
UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR IT SECURITY

# Examining Control Flow Leakage Attacks on TEEs

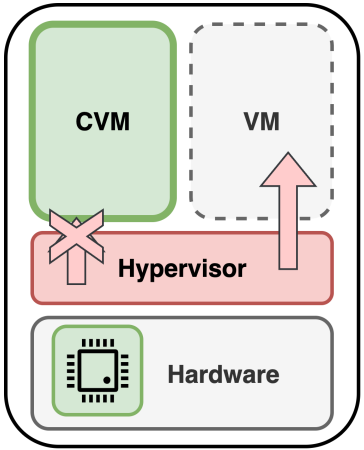
Luca Wilke*, Florian Sieck* and Thomas Eisenbarth
(*equal contribution)

Intel Product Assurance and Security - Tech Sharing - November 5, 2024

# Overview

1. Attacks on TDX
2. Finding Control Flow Leakages
3. Countermeasures

Goal: Remove cloud provider from TCB

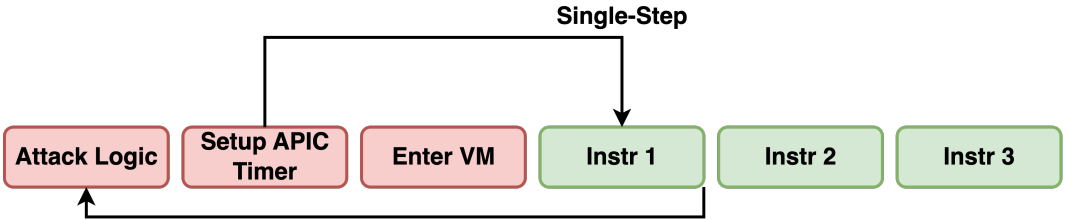



- AMD SEV-SNP
- Intel TDX
- (ARM CCA)

