A Cache Timing Analysis of HC-256

Erik Zenner

Technical University Denmark (DTU)
Institute for Mathematics
e.zenner@mat.dtu.dk

SAC 2008, Aug. 14, 2008

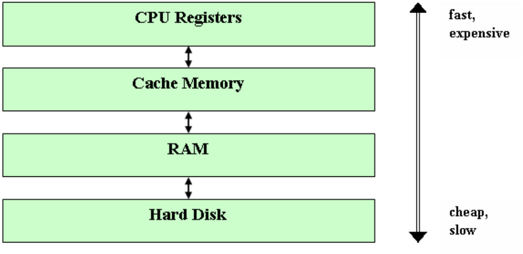
1. Cache Timing Attacks
2. Our Attack Model
3. Attacking HC-256
4. Conclusions

Outline

1. Cache Timing Attacks
2. Our Attack Model
3. Attacking HC-256
4. Conclusions

Memory Hierarchy (Simplified)

In a modern computer, different types of memory are used (simplified):

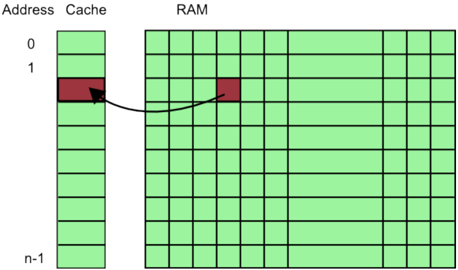


While CPU registers, RAM, and hard disk are protected against other users on the same machine, the cache is not.

Cache Workings (Simplified)

Working principle: Let n be the cache size.
When data from RAM address a is requested by the CPU:

- 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.

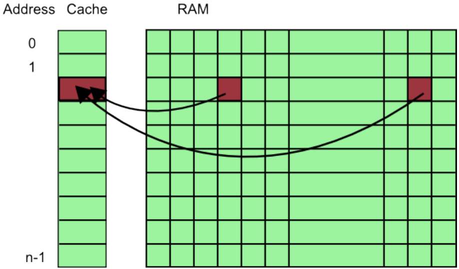


⇒ Next time data from address a can be loaded faster.

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 Eve and Alice sharing a CPU. If Eve knows that Alice is about to encrypt, she can proceed as follows:

1. Eve fills all of the cache with her own data, then stops working.
2. Alice does her encryption.
3. Eve measures loading times to find out which of her entries have been pushed out of the cache.

This way, Eve learns which cache addresses have been used by Alice.

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

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

Outline

1. Cache Timing Attacks
2. Our Attack Model
3. Attacking HC-256
4. Conclusions

Standard Adversary

Standard Oracles:

In standard analysis of stream ciphers, the adversary has access to the following oracles:

- KEYSETUP: Sets up a new cipher instance. Does not return any output.
- IVSETUP(N): Resets the cipher instance with initialisation vector N, as chosen by the adversary. Does not return any output.
- KEYSTREAM(i): Returns the keystream block i.

Note:

- These oracles overestimate the abilities of a real-world adversary, but they are widely used for analysing stream ciphers.
- We want to define additional oracles for a cache-timing adversary that are equally universal.

Synchronous Cache Adversary

Motivation:

- Abstract away technical details of the cache timing attacks.

Available Oracles:

A synchronous cache adversary (SCA) has access to the following additional oracles:

- SCA_KEYSETUP: Returns an accurate list of the cache blocks accessed while running KEYSETUP.
- $\text{SCA\_IVSETUP}(N)$: Returns an accurate list of the cache blocks accessed while running $\text{IVSETUP}(N)$.
- $\text{SCA\_KEYSTREAM}(i)$: Returns an accurate list of the cache blocks accessed while running $\text{KEYSTREAM}(i)$.

Discussion

Criticism:
This model is rather generous towards the adversary. In the real world, he may not be able to

- observe every encryption operation,
- get a precise list of cache block accesses,
- choose the IV, or
- observe the keystream.

⇒ Attacks in this model are not necessarily attacks in the real world.

Justification:

- The model is meant for use in cipher *design*.
- Designers must not rely on things that the adversary *might* not be able to do!

⇒ The cache adversary model *has* to be generous towards the adversary.

Outline

1 Cache Timing Attacks

2 Our Attack Model

3 Attacking HC-256

4 Conclusions

About HC-256

- Stream cipher (FSE 2004), eStream software finalist.
- **Key/IV:** 256 bit each.
- **Inner State:** Two tables, $1024 \cdot 32$ bit each.
  $\Rightarrow 65,536$ bits of inner state.
- **One Round:**
  - Update one of the tables.
  - Produce 32 bit of output.
- **Performance:**
  - Designed for software.
  - Slow key/IV setup (due to table initialisation).
  - Fast keystream generation.

Sketch of the Attack

The adversary uses the following oracles:

- 2048 calls to $\textsc{Keystream}(i)$.
- 6148 calls to $\textsc{SCA\_Keystream}(i)$.

Then he uses three layers of guess-and-verify to determine the inner state:

1. Determine the block access ordering.
2. Guess-and-eliminate step.
3. Guess-and-determine step.

We assume a textbook implementation of the cipher:

- One call to $\textsc{Keystream}(i)$ gives 32 output bits.

This excludes the optimised eStream implementation (512 output bits).

Step 1: Block Access Ordering

Adversary makes 6148 calls to $\mathrm{SCA\_K{\scriptsize EYSTREAM}}(i)$ and maps the resulting observations to inner state bits.

Problem: How to map cache accesses to state variables?

- Each oracle call: 5 cache accesses, e.g.:
  001011*xxxx*, 011100*xxxx*, 010011*xxxx*, 101101*xxxx*, 111110*xxxx*
- How to assign them to internal state variables? E.g.:
  $(00||P_{13}^{(7..0)})$, $(01||P_{13}^{(15..8)})$, $(10||P_{13}^{(23..16)})$, $(11||P_{13}^{(31..24)})$, $(P_{22} \oplus P_{-998})^{(9..0)}$

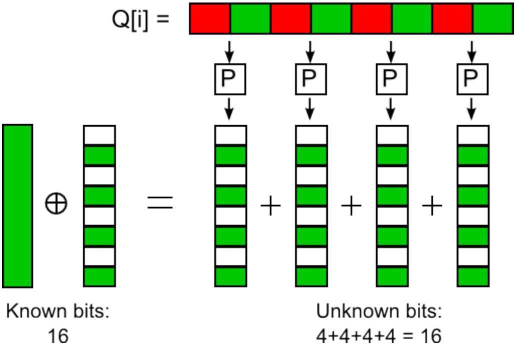
Solution: Simple internal consistency test works with high probability!

End of step 1:

For almost all inner state words, we know all upper half-bytes.
$\Rightarrow 2^{16}$ candidates for each inner state word.

Step 2: Guess-and-Eliminate Step (1)

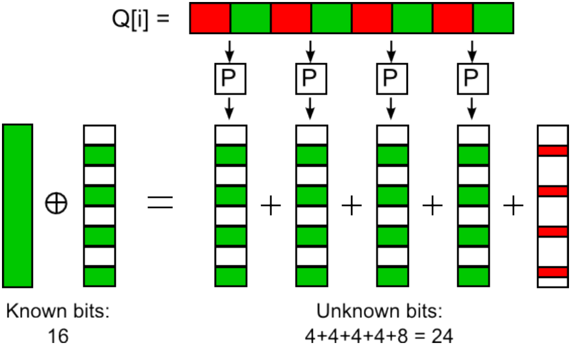
Adversary makes 2048 calls to $\textsc{Keystream}(i)$ and uses an internal equation to further reduce the number of candidates.



Problem: Carry bits complicate the equation.

Step 2: Guess-and-Eliminate Step (2)

Solution: Guess the carry bits, too.



End of step 2:

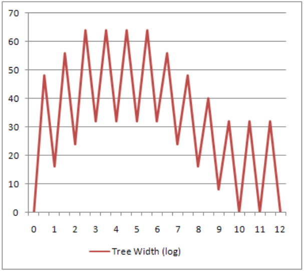
2^8 remaining candidates for each inner state word.
⇒ Store in a table (size ≈ 3 MByte).

Step 3: Guess-and-Determine Step

Adversary uses guess-and-determine strategy with a different equation to determine the rest of the inner state.

Problems:

- Many bits (48) have to be guessed before verification becomes possible.
- Too few verification bits (32) available.



Solution: Guesses start to overlap.

- Search tree grows slower than in the beginning, then starts shrinking.
- Maximum tree width: 2^{64} guesses.

End of step 3:

Full inner state for one point in time has been recovered.

The Attack in a Nutshell

Requirements:

- 6148 precise cache timing measurements.
- 2^{16} known plaintext bits.
- Computational effort corresponding to testing $\approx 2^{55}$ keys.
- $\approx$ 3 MByte of memory.

Result:

- Reconstruction of full inner state.
- Allows to create arbitrary output bits.
- Also allows to reconstruct the key.

Outline

1. Cache Timing Attacks
2. Our Attack Model
3. Attacking HC-256
4. Conclusions

Practical Relevance

Question: So is HC-256 broken?

Answer: Not unless you already stopped using AES for security reasons.

- Attack uses very strong assumptions.
- AES would be completely broken under the same assumptions.

But: Relevance of cache timing attacks is currently an open issue.

- A distinguisher using 2^{60} known plaintexts is sufficient to discard a cipher.
- How about a key recovery attack using $\approx 6,000$ precise cache timings?

Other eStream Software Finalists (1)

Cipher	Tables	Relevant
CryptMT	*none*	-
Dragon	Two 8 × 32-bit S-Boxes	?
HC-128	Two 9 × 32-bit tables	?
HC-256	Two 10 × 32-bit tables	†
LEX-128	Eight 8 × 32-bit S-Boxes (opt. code)	†
NLS	One 8 × 32-bit S-Box	?
Rabbit	*none*	-
Salsa-20/x	*none*	-
Sosemanuk	One 8 × 32-bit table (opt. code)	?

'†' = vulnerable '?' = potentially vulnerable '-' = immune

Other eStream Software Finalists (2)

- **Expectation:** When starting analysis in the above (generous) model, we expected most eStream candidates to break down completely.
- **Surprise:** Most candidates seem to withstand analysis even in the generous model surprisingly well, even though they were not designed to that purpose (exception: Salsa).
- Work on cryptanalysis is still in progress.
  - No one-size-fits-all attack
  - Different ciphers pose different problems
  - Individual analysis required
- **Guess:** Attacks are possible, but require some thought.

And finally...

Questions? Comments?