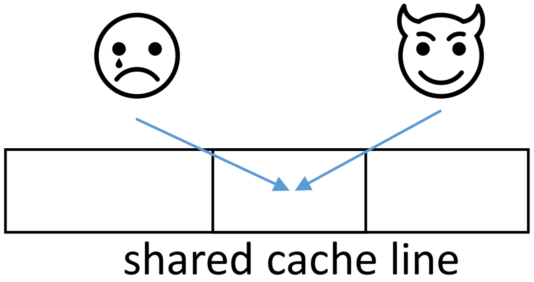
DESIGN, AUTOMATION & TEST IN EUROPE
17 – 19 April 2023 · Antwerp, Belgium
The European Event for Electronic System Design & Test

Cache Side-channel Attacks and Defenses of the Sliding Window Algorithm in TEEs

Zili KOU [1], Sharad Sinha [2], Wenjian HE [1], and Wei ZHANG [1]
[1] Hong Kong University of Science and Technology
[2] Indian Institute of Technology Goa

Cache Side-channel Attacks (SCA)

- **Cache side-channel**
  - Utilize the timing difference between cache hit and miss



Cache hit: ≈ 20 cpu cycles

Cache miss: ≈ 100 cpu cycles

- **Attack scenarios**
  - covert channel communications
  - extracting cryptographic keys (RSA, AES, etc.)
  - speculative execution attacks

SCA-resistant Cryptographic Algorithm

- **RSA**
  - Usually adopt sliding window algorithm, which is initially vulnerable against SCA

Algorithm 1: Sliding Window Algorithm

Input: base b, modulus m, exponent $d = \{d_n ... d_1\}_2$
Output: $b^d \pmod m$

```
precompute multipliers M[2^{w-1}] to M[2^w - 1].
r ← 1, i ← n
while i > w − 1 do
    if d_i = 0 then
        r ← r^2 (mod m)            // modular square r
        i ← i − 1
    else
        repeat w times
            r ← r^2 (mod m)        // modular square r
        end
        j ← {d_i ... d_{i−w+1}}_2
        r ← r × M[j] (mod m)       // modular multiply r by M[j]
        i ← i − w
    end
end
while i > 0 do
    r ← r^2 (mod m)
    i ← i − 1
    if d_i = 1 then
        r ← r × M[1] (mod m)       // process remaining bits
    end
end
return r
```

"Secret-dependent memory loading"

each M[j] locates in different cache set…
when M[j] is loaded, the corresponding cache set is accessed!

- **Many defenses already deployed**
  - To **somewhat** make it secure…

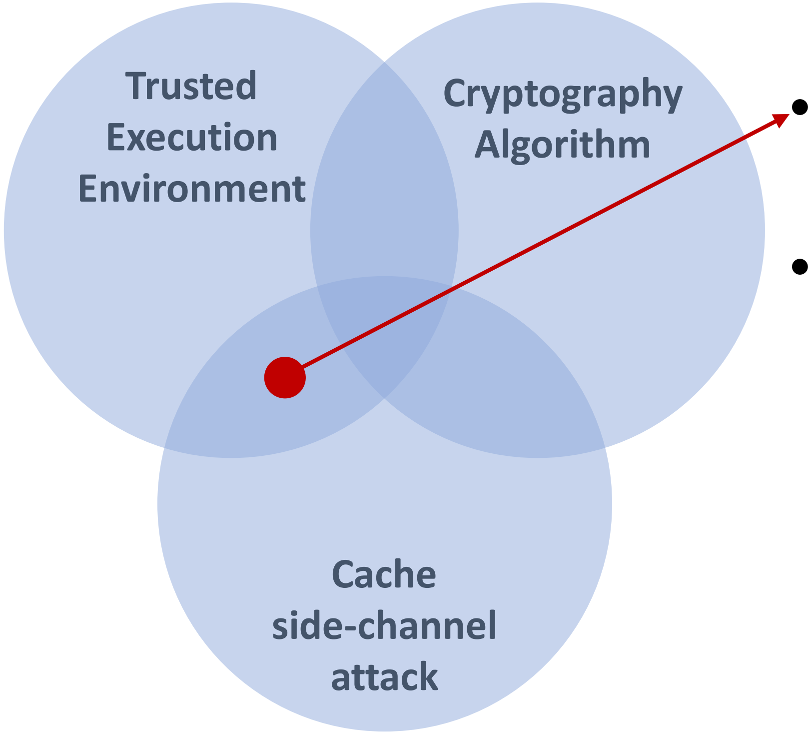
Trusted Execution Environment (TEE)

- **Offers both HW and SW isolation** between the "secure world" and the "rich world"
  - Commercial products: Intel SGX and ARM TrustZone
- **Upgraded protection brings upgraded threat model**
  - TEE is designed to defend even against malicious OS in rich world
  - However, <u>cache side-channel attackers in TEEs</u> are much more capable!