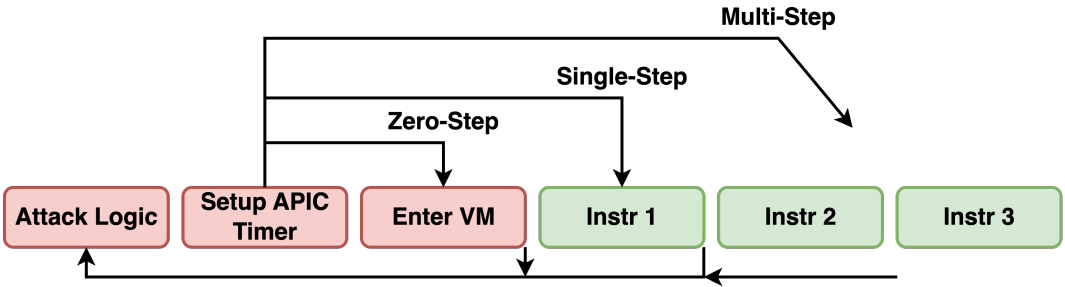
Available in the wild on

# Single-Stepping Attacks

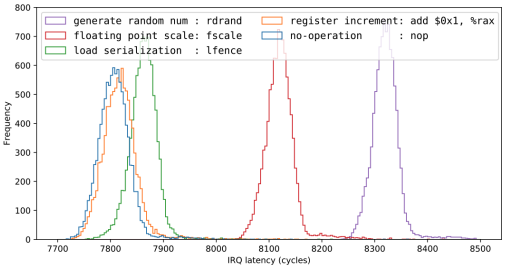
# Single-Stepping



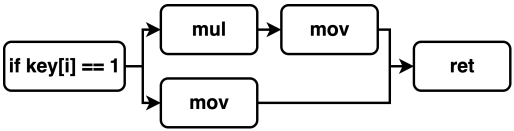
# Single-Stepping



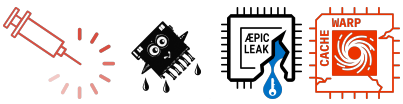
# Single-Stepping: The bane of TEEs



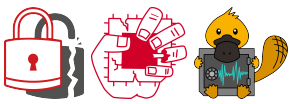

Interrupt Latency Attacks




Instruction Counting Attacks



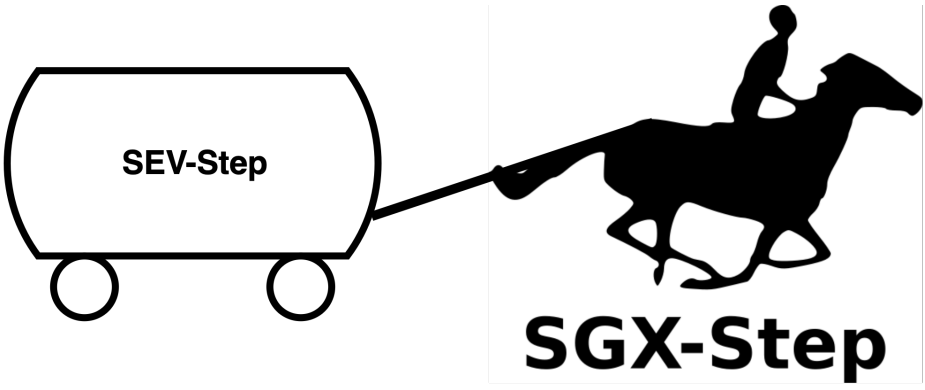

Amplifier



Zero-Stepping Attacks

# History of Single-Stepping

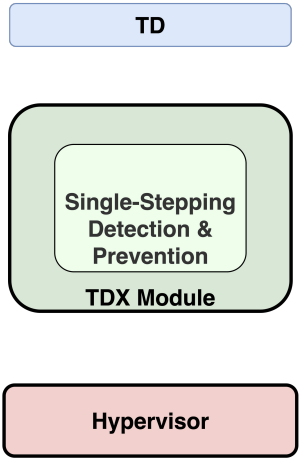
# History of Single-Stepping



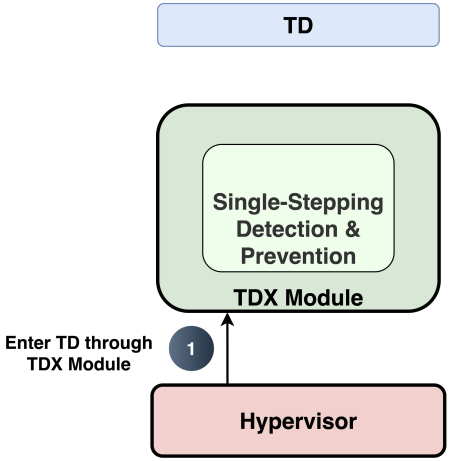
# History of Single-Stepping

# Countermeasure in TDX

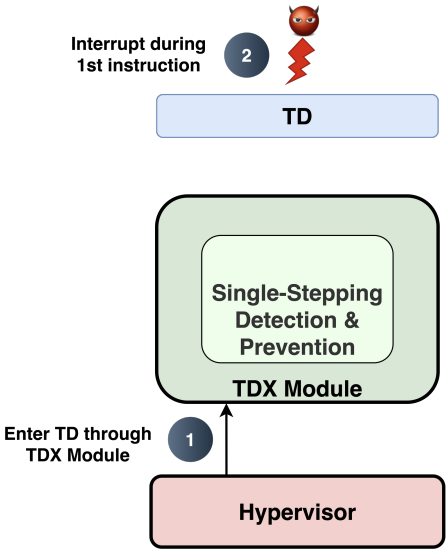
# Single-Stepping Countermeasure



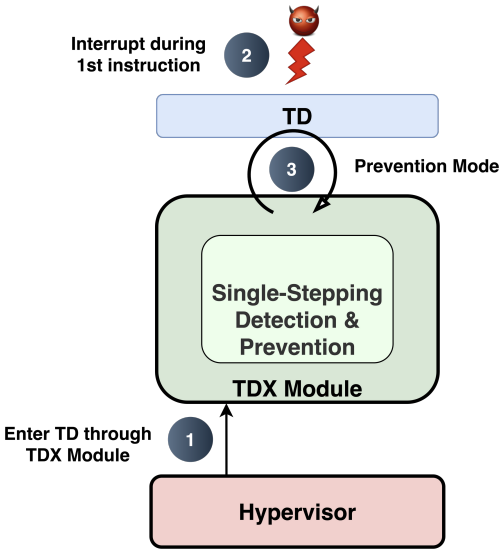
# Single-Stepping Countermeasure



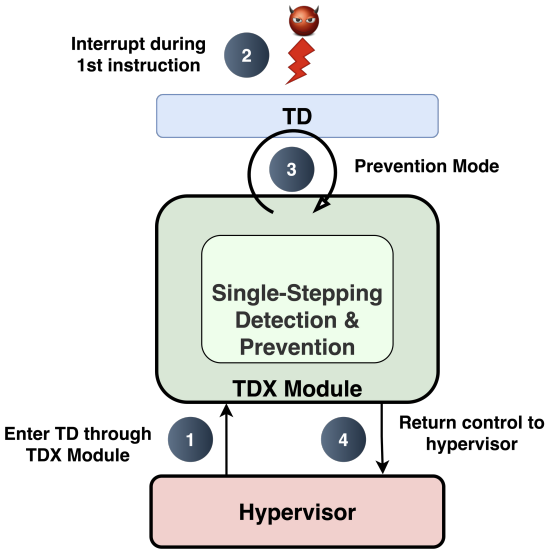
# Single-Stepping Countermeasure



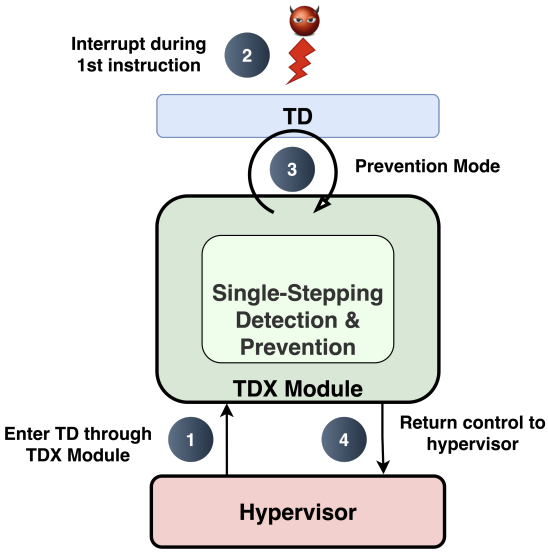
# Single-Stepping Countermeasure



# Single-Stepping Countermeasure



# Single-Stepping Countermeasure



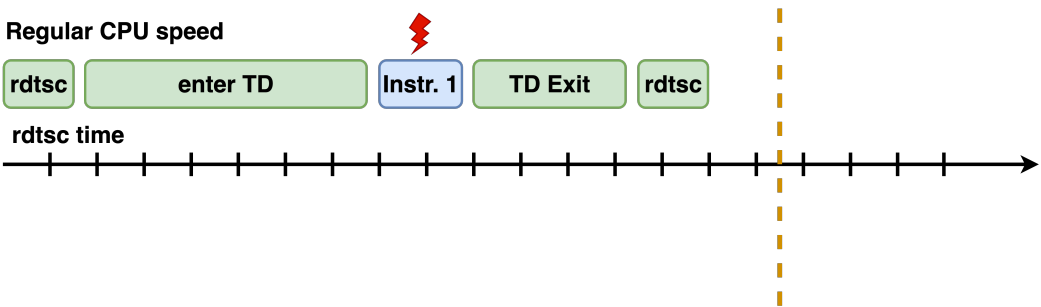

## Two attacks

1. Single-Stepping
2. StumbleStepping

# Single-Stepping TDX

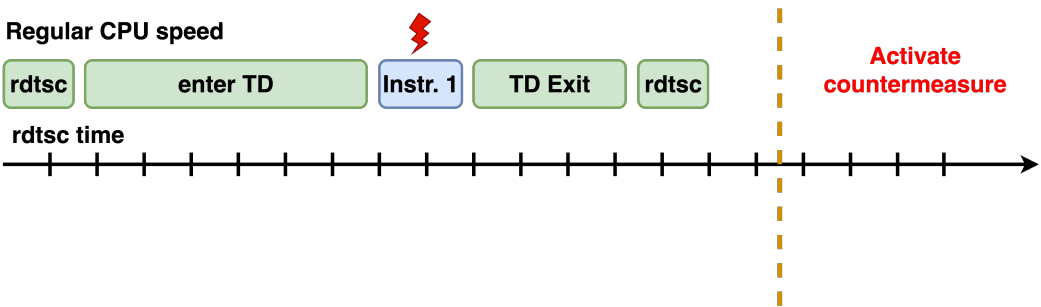
# Re-enable Single-Stepping

**Classified as benign if : "*> 2 Instructions*" OR "*> THRESH cycles*"**



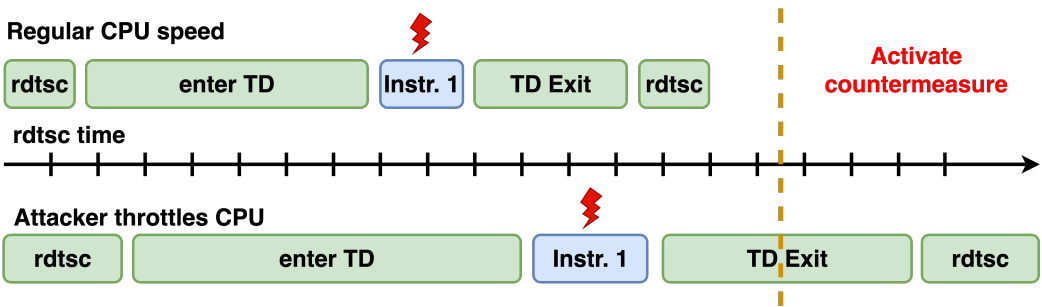
# Re-enable Single-Stepping

**Classified as benign if : "*> 2 Instructions*" OR "*> THRESH cycles*"**



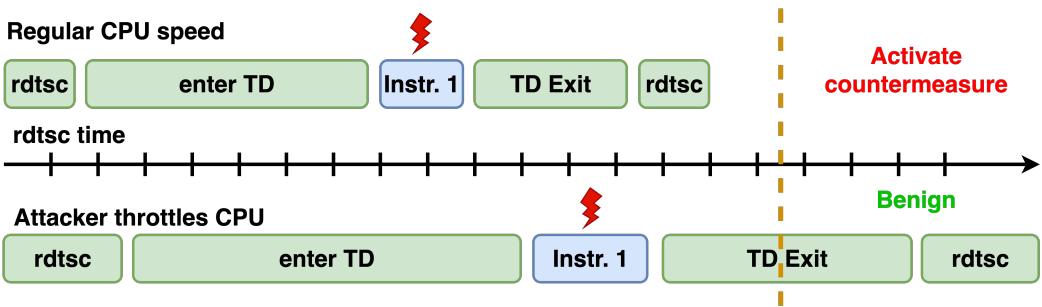
# Re-enable Single-Stepping

**Classified as benign if : "> *2 Instructions*" OR "> *THRESH cycles*"**



# Re-enable Single-Stepping

**Classified as benign if : "> *2 Instructions*" OR "> *THRESH cycles*"**



# Filtering Zero-Steps

- SGX-Step: Check if “Accessed”-bit is set

# Filtering Zero-Steps

- SGX-Step: Check if “Accessed”-bit is set ⇒ 2nd level page tables inaccessible with TDX

# Filtering Zero-Steps

- SGX-Step: Check if “Accessed”-bit is set ⇒ 2nd level page tables inaccessible with TDX
- SEV-Step: Performance monitoring counters

# Filtering Zero-Steps

- SGX-Step: Check if “Accessed”-bit is set ⇒ 2nd level page tables inaccessible with TDX
- SEV-Step: Performance monitoring counters ⇒ isolated by TDX

# Filtering Zero-Steps

- SGX-Step: Check if “Accessed”-bit is set ⇒ 2nd level page tables inaccessible with TDX
- SEV-Step: Performance monitoring counters ⇒ isolated by TDX
- Cache attack on TD code page: works

# Interlude: Zero-Stepping Attacks

# Timer-based Zero-Stepping Attacks

## Idea

1. Repeatedly trigger context switch without state change
2. Leak information from context switch
3. Example: RAPL software power measurements

# Timer-based Zero-Stepping Attacks



## Idea

1. Repeatedly trigger context switch without state change
2. Leak information from context switch
3. Example: RAPL software power measurements

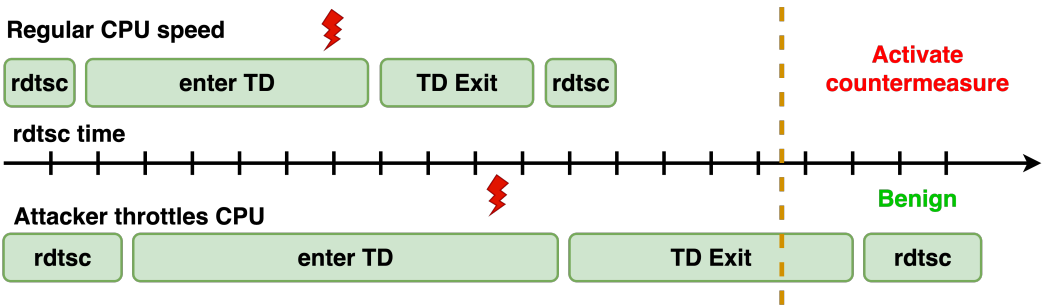
# Timer-based Zero-Stepping Attacks



## Idea

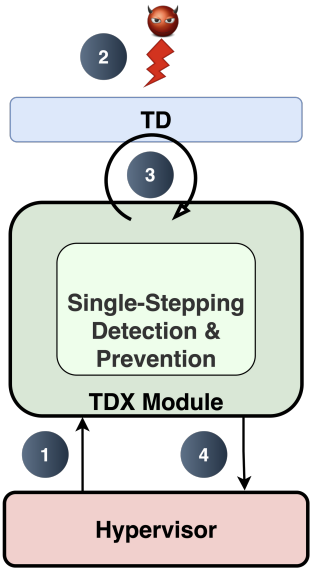
1. Repeatedly trigger context switch without state change
2. Leak information from context switch
3. Example: RAPL software power measurements

