Cache-Access Pattern Attack on Disaligned AES T-Tables

Raphael Spreitzer and Thomas Plos

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a, A-8010 Graz, Austria

{raphael.spreitzer, thomas.plos}@iaik.tugraz.at

Outline

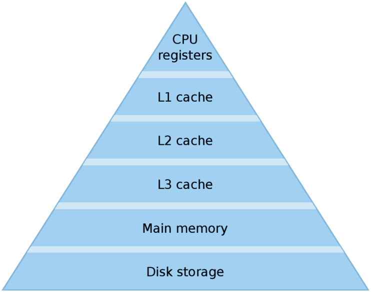
- Introduction and motivation
- Preliminaries
    - CPU caches
    - Advanced Encryption Standard
    - Aligned and disaligned T-tables
- Attack concept of the cache-access pattern attack
- Practical results on a Google Nexus S
- Conclusion

Introduction

- Motivation
  - Wide-spread usage of mobile devices
  - Protection of private information
- Implementation attacks
  - CPU caches are a potential side channel [Koc96, KSWH00]
- Cache attacks on mobile devices?
  - Only testbeds so far, e.g., [BEPW10, GK11, WHS12]
- Our contribution
  - Attack an Android-based Google Nexus S
  - Attack is implemented purely in software
  - Focus on disaligned AES T-tables

CPU Caches

- Memory hierarchy

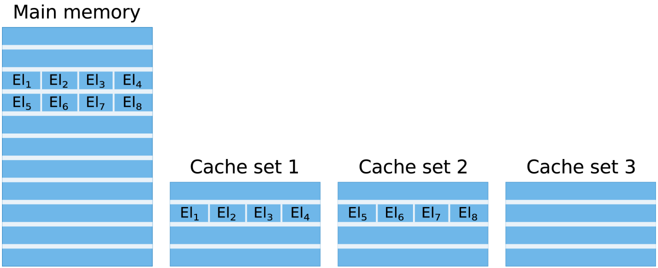


- Problems
  - Memory accesses are not performed in constant time
  - Cache is a shared resource $\longrightarrow$ manipulation

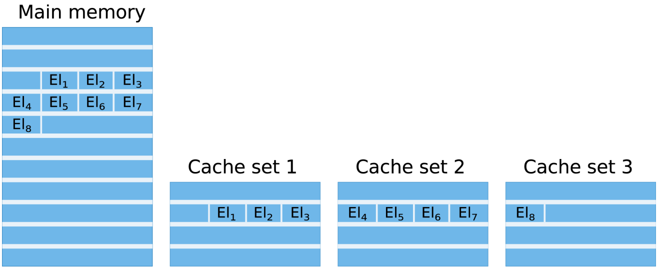
Advanced Encryption Standard

- Block cipher, 128-bit state, 4 round transformations
- Software implementations employ T-tables
- Problems
  - Key-dependent look-up indices $\mathbf{T}\,[\mathbf{p}_i \oplus \mathbf{k}_i]$
  - T-table elements might be within
    - CPU cache
    - Main memory

Aligned AES T-Tables



Disaligned AES T-Tables



ARM Cortex-A8 Processor

- Designed for mobile devices
- Also employs CPU caches
  - Set-associative cache
  - Random-replacement policy
  - Cache-line size of 64 bytes
- Performance monitor registers (*Cycle Count Register*)

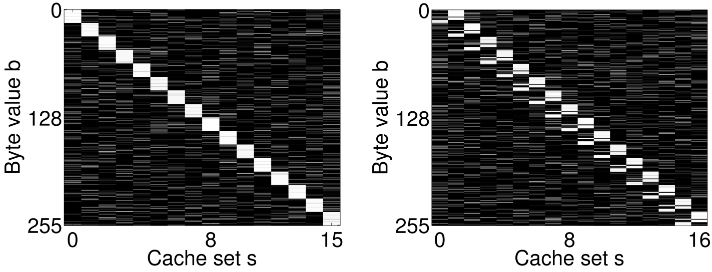
Cache-Access Pattern Attack (1/3)

- Based on the work of Tromer et al. [TOS10]
- Online phase: step 1
- Offline phases: steps 2-4

1) Gather cache-access patterns
   - Assume knowledge of where T-table **T** resides
   - Encrypt a plaintext **p**
   - Evict a specific cache set *s*
   - Measure the encryption time of **p** again
   - Collect timing information for each key byte $\mathbf{k}_i$

