Cache Timing Analysis of eStream Finalists

Erik Zenner

Technical University Denmark (DTU)
Department of Mathematics
e.zenner@mat.dtu.dk

Dagstuhl, Jan. 15, 2009

Outline

Cache Motivation

What is a CPU cache?

- Intermediate memory between CPU and RAM
- Stores data that was recently fetched from RAM

What is is good for?

- Loading data from cache is much faster than loading data from RAM (e.g. RAM access ≈ 50 cycles, cache access ≈ 3 cycles).
- Data that is often used several times.
- $\Rightarrow$ Keeping copies in cache reduces the average loading time.

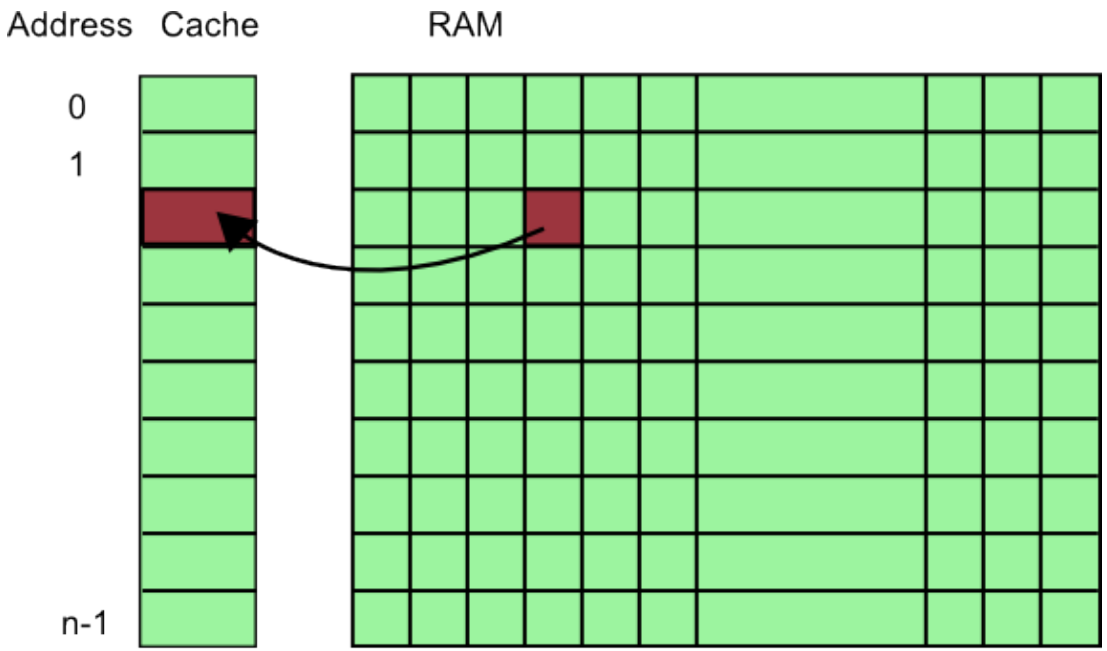
Why is this a problem?

- As opposed to RAM, cache is shared between users.
- $\Rightarrow$ Cryptographic side-channel attack becomes possible.

Cache Workings

Working principle (simplified): Let n be the cache size.
When we read from (or write to) RAM address a, proceed as follows:

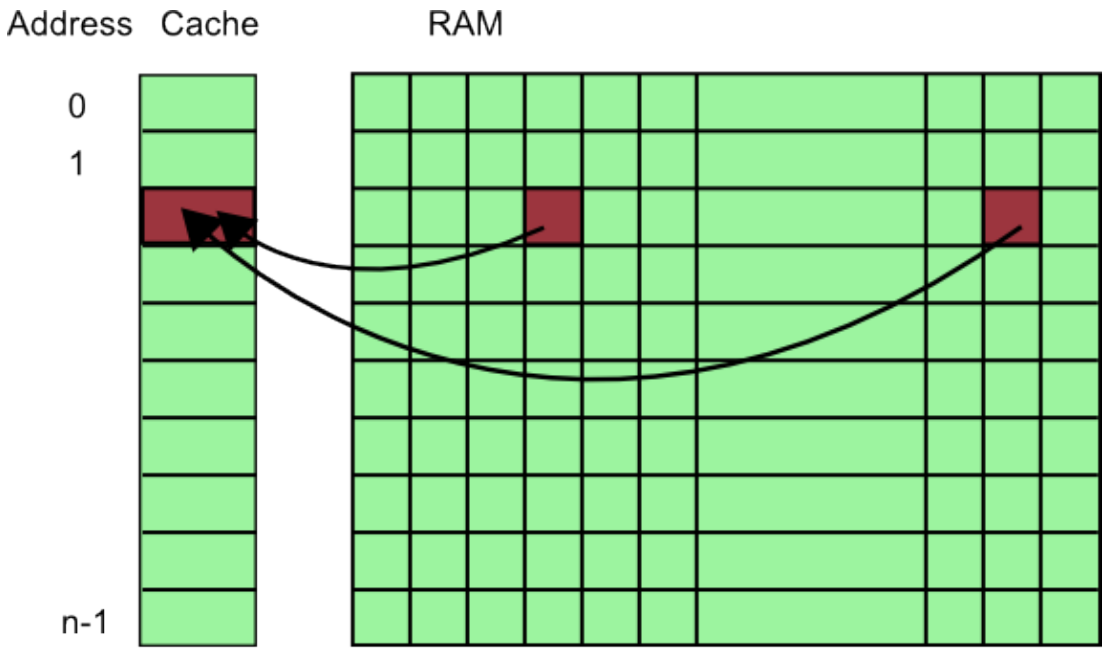
- Check whether requested data is at cache address (a mod n).
- If not, load data into cache address (a mod n).
- Load data item directly from cache.



Cache Eviction (Simplified)

Problem: Cache is much smaller than RAM.

Consequence: Many RAM entries compete for the same place in cache.



Handling: New data overwrites old data (First in, first out).

Sample Attack Setting

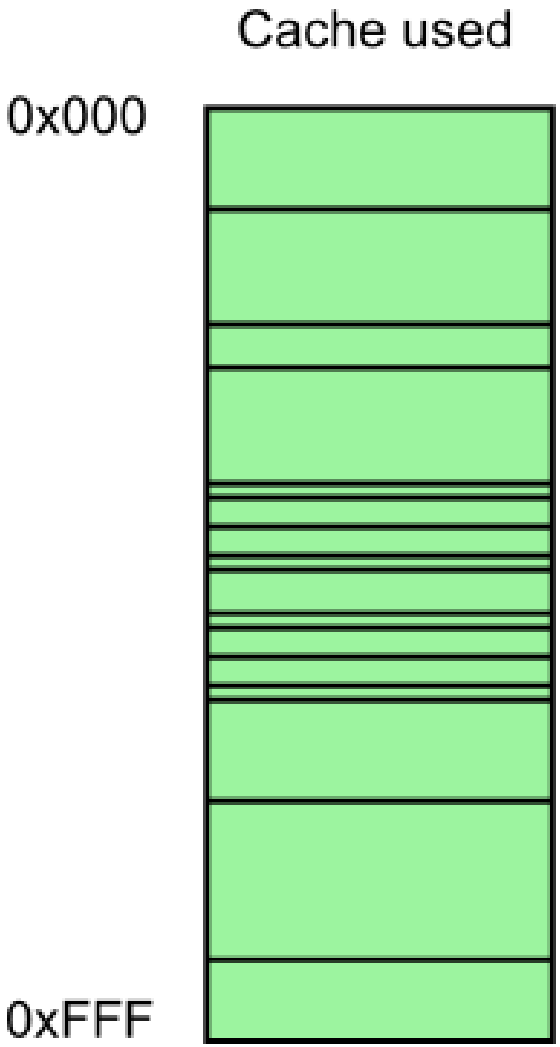
Starting point: Reading data is faster if it is in cache (cache hit), and slower if it has to be loaded (cache miss).

Sample attack (prime-then-probe): Imagine two users A and B sharing a CPU. If user A knows that user B is about to encrypt, he can proceed as follows:

1. A fills all of the cache with his own data, then he stops working.
2. B does his encryption.
3. A measures loading times to find out which of his data have been pushed out of the cache.

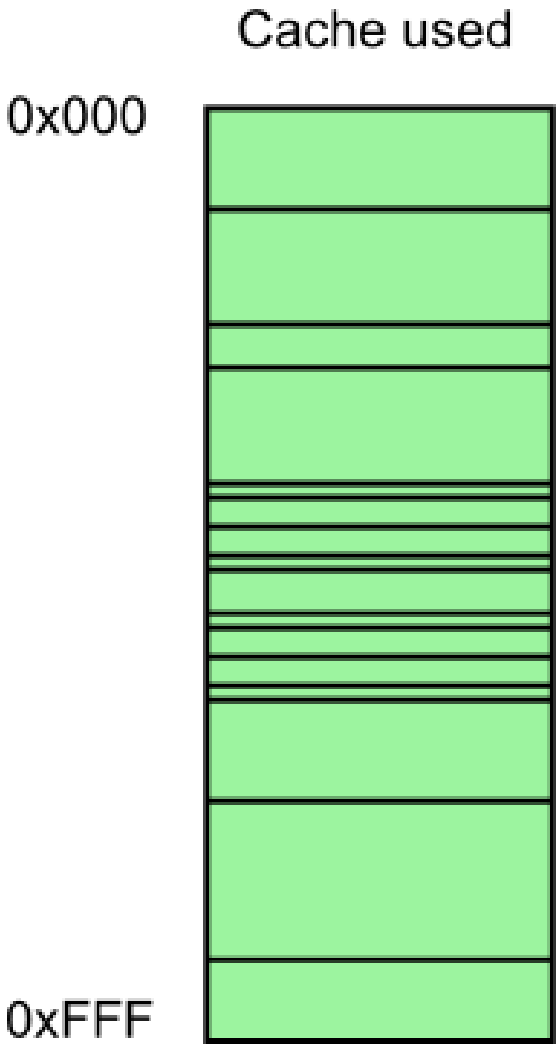
This way, A learns which cache addresses have been used by B.

Example



1. Running a cache timing attack gives the adversary a table with this structure.

Example



1. Running a cache timing attack gives the adversary a table with this structure.
2. We can clearly see that B used a table (e.g. S-Box, lookup-table etc.).
3. We can also see which table entries have been used.

Note: Adversary learns only the table **indices** used by B, but not the table **contents**!

Practical Difficulties

For didactical reasons, we worked with a simplified cache model.

Real-world complexities include:

- Cache data is not organised in bytes, but in blocks.
  $\Rightarrow$ We do not learn the exact index, but only some index bits.
- Other processes (e.g. system processes) use the cache, too.
  $\Rightarrow$ We can not tell "encryption" cache accesses apart from others.
- Timing noise disturbs the measurement.
  $\Rightarrow$ Not all slow timings are due to cache misses.
- Cache hierarchy is more complex.
  $\Rightarrow$ Several layers of cache, several cache blocks for each memory block.

Nonetheless, these difficulties can often be overcome in practice (Bernstein 2005, Osvik/Shamir/Tromer 2005, Bonneau/Mironov 2006).

Outline

Attacking Algorithms vs. Implementations

Basically, side-channel attacks target the **implementation**, not the **algorithm**.

Who is responsible - cryptographers or implementers?

Attacking Algorithms vs. Implementations

Basically, side-channel attacks target the **implementation**, not the **algorithm**.

Who is responsible - cryptographers or implementers? $\Rightarrow$ **Both!**

- Ideal: Cryptographers design algorithms that are not vulnerable to side-channel attacks.
- This saves **all** implementers the trouble of introducing protection measures.
- However: Cryptographers have to make assumptions (model) about the target system.

Assumptions for our Cryptanalysis

Available oracles:

- Adversary can trigger key/IV setup with IV of his choice.
- Adversary can step through the stream cipher, one round at a time (Osvik et al.: "synchronous" attack)
- Adversary can obtain any keystream block of his choice.
- Adversary can obtain any precise cache measurement of his choice. (**new!**)

Limitations:

- Adversary is limited to "realistic" number of keystream blocks.
- Adversary is limited to small number of cache measurements.
- Adversary is limited to "realistic" computational resources.

Outline

What is eStream?

Project: eStream was a subproject of the European ECRYPT project (2004-2008).
Purpose: Advance the understanding of stream ciphers and propose a portfolio of recommended algorithms.