**Classified as benign if : "*> 2 Instructions*" OR "*> THRESH cycles*"**

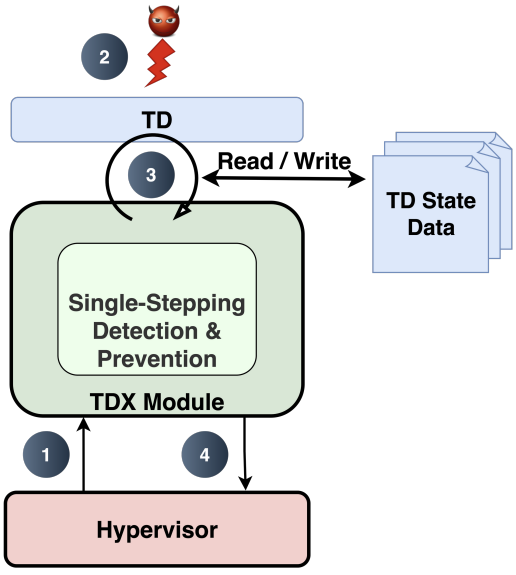


# StumbleStepping TDX

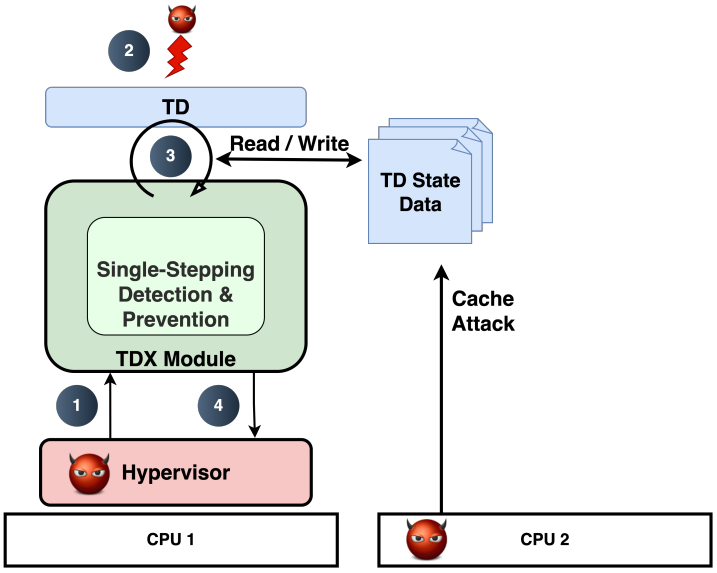
# StumbleStepping



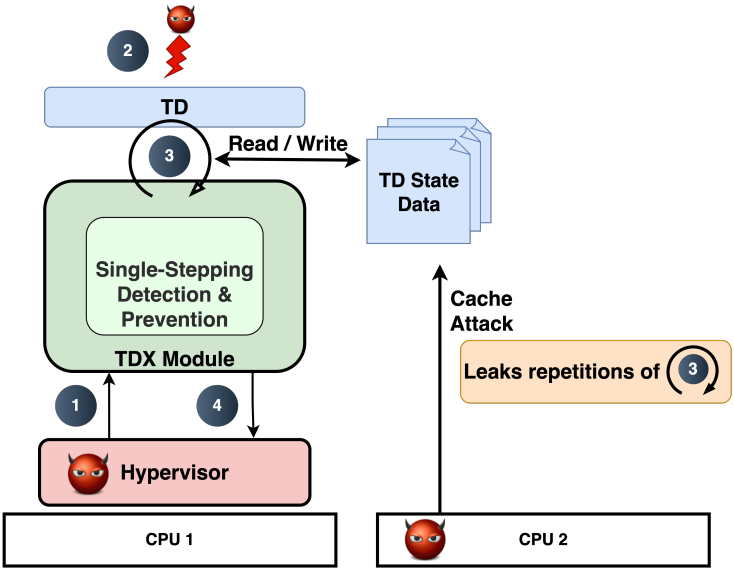
# StumbleStepping



# StumbleStepping



# StumbleStepping



# StumbleStepping Details

1. StumbleStepping needs high frequency cache observations
2. Regular *Flush+Reload* blocked by MKTME
3. But also: MKTME coherency mechanism enables KeyID-based *Flush+Reload*

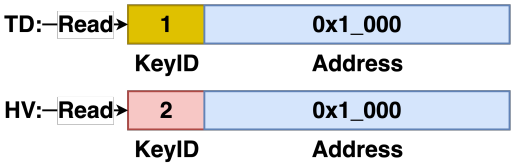
# StumbleStepping Details

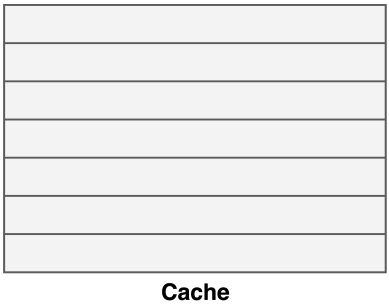
1. StumbleStepping needs high frequency cache observations
2. Regular *Flush+Reload* blocked by MKTME
3. But also: MKTME coherency mechanism enables KeyID-based *Flush+Reload*

# StumbleStepping Details

1. StumbleStepping needs high frequency cache observations
2. Regular *Flush+Reload* blocked by MKTME
3. But also: MKTME coherency mechanism enables KeyID-based *Flush+Reload*

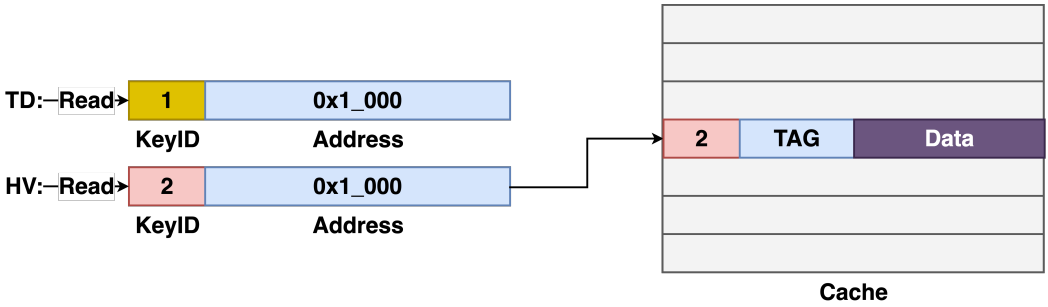
# Exploiting MKTME's coherency mechanism



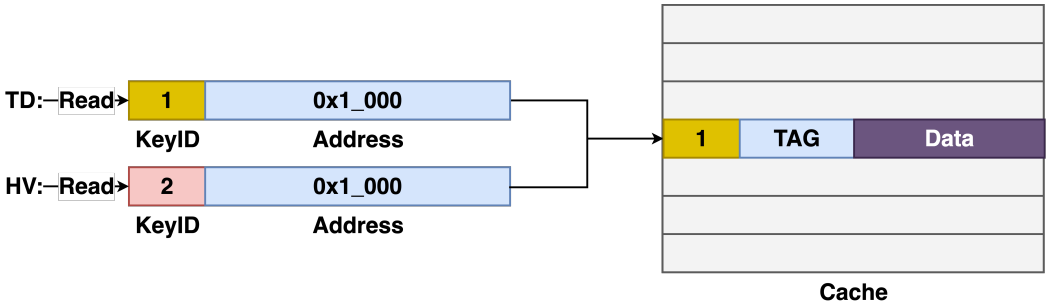

KeyID based coherency mechanism for flushing cache lines

# Exploiting MKTME's coherency mechanism



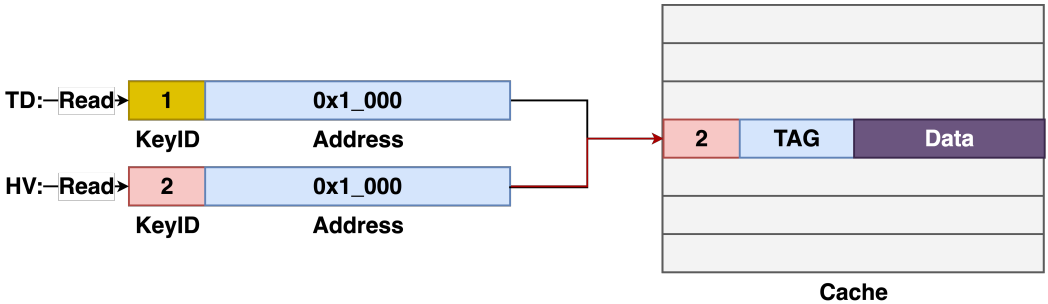

KeyID based coherency mechanism for flushing cache lines

# Exploiting MKTME's coherency mechanism




KeyID based coherency mechanism for flushing cache lines

# Exploiting MKTME's coherency mechanism




KeyID based coherency mechanism for flushing cache lines

# StumbleStepping Details

- Cache attack ✓

# StumbleStepping Details

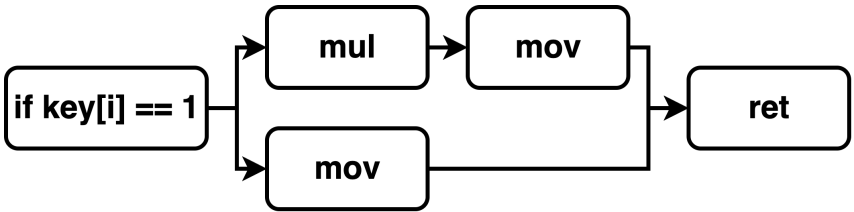
- Cache attack ✓
  - Frequency throttling improves temporal resolution even more

