CopyCat: Controlled Instruction-Level Attacks on Enclaves

- **Daniel Moghimi**
- Jo Van Bulck
- Nadia Heninger
- Frank Piessens
- Berk Sunar

29TH USENIX SECURITY SYMPOSIUM

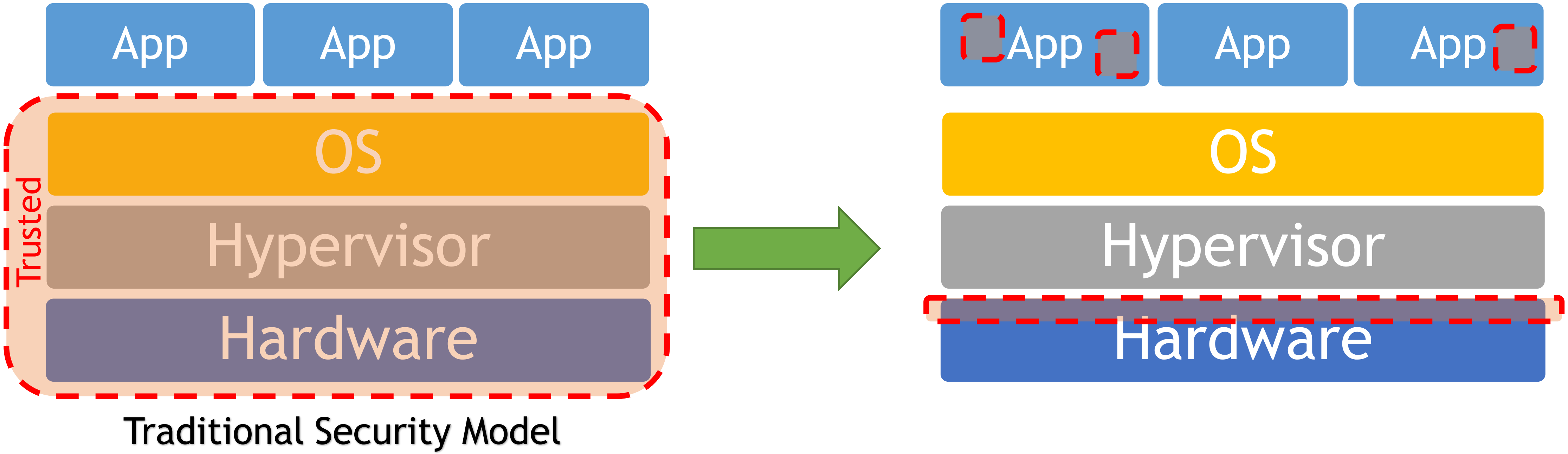
AUGUST 12–14, 2020

DistriNet

KU LEUVEN

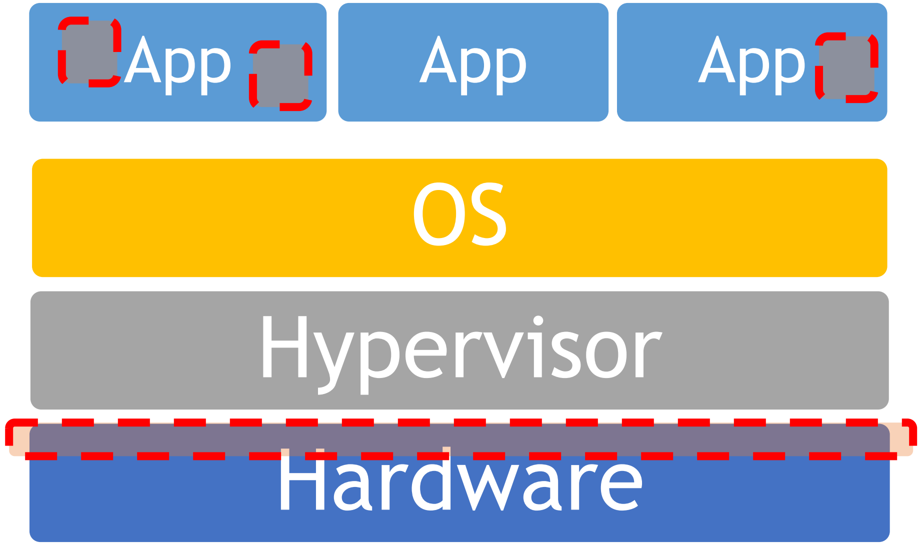
Trusted Execution Environment (TEE) - Intel SGX

- Intel Software Guard eXtensions (SGX)



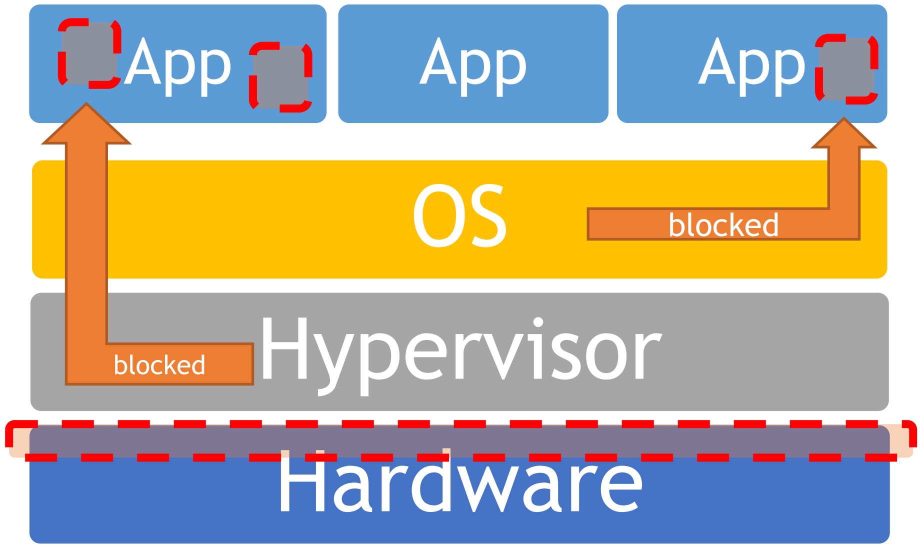
Trusted Execution Environment (TEE) – Intel SGX

- Intel Software Guard eXtensions (SGX)
- **Enclave:** Hardware protected user-level software module
  - Mapped by the Operating System
  - Loaded by the user program
  - Authenticated and Encrypted by CPU



Trusted Execution Environment (TEE) - Intel SGX

- Intel Software Guard eXtensions (SGX)
- **Enclave:** Hardware protected user-level software module
  - Mapped by the Operating System
  - Loaded by the user program
  - Authenticated and Encrypted by CPU
- Protects against system level adversary

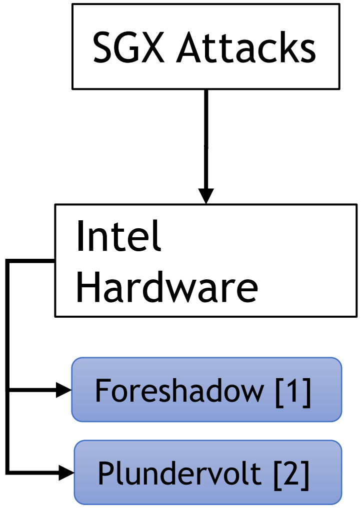


New Attacker Model:

Attacker gets full control over OS

Intel SGX Attack Taxonomy

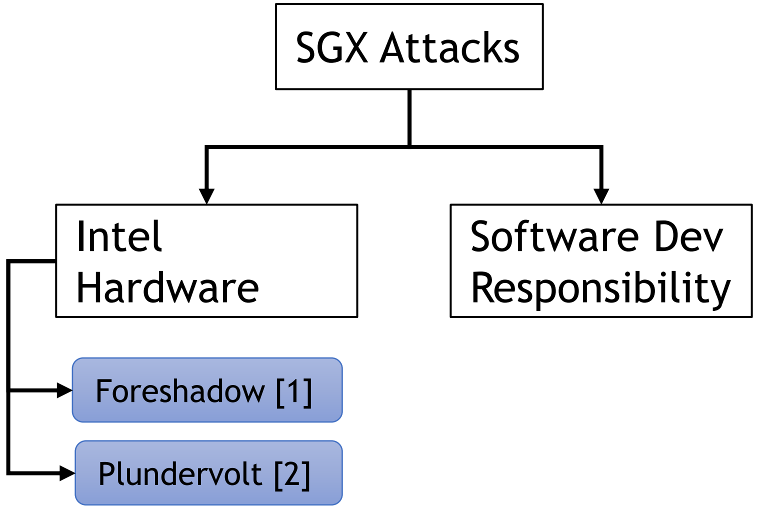
- Intel's Responsibility
  - Microcode Patches / Hardware mitigation
  - TCB Recovery
    - Old Keys are Revoked
    - Remote attestation succeeds only with mitigation.