Kernel-privileged attackers

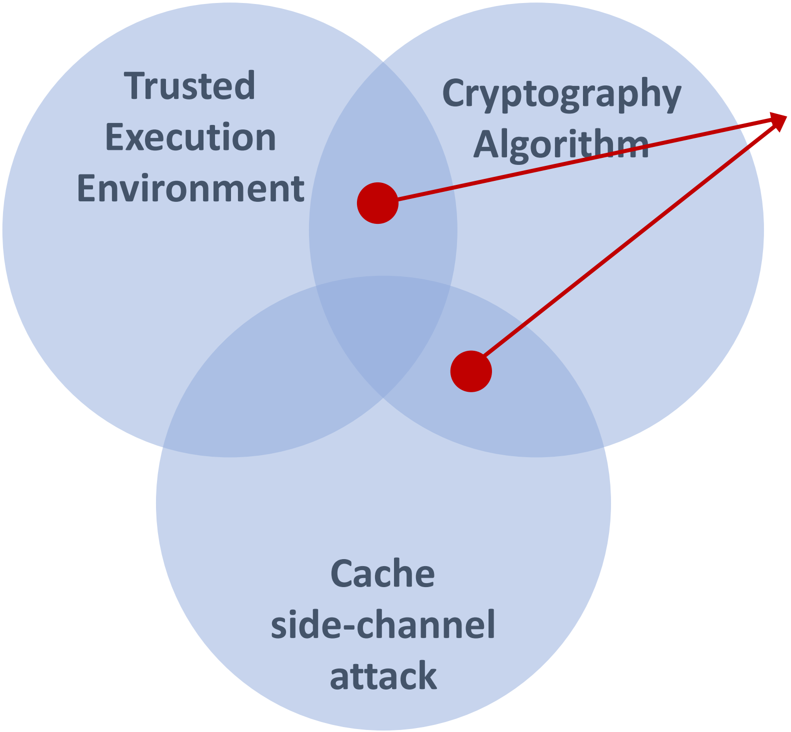
Invoke **interruptions**
read page tables
access performance counters
…

Put It All together!



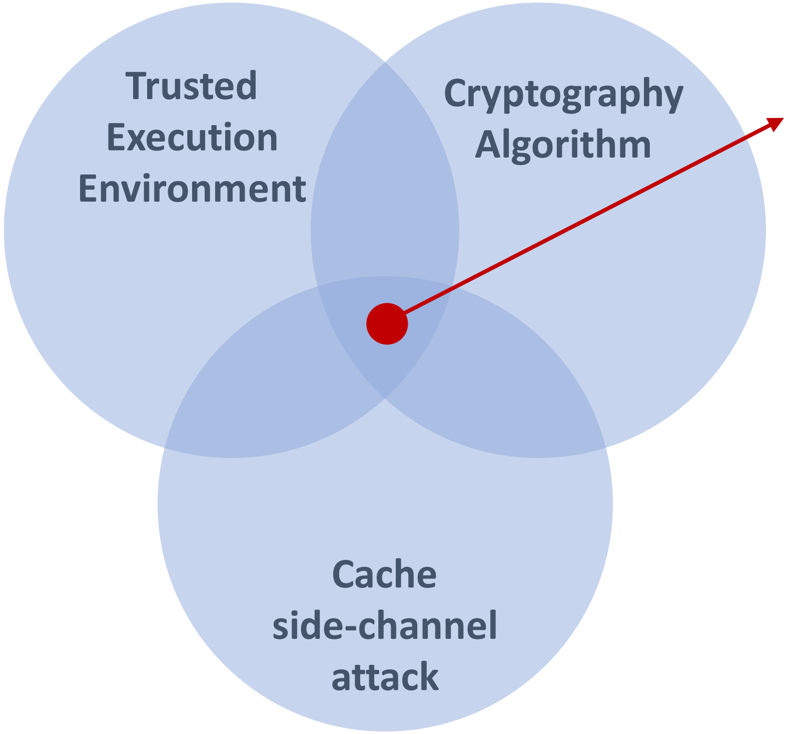
- Kernel privileged cache side-channel attackers are much more capable!
- New vulnerabilities, new attack surfaces are introduced...

Put It All together!



- There existed vulnerabilities in cryptography algorithms, though, they were reported and patched
- <u>Must admit any new attack technique (introduced either by cache SCA or TEE) still threatens the security.</u>

Put It All together!



Are current cryptography algorithms in TEEs really secure under the threat of kernel-privileged cache side-channel attackers?

How and to what extent does kernel-privileged cache side-channel attackers breach the SOTA defenses?

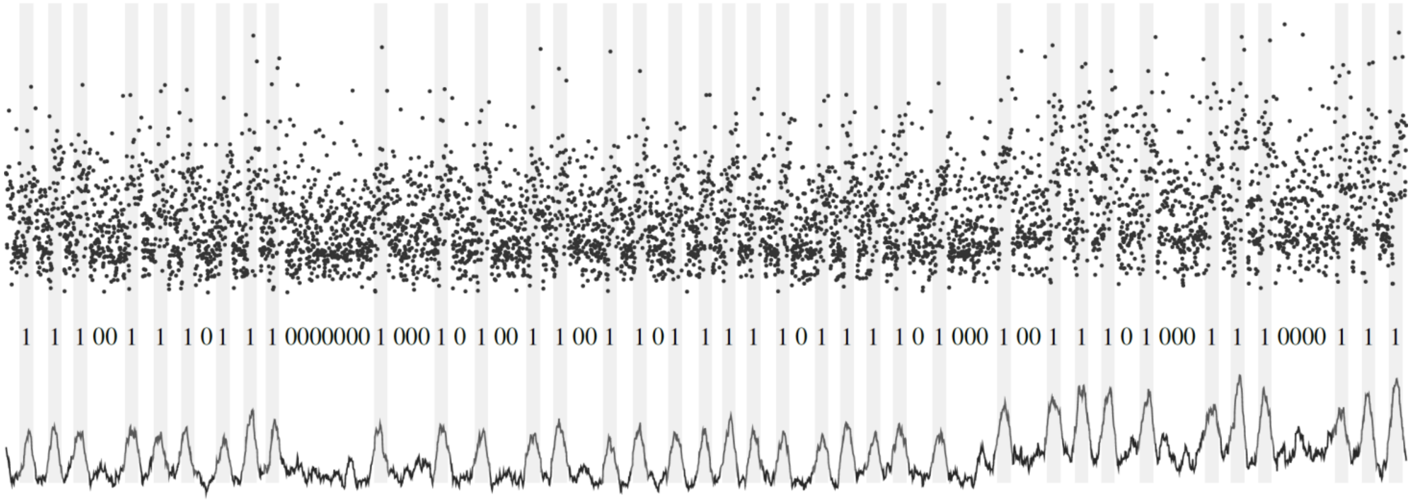
Our Contribution

(Specifically focus on RSA)

- Systematically Scrutinize the sliding window algorithm from the kernel-privileged attacker’s point of view
- Supported by real-world attacks, we reveal that the RSA implementation in the latest Mbed TLS library is still vulnerable
- Propose mitigation and analyze the trade-offs

Attack Naïve Implementations

- Attacker and Victim run simultaneously
- Attacker keeps monitoring the status of cache sets



[1] Access pattern of the target cache line, a higher peak indicates the key bit is ‘1’

Defense 1: Exponent Blinding

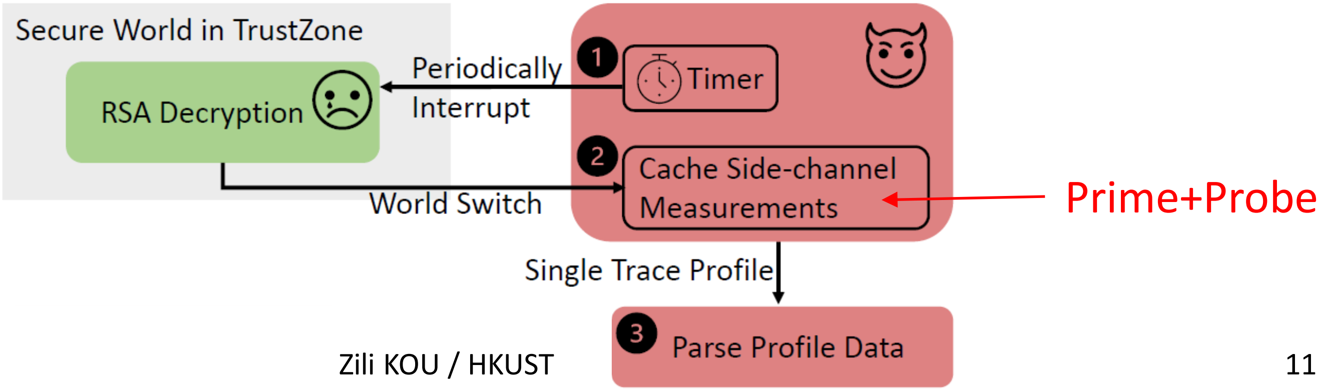
- **Multiple profiling traces** are required by attackers to remove the noise and misalignment (to improve overall confidence of guessing)

- **Exponent Blinding**
  - Randomize the private key for every decryption
  - Different profiling trace detects different key bits, impossible to improve the overall confidence by combining multiple traces!

$m = s^d \pmod{N}$ → $m = s^{d+r(p-1)(q-1)} \pmod{N}$, where r is a random number

Defense 1: Exponent Blinding

- However, TEE cache SCA are capable to recover the full key by a **single trace profiling**
  - SGX-Step[1], Load-Step[2] utilize the interrupt (IRQ) mechanism of OS to achieve much higher precision of cache side-channel attacks.
- We reproduce a similar IRQ-based cache side-channel attack on a real-world board Hikey 960 [3] (an ARM SoC with TrustZone)



Defense 1: Exponent Blinding

- Exponent Blinding is ineffective if attackers can fully recovery the key by a single trace profiling:

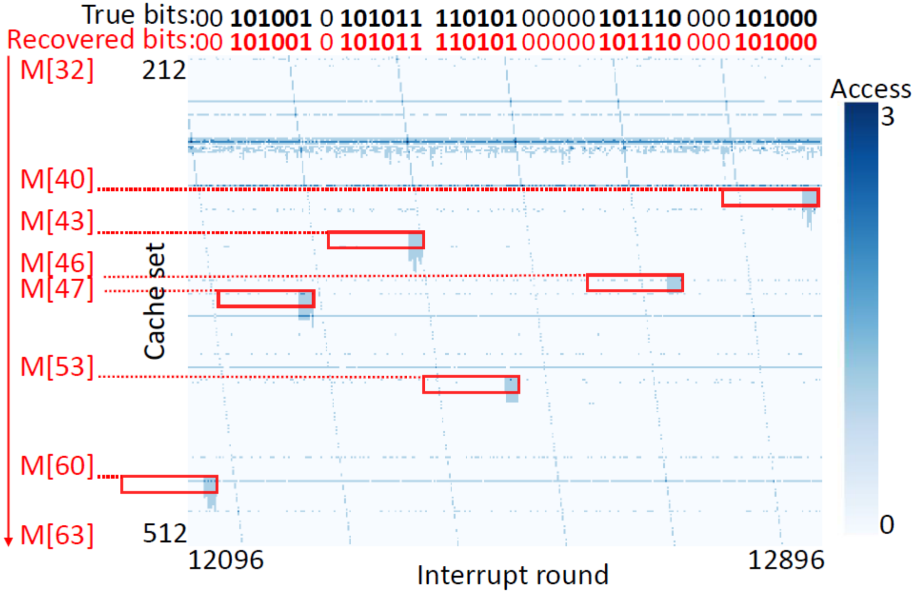


Fig. 1. Our attack against the sliding window algorithm in Mbed TLS 2.26.0.