# StumbleStepping Details

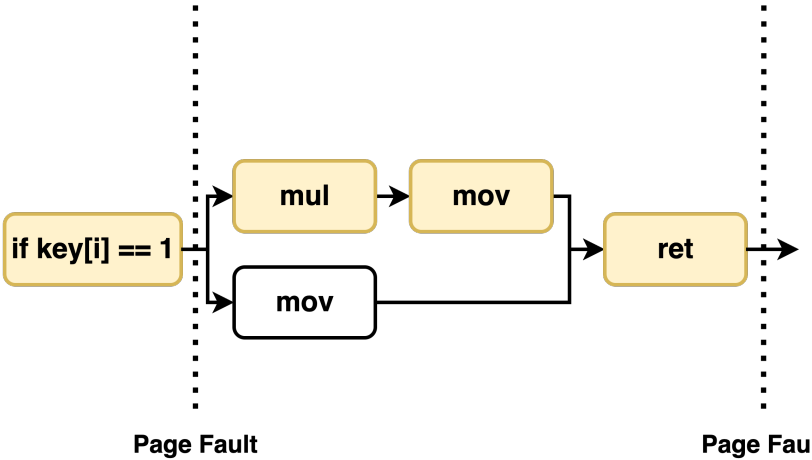
- Cache attack ✓
  - Frequency throttling improves temporal resolution even more
- Page faults for precise termination of prevention mode

# Primitive Evaluation

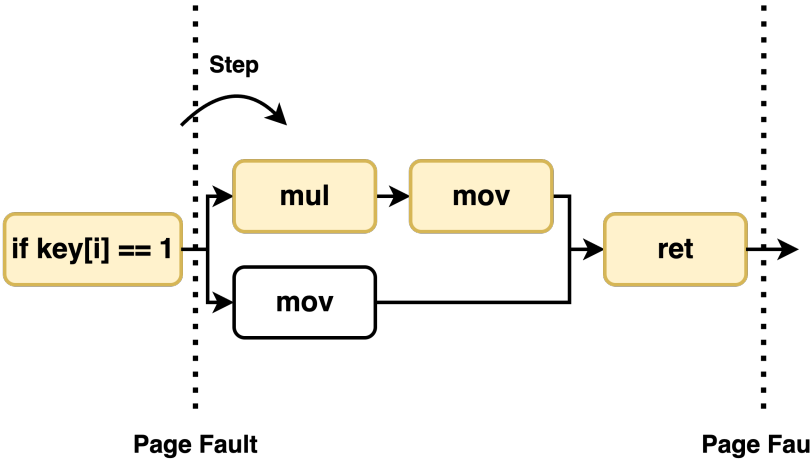
# Secret Dependent Control-Flow



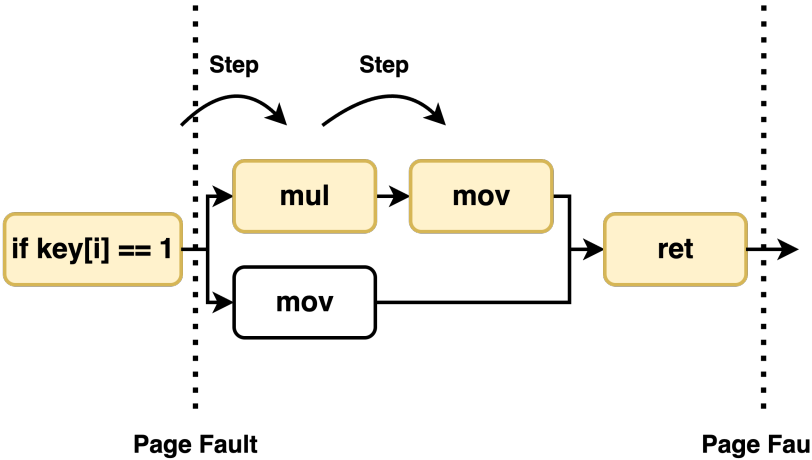
# Exploitation with Single-Stepping



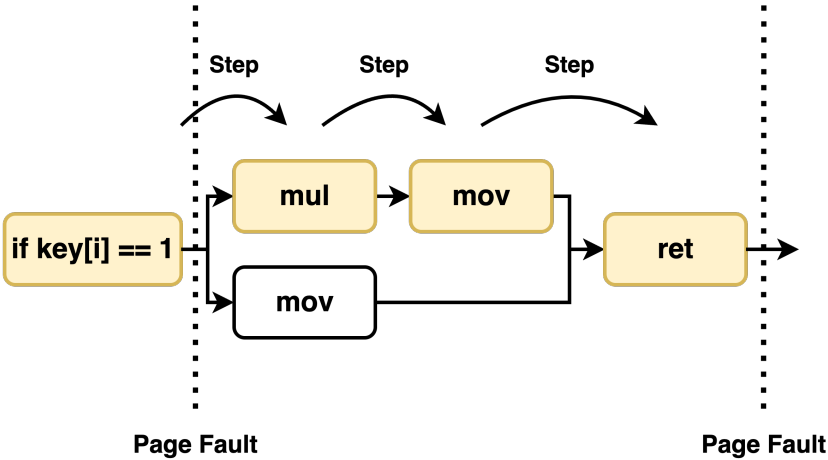
# Exploitation with Single-Stepping



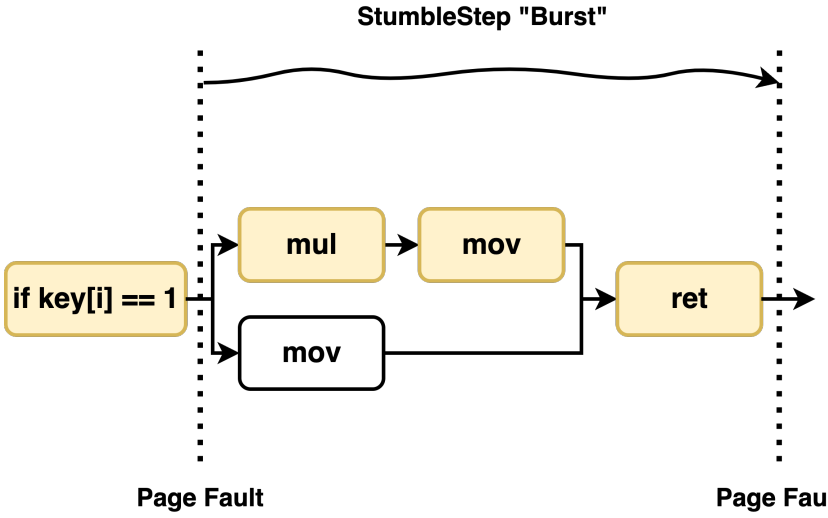
# Exploitation with Single-Stepping



# Exploitation with Single-Stepping

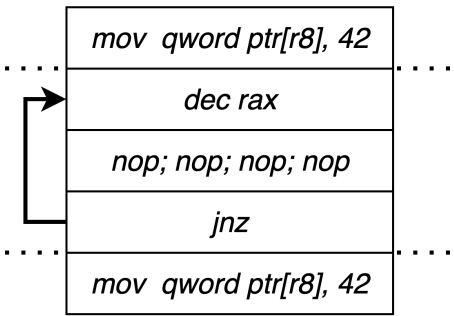


# Exploitation with StumbleStepping



# Synthetic Single-Stepping Evaluation

**Goals**: Filter all zero-steps & show absence of multi-steps

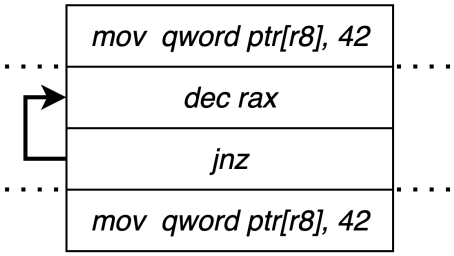



Evaluation target

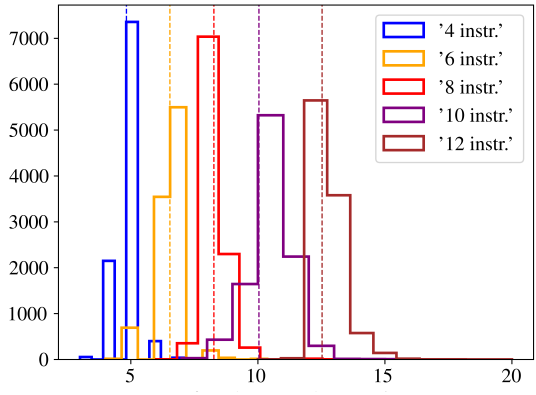
- 3 configs: 1, 9 and 10 loop iterations
  - corresponds to 8, 56 and 62 instructions
- 10 000 measurements for each config
- *Found no errors*
- only 0.8% zero-step events

