On the Applicability of Time-Driven Cache Attacks on Mobile Devices*,**

Raphael Spreitzer and Thomas Plos

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{raphael.spreitzer,thomas.plos}@iaik.tugraz.at

Abstract. Cache attacks are known to be sophisticated attacks against cryptographic implementations on desktop computers. Recently, investigations of such attacks on specific testbeds with processors that are employed in mobile devices have been done. In this work we investigate the applicability of Bernstein's [2] timing attack and the cache-collision attack by Bogdanov *et al.* [4] in real environments on three state-of-the-art mobile devices: an *Acer Iconia A510*, a *Google Nexus S*, and a *Samsung Galaxy SIII*. We show that T-table based implementations of the Advanced Encryption Standard (AES) leak enough timing information on these devices in order to recover parts of the used secret key using Bernstein's timing attack. We also show that systems with a cache-line size larger than 32 bytes exacerbate the cache-collision attack of Bogdanov *et al.* [4].

Keywords: AES, ARM Cortex-A series processors, time-driven cache attacks, cache-collision attacks.

1 Introduction

Cache attacks are a specific form of implementation attacks that focus on the exploitation of variations within the execution time of a cryptographic algorithm due to different access times within the memory hierarchy. For instance, the central-processing unit (CPU) is able to access data within the CPU cache an order of magnitude faster than data within the main memory. Cache attacks can be separated into three categories: (1) *time-driven attacks*, (2) *access-driven attacks*, and (3) *trace-driven attacks*. Time-driven attacks [2] exploit the overall encryption time and, thus, require many measurement samples. In contrast, access-driven attacks [6, 12] and trace-driven attacks [3] focus on more fine-grained information leakage and require far less measurement samples than time-driven attacks. However, access-driven attacks and trace-driven attacks require sophisticated knowledge about the hardware and the software under attack.

Today's mobile devices also employ CPU caches and investigations of implementation attacks—and cache attacks in particular—are necessary in order to ensure the

* An extended version of this paper can be found at [11].

** This work has been supported by the Austrian Science Fund (FWF) under grant number TRP 251-N23 (Realizing a Secure Internet of Things - ReSIT). Furthermore, it has been supported by the Austrian Research Promotion Agency (FFG) and the Styrian Business Promotion Agency (SFG) under grant number 836628 (SeCoS).

J. Lopez, X. Huang, and R. Sandhu (Eds.): NSS 2013, LNCS 7873, pp. 656–662, 2013.
The final publication is available at link.springer.com.

user's privacy and security on these devices. Especially due to the wide-spread usage of mobile devices, *e.g.*, smartphones and tablet computers, and their manifold application scenarios, security and privacy issues on these devices are of utmost importance. Additional applications and widgets allow for further enhancements of capabilities on these devices and potentially contain security-relevant algorithms. Since these algorithms might be vulnerable to implementation attacks, the investigation of such attacks shall raise the awareness of implementation attacks among developers, leading to more secure systems in general. However, until recently these attacks mainly focused on desktop machines [2, 6, 8, 12]. Only minor efforts have been made towards the investigation of these attacks on mobile devices [10], where mainly testbeds simulating specific mobile-device configurations [4, 5, 13] have been used.

In 2010, Bogdanov *et al.* [4] proposed a cache-collision attack by exploiting collisions between consecutive encryptions of pairs of chosen plaintexts. The attack environment was an ARM9 board running the AES implementation of OpenSSL [9] which was queried via an Ethernet interface. Gallais and Kizhvatov [5] investigated trace-driven cache attacks on an ARM7 microcontroller. In 2012, Weiß *et al.* [13] investigated the applicability of Bernstein's [2] time-driven cache attack on a Beagleboard employing an ARM Cortex-A8 processor, running the *Fiasco.OC microkernel* and the *L4Re runtime environment* on top. Nevertheless, they claim that further research regarding the impact of real noise is necessary.