Y-axis: **which** multiplier M is loaded, determine decode M_j into $\{j\}_2$

X-axis: **when** a multiplier M is loaded determine locations of $\{j_1\}_2, \{j_2\}_2, \ldots$

Defense 2: Multiplier Obfuscation

- Conceal or obfuscate **which** multiplier M is loaded
  - Scatter-gather in “OpenSSL”
    - avoid the accesses of the multipliers at granularity that coarser than cache line
  - Traverse-select in “Libgcrypt” and “Mbed TLS”
    - Allocate a buffer to traverse all multipliers by sequence, while only the target multiplier would finally remain in the buffer.

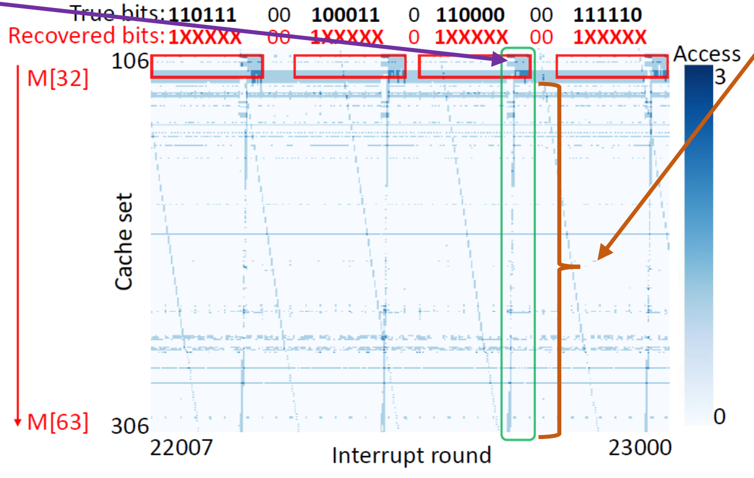


Fig. 2. Our attack against the sliding window algorithm in Mbed TLS 3.1.0.

Defense 2: Multiplier Obfuscation

- However, there are still some hints for key bits
  - Suppose w denotes "window size"
  - 1. $M_j, j \in [2^{w-1}, 2^w - 1]$, i.e., M_j must be decoded into a bit string "**1**xxxx"
  - 2. Between two multipliers, "**0**"s can be filled in

Attackers can partially know the bits!
And the number of known bits are determined by window size!

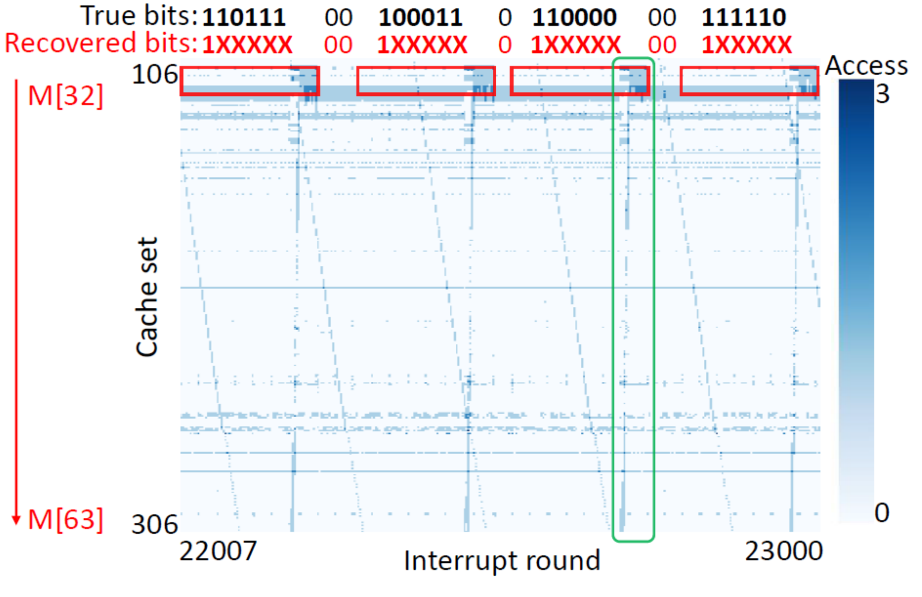
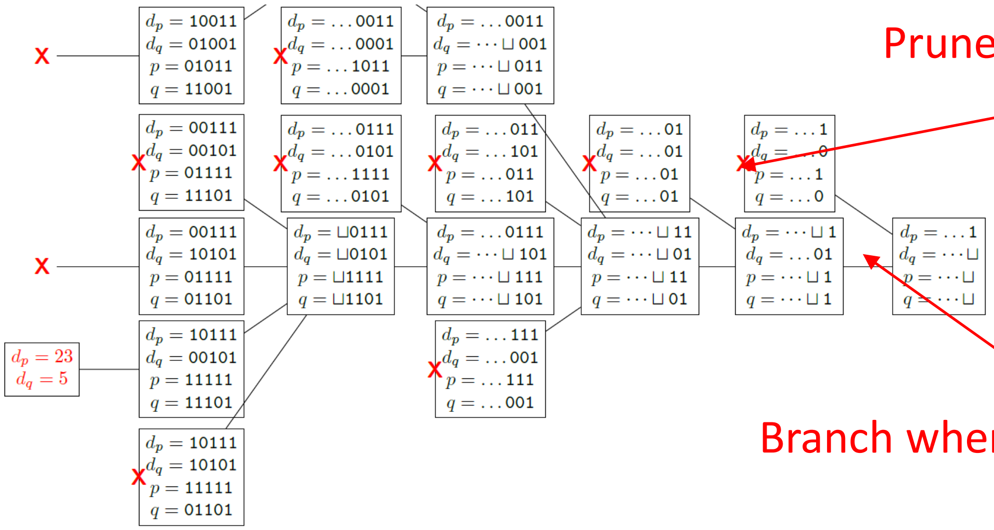