[1] Van Bulck et al. "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution." USENIX Security 2018.
[2] Murdock et al. "Plundervolt: Software-based fault injection attacks against Intel SGX." IEEE S&P 2020.

Intel SGX Attack Taxonomy

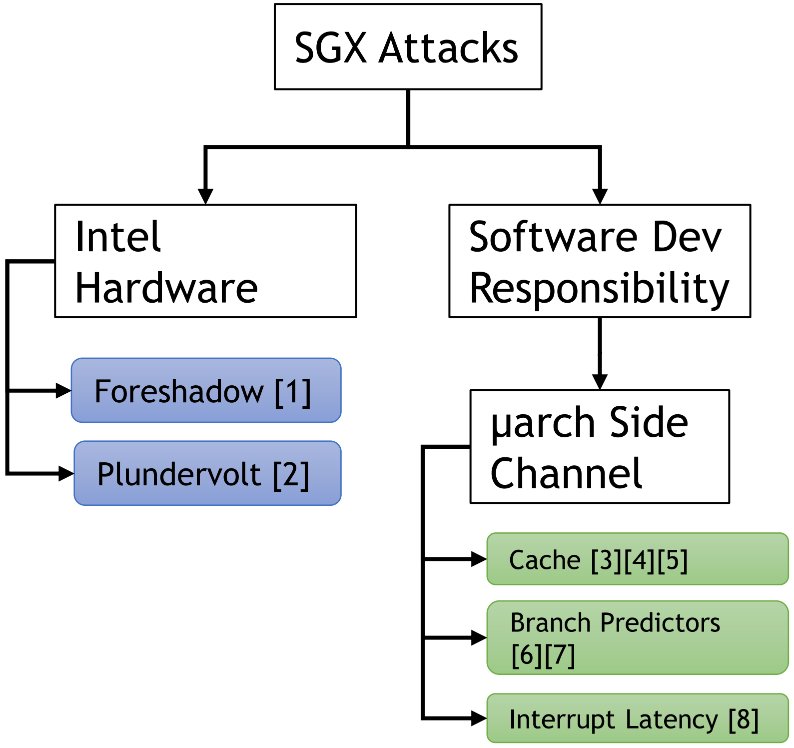
- Intel's Responsibility
  - Microcode Patches / Hardware mitigation
  - TCB Recovery
    - Old Keys are Revoked
    - Remote attestation succeeds only with mitigation.



[1] Van Bulck et al. "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution." USENIX Security 2018.
[2] Murdock et al. "Plundervolt: Software-based fault injection attacks against Intel SGX." IEEE S&P 2020.

Intel SGX Attack Taxonomy

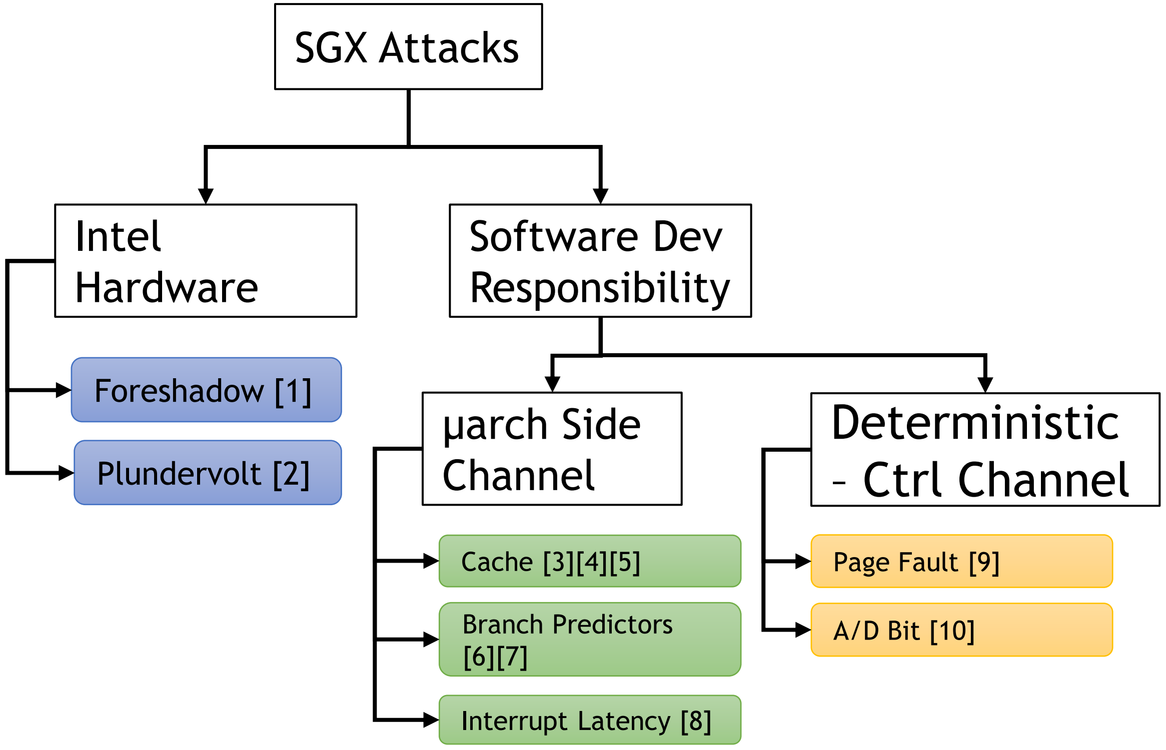
- Intel's Responsibility
  - Microcode Patches / Hardware mitigation
  - TCB Recovery
    - Old Keys are Revoked
    - Remote attestation succeeds only with mitigation.
  - Hyperthreading is out
    - Remote Attestation Warning
- μarch Side Channel
  - Constant-time Coding
  - Flushing and Isolating buffers
  - Probabilistic



[1] Van Bulck et al. "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution." USENIX Security 2018.
[2] Murdock et al. "Plundervolt: Software-based fault injection attacks against Intel SGX." IEEE S&P 2020.
[3] Moghimi et al. "Cachezoom: How SGX amplifies the power of cache attacks." CHES 2017.
[4] Brasser et al. "Software grand exposure:{SGX} cache attacks are practical." USENIX WOOT 2017.
[5] Schwarz et al. "Malware guard extension: Using SGX to conceal cache attacks." DIMVA 2017.
[6] Evtyushkin, Dmitry, et al. "Branchscope: A new side-channel attack on directional branch predictor." ACM SIGPLAN 2018.
[7] Lee, Sangho, et al. "Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing." USENIX Security 2017.
[8] Van Bulck et al. "Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic." ACM CCS 2018.