Brief history:

- 2004 (Fall): Call for contributions.
- 2005 (Spring): Submission of 34 stream ciphers for evaluation.
- 2006 (Spring): End of evaluation phase 1, reduction to 27 candidates.
- 2007 (Spring): End of evaluation phase 2, reduction to 16 finalists.
- 2008 (April 15): Announcement of the final portfolio of 8 ciphers.
- 2008 (Sept. 8): Reduction to 7 ciphers due to new cryptanalysis.

Portfolio (Software): HC-128, Rabbit, Salsa20/12, Sosemanuk
Portfolio (Hardware): Grain, MICKEY (v2), Trivium

eStream Software Finalists

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)	†

†: Uses tables, thus potentially vulnerable

Dragon

Table use:

Dragon uses two 8×32-bit S-Boxes.

- Each S-Box fills 16 cache blocks (Pentium 4).
- For each round, each S-box is called 12 times.
- For each S-Box, up to 12 out of 16 cache blocks are accessed (on average: 8.6).
  $\Rightarrow$ Less information than we hoped for.
- 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.

Status:

Not fully analysed yet.

HC-256

Table use:

Two 1024×32-bit tables.

- Main problem: huge inner state.
- Attack at SAC 2008:
  - Computation time: equivalent to 2^{55} key setups.
  - Memory requirement: 3 MByte
  - Known keystream: 8 kByte
  - Precise cache measurements: 6148 rounds

Status:

Theoretically broken, but not relevant in practice.

HC-128

Table use:

Two 512×32-bit tables.

- Surprisingly big changes compared to HC-256.
- Very relevant for the cache timing attack.
- Attack from SAC 2008 can not be transferred.

Status:

Not fully analysed yet.

LEX-128

Table use:

Eight 8×32-bit S-Boxes (optimised code).

- Based on AES.
- Similar attacks applicable, both against key/IV setup and against keystream generation.
- Known protection measures (smaller S-boxes, bitslice implementation) applicable.

Status:

Optimised implementation breakable in practice. Protection measures have to be applied.

NLS v2

Table use:

One 8×32-bit S-Box.

- Work submitted for publication (Joint work with Gregor Leander).
- Attack retrieves the uppermost byte of each inner state word:
  - Computation time: 2^{45} guess-and-determine steps.
  - Memory requirement: negligible
  - Known keystream: 23 upper bytes
  - Precise cache measurements: 26 rounds
- Not obvious how to retrieve the lowermost bytes
  (S-box removed, but need to solve AXR system)

Status:

Theoretical weakness which does not seem to lead to a practical vulnerability.

Sosemanuk (1)

Table use:

One 8×32-bit table to speed up computations in GF(2^{32}),
some implementations also eight 4×4-bit S-Boxes (not used for analysis)

- Work submitted for publication (Joint work with Gregor Leander).
- Attack targets LFSR:
  - Any (cache timing) information about the inner state can be incorporated into linear equation system.
  - Ordering problem ($\rightarrow$ Dragon) can be solved by using slightly more measurements.
  - Retrieving of LFSR state (320 bit) by solving linear equation system.
  - Retrieving the nonlinear state (64 bit) by 2^{32} guessing steps.

Sosemanuk (2)

- Attack parameters:
  - Computation time: 2^{32} guess-and-determine steps,
    + solving a linear equation system with 320 unknowns in GF(2).
  - Memory requirement: 12.5 kByte (eq. system)
  - Known keystream: 1 output block (16 bytes)
  - Precise cache measurements: 20-40 rounds
    (depending on cache block size)
- Attack applies to all current designs with LFSRs over GF(2^{32}): Snow, Sober, Turing,...

Status:

Practical break of Sosemanuk, Snow, Sober, Turing.
Protection of the implementation necessary.

Outline

Conclusions and Observations

- LFSR-based solutions (over large fields) very vulnerable due to combination of lookup-table and linearity.
- Most other stream ciphers surprisingly resistant against cache timing attacks:
  - Given significant information about the inner state, we still can't break them efficiently!
  - Overdesigned for normal purposes?
  - Significant speed-up possible if we drop some of the more extreme security requirements?
- Toolbox for cryptanalysis pretty empty:
  - Most analysis methods require huge amounts of data and computational resources (correlation attacks, non-trivial algebraic attacks, BDD attacks, distinguishers based on small biases etc.).
  - Efficient tools: guess-and-determine, solving linear equations, others?
  - Tools for solving AXR problem ($\rightarrow$ Ralf-Philipp's talk) would come in handy!