Fig. 2. Our attack against the sliding window algorithm in Mbed TLS 3.1.0.

Defense 2: Multiplier Obfuscation

- Partial Key Recovery
  - Techniques for key recovery when only part of key bits are known, see survey [1]
  - **Branch-and-Prune** works when d_p and d_q are partially known in RSA with CRT
    - Evaluate $(ed_p - 1 + k_p)(ed_q - 1 + k_q) == k_p k_q N \pmod{2^i}$ bit by bit



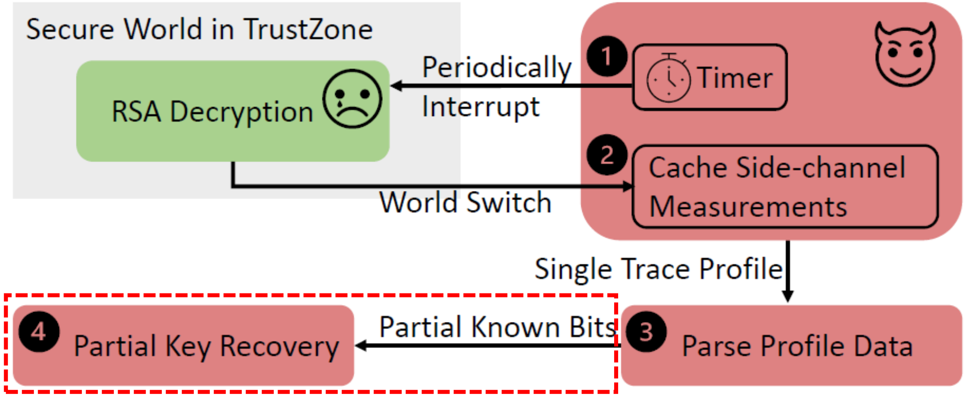
Prune when evaluation failed

Branch when bits are unknown

Visualization sample given by [1]

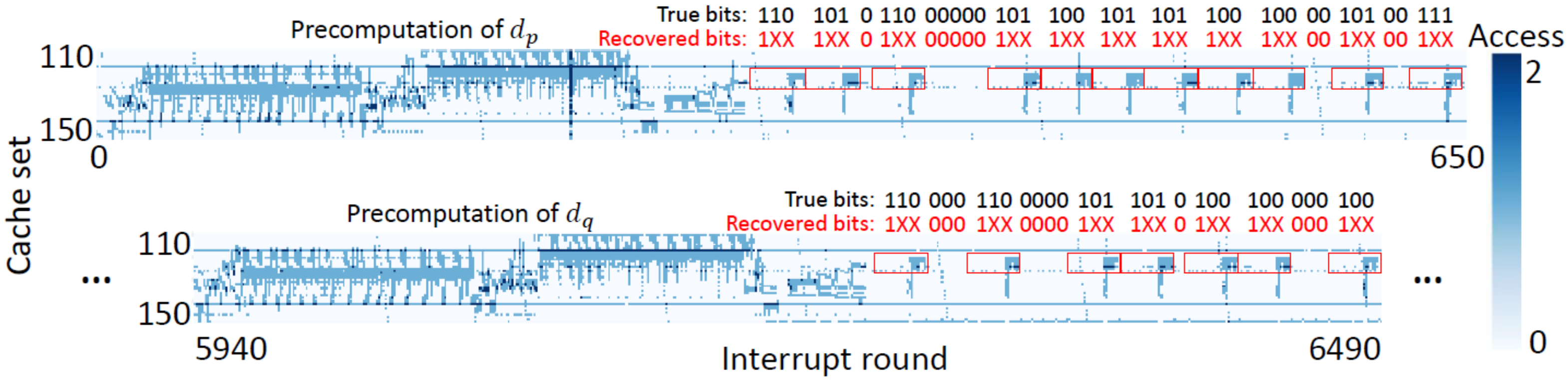
Defense 2: Multiplier Obfuscation

- Our framework for kernel-privileged cache side-channel attacker
  - Attack “Mbed TLS 3.1.0”, latest at the time of paper writing
  - Applied defenses: exponent blinding + multiplier obfuscation



Defense 2: Multiplier Obfuscation

- Single trace cache profiling



- Results
  - Heuristically, branch-and-prune works when more than 50% bits are known
  - When windows size is smaller than 4, the private key is leaked!

TABLE II
EXPERIMENT RESULTS.