# Synthetic StumbleStepping Evaluation

**Goals**: Evaluate Accuracy of inferred instruction count



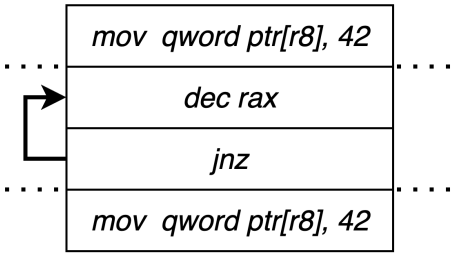

Evaluation Target



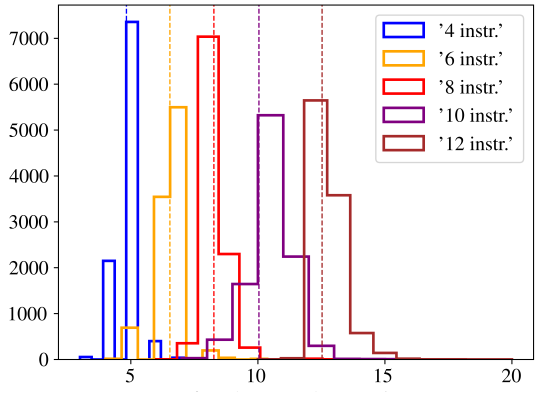

Inferred Executed Instructions

# Synthetic StumbleStepping Evaluation

**Goals**: Evaluate Accuracy of inferred instruction count



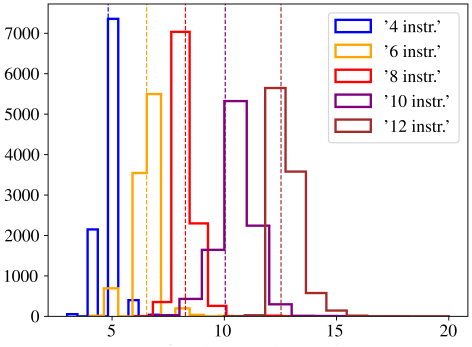

Evaluation Target



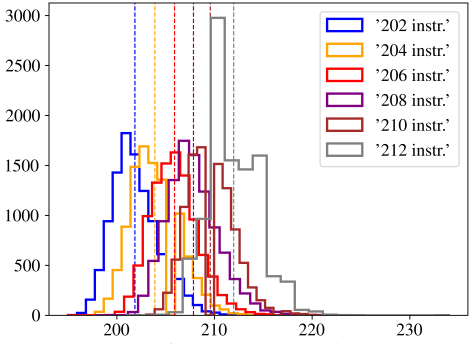

Inferred Executed Instructions

**However...**

Noise grows with observation length

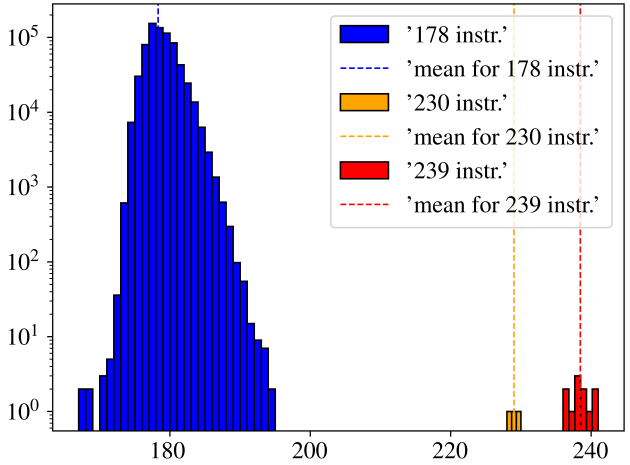



Inferred Executed Instructions (short)




Inferred Executed Instructions (long)

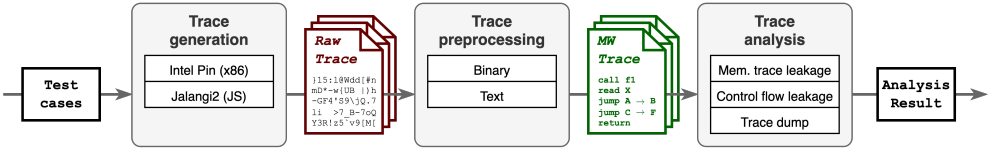
# StumbleStepping E2E Attack Sneak Peek




Control Flow events for secp160r1 ECDSA in wolfSSL

# Finding Control Flow Leakages

# Finding Control-Flow Leakages with Microwalk




Microwalk

# Modular Reduction in ECDSA

ECDSA is a signature scheme, it requires a nonce k that must remain secret

1. Goal: Get random nonce $k < n$

# Modular Reduction in ECDSA

ECDSA is a signature scheme, it requires a nonce k that must remain secret

1. Goal: Get random nonce $k < n$
2. Also: Get it fast

# Modular Reduction in ECDSA

ECDSA is a signature scheme, it requires a nonce k that must remain secret

1. Goal: Get random nonce $k < n$
2. Also: Get it fast
3. Modular reduction approach:

# Modular Reduction in ECDSA

ECDSA is a signature scheme, it requires a nonce k that must remain secret

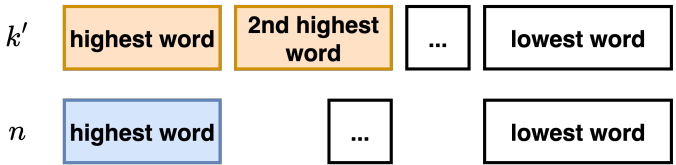
1. Goal: Get random nonce $k < n$
2. Also: Get it fast
3. Modular reduction approach:
   3.1 Sample candidate nonce $k'$

# Modular Reduction in ECDSA

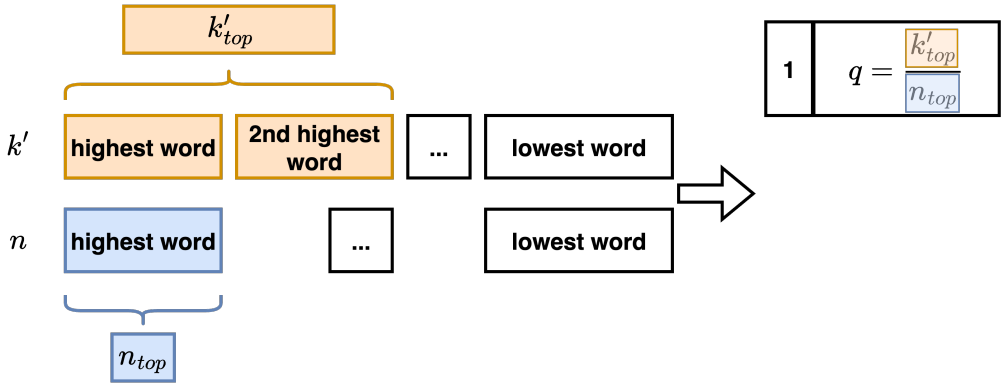
ECDSA is a signature scheme, it requires a nonce k that must remain secret

1. Goal: Get random nonce $k < n$
2. Also: Get it fast
3. Modular reduction approach:
   - 3.1 Sample candidate nonce $k'$
   - 3.2 Compute $k$ as $k' \mod n$

