Cache-based attack on AES

Mikaël Fourrier
Contribution of this paper

- Get the secret key from AES in 3s + 3min
- Very weak assumptions
- No known plaintext needed
- No special rights needed, only to be able to spawn and control threads
Side-channel attack

- **Goal:** attack the implementation instead of brute force or theoretical weaknesses
  - Timing attacks
  - Power-monitoring attacks
  - Electromagnetic attack
  - Acoustic cryptanalysis
CPU Cache

- The CPU is way faster than the RAM, so we add caches so that we don't have to interact with the RAM
  - The caches are divided in blocks (64-128 bytes)
  - L1 ~10kb
  - L2 ~1M
  - L2 14x slower than L1
  - RAM 20x slower than L2, 200x slower than L1
AES

\[ X = \begin{pmatrix} x_0 & x_4 & x_8 & x_C \\ x_1 & x_5 & x_9 & x_D \\ x_2 & x_6 & x_A & x_E \\ x_3 & x_7 & x_B & x_F \end{pmatrix} = (x_0 \; x_1 \; x_2 \; x_3) \]

\[ \text{ShiftRows}(X) = \tilde{X} = \begin{pmatrix} x_0 & x_4 & x_8 & x_C \\ x_5 & x_9 & x_D & x_1 \\ x_A & x_E & x_2 & x_6 \\ x_F & x_3 & x_7 & x_B \end{pmatrix} \]

\[ s(\tilde{X}) = \begin{pmatrix} s(x_0) & s(x_4) & s(x_8) & s(x_C) \\ s(x_5) & s(x_9) & s(x_D) & s(x_1) \\ s(x_A) & s(x_E) & s(x_2) & s(x_6) \\ s(x_F) & s(x_3) & s(x_7) & s(x_B) \end{pmatrix} \]

\[ \text{MixColumns}(s(\tilde{X})) = M \cdot s(\tilde{X}) = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot s(\tilde{X}) \]
AES implementation

- Exploit redundancy in the matrix multiplications to speed up the calculation
- Massive use of precomputed tables
  - If we know which entry is used when, we can deduce the private key
  - Each round we get a probability that a byte sequence is part of the key
Main idea

1) Fill L1 with known data
2) Let the target execute one table load (will be a miss)
3) Detect which cache line has been changed
4) Deduce which part of the table has been loaded
5) Repeat!
Completely fair scheduler

• Linux process scheduler
• Goal: as with n processes executing on n processors at 1/n the speed.
  → Execute first the process which had less execution time
The attack

- DoS on CFS: hundreds of threads + one dummy thread
• Read a big array, if response time above a threshold → cache miss → the target process used this cache line
Neural networks

- One neuron has multiple inputs and one output
- Each input has an associated weight
- The networks learns by changing the weights
Post-processing

- Use of two neural networks:
  - Noise reduction (right)
  - Estimation on the number of memory access at $t$

(a) Input of the neural network.

(b) Output of the neural network.
Results

- 250 threads, 100 encryptions
  - 10ms → 2.8s
- Noise reduction: 21s, normal process
- Preparