Cache Timing Analysis of eStream Finalists (Extended Abstract)

Erik Zenner
Technical University of Denmark
e.zenner@mat.dtu.dk

March 9, 2009

Abstract

Cache Timing Attacks have attracted a lot of cryptographic attention due to their relevance for the AES. However, their applicability to other cryptographic primitives is less well researched. In this talk, we give an overview over our analysis of the stream ciphers that were selected for phase 3 of the eStream project.

1 Introduction

1.1 Motivation

Cache Timing Attacks are a class of side-channel attacks. They assume that the adversary can use timing measurements to learn something about the cache accesses of a legitimate party, which turns out to be the case in some practical applications. In 2005, Bernstein [2] and Osvik, Shamir, and Tromer [18, 19] showed in independent work that the Advanced Encryption Standard (AES) is particularly vulnerable to this type of side-channel attack, generating a lot of attention for the field. Subsequent work dealt with verifying the findings [17, 16, 15, 23, 8], improving the attack [20, 3, 14, 6], and devising and analyzing countermeasures [7, 5, 24].

However, the cryptanalytic attention was mainly focussed on AES, while other ciphers were treated only handwavingly. For example, the eStream report on side-channel attacks [11] simply categorizes all stream ciphers that use tables in their implementations as vulnerable, independent of whether or not a cache timing attack was actually feasible. Since then, some analysis of the eStream phase 3 candidates has been conducted by the authors [27, 13], and in the following, we give a brief overview over the results.

Dagstuhl Seminar Proceedings 09031
Symmetric Cryptography
http://drops.dagstuhl.de/opus/volltexte/2009/1943

1.2 Attack Model

As all side-channel attacks, cache timing attacks are not inherently attacks against the algorithm, but against its implementation. Thus, there are basically two ways of analyzing the cache timing resistance of a cipher. One can either consider a concrete implementation of the cipher, or do a general analysis in the framework of a model which gives the adversary certain rights, which can be modeled as oracle accesses. In the latter case, a "break" within the model does not necessarily imply a break of all practical implementations, but it can indicate that extra care has to be taken when implementing the cipher.

The model that we use for our analysis follows the second approach and was proposed in [27]. It models a synchronous cache attacker, i.e. an adversary who can only access (and thus perform measurements on) the cache after certain elementary operations by the legitimate users have finished. In particular, a synchronous cache adversary can do cache measurements before and after a full update of the stream cipher's inner state, but not while the update is in progress. Thus, he has access to two oracles: The KEYSTREAM(i) oracle returns keystream block z_i (this is the standard oracle in stream cipher cryptanalysis), and the SCA_KEYSTREAM(i) gives him an (unordered) list of cache accesses made by KEYSTREAM(i).

Note that the SCA_KEYSTREAM(i) oracle gives the adversary more information than will typically be available in a real-world side-channel setting, since it assumes that his measurements are undisturbed by noise. The task of the cryptanalyst will be made more difficult by the noise, but without knowing the precise practical setting, the exact extend of the extra work to be done by the attacker can not be estimated. On the other hand, given these strong assumptions, we only accept an attack as valid if it is feasible in a very practical sense, i.e. requiring only a "realistic" number of keystream blocks, cache measurements, and computational steps.

2 Analysing eStream Finalists

eStream was a subproject of the European ECRYPT project (2004-2008) that was dedicated to advancing the understanding of stream ciphers and to proposing a portfolio of recommended algorithms. It started by a call for contributions in Fall 2004 and 34 submissions in Spring 2005. The number of candidates was reduced to 27 (phase 2, Spring 2006) and 16 (phase 3, Spring 2007) before finally creating a portfolio of 8 ciphers in Spring 2008[1].

In the following, we concentrate on the finalists (i.e. phase 3 ciphers) in the "software" category, since only software implementations will be potential targets for cache timing attacks. An overview over those candidates is given in Table 1. Ciphers marked with the '†' symbol are potentially vulnerable to cache timing attacks due to their use of tables.