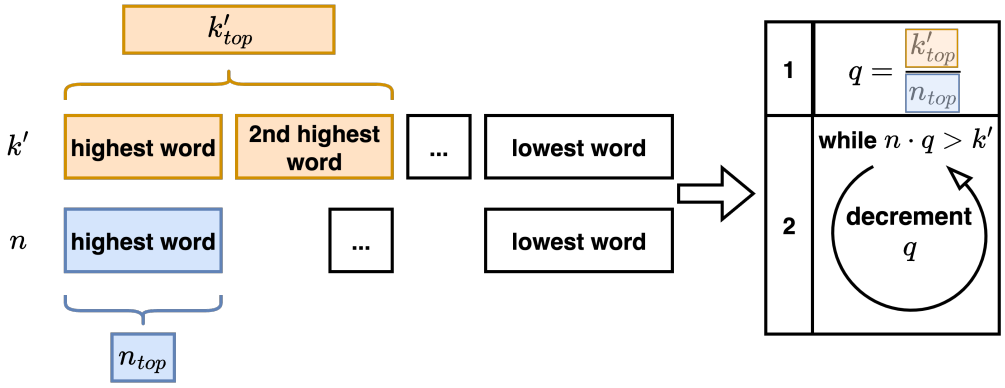
# Compute $k'$ mod $n$



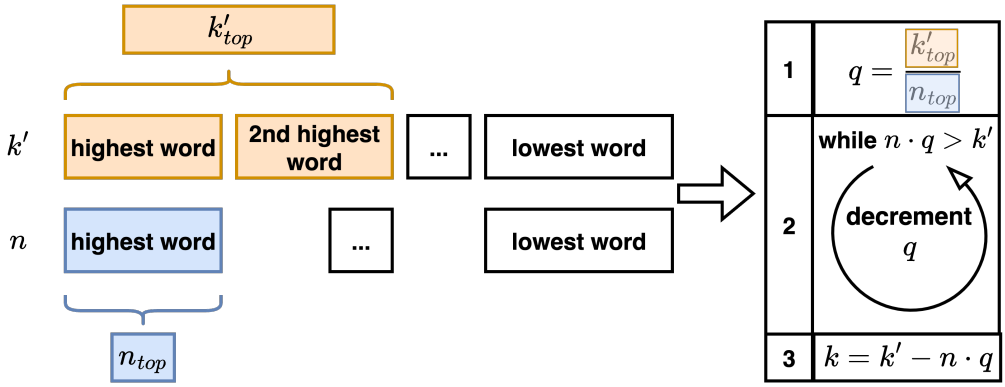
# Compute $k'$ mod $n$



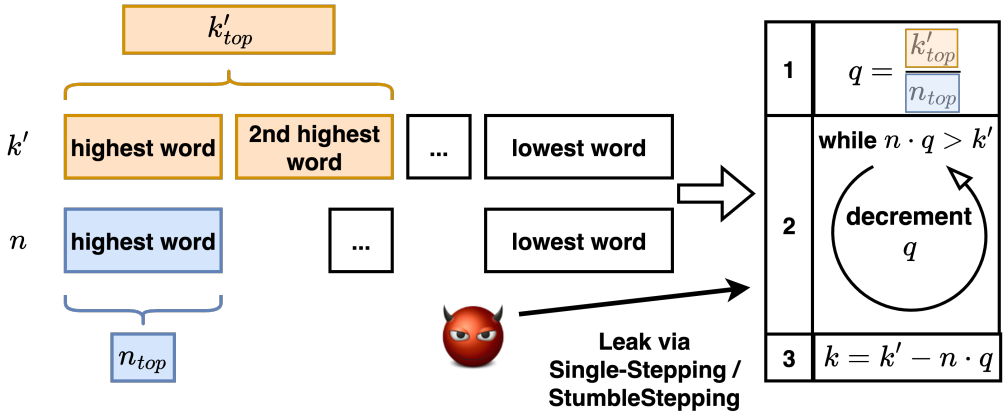
# Compute $k'$ mod $n$



# Compute $k'$ mod $n$



# Compute $k'$ mod $n$

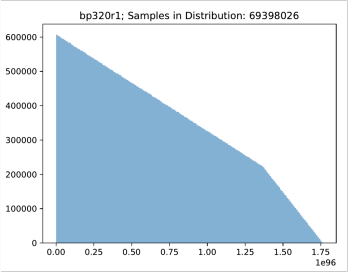


# Attack Case Study

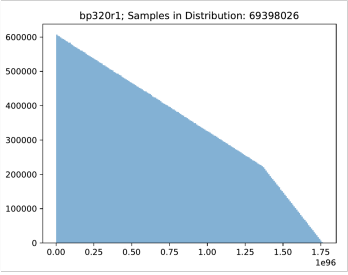
# ECDSA Nonce Truncation in wolfSSL

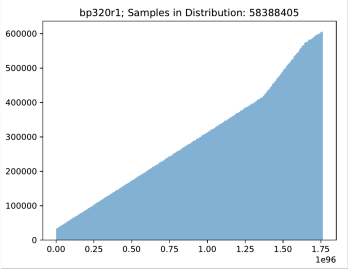
```
1  int _sp_div_impl(sp_int* a, d, r, trial) {
2      for (i = a->used - 1; i >= d->used; i--) {
3          //Calculate trial quotient
4          t = sp_div_word(a->dp[i], a->dp[i-1], dt);
5          do {
6              for (j = 0; j < d->used; j++) {...}
7              for (j = d->used; j > 0; j--)
8                  //Event W2
9                  if (trial->dp[j] != a->dp[j + o])
10                     break;
11             if (trial->dp[j] > a->dp[j + o]) { t--; }
12             //Event W1
13         } while (trial->dp[j] > a->dp[j + o]);
14     }
15 };
```

# Analyzing the Distributions - Sampling the nonce k 128 million times

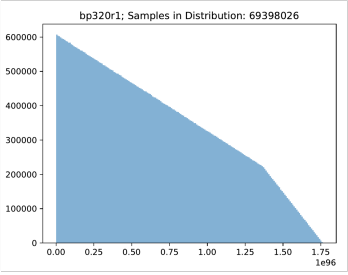


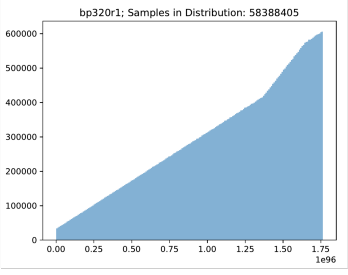
# Analyzing the Distributions - Sampling the nonce k 128 million times

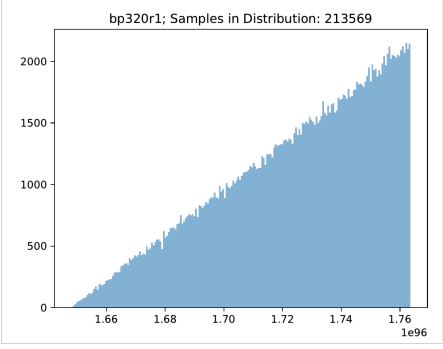


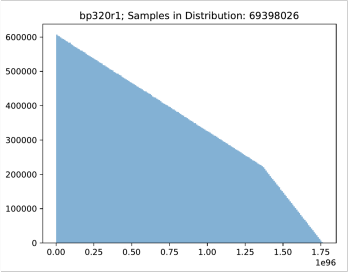
# Analyzing the Distributions - Sampling the nonce k 128 million times

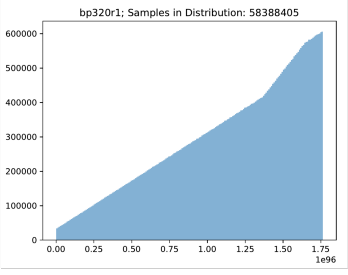


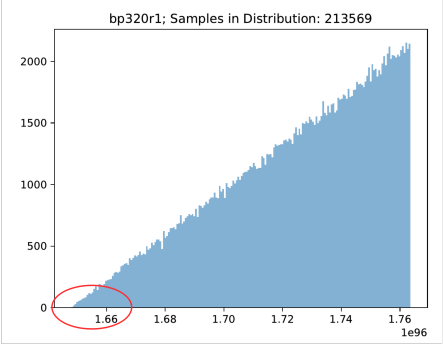



# Analyzing the Distributions - Sampling the nonce k 128 million times

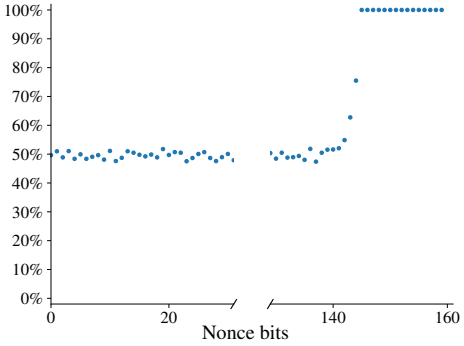


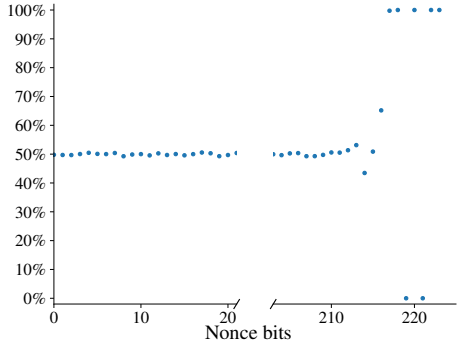



# Nonce Bias

Nonce bit distribution given leaked loop iterations for *secp160r1* and brainpoolP224r1.