Intel SGX Attack Taxonomy

- Intel's Responsibility
  - Microcode Patches / Hardware mitigation
  - TCB Recovery
    - Old Keys are Revoked
    - Remote attestation succeeds only with mitigation.
  - Hyperthreading is out
    - Remote Attestation Warning
- µarch Side Channel
  - Constant-time Coding
  - Flushing and Isolating buffers
  - Probabilistic
- Deterministic Attacks
  - Page Fault, A/D Bit, etc. (4kB Granularity)

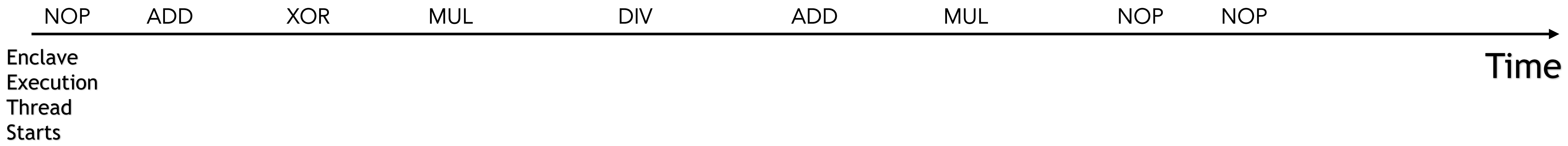


[1] Van Bulck et al. "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution." USENIX Security 2018.
[2] Murdock et al. "Plundervolt: Software-based fault injection attacks against Intel SGX." IEEE S&P 2020.
[3] Moghimi et al. "Cachezoom: How SGX amplifies the power of cache attacks." CHES 2017.
[4] Brasser et al. "Software grand exposure:{SGX} cache attacks are practical." USENIX WOOT 2017.
[5] Schwarz et al. "Malware guard extension: Using SGX to conceal cache attacks." DIMVA 2017.
[6] Evtyushkin, Dmitry, et al. "Branchscope: A new side-channel attack on directional branch predictor." ACM SIGPLAN 2018.
[7] Lee, Sangho, et al. "Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing." USENIX Security 2017.
[8] Van Bulck et al. "Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic." ACM CCS 2018.
[9] Xu et al. "Controlled-channel attacks: Deterministic side channels for untrusted operating systems." IEEE S&P 2015.
[10] Wang, Wenhao, et al. "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX." ACM CCS 2017.

CopyCat Attack

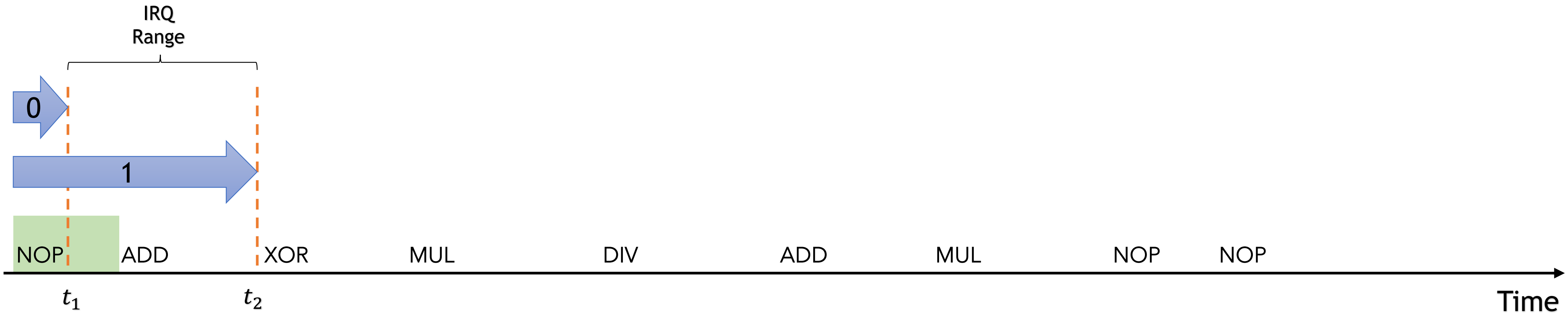
CopyCat Attack

- Malicious OS controls the interrupt handler



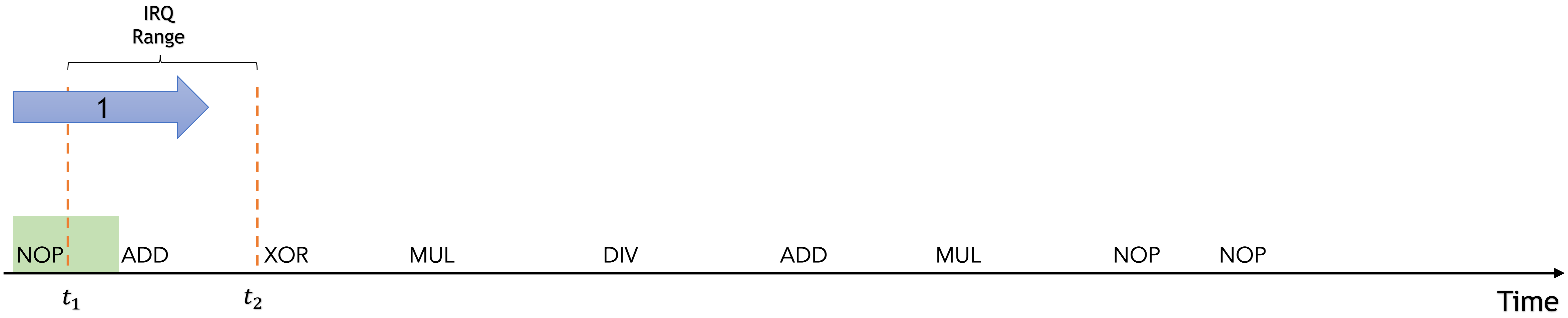
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



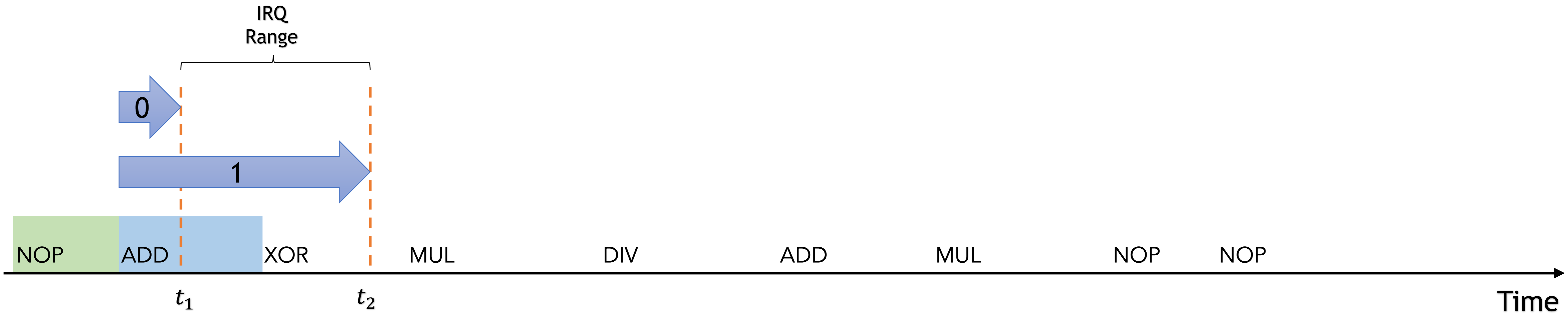
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



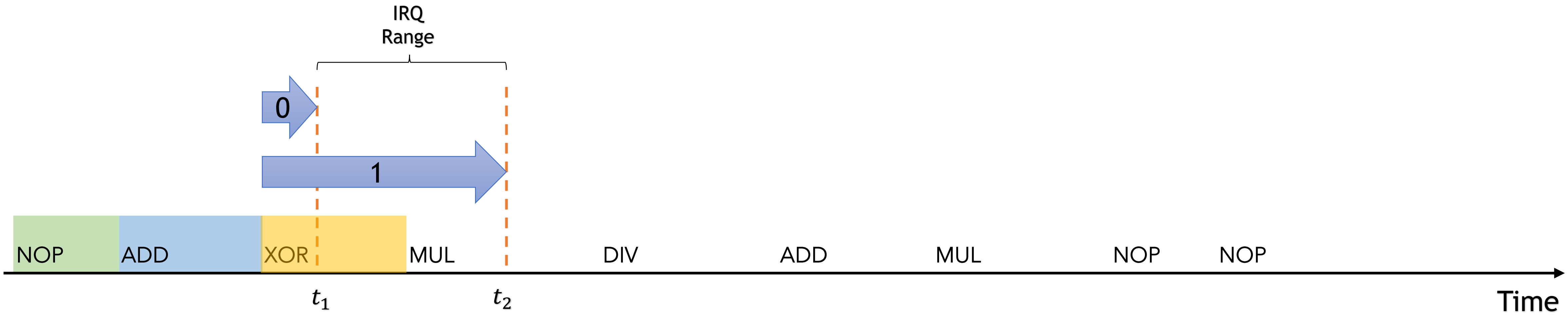
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