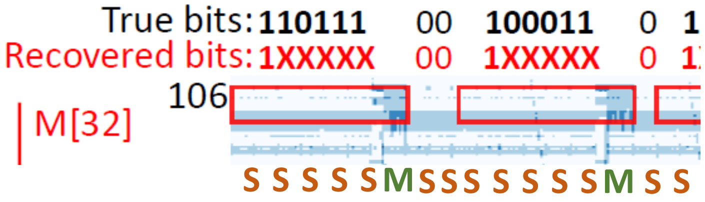
Window Size	Interrupt Round	Partial Known Bits	Execution Time
$w = 5$	10564	34.6%	10.5 s + 3 s + > 48 h
$w = 4$	11869	40.5%	11.7 s + 3 s + > 48 h
$w = 3$	12967	50.8%	12.6 s + 3 s + 0.6 s
$w = 2$	13294	66.8%	13.0 s + 3 s + 0.03 s
$w = 1$	16630	100%	15.2 s + 3 s + 0 s

Defense 3: Square&Multiply Obfuscation

- Need to conceal **both** <u>when</u> and <u>which</u> multiplier is loaded!
- Multiplier is loaded to do a multiplication after *w square operations*

Algorithm 1: Sliding Window Algorithm

Input: base b, modulus m, exponent $d = \{d_n \dots d_1\}_2$
Output: $b^d \pmod m$
precompute multipliers $M[2^{w-1}]$ to $M[2^w - 1]$.
$r \leftarrow 1$, $i \leftarrow n$
while $i > w - 1$ **do**
  if $d_i = 0$ **then**
    $r \leftarrow r^2 \pmod m$ *// modular square r*
    $i \leftarrow i - 1$
  else
    repeat w **times**
      $r \leftarrow r^2 \pmod m$ *// modular square r*
    end
    $j \leftarrow \{d_i \dots d_{i-w+1}\}_2$
    $r \leftarrow r \times M[j] \pmod m$ *// modular multiply r by M[j]*
    $i \leftarrow i - w$
  end
end

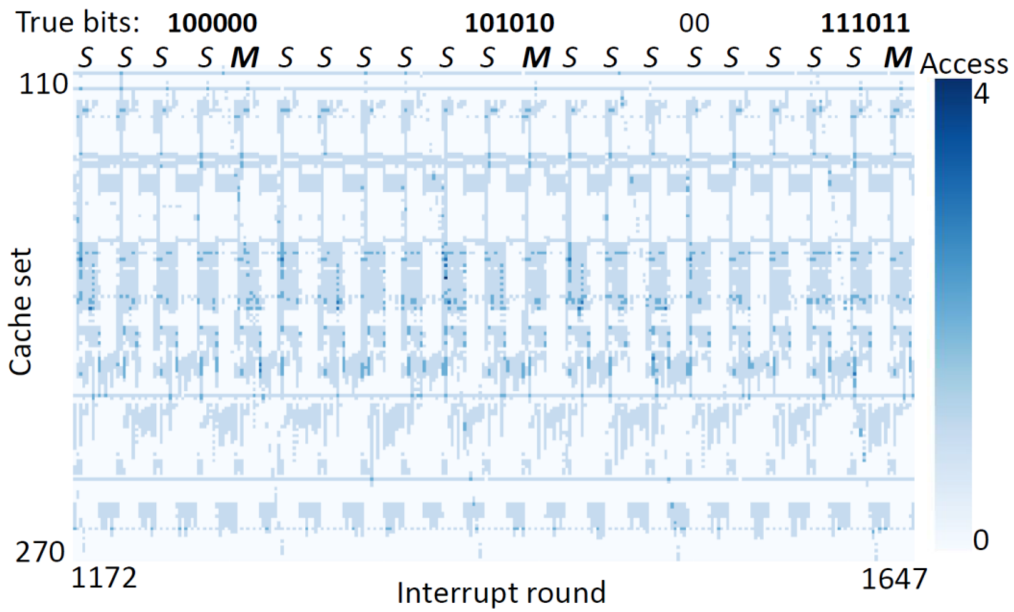


S: square operation
M: multiply operation

- Idea
  - Always traverse all multipliers **regardless** of square and multiplication!

Defense 3: Square&Multiply Obfuscation

- Single trace profiling
  - No longer see any hint of key bit



Defense 3: Square&Multiply Obfuscation

- However, performance degrades a lot
  - Square appears more frequently than multiplication
    - Much more unnecessary memory loadings!

- We naively implemented Square&Multiply Obfuscation on Mbed TLS:

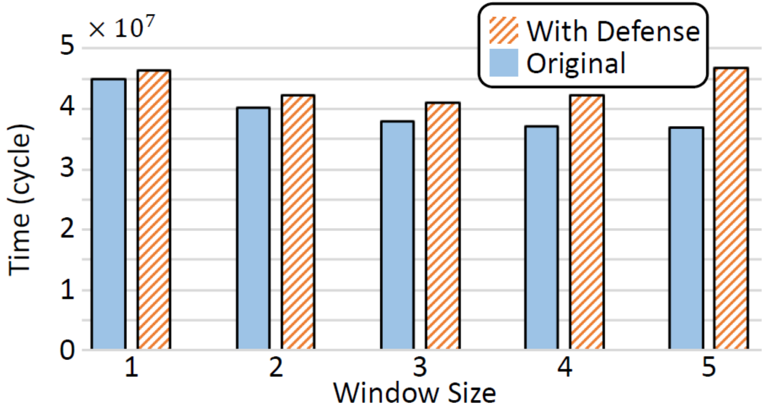


Fig. 6. The Execution time of the sliding window algorithm of Mbed TLS 3.1.0, with or without the square&multiply obfuscation applied.