In this work, we focus on the investigation of time-driven cache attacks in more realistic environments by analyzing the applicability of the attack by Bernstein [2] and the attack by Bogdanov *et al.* [4] on three Android-based mobile devices. We aim at analyzing whether T-table based implementations of the Advanced Encryption Standard (AES) on state-of-the-art Android-based mobile devices, *i.e.*, featuring a full-blown operating system, leak enough timing information to deduce the used secret key.

The presented paper is organized as follows. Section 2 outlines the required preliminaries and illustrates the basic concepts of the two investigated cache attacks. We state the main findings regarding the analysis of these two attacks on mobile devices in Section 3. Finally, we conclude this work in Section 4.

2 Background Knowledge

In this section we introduce the necessary preliminaries and outline the basic concepts of the conducted attacks.

Advanced Encryption Standard. The Advanced Encryption Standard (AES) [7] is a block cipher operating on a 128-bit state denoted as a series of bytes $\mathbf{S} = \{\mathbf{s}_0, \ldots, \mathbf{s}_{15}\}$. The AES consists of four round transformations: *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. Since *SubBytes* and *MixColumns* perform complex mathematical operations, software implementations usually operate on look-up tables $\mathbf{T}$ which hold precomputed values for these two round transformations. The fact that these look-up tables—each consisting of 256 4-byte values—are partially cached during the encryption and the fact that the look-up indices are key dependent, *i.e.*, $\mathbf{s}_i = \mathbf{p}_i \oplus \mathbf{k}_i$ within the first round, leads to AES implementations which are susceptible to cache attacks.

Table 1. Detailed device specifications for the three mobile devices under attack.

	Acer Iconia A510	**Google Nexus S**	**Samsung Galaxy SIII**
Processor	Cortex-A9	Cortex-A8	Cortex-A9
Processor implementation	Nvidia Tegra 3 Quad 1.4 GHz	Exynos 3 Single 1 GHz	Exynos 4 Quad 1.4 GHz
L1 cache size	32 KB	32 KB	32 KB
L1 cache associativity	4 way	4 way	4 way
L1 cache-line size	32 byte	64 byte	32 byte
L1 cache sets	256	128	256
Operating system	Android 4.0.4	Android 2.3.4	Android 4.0.4

ARM Architecture. The ARM Cortex-A series processors [1] are employed in many modern mobile devices, *e.g.*, smartphones and tablet computers. Processors of this series typically employ a 4-way set-associative data cache with a cache-line size of either 32 or 64 bytes and a total size of 32 KB. The crucial difference between most desktop CPU caches and ARM CPU caches is the mechanism to evict a cache line from a cache set. While desktop CPU caches usually employ a deterministic replacement policy, ARM processors usually evict a cache line randomly. Since time-driven cache attacks rely on statistical analysis of measurement samples, the random replacement policy might have a negative impact on the number of required measurement samples.

Time-Driven Cache Attacks. The basic idea of these attacks is to exploit the overall execution time of cryptographic primitives employing precomputed look-up tables.

Timing Attack. In 2005, Bernstein [2] suggested a time-driven cache attack against the AES T-table implementation of OpenSSL [9]. The attack is based on the assumption that the overall encryption time correlates with the timing leakage of specific look-up operations. By correlating measurement samples of encryptions under a known key $\mathbf{K}$ with measurement samples under an unknown key $\tilde{\mathbf{K}}$ one tries to deduce the used secret key. For further details about this attack we refer to [2, 8].

Collision Attack. Cache-collision attacks exploit collisions between look-up indices of intermediate state bytes. Given information about such collisions an attacker tries to infer relations between key bytes. Bogdanov *et al.* [4] suggested to choose pairs of plaintexts $(\mathbf{P}_1, \mathbf{P}_2)$ such that five S-Box or T-table look-ups within the encryption of $\mathbf{P}_2$ collide with S-Box or T-table look-ups of $\mathbf{P}_1$. This is what they call a *wide collision*. The encryption time of the plaintext $\mathbf{P}_2$ is used as an indicator to determine whether such a *wide collision* occurred.

3 Analysis and Practical Results

We launch the above outlined attacks on state-of-the-art Android-based mobile devices: (1) an *Acer Iconia A510 tablet computer*, (2) a *Google Nexus S*, and (3) a *Samsung Galaxy SIII*. Table 1 provides a detailed specification of these devices. The attack runs in unprivileged mode, though we need root access once after powering up the device to grant unprivileged applications access to the cycle-count register. As already suggested

Table 2. Sample output of Bernstein's time-driven cache attack on a *Samsung Galaxy SIII*.

# of key candidates	Key byte	Possible values									
3	0	**b5**	b4	b8							
125	1	00	a2	be	c2	b8	1d	f6	...	**93**	...
165	2	87	03	51	17	1b	1f	c7	...	**11**	...
⋮	⋮	⋮									
2	13	**4a**	4b								
40	14	6a	7a	7b	74	61	7c	64	**6b**	78	...
2	15	9c	**9d**								

by Neve [8] we perform the attacks within a single application, *i.e.*, the attack application performs the AES encryption (standard C implementation of OpenSSL [9]) and computes the relevant information. In this section we briefly state the main findings of the conducted attacks on the three mobile devices.

3.1 Timing Attack

The timing attack by Bernstein [2] requires measurement samples under a known key $\mathbf{K}$ and an unknown key $\tilde{\mathbf{K}}$. Gathering 2^{30} measurement samples under the known key $\mathbf{K}$ and the unknown key $\tilde{\mathbf{K}}$ takes about 6 hours on the *Google Nexus S* and about 4 hours on the *Acer Iconia A510* and the *Samsung Galaxy SIII*. Table 2 illustrates an excerpt of a sample output on the *Samsung Galaxy SIII* after correlating the measurement results. The columns state the number of remaining key candidates, the index of the corresponding key byte and all possible key bytes with the correct key marked in bold. A series of dots illustrates omitted key bytes. The possible key candidates are sorted according to the computed correlation and, thus, the position also indicates the probability of the corresponding key candidate being the correct key byte. One clearly observes that timing information is leaking, though the number of remaining key bits is still too large for an exhaustive key search. In order to retrieve more key bits one might apply the second-round attack as suggested by Neve [8]. From Table 2 we also observe that for some key bytes the number of possible key candidates has been reduced significantly, *e.g.*, to only 2 key candidates. However, some key bytes have not been reduced significantly.

Table 3 lists two runs with the lowest number of remaining key bits for each of the three mobile devices. The number of generated measurement samples seems to be a crucial part. For the same number of measurement samples we observed runs where the key space was not reduced significantly and runs where the key space was reduced too much, *i.e.*, the correct key byte was not present among the possible key candidates anymore in which case the attack would fail. Thus, more measurement samples do not necessarily yield better results in terms of remaining key bits.

Timing variations within the encryption time only occur if cache evictions happen frequently. Bernstein [2] generated the required cache evictions by sending data of different length to the server and the server in turn performed memory accesses on the transmitted data. We also launched this attack in a more realistic scenario where we mounted the attack while watching videos or while watching an image slideshow on the mobile devices. Nevertheless, running external applications on purpose did not leak

Table 3. Results of Bernstein's time-driven attack on the three mobile devices.

Device	Samples in		Remaining key space
	study phase	attack phase	
Acer Iconia A510	2^{30}	2^{27}	73 bits
	2^{30}	2^{29}	78 bits
Google Nexus S	2^{30}	2^{29}	65 bits
	2^{29}	2^{28}	69 bits
Samsung Galaxy SIII	2^{30}	2^{29}	58 bits
	2^{30}	2^{30}	61 bits

more information and, hence, did not reduce the key space further. We conclude that these external applications either affected the wrong cache sets or lead to uncontrollable noise that corrupted the timing measurements. Furthermore, on multi-core devices, *e.g.*, the *Acer Iconia A510* and the *Samsung Galaxy SIII*, the two applications might be executed on different cores. Thus, a fairly realistic approach would be to wrap the attack in a fine-grained application and to control the memory accesses and potentially also the number of active cores within this application.