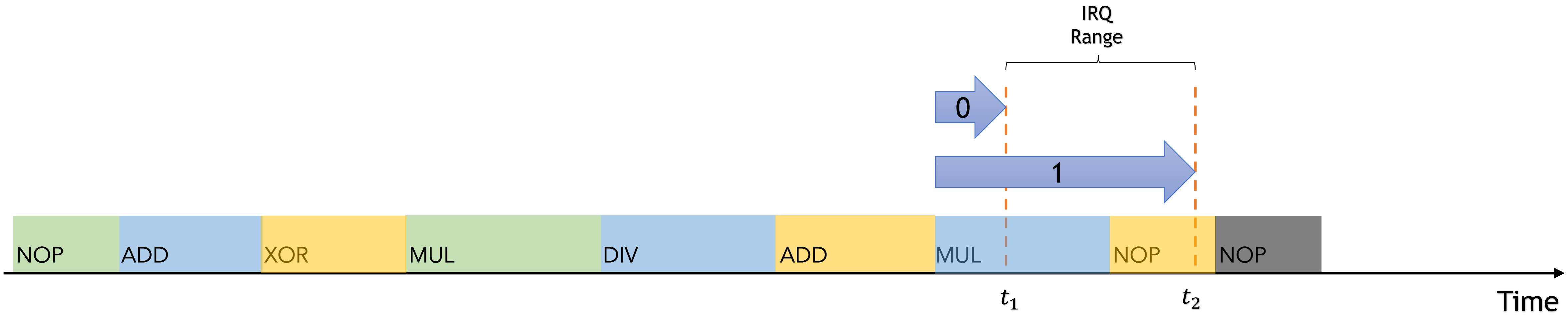
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



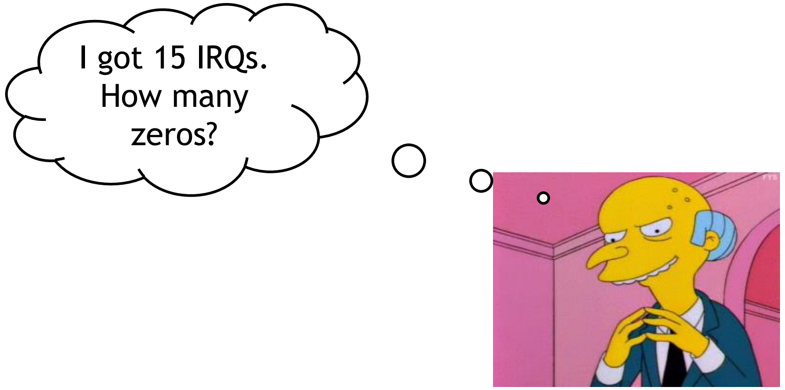
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



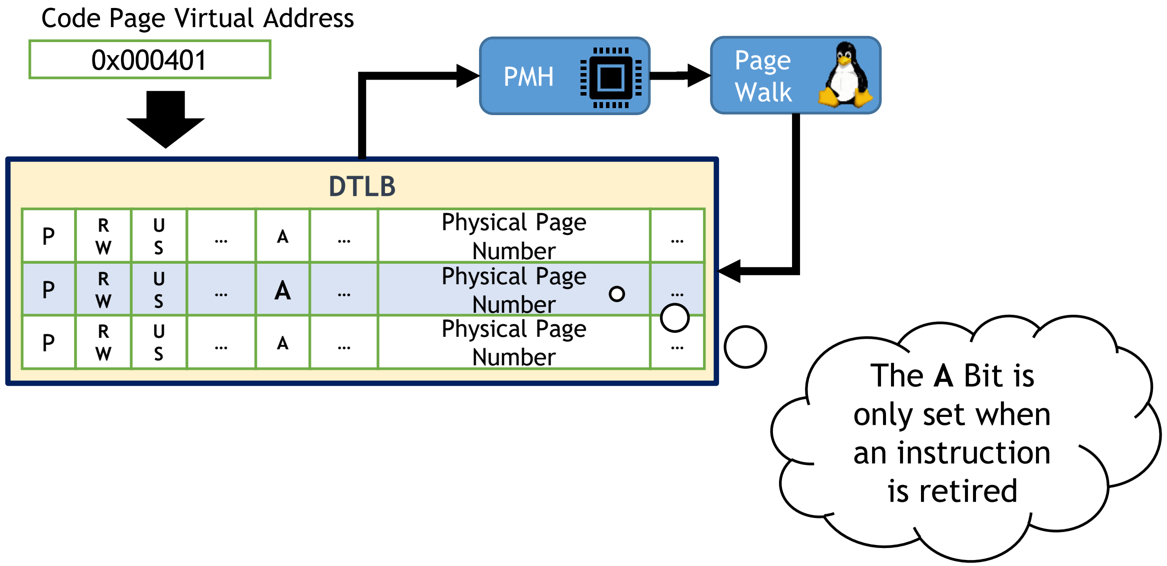
CopyCat Attack

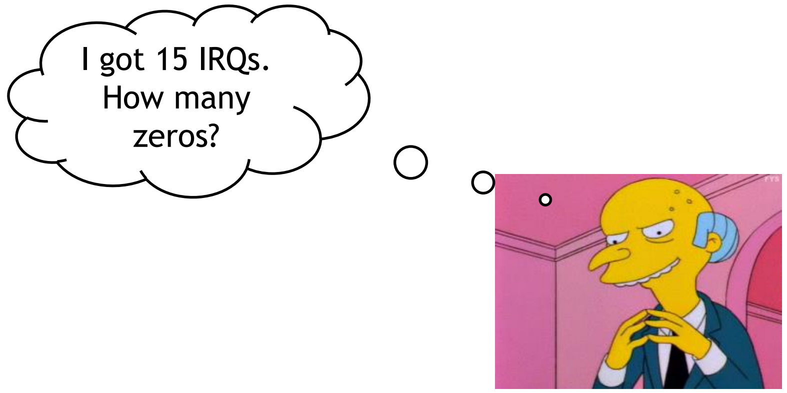
- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after



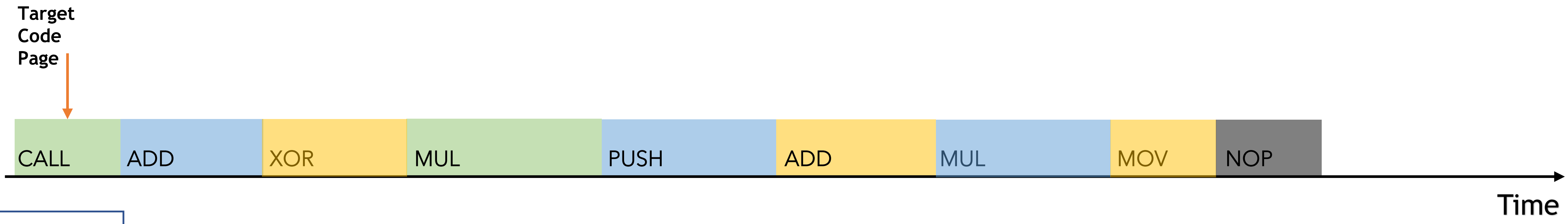


CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after
- Deterministic Instruction Counting

