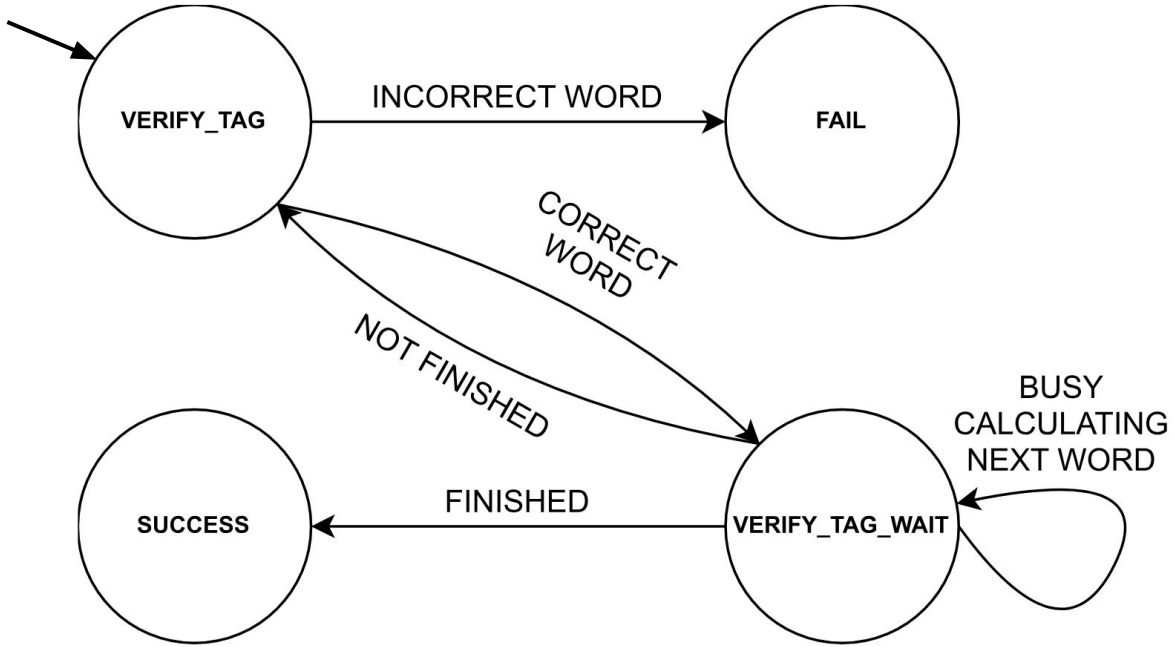


# Embedded systems: timing leakage



# Timing leakage in the hardware design

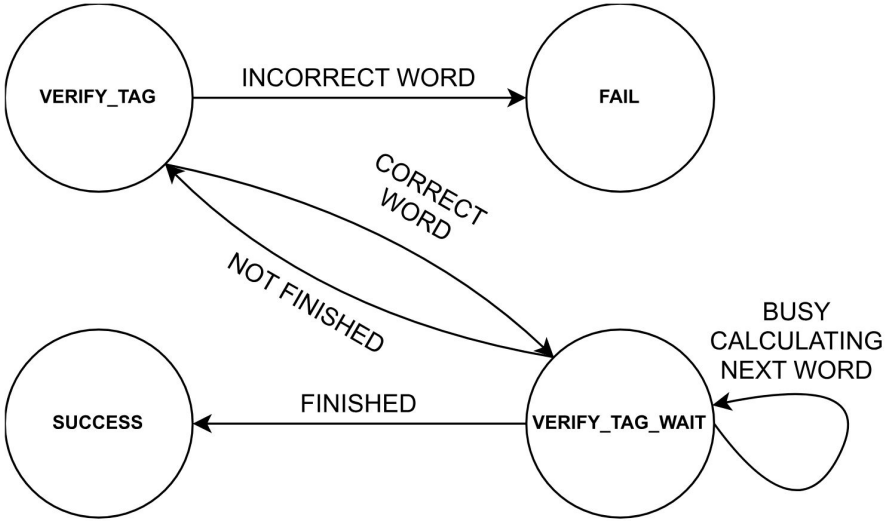
- Sancus features a cryptographic engine
  - Tag comparison for authenticated encryption

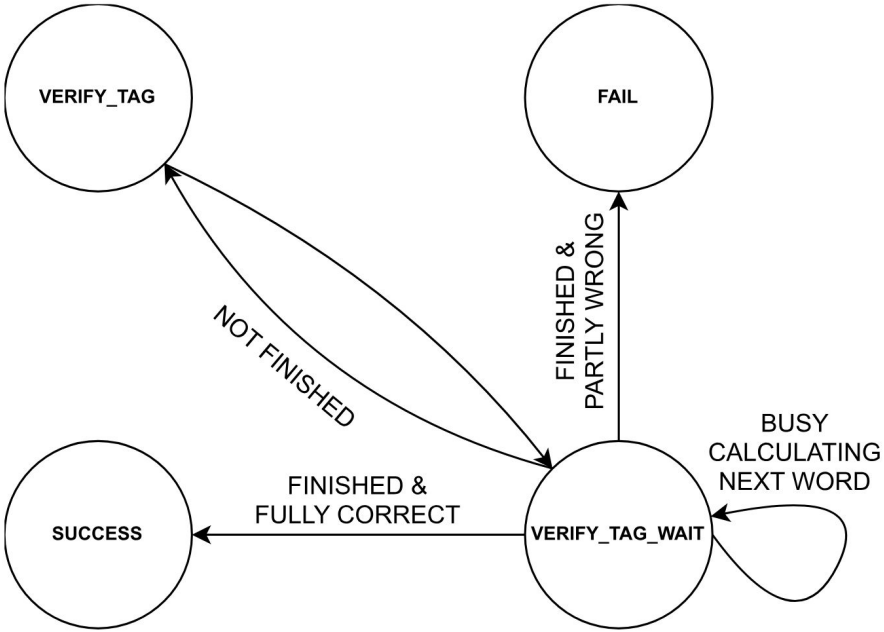


# Mitigations

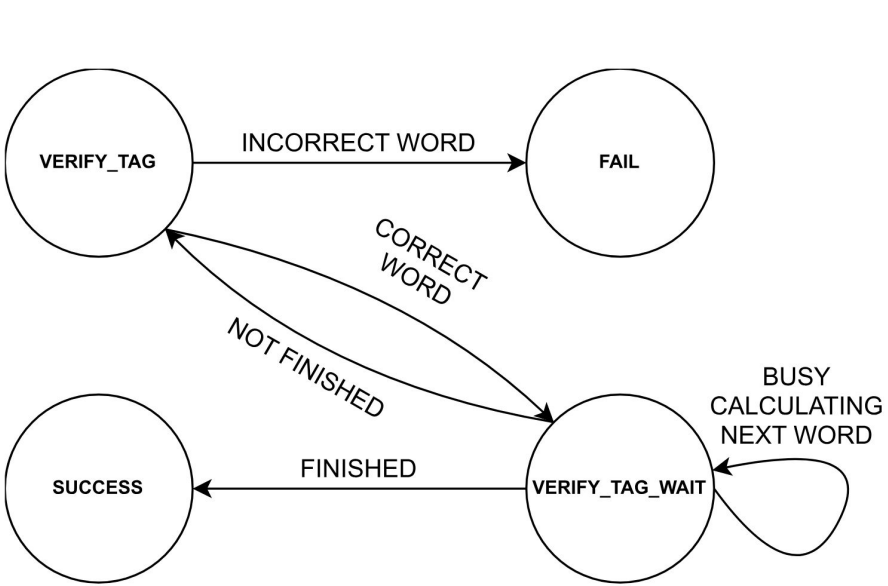
- *Library:* use **constant-time** functions
- *Compiler:* use **binary analysis** to validate security
- *Hardware:* error-prone software mitigation
- *Cryptographic unit*: straightforward state machine fix