Conclusion

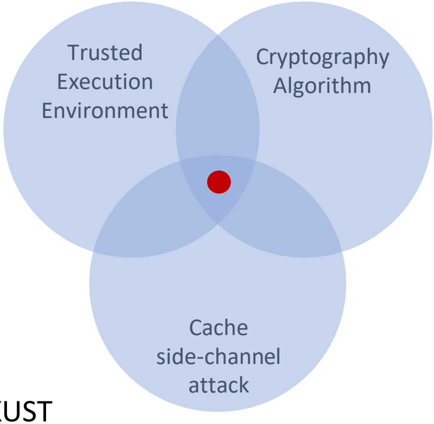
- Defense status of cryptography libraries

TABLE I
THREE TYPES OF DEFENSES IN CRYPTOGRAPHY LIBRARIES.

Cryptography Library	Exponent Blinding	Multiplier Obfuscation	Square&Multiply Obfuscation
Libgcrypt 1.10.1	✓	✓	✓
OpenSSL 3.1.0		✓	
WolfSSL 5.3.0*			
Mbed TLS 2.26.0	✓		
Mbed TLS 3.1.0	✓	✓	

*designed to be lightweight and portable.

- Our practical attack
  - Recognized and patched by Mbed TLS community
  - Assigned CVE-2021-46392 as the public identifier
- Discussion?
  - Never be too cautious…
    - for potential attack surfaces



Thanks for listening!

Page 9:

[1] Schwarz, Michael, et al. "Malware guard extension: Using SGX to conceal cache attacks." International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, Cham, 2017.

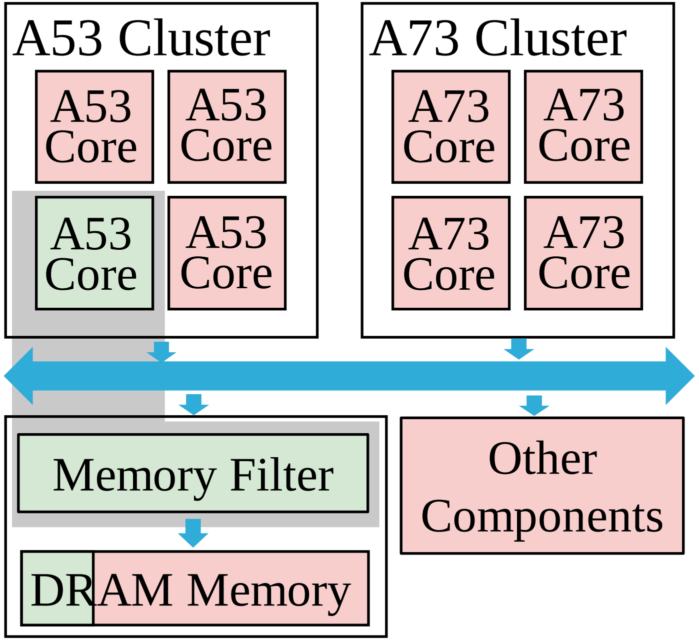
Page 11:

[1] Van Bulck, Jo, Frank Piessens, and Raoul Strackx. "SGX-Step: A practical attack framework for precise enclave execution control." Proceedings of the 2nd Workshop on System Software for Trusted Execution. 2017.

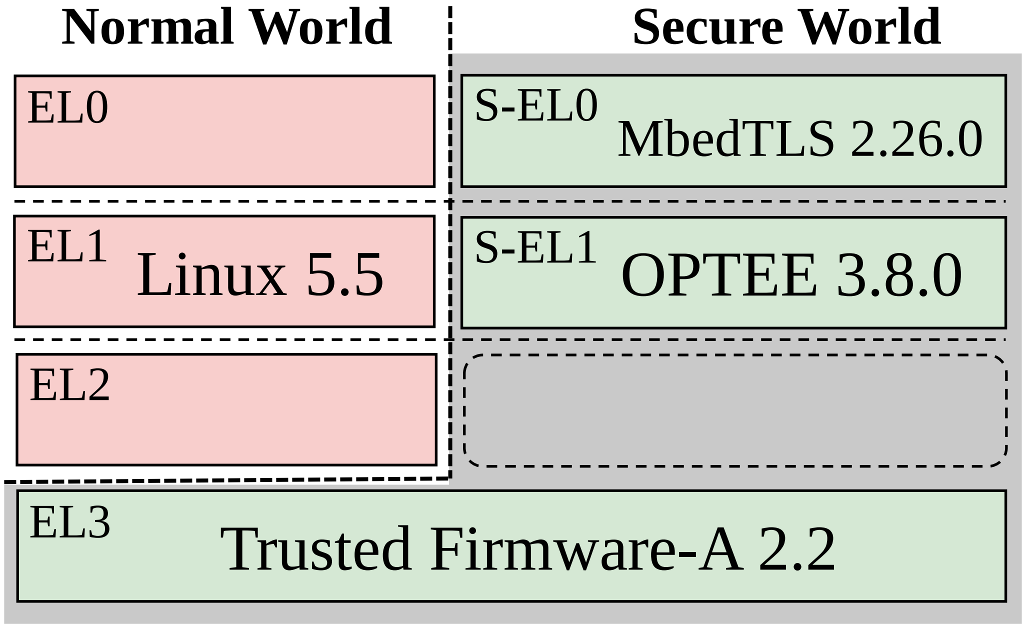
[2] Kou, Zili, et al. "Load-Step: A Precise TrustZone Execution Control Framework for Exploring New Side-channel Attacks Like Flush+ Evict." 2021 58th ACM/IEEE Design Automation Conference (DAC). IEEE, 2021.