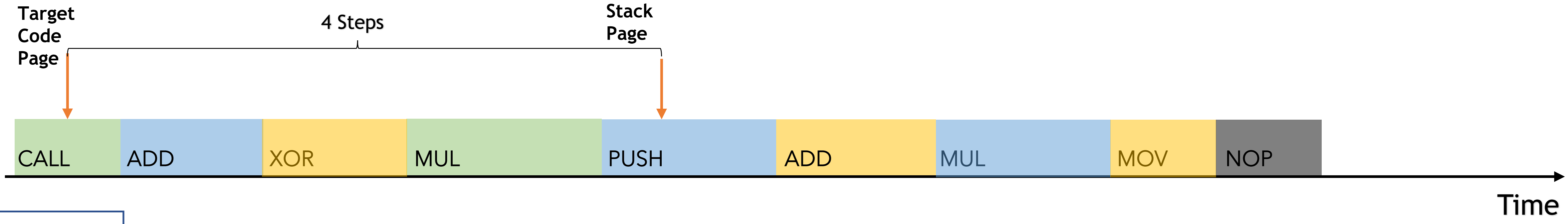
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
  - A Secondary oracle
  - Page table attack as a deterministic secondary oracle



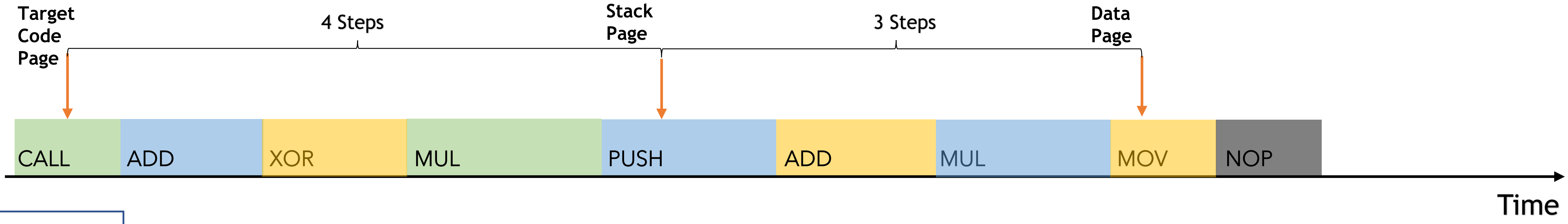
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
  - A Secondary oracle
  - Page table attack as a deterministic secondary oracle



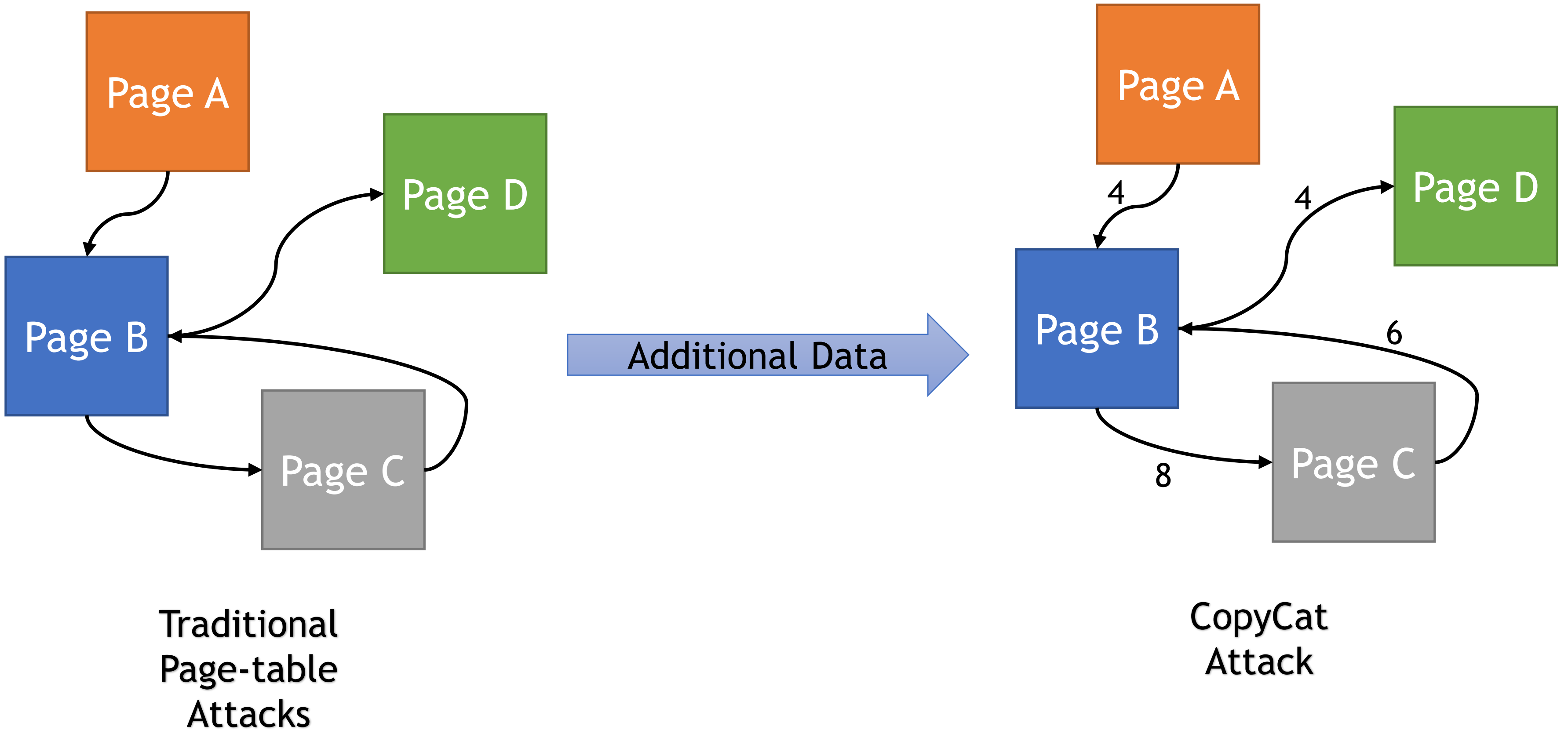
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
    - A Secondary oracle
    - Page table attack as a deterministic secondary oracle



CopyCat Attack

- Previous Controlled Channel attacks leak Page Access Patterns
- CopyCat additionally leaks number of instructions per page



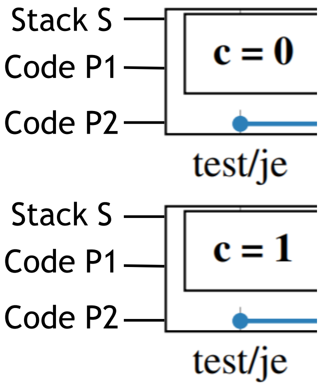
CopyCat – Leaking Branches


```c
if(c == 0) {
  r = add(r, d);
}
else {
  r = add(r, s);
}
```

C Code

Compile

```
test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
```



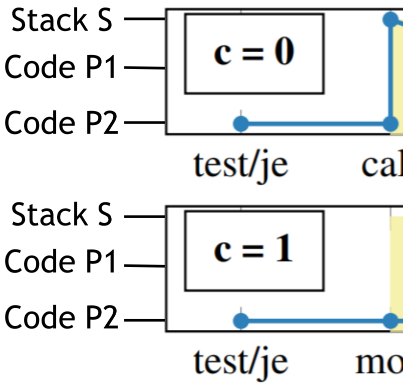
CopyCat – Leaking Branches

```c
if(c == 0) {
  r = add(r, d);
}
else {
  r = add(r, s);
}
```

C Code

Compile

```
test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
```



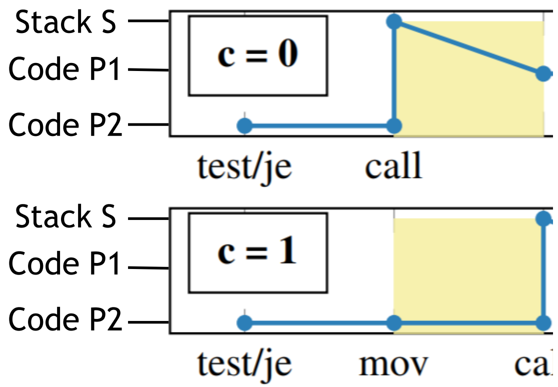
CopyCat - Leaking Branches


```c
if(c == 0) {
  r = add(r, d);
}
else {
  r = add(r, s);
}
```

C Code

Compile

```asm
test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
```



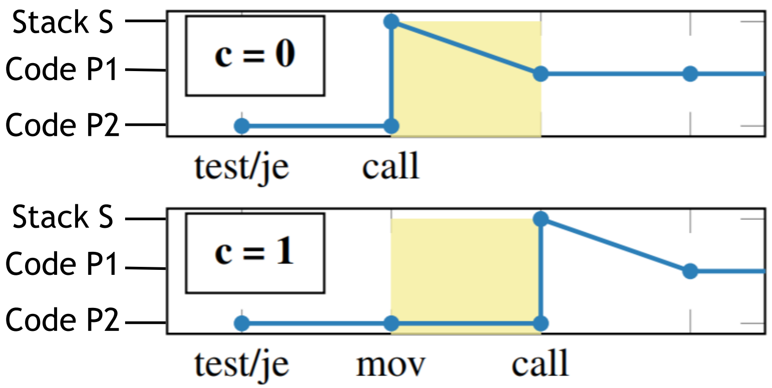
CopyCat - Leaking Branches


```c
if(c == 0) {
  r = add(r, d);
}
else {
  r = add(r, s);
}
```

C Code

Compile

```
test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
```



Crypto means Crpyptoattacks

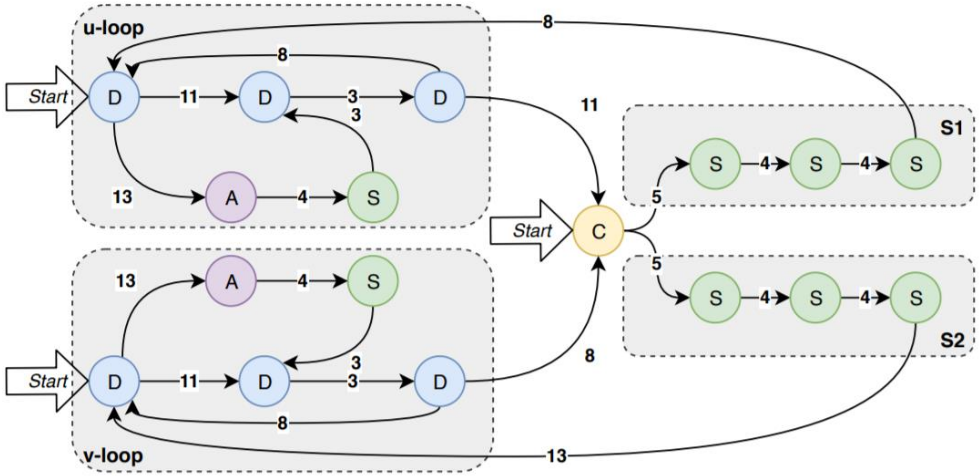
Binary Extended Euclidean Algorithm (BEEA)

- Previous attacks only leak some of the branches w/ some noise

procedure $\textsc{ModInv}(u, \text{modulus } v)$
    $b_i \leftarrow 0\ d_i \leftarrow 1, u_i \leftarrow u, v_i = v,$
    while $isEven(u_i)$ **do**
        $u_i \leftarrow u_i/2$
        if $isOdd(b_i)$ **then**
            $b_i \leftarrow b_i - u$
        $b_i \leftarrow b_i/2$
    while $isEven(v_i)$ **do**
        $v_i \leftarrow v_i/2$
        if $isOdd(d_i)$ **then**
            $d_i \leftarrow d_i - u$
        $d_i \leftarrow d_i/2$
    if $u_i > v_i$ **then**
        $u_i \leftarrow u_i - v_i,\ b_i \leftarrow b_i - d_i$
    else
        $v_i \leftarrow v_i - u_i,\ d_i \leftarrow d_i - b_i$
    return d_i

Binary Extended Euclidean Algorithm

- Previous attacks only leak some of the branches w/ some noise
- CopyCat synchronously leaks all the branches wo/ any noise



procedure $\textsc{ModInv}(u, \text{modulus } v)$
  $b_i \leftarrow 0\ d_i \leftarrow 1, u_i \leftarrow u, v_i = v,$
  while $isEven(u_i)$ **do**
    $u_i \leftarrow u_i/2$
    if $isOdd(b_i)$ **then**
      $b_i \leftarrow b_i - u$
    $b_i \leftarrow b_i/2$
  while $isEven(v_i)$ **do**
    $v_i \leftarrow v_i/2$
    if $isOdd(d_i)$ **then**
      $d_i \leftarrow d_i - u$
    $d_i \leftarrow d_i/2$
  if $u_i > v_i$ **then**
    $u_i \leftarrow u_i - v_i,\ b_i \leftarrow b_i - d_i$
  else
    $v_i \leftarrow v_i - u_i,\ d_i \leftarrow d_i - b_i$
  return d_i

CopyCat on WolfSSL - Cryptanalysis

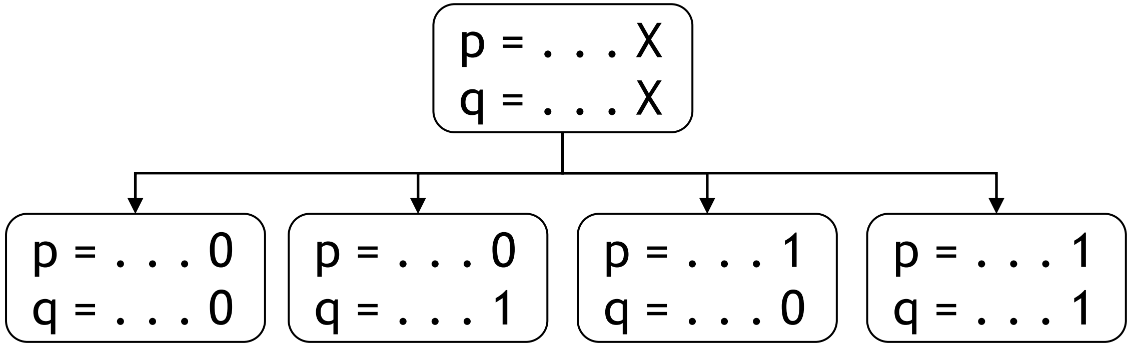
- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \, mod \, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = k_1^{-1}(h - r_1.x) mod \, n$ → get private key x

CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1}\, mod\, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = k_1^{-1}(h - r_1 . x) mod\, n$ → get private key x
- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1}\, mod\, p$
  - We know that $\mathrm{p}.\mathrm{q} = \mathrm{N}$

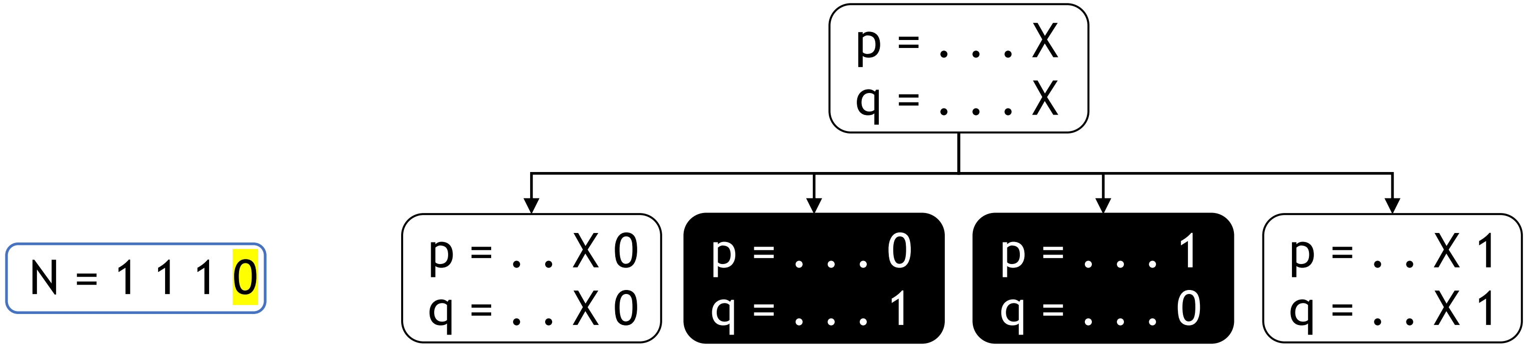
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \, mod \, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = k_1^{-1}(h - r_1 . x) mod \, n$ → get private key x
- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $\mathrm{p}.\mathrm{q} = \mathrm{N}$
  - Branch and prune Algorithm with the help of the recovered trace