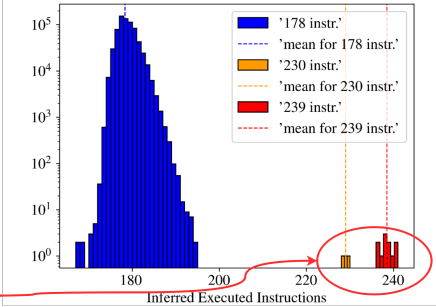



# Leakage Overview in wolfSSL and OpenSSL

| | wolfSSL | | OpenSSL | |
|---|---|---|---|---|
| Curve | Event | MI / FB | Event | MI / FB |
| | $(W_1, W_2)$<br>$Pr[A = a]$ | [bit / bit] | $(O_1, O_2)$<br>$Pr[A = a]$ | [bit / bit] |
| bp224r1 | (2, *)<br>0.09 | 1.6 / 1 | (1, 0)<br>$1.6 \cdot 10^{-4}$ | 7 / 6 |
| bp320r1 | (3, *)<br>$< 0.002$ | 3 / 3 | (2, *)<br>$1.7 \cdot 10^{-3}$ | 3 / 3 |
| bp384r1 | (2, *)<br>0.05 | 3.5 / 0 | (1, *)<br>0.05 | 3.5 / 0 |
| secp160r1 | (2, *)<br>$1.5 \cdot 10^{-5}$ | 15.6 / 15 | (1, *)<br>$1.3 \cdot 10^{-5}$ | 15.8 / 15 |

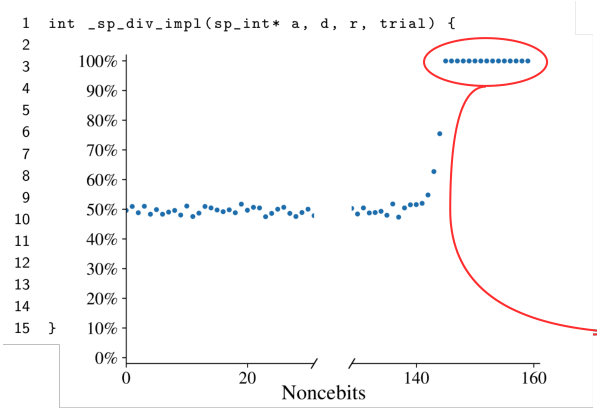
# StumbleStepping the Nonce Bias

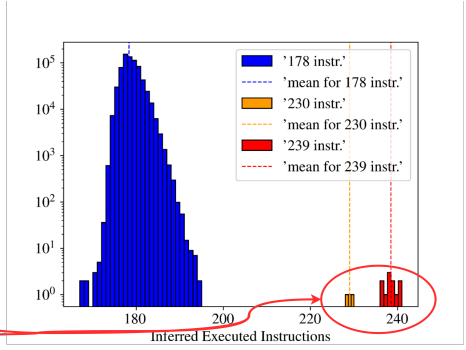
```
1  int _sp_div_impl(sp_int* a, d, r, trial) {
2      for (i = a->used - 1; i >= d->used; i--) {
3          //Calculate trial quotient
4          t = sp_div_word(a->dp[i], a->dp[i-1], dt);
5          do {
6              for (j = 0; j < d->used; j++) {...}
7              for (j = d->used; j > 0; j--)
8                  //Event W2
9                  if (trial->dp[j] != a->dp[j + o])
10                     break;
11             if (trial->dp[j] > a->dp[j + o]) { t--; }
12         //Event W1
13         } while (trial->dp[j] > a->dp[j + o]);
14     }
15 };
```



# StumbleStepping the Nonce Bias

```
1  int _sp_div_impl(sp_int* a, d, r, trial) {
2
3
4
5
6
7
8
9
10
11
12
13
14
15 }
```



# Countermeasures

# Improved Single-Stepping Heuristic

Current Heuristic: Benign if

$$\texttt{rip\_delta} > \texttt{INTEL64\_MAX\_INST\_LEN} * 2$$

OR

$$\texttt{vcpu\_tsc\_delta}(\texttt{ld\_p}) > \texttt{STEPPING\_TSC\_THRESHOLD}$$

Suggested Improvements:

- Only rely on instruction pointer progress
- AEX-Notify shows that reliable “n-stepping” is not possible

# Improved Single-Stepping Heuristic

Current Heuristic: Benign if

$$\texttt{rip\_delta} > \texttt{INTEL64\_MAX\_INST\_LEN} * 2$$

OR

$$\texttt{vcpu\_tsc\_delta}(\texttt{ld\_p}) > \texttt{STEPPING\_TSC\_THRESHOLD}$$

Suggested Improvements:

- Only rely on instruction pointer progress
- AEX-Notify shows that reliable “n-stepping” is not possible

# Intel's TDX Module Patch

```
if ((rip_delta > INTEL64_MAX_INST_LEN * 2) || (vcpu_tsc_delta(ld_p) > STEPPING_TSC_THRESHOLD))
// Always use instruction count heuristic if Perfmon is disabled, regardless of TDCS.ATTRIBUTES.
ICSSD
if (!perfmon_enabled)
{      mvainer, 3 weeks ago • TDX 1.5.06 …
    uint64_t inst_retired = ia32_rdmsr(IA32_FIXED_CTR0_MSR_ADDR);
    uint64_t rcx_delta = ld_p->guest_rcx_on_td_entry - ld_p->vp_ctx.tdvps->guest_state.gpr_state.
    rcx;

    if ((inst_retired > 1) || ((0 == inst_retired) && (rcx_delta > 1)))
    {
        return FILTER_OK_CONTINUE;
    }
}
else if ((rip_delta > INTEL64_MAX_INST_LEN * 2) || (vcpu_tsc_delta(ld_p) > STEPPING_TSC_THRESHOLD))
{
    return FILTER_OK_CONTINUE;
}
```

# Preventing Instruction Counting Attacks

- StumbleStepping attack won't be mitigated by TDX module
- Protecting crypto code via constant time programming is feasible
- Protecting databases, image decoding, etc. is *not feasible*

# Preventing Instruction Counting Attacks

- StumbleStepping attack won't be mitigated by TDX module
- Protecting crypto code via constant time programming is feasible
- Protecting databases, image decoding, etc. is *not feasible*

# Preventing Instruction Counting Attacks

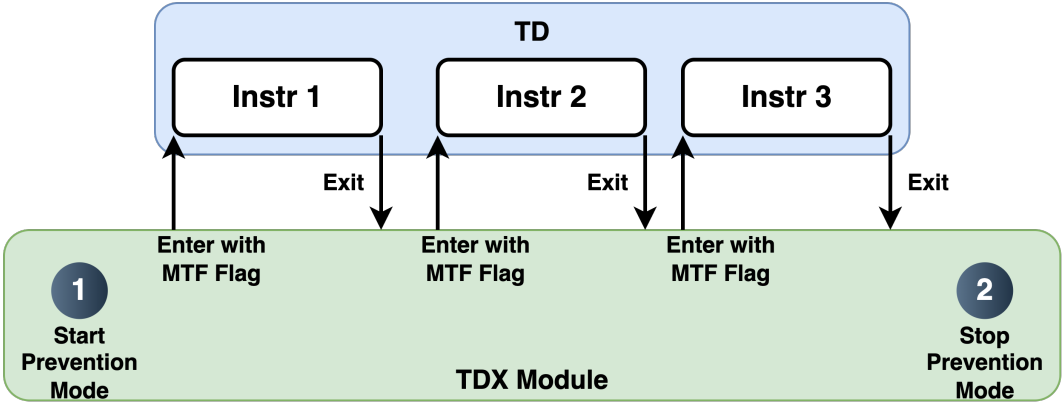
- StumbleStepping attack won't be mitigated by TDX module
- Protecting crypto code via constant time programming is feasible
- Protecting databases, image decoding, etc. is *not feasible*

# Preventing Instruction Counting Attacks

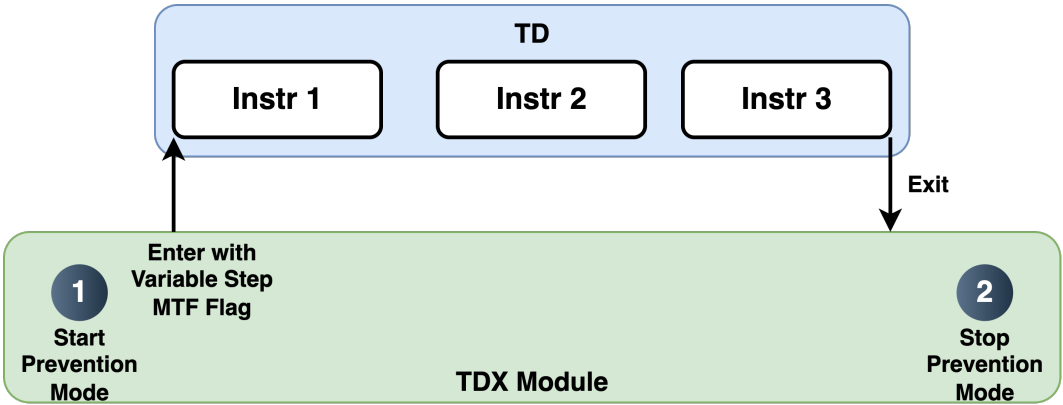
- StumbleStepping attack won't be mitigated by TDX module
- Protecting crypto code via constant time programming is feasible
- Protecting databases, image decoding, etc. is *not feasible*