Cache-Access Pattern Attack (2/3)

- $\mathbf{s}_i = \mathbf{p}_i \oplus \mathbf{k}_i \longrightarrow \mathbf{k}_i = \mathbf{s}_i \oplus \mathbf{p}_i$
- Plot for a specific key byte (key=`0x0C`)

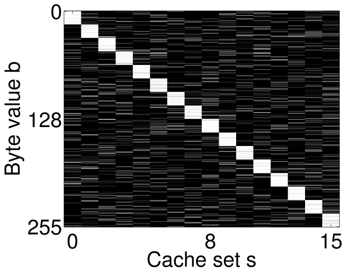


(a) Aligned T-table.

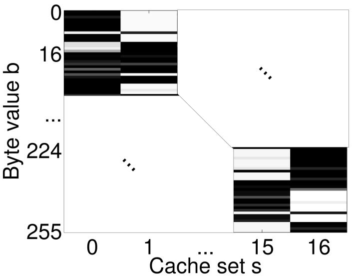
(b) Disaligned T-table.

Cache-Access Pattern Attack (2/3)

- $\mathbf{s}_i = \mathbf{p}_i \oplus \mathbf{k}_i \longrightarrow \mathbf{k}_i = \mathbf{s}_i \oplus \mathbf{p}_i$
- Plot for a specific key byte (key=`0x0C`)



(a) Aligned T-table.



(b) Disaligned T-table.

- Disaligned T-tables leak more information

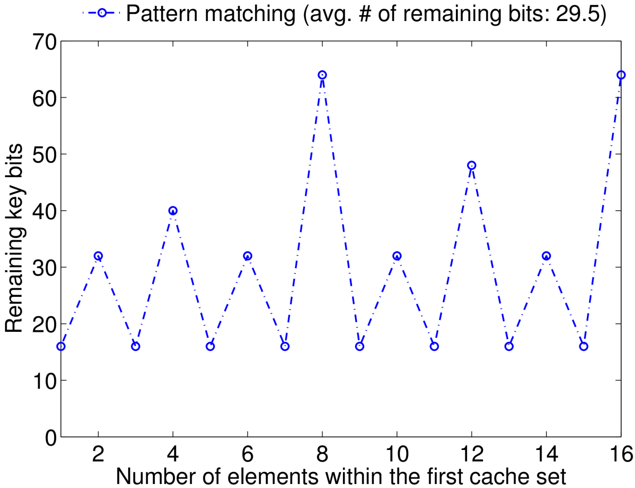
Cache-Access Pattern Attack (3/3)

2) Compute possible cache-access patterns
   - For all possible key bytes and disalignments, for a specific cache set
   - Pattern $\longrightarrow$ possible key candidates
3) Pattern matching and extraction of key candidates
   - Query with cache-access pattern
4) Brute-force key search
   - Sometimes not even necessary

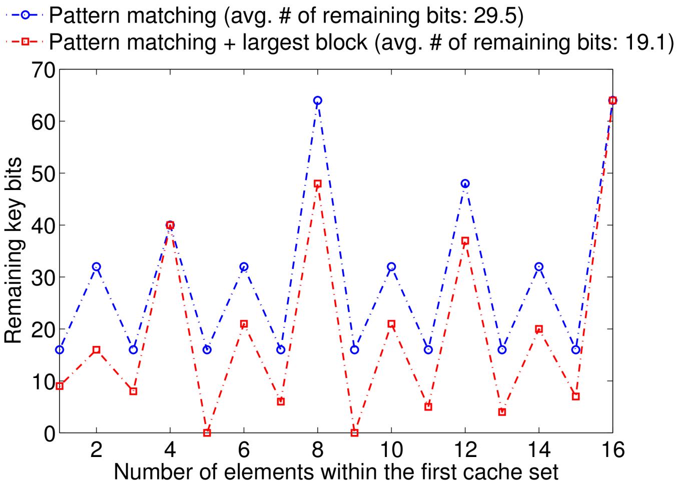
Practical Results(1/3)

- Google Nexus S
- 2^{21} AES encryptions (step 1)
- 40–80 seconds
  - Steps 1-3 (excluding the final remaining key search)
  - Might be reduced even further (few seconds)
- Some disalignments reveal the whole key immediately