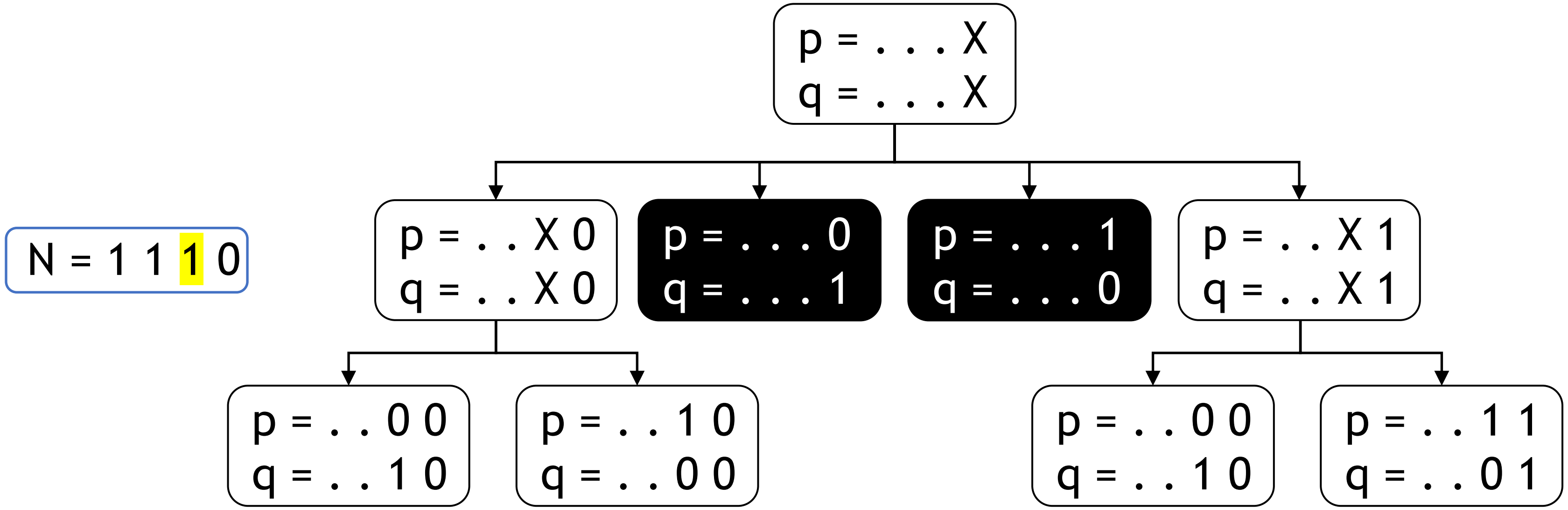
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \, mod \, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = k_1^{-1}(h - r_1.x) mod \, n$ → get private key x
- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $\mathrm{p}.\mathrm{q} = \mathrm{N}$, and N is public
  - Branch and prune Algorithm with the help of the recovered trace



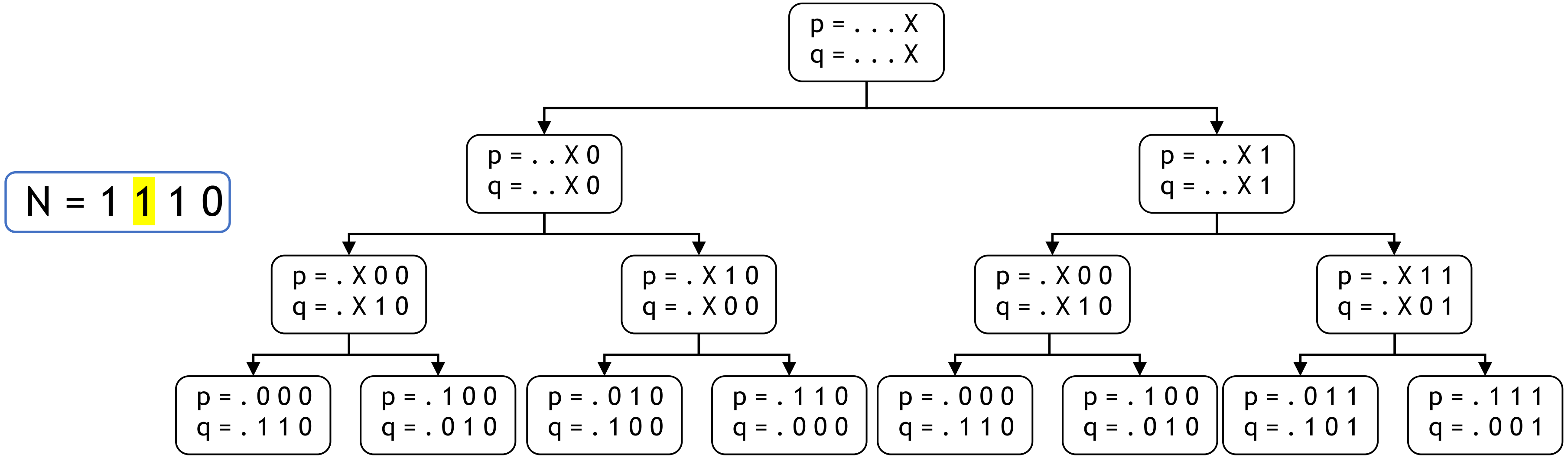
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \, mod \, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = k_1^{-1}(h - r_1.x) mod \, n$ → get private key x
- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $\mathrm{p.q} = \mathrm{N}$, and N is public
  - Branch and prune Algorithm with the help of the recovered trace



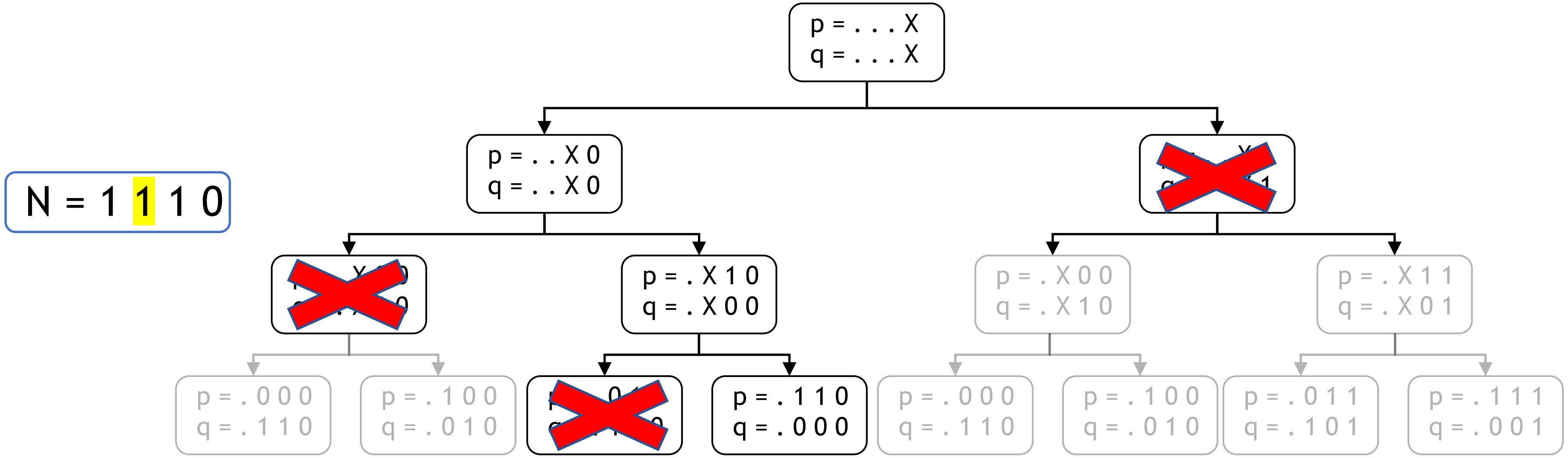
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \, mod \, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = k_1^{-1}(h - r_1 . x) mod \, n$ → get private key x
- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $\mathrm{p}.\mathrm{q} = \mathrm{N}$, and N is public
  - Branch and prune Algorithm with the help of the recovered trace



CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \, mod \, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = k_1^{-1}(h - r_1.x) mod \, n$ → get private key x
- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $\mathrm{p.q} = \mathrm{N}$, and N is public
  - Branch and prune Algorithm with the help of the recovered trace



CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \, mod \, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = k_1^{-1}(h - r_1.x) mod \, n$ → get private key x
- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $\mathrm{p}.\mathrm{q} = \mathrm{N}$, and N is public
  - Branch and prune Algorithm with the help of the recovered trace
- Single-trace Attack during RSA Key Generation: $d = e^{-1} \, mod \, \lambda(N)$

CopyCat on WolfSSL – Cryptanalysis Results

- Executed each attack 100 times.
- DSA $k^{-1}\ mod\ n$
  - Average 22,000 IRQs
  - 75 ms to iterate over an average of 6,320 steps
- RSA $q^{-1}\ mod\ p$
  - Average 106490 IRQs
  - 365 ms to iterate over an average of 39,400 steps
- RSA $e^{-1}\ mod\ \lambda(N)$
  - $e^{-1}\ mod\ \lambda(N)$
  - Average 230,050 IRQs
  - 800ms to iterate over an average of 81,090 steps
- Experimental traces always match the leakage model in all experiments → Successful single-trace key recovery

How about other Crypto libraries?

- Libgcrypt uses a variant of BEEA
  - Single trace attack on DSA, Elgamal, ECDSA, RSA Key generation
- OpenSSL uses BEEA for computing GCD
  - Single trace attack on RSA Key generation when computing $\gcd(q-1, p-1)$

	Operation (Subroutine)	Implementation	Secret Branch	Exploitable	Computation → Vulnerable Callers	Single-Trace Attack
WolfSSL	Scalar Multiply (wc_ecc_mulmod_ex)	Montgomery Ladder w/ Branches	✔	✔	$(k \times G) \rightarrow$ wc_ecc_sign_hash	✘
	Greatest Common Divisor (fp_gcd)	Euclidean (Divisions)	✔	✘	N/A	N/A
	Modular Inverse (fp_invmod)	BEEA	✔	✔	$(k^{-1} \bmod n) \rightarrow$ wc_DsaSign $(q^{-1} \bmod p) \rightarrow$ wc_MakeRsaKey $(e^{-1} \bmod \Lambda(N)) \rightarrow$ wc_MakeRsaKey	✔ ✔ ✔
Libgcrypt	Greatest Common Divisor (mpi_gcd)	Euclidean (Divisions)	✔	✘	N/A	N/A
	Modular Inverse (mpi_invm)	Modified BEEA [43, Vol II, §4.5.2]	✔	✔	$(k^{-1} \bmod n) \rightarrow$ {dsa,elgamal}.c::sign,_gcry_ecc_ecdsa_sign $(q^{-1} \bmod p) \rightarrow$ generate_{std,fips,x931} $(e^{-1} \bmod \Lambda(N)) \rightarrow$ generate_{std,fips,x931}	✔ ✔ ✔
OpenSSL	Greatest Common Divisor (BN_gcd)	BEEA	✔	✔	$gcd(q-1, p-1) \rightarrow$ RSA_X931_derive_ex	✔
	Modular Inverse (BN_mod_inverse_no_branch)	BEEA w/ Branches	✘	N/A	N/A	N/A
IPP Crypto	Greatest Common Divisor (ippsGcd_BN)	Modified Lehmer's GCD	✔	? ?	$gcd(q-1, e) \rightarrow$ cpIsCoPrime $gcd(p-1, q-1) \rightarrow$ isValidPriv1_rsa	N/A N/A
	Modular Inverse (cpModInv_BNU)	Euclidean (Divisions)	✔	✘	N/A	N/A

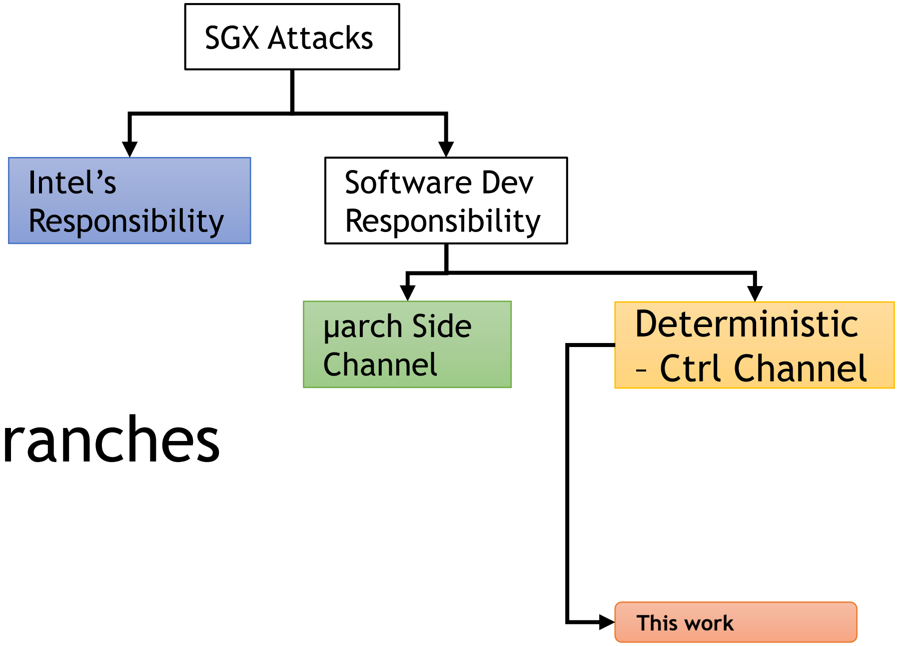
Responsible Disclosure

- WolfSSL fixed the issues in 4.3.0 and 4.4.0
  - Blinding for $k^{-1} \bmod n$ and $e^{-1} \bmod \lambda(N)$
  - Alternate formulation for $q^{-1} \bmod p$: $q^{p-2} \bmod p$
  - Using a constant-time (branchless) modular inverse [11]
- Libgcrypt fixed the issues in 1.8.6
  - Using a constant-time (branchless) modular inverse [11]
- OpenSSL fixed the issue in 1.1.1e
  - Using a constant-time (branchless) GCD algorithm [11]

[11] Bernstein, Daniel J., and Bo-Yin Yang. "Fast constant-time gcd computation and modular inversion." *CHES 2019*.

Conclusion

- Instruction Level Granularity
  - Imbalance number of instructions
  - Leak the outcome of branches
- Fully Deterministic and reliable
  - Millions of instructions tested
  - Attacks match the exact leakage model of branches
- Easy to scale and replicate
  - No reverse engineering of branches and microarchitectural components
  - Tracking all the branches synchronously
- Branchless programming is hard!



Questions?!

Daniel Moghimi
@danielmgmi

https://github.com/jovanbulck/sgx-step