3.2 Collision Attack

Bogdanov *et al.* [4] aim at recovering 4-byte key chunks at once. After recovering all four potential 4-byte key chunks these are enumerated exhaustively in order to recover the whole key. Recovering 4-byte key chunks at once requires at least four real *wide collisions* between chosen pairs of plaintexts ($\mathbf{P}_1$, $\mathbf{P}_2$). However, given the overall encryption time of multiple plaintexts the critical part of this attack is to distinguish encryptions that lead to *wide collisions* from encryptions that do not lead to *wide collisions*. This in turn means that a high expectation rate of *false positives*[1] must be overcome by taking more plaintexts—that possibly lead to *wide collisions*—into consideration.

Figure 1 illustrates two histograms of encryption times of five plaintexts that lead to *wide collisions* in light gray and five plaintexts that do not lead to wide collisions in dark gray. The presented histograms are based on measurement samples gathered on the ARM Cortex-A8 processor. Due to reasons of noise each of the five chosen plaintexts is encrypted multiple times. In case of the 3-round AES implementation we clearly observe easily separable encryption times for plaintexts which lead to *wide collisions* and plaintexts which do not lead to *wide collisions*. Thus, by taking $n = 4$ plaintexts with the lowest encryption times we might indeed detect 4 real *wide collisions* with a high probability. In contrast, in case of the 7-round AES implementation we observe that the encryption times of these two categories of plaintexts cannot be distinguished anymore. Obviously, the number of false positives increases drastically and, hence, we need to consider a higher number n of plaintexts that possibly lead to *wide collisions*. The n plaintexts are used to find possible candidates of 4-byte subkeys by iterating over all possible 4-byte keys (2^{32}). Bogdanov *et al.* [4] state the number of expected

[1] False positives are diagonal pairs which are supposed to lead to *wide collisions* due to their encryption time, but in fact do not lead to wide collisions.

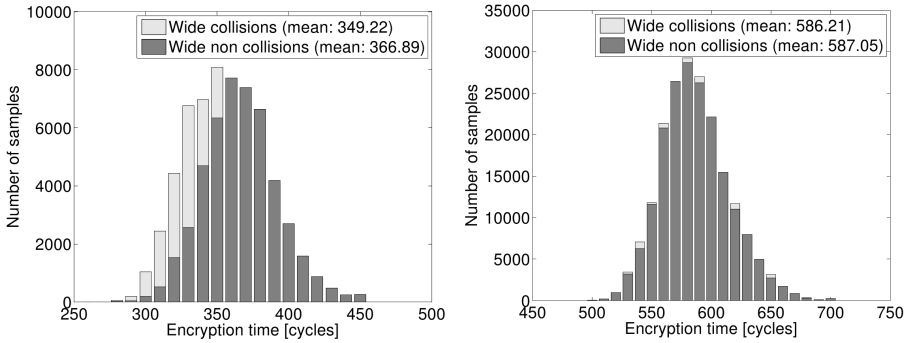

Fig. 1. Histogram of encryption times for a 3-round AES (left) and a 7-round AES (right).

4-byte subkey candidates per attacked 4-byte subkey as $(\binom{n}{4} \cdot 256)$ and since these subkeys must be enumerated exhaustively for all four 4-byte subkeys this yields an overall complexity of $(\binom{n}{4} \cdot 256)^4$ AES encryptions to recover the whole key.

We blame the larger cache-line size of 64 bytes on the ARM Cortex-A8 for the challenging detection of wide collisions. In contrast, Bogdanov *et al.* [4] launched the attack on an ARM9 board with a cache-line size of 32 bytes. Since each T-table is composed of 256 4-byte elements, a 32-byte cache line holds $\delta = 8$ T-table elements, whereas a 64-byte cache line holds $\delta = 16$ T-table elements. In case of a cache miss the Cortex-A8 loads 16 consecutive T-table elements into the cache, whereas the ARM9 board loads only 8 elements into the cache at once. If we take probability theory into consideration the problem becomes clear. Since the last round of the AES T-table implementation usually employs a different T-table, there are $4 \cdot 9$ look-up operations into the same T-table within the rounds 1–9. The probability for δ consecutive T-table elements—mapping to the same cache line—still not being cached after one encryption is $(1 - \frac{\delta}{256})^{4 \cdot 9}$. In case of $\delta = 16$ this yields a probability of 0.098 that a specific block of T-table elements is still not being cached after one encryption. In case of $\delta = 8$ this yields a probability of 0.319. Hence, the probability for a specific T-table element already being cached after the first encryption is $1 - 0.098 = 0.902$ and $1 - 0.319 = 0.681$. This in turn means that the probability for additional cache collisions, besides the required *wide collisions*, is far greater on systems with a cache-line size of 64 bytes. The overall encryption time of $\mathbf{P}_2$ decreases and makes *wide collisions* nearly indistinguishable from non wide collisions. We conclude that the larger cache-line size on the ARM Cortex-A8 exacerbates the detection of *wide collisions* and, thus, the applicability of this attack in general.