Practical Results (2/3)



Practical Results (3/3)



Conclusion

- Access-driven attack on disaligned AES T-tables
- First access-driven attack on ARM Cortex-A series
- Improvement: correct key byte is always within the largest block
- Attack implemented purely in software
- Cache attacks pose a serious threat
- Aligned T-tables reduce the amount of leaked key bits
  - Declare T-tables as *__attribute__(aligned(64))*
  - Only 64 key bits can be recovered immediately

Cache-Access Pattern Attack on Disaligned AES T-Tables

Raphael Spreitzer and Thomas Plos

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a, A-8010 Graz, Austria

{raphael.spreitzer, thomas.plos}@iaik.tugraz.at

Bibliography

[BEPW10] Andrey Bogdanov, Thomas Eisenbarth, Christof Paar, and Malte Wienecke.
Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs.
In *Topics in Cryptology - CT-RSA 2010*, volume 5985 of *LNCS*, pages 235–251. Springer Berlin / Heidelberg, 2010.

[GK11] Jean-François Gallais and Ilya Kizhvatov.
Error-Tolerance in Trace-Driven Cache Collision Attacks.
In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, COSADE 2011, pages 222–232, 2011.

[Koc96] Paul C. Kocher.
Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.
In *Advances in Cryptology - CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer Berlin / Heidelberg, 1996.

[KSWH00] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall.
Side Channel Cryptanalysis of Product Ciphers.
Journal of Computer Security, 8(2–3):141–158, 2000.

[TOS10] Eran Tromer, Dag Arne Osvik, and Adi Shamir.
Efficient Cache Attacks on AES, and Countermeasures.
Journal of Cryptology, 23(1):37–71, 2010.

[WHS12] Michael Weiß, Benedikt Heinz, and Frederic Stumpf.
A Cache Timing Attack on AES in Virtualization Environments.
In *Financial Cryptography and Data Security*, volume 7397 of *LNCS*, pages 314–328. Springer Berlin Heidelberg, 2012.

Backup Slide 1

Correct key byte is always within the largest block of the first set

- Let α be the number of look-up indices s_i within the first cache set
- Assume the key $k_i = \texttt{0x0C}_{16} = \texttt{1100}_2$

α	$\lceil \log_2 \alpha \rceil$	s_i		$p_i = s_i \oplus k_i$	
1	0	0000	0	1100	**12**
2	1	0001	1	1101	13
3	2	0010	2	1110	14
4	2	0011	3	1111	15
5	3	0100	4	1000	8
6	3	0101	5	1001	9
7	3	0110	6	1010	10

- $p_i = s_i \oplus k_i$
  - Upper $8 - \lceil \log_2 \alpha \rceil$ bits flip to the same state
  - Lower $\lfloor \log_2 \alpha \rfloor$ bits form the largest group of $2^{\lfloor \log_2 \alpha \rfloor}$ indices, with 0 always being part of this group

Backup Slide 2

Correct key byte is within the largest block of the last set

- Let α be the number of look-up indices s_i within the last cache set
- Assume the key $k_i = \texttt{0x0C}_{16} = \texttt{00001100}_2$

α	$\lceil \log_2 \alpha \rceil$	s_i		$p_i = s_i \oplus k_i$		$k_i = p_i \oplus \texttt{0xFF}$
1	0	11111111	255	11110011	243	**12**
2	1	11111110	254	11110010	242	13
3	2	11111101	253	11110001	241	14
4	2	11111100	252	11110000	240	15
5	3	11111011	251	11110111	247	8
6	3	11111010	250	11110110	246	9
7	3	11111001	249	11110101	245	10

- $p_i = s_i \oplus k_i$
  - Upper $8 - \lceil \log_2 \alpha \rceil$ bits flip to the same state
  - Lower $\lfloor \log_2 \alpha \rfloor$ bits form the largest group of $2^{\lfloor \log_2 \alpha \rfloor}$ indices, with 0 always being part of this group
  - XOR `0xFF` since we attack the last look-up index

Backup Slide 3

How to determine the location of the T-tables

- Assume knowledge of the number of cache sets
- Allocate a data structure (3 times the cache size)
- Encrypt random plaintext **p**
- Evict a specific cache set
- Measure encryption time of the same plaintext **p**
- Search for the longest sequence of cache sets where the performance decreases