Need principled mitigations as part of the TEE

# Approach 1: Changing MTF flag



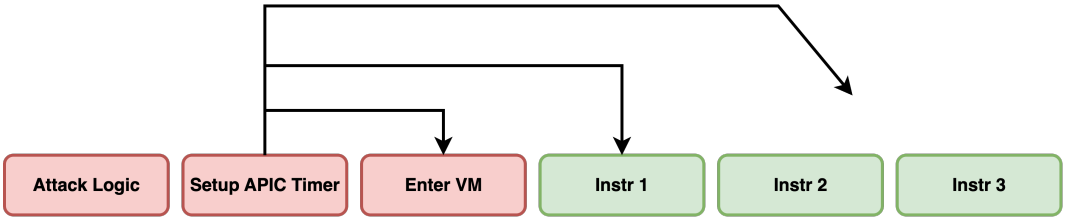
# Approach 1: Changing MTF flag




Implementation effort hard to judge for us

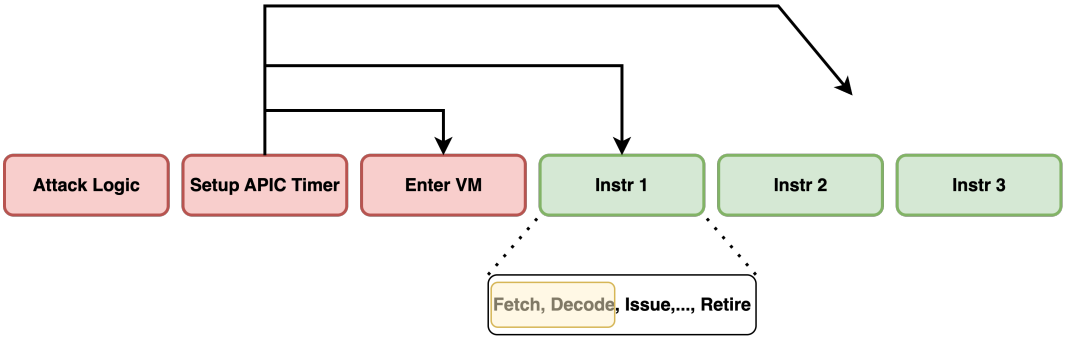
# Approach 2: AEX-Notify

Recap Single-Stepping:



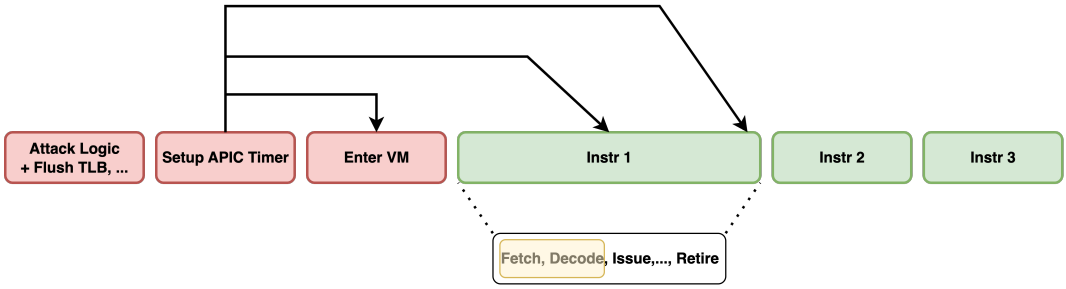
# Approach 2: AEX-Notify

Recap Single-Stepping:



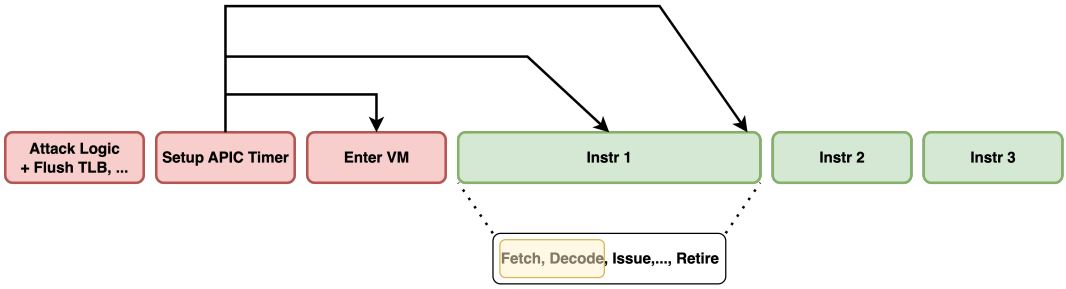
# Approach 2: AEX-Notify

Recap Single-Stepping:



# Approach 2: AEX-Notify

Recap Single-Stepping:



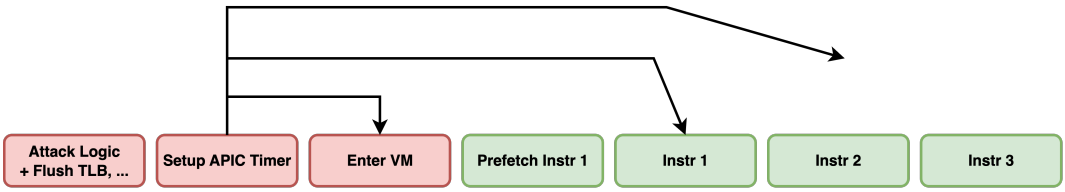

Without slowing down `Instr 1`, reliable, repeated single-stepping is not possible

# Approach 2: AEX-Notify cont.

AEX-Notify idea:

# Approach 2: AEX-Notify cont.

AEX-Notify idea:




- Prefetch code is constant time
- Small, atomic part at the end

# Approach 2: Integrate AEX-Notify with CVMs

Idea:

1. Build on existing interrupt injection mechanisms
2. Execute AEX-Notify prefetch as part of each interrupt
3. For VM enter without interrupt injection: Force injection of dummy interrupt

**Pro**: Minimial HW-RoT changes $\Rightarrow$ portable across CVMs?
**Con**: Overhead of prefetch, Zero-Stepping security?

# Approach 2: Integrate AEX-Notify with CVMs

Idea:

1. Build on existing interrupt injection mechanisms
2. Execute AEX-Notify prefetch as part of each interrupt
3. For VM enter without interrupt injection: Force injection of dummy interrupt

**Pro**: Minimial HW-RoT changes $\Rightarrow$ portable across CVMs?
**Con**: Overhead of prefetch, Zero-Stepping security?

# Approach 2: Integrate AEX-Notify with CVMs

Idea:

1. Build on existing interrupt injection mechanisms
2. Execute AEX-Notify prefetch as part of each interrupt
3. For VM enter without interrupt injection: Force injection of dummy interrupt

**Pro**: Minimial HW-RoT changes $\Rightarrow$ portable across CVMs?
**Con**: Overhead of prefetch, Zero-Stepping security?

# Approach 2: Integrate AEX-Notify with CVMs

Idea:

1. Build on existing interrupt injection mechanisms
2. Execute AEX-Notify prefetch as part of each interrupt
3. For VM enter without interrupt injection: Force injection of dummy interrupt

Pro: Minimial HW-RoT changes $\Rightarrow$ portable across CVMs?
Con: Overhead of prefetch, Zero-Stepping security?

# Approach 2: Integrate AEX-Notify with CVMs

Idea:

1. Build on existing interrupt injection mechanisms
2. Execute AEX-Notify prefetch as part of each interrupt
3. For VM enter without interrupt injection: Force injection of dummy interrupt

**Pro**: Minimial HW-RoT changes $\Rightarrow$ portable across CVMs?
Con: Overhead of prefetch, Zero-Stepping security?

# What about page fault side channel

- Page granular leakage already sufficient to leak image from jpeg decoding
- TDX already protects the 2nd level page tables but exports page blocking API
- Why not completely restrict forcing page faults?

# What about page fault side channel

- Page granular leakage already sufficient to leak image from jpeg decoding
- TDX already protects the 2nd level page tables but exports page blocking API
- Why not completely restrict forcing page faults?

# What about page fault side channel

- Page granular leakage already sufficient to leak image from jpeg decoding
- TDX already protects the 2nd level page tables but exports page blocking API
- Why not completely restrict forcing page faults?

# Summary

# Summary

- Attacks on TDX
  - full single-stepping
  - instruction counting via *StumbleStepping*;
- Finding & Exploiting Control Flow Leakages
  - Microwalk + Distribution Analysis
  - Nonce truncation in wolfSSL and OpenSSL leaks for certain curves
- Countermeasures
  - Improved MTF flag, AEXNotify for CVMs
  - Preventing page fault side-channel?
- Responsible Disclosure:
  - Intel fixed single-stepping with TDX module 1.5.06 but not StumbleStepping
  - wolfSSL and OpenSSL switched to rejection sampling

# Backup Slides