Even on the Cortex-A9 the detection of at least four *wide collisions* among a small number of chosen plaintexts, *e.g.*, $n \leq 6$, is a challenging task and a larger number of n drastically increases the remaining brute-force complexity. Our observations showed that we are able to detect enough *wide collisions* among $n = 7$ chosen plaintexts, but in this case the complexity of the exhaustive key search is impractical.

4 Conclusion

Recent investigations of cache attacks on mobile devices focused on specific testbeds and stressed the importance of analyzing these attacks in more realistic environments. Thus, we investigated the applicability of two time-driven cache attacks on state-of-the-art Android-based mobile devices. We observed that timing information also leaks on these devices and can be used to reduce the key space of cryptographic algorithms significantly. Though time-driven cache attacks usually require an enormous number of measurement samples we consider the attack of Bernstein [2] a threat for cryptographic implementations on mobile devices. In addition, we analyzed the attack of Bogdanov *et al.* [4] according to its applicability on mobile devices. We showed that a cache-line size of 64 bytes exacerbates this attack and even on systems with a cache-line size of 32 bytes the detection of *wide collisions* seems to be a challenging task. Our observations revealed that, in practice, encryptions where *wide collisions* occur and encryptions where no wide collisions occur are hardly distinguishable. Even though a high number of false positives might be overcome by taking more diagonal pairs into consideration, this drastically increases the complexity of the following key-search phase.

References

[1] ARM Ltd. Cortex-A Series. Available online at `http://www.arm.com/products/processors/cortex-a/index.php`, 2012.

[2] D. J. Bernstein. Cache-timing attacks on AES. Available online at `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005.

[3] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo. AES Power Attack Based on Induced Cache Miss and Countermeasure. In *ITCC (1)*, pages 586–591, 2005.

[4] A. Bogdanov, T. Eisenbarth, C. Paar, and M. Wienecke. Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs. In *CT-RSA*, pages 235–251, 2010.

[5] J.-F. Gallais and I. Kizhvatov. Error-Tolerance in Trace-Driven Cache Collision Attacks. In *COSADE*, pages 222–232, Darmstadt, 2011.

[6] D. Gullasch, E. Bangerter, and S. Krenn. Cache Games - Bringing Access-Based Cache Attacks on AES to Practice. In *IEEE SP*, pages 490–505, 2011.

[7] National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001.

[8] M. Neve. *Cache-based Vulnerabilities and SPAM Analysis*. PhD thesis, UCL, 2006.

[9] OpenSSL Software Foundation. OpenSSL Project. Available online at `http://www.openssl.org/`, 2012.

[10] R. Spreitzer and T. Plos. Cache-Access Pattern Attack on Disaligned AES T-Tables. In *COSADE 2013*, LNCS. Springer, 2013. In press.

[11] R. Spreitzer and T. Plos. On the Applicability of Time-Driven Cache Attacks on Mobile Devices (Extended Version). Cryptology ePrint Archive, Report 2013/172, 2013. `http://eprint.iacr.org/`.

[12] E. Tromer, D. A. Osvik, and A. Shamir. Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[13] M. Weiß, B. Heinz, and F. Stumpf. A Cache Timing Attack on AES in Virtualization Environments. In *FC*, pages 314–328. Springer Berlin Heidelberg, 2012.