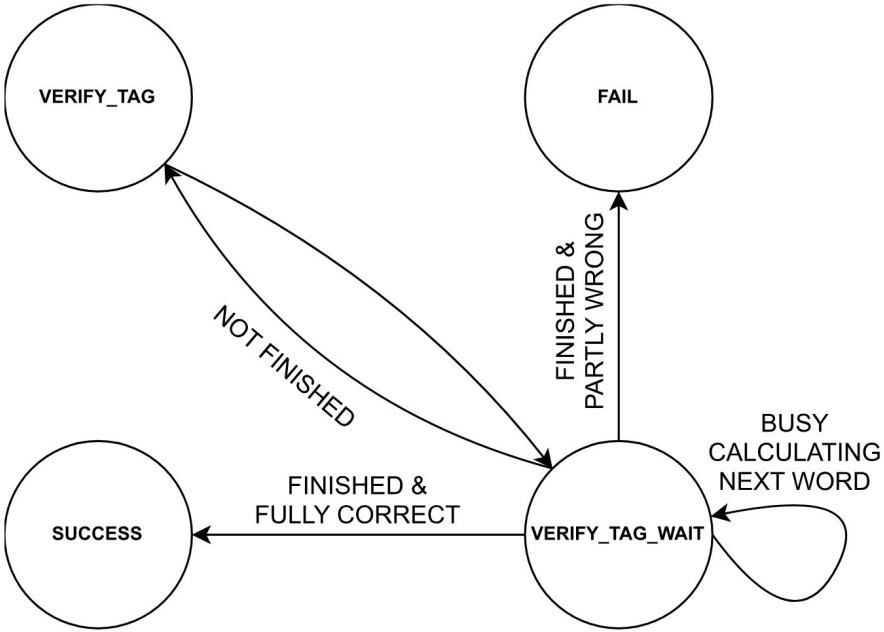
# Mitigation with extra state register



# Mitigation with extra state register




| | *Total* | | *Crypto unit* | |
|---|---|---|---|---|
| **Architecture** | **LUT** | **FF** | **LUT** | **FF** |
| Original | 5,427 | 2,240 | 1,436 | 592 |
| Extra register | 5,407 | 2,241 | 1,414 | 593 |
| Extra states | 5,457 | 2,240 | 1,482 | 592 |

# Conclusion

## Wait a Cycle: Eroding Cryptographic Trust in Low-End TEEs via Timing Side Channels

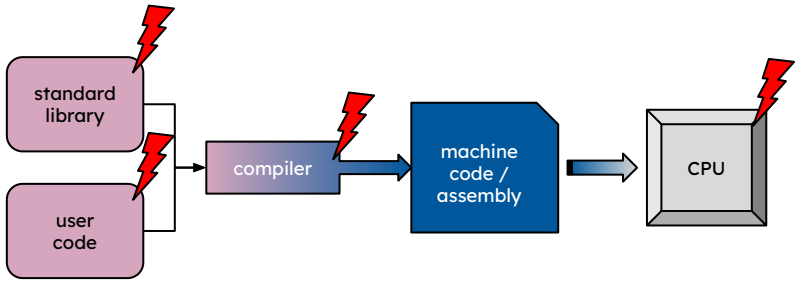
CI passing

SysTEX'25 Artifact Evaluated Available

SysTEX'25 Artifact Evaluated Functional

SysTEX'25 Artifact Evaluated Reusable

https://github.com/dnet-tee/wait-a-cycle




| System | Library | == operator | Hardware |
|---|---|---|---|
| VRASED+, RATA, ACFA, TRAIN | ✖ | | |
| VatiCAN | ✖ | | |
| LEIA | | ✖ | |
| VulCAN | | ✖ | |
| Sancus, Authentic Execution | ✖ | | ✖ |