[1]The portfolio was reduced to 7 ciphers in Fall 2008 because one of the chosen designs was broken.

Cipher	Tables	Relevant
CryptMT	none	-
Dragon	Two 8×32-bit S-Boxes	†
HC-128	Two 512×32-bit tables	†
HC-256	Two 1024×32-bit tables	†
LEX-128	One 8×8-bit S-Box (ref. code) Eight 8×32-bit S-Boxes (opt. code)	†
NLS	One 8×32-bit S-Box	†
Rabbit	none	-
Salsa-20	none	-
Sosemanuk	One 8×32-bit table, eight 4×4-bit S-Boxes (ref. code)	†

Table 1: Table of eStream software finalists

On the other hand, note that CryptMT, Rabbit, and Salsa-20 do not use tables at all and are thus not even theoretically vulnerable to cache timing attacks. In the following, we discuss the remaining ciphers.

2.1 Dragon

No full analysis of Dragon [9] exists yet. Preliminary analysis seems to indicate that attacking the cipher with a cache timing attack will not be easy due to the heavy use of table lookups. The problem is that the cipher uses two 8×32-bit S-boxes, each of which fills 16 cache blocks (on a Pentium 4). For each call to KEYSTREAM(i), each S-box is called 12 times. Thus, for each S-box, up to 12 out of 16 cache blocks are accessed (on average: 8.6). For the cryptanalyst, this is a problem since (a) almost all possible cache blocks are used and thus only little information is retrieved, and (b) it is unclear in which order those cache blocks were accessed. If a full 12 different blocks were accessed for both S-boxes, there would be $2^{57.7}$ possible ways of ordering them. Thus, unless the ordering problem can be solved somehow, the cipher seems to be safe against practical cache-timing attacks. However, further analysis is necessary to confirm this conjecture.

2.2 HC-256

HC-256 [25] has a rather large inner state (two 1024×32-bit tables), all of which a cache timing attack has to reconstruct in order to be able to recover the key. Nonetheless, an attack was presented at SAC 2008 [27], requiring computation time corresponding to 2^{55} key setups, 3 MByte of memory, 8 kByte of known keystream, and precise cache measurements for 6148 rounds. Thus, the cipher is vulnerable to a cache timing attack in theory. Note, however, that the attack effort still corresponds to a brute-force attack against the Data

Encryption Standard (DES), and in the real world, noisy measurements will probably make it completely infeasible.

2.3 HC-128

HC-128 [26] is the "little brother" of HC-256 and has a slightly smaller inner state (two 512×32-bit tables) and surprisingly big changes of the internal workings. Most state update equations are modified, and this has a profound impact on the above cache timing attack. It turns out that the attack can not be transferred to HC-128 in a straightforward way. Thus, further analysis of HC-128 is necessary to determine its resistance against cache timing attacks.

2.4 LEX-128

The stream cipher LEX [4] is based to a large extend on AES, and its optimised code uses eight 8×32-bit S-boxes. It turns out that most known cache timing attacks against AES can be applied either against the key/IV setup or against the keystream generation. On the other hand, known protection measures (like using smaller S-boxes or bitsliced implementations) are also applicable and help mitigating the problem. Concluding, a fully optimised implementation of LEX-128 is breakable in practice, and protection measures have to be used where cache timing attacks are of concern.

2.5 NLS v2

NLS [22] uses one 8×32-bit S-box which can be the target of a cache timing attack. In [13], it is shown how this information can be used to retrieve the uppermost byte of each inner state word. This partial attack uses 2^{45} guess-and-determine steps, negligible memory, 23 known (upper) keystream bytes and 26 calls to SCA_KEYSTREAM(i).

However, it is not obvious how knowledge of the upper bytes (even for many subsequent inner states) can be used to retrieve the remaining inner state. In fact, if the uppermost byte is known, the cipher can be re-written without an S-box in a purely arithmetical fashion, using only addition, xor, and rotation (AXR) operators. It is known that solving such mixed equation systems can be very difficult (as an example, the security of Salsa-20 rests entirely on this observation), and in the case of NLS v2, they are sufficiently complicated. Concluding, a theoretical cache timing weakness was identified but does not seem to lead to a practical vulnerability.