[3] Hikey 960, https://www.96boards.org/product/hikey960/

ARM TrustZone



Hardware

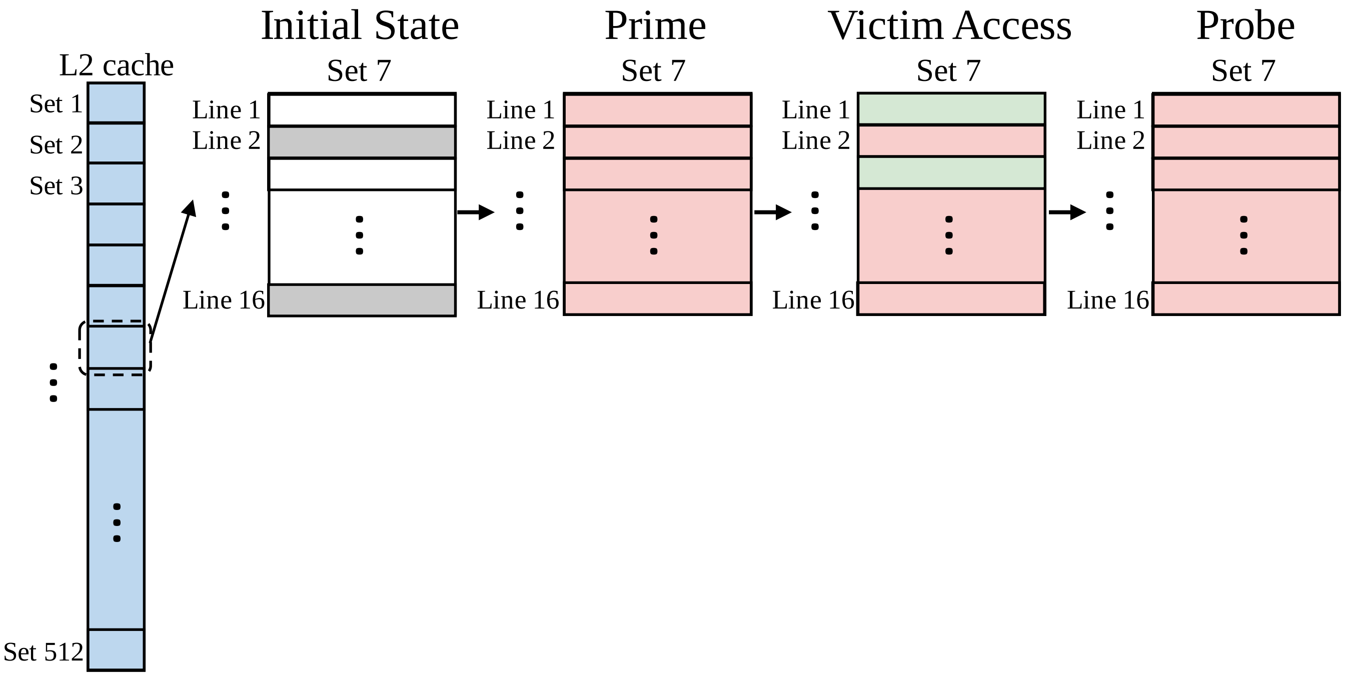


Software

Partial Key Recovery

Scheme	Secret information	Bits known	Technique	Section
RSA	$p \geq 50\%$ most significant bits		Coppersmith's method	§4.2.2
RSA	$p \geq 50\%$ least significant bits		Coppersmith's method	§4.2.3
RSA	p middle bits		Multivariate Coppersmith	§4.2.4
RSA	p multiple chunks of bits		Multivariate Coppersmith	§4.2.4
RSA	$> \log\log N$ chunks of p		Open problem	
RSA	$d \bmod (p-1)$ MSBs		Coppersmith's method	§4.2.7
RSA	$d \bmod (p-1)$ LSBs		Coppersmith's method	§4.2.7 and §4.2.3
RSA	$d \bmod (p-1)$ middle bits		Multivariate Coppersmith	§4.2.7 and §4.2.4
RSA	$d \bmod (p-1)$ chunks of bits		Multivariate Coppersmith	§4.2.7 and §4.2.4
RSA	d most significant bits		Not possible	§4.2.8
RSA	$d \geq 25\%$ least significant bits		Coppersmith's method	§4.2.9
RSA	$\geq 50\%$ random bits of p and q		Branch and prune	§4.3.1
RSA	$\geq 50\%$ of bits of $d \bmod (p-1)$ and $d \bmod (q-1)$		Branch and prune	§4.3.2
(EC)DSA	MSBs of signature nonces		Hidden Number Problem	§5.2
(EC)DSA	LSBs of signature nonces		Hidden Number Problem	§5.2
(EC)DSA	Middle bits of signature nonces		Hidden Number Problem	§5.2
(EC)DSA	Chunks of bits of signature nonces		Extended HNP	§5.2.4
EC(DSA)	Many bits of nonce		Scales poorly	
Diffie-Hellman	Most significant bits of shared secret g^{ab}		Hidden Number Problem	§6.2
Diffie-Hellman	Secret exponent a		Pollard kangaroo method	§6.3
Diffie-Hellman	Chunks of bits of secret exponent		Open problem	

Prime+Probe



Prime+Probe

Difference:
2 cache lines are refilled
Measured by:
Time of loading
Read PMU of Core