2.6 Sosemanuk

In principle, Sosemanuk [1] can be implemented without the use of tables. However, software implementations optimised for speed require two 8×32-bit tables to speed up computations in GF(2^{32}). In [13], it was shown that these tables can be used for a very efficient cache timing attack against Sosemanuk.

As it turns out, any cache timing information about the inner state can be incorporated into a linear equation system. The ordering problem that was mentioned in connection with Dragon can be solved efficiently using slightly more measurements. Then the LFSR state (320 bit) can be retrieved by solving a system of linear equations, and the remaining 64 bit of nonlinear state can be found by 2^{32} simple guess-and-determine steps.

It turns out that the attack is also applicable against other designs using LFSRs over GF(2^{32}), such as Snow [10], Sober [12], and Turing [21]. The attack is so efficient that it has to be considered a practical cache timing break, and additional protection measures are necessary to protect the cipher where cache timing attacks are considered a threat.

3 Observations

It is interesting that most stream ciphers turn out to be surprisingly resistant against cache timing attacks. Note that the above model gives the attacker significant extra information about the inner state. Nonetheless, it seems that the ciphers can not be broken efficiently (it might be possible that a distinguishing attack with significant workload becomes possible, but this is not the goal here). This seems to indicate that modern stream ciphers have quite a large security margin. So the question is whether the ciphers in fact are overdesigned for normal purposes and whether significant speed-ups would become possible if the designers dropped some of the more extreme security requirements (e.g. large number of available keystream bits without re-keying).

Another observation is that in the above scenarios, the toolbox for cryptanalysis is pretty empty. Most known analysis methods require huge amounts of data and computational resources once the designer is aware of them and takes them into account (correlation attacks, non-trivial algebraic attacks, BDD attacks, distinguishers based on small biases etc.). The only efficient tools we had available for the above attacks were guess-and-determine and the solving of linear equations. In particular, our inability to efficiently deal with AXR systems consisting only of addition, xor, and rotations frustrated many potential attacks. Thus, research in this direction seems to have high potential of improving the cryptanalyst's toolbox.

Acknowledgements

The author wishes to express his thanks to Gregor Leander who co-authored some of the results discussed above, and to Philip Hawkes who provided valuable input on the initial attack ideas against Sosemanuk.

References

[1] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert. SOSEMANUK, a fast software-oriented stream cipher. eStream submission,
http://www.ecrypt.eu.org/stream/sosemanuk.html, 2005.

[2] D. Bernstein. Cache timing attacks on AES.
http://cr.yp.to/papers.html#cachetiming, 2005.

[3] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo. AES power attack based on induced cache miss and countermeasure. In *International Symposium on Information Technology: Coding and Computing (ITCC 2005)*, volume 1, pages 586–591. IEEE Computer Society, 2005.

[4] A. Biryukov. The design of a stream cipher LEX. In E. Biham and A. Youssef, editors, *Proc. SAC 2006*, volume 4356 of *LNCS*, pages 67–75. Springer, 2007.

[5] J. Blömer and V. Krummel. Analysis of countermeasures against access driven cache attacks on AES. In C. Adams, A. Miri, and M. Wiener, editors, *Proc. SAC 2007*, volume 4876 of *LNCS*, pages 96–109. Springer, 2007.

[6] J. Bonneau and I. Mironov. Cache-collision timing attacks against AES. In L. Goubin and M. Matsui, editors, *Proc. CHES 2006*, volume 4249 of *LNCS*, pages 201–215. Springer, 2006.

[7] E. Brickell, G. Graunke, M. Neve, and S. Seifert. Software mitigations to hedge AES against cache-based software side-channel vulnerabilities.
http://eprint.iacr.org/2006/052.pdf, 2006.

[8] A. Canteaut, C. Lauradoux, and A. Seznec. Understanding cache attacks. Technical Report 5881, INRIA, 2006.

[9] K. Chen, M. Henricksen, W. Millan, J. Fuller, L. Simpson, E. Dawson, H. Lee, and S. Moon. Dragon: A fast word based stream cipher. In C. Park and S. Chee, editors, *Proc. ICISC 2004*, volume 3506 of *LNCS*, pages 33–50. Springer, 2005.

[10] P. Ekdahl and T. Johansson. A new version of the stream cipher SNOW. In H. Heys and K. Nyberg, editors, *Proc. SAC 2002*, volume 2595 of *LNCS*, pages 47–61. Springer, 2002.

[11] B. Gierlichs, L. Batina, C. Clavier, T. Eisenbarth, A. Gouget, H. Handschuh, T. Kasper, K. Lemke-Rust, S. Mangard, A. Moradi, and E. Oswald. Susceptibility of eSTREAM candidates towards side channel analysis. In

C. de Cannière and O. Dunkelmann, editors, *SASC '08 Workshop Record*, pages 123–150, 2008.

[12] P. Hawkes and G. Rose. Primitive specification and supporting documentation for Sober-t32. NESSIE project submission, October 2000.

[13] G. Leander and E. Zenner. Cache timing analysis of the Sosemanuk and NLS stream ciphers. Manuscript available from the authors, 2009.

[14] M. Neve and J. Seifert. Advances on access-driven cache attacks on AES. In E. Biham and A. Youssef, editors, *Proc. SAC 2006*, volume 4356 of *LNCS*, pages 147–162. Springer, 2006.

[15] M. Neve, J. Seifert, and Z. Wang. Cache time-behavior analysis on AES. http://www.cryptologie.be/document/Publications/AsiaCSS_full_06.pdf, 2006.

[16] M. Neve, J. Seifert, and Z. Wang. A refined look at bernstein's AES side-channel analysis. In *Proc. AsiaCSS 2006*, page 369. ACM, 2006.

[17] M. O'Hanlon and A. Tonge. Investigation of cache-timing attacks on AES. http://www.computing.dcu.ie/research/papers/2005/0105.pdf, 2005.

[18] D. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of AES.
http://eprint.iacr.org/2005/271.pdf, 2005.

[19] D. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of AES. In D. Pointcheval, editor, *Proc. CT-RSA 2006*, volume 3860 of *LNCS*, pages 1–20. Springer, 2006.

[20] C. Percival. Cache missing for fun and profit. Paper accompanying a talk at BSDCan 2005; available at
http://www.daemonology.net/papers/htt.pdf, 2005.

[21] G. Rose and P. Hawkes. Turing: A fast stream cipher. In T. Johansson, editor, *Proc. Fast Software Encryption 2003*, volume 2887 of *LNCS*, pages 290–306. Springer, 2003.

[22] G. Rose, P. Hawkes, M. Paddon, and M. Wiggers de Vries. Primitive specification for NLSv2. eStream submission,
http://www.ecrypt.eu.org/stream/nlsp2.html, 2006.

[23] R. Salembier. Analysis of cache timing attacks against AES. Scholarly Paper, ECE Department, George Mason University, Virginia; available from: http://ece.gmu.edu/courses/ECE746/project/F06_Project_resources/ Salembier_Cache_Timing_Attack.pdf, May 2006.

[24] Z. Wang and R. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proc. ISCA 2007*, pages 494–505. ACM, June 2007.

[25] H. Wu. A new stream cipher HC-256. In B. Roy and W. Meier, editors, *Proc. FSE 2004*, volume 3017 of *LNCS*, pages 226–244. Springer, 2004.

[26] H. Wu. The stream cipher HC-128. http://www.ecrypt.eu.org/stream/hcp3.html, 2006.

[27] E. Zenner. A cache timing analysis of HC-256. In *Proc. SAC '08*, LNCS. Springer, 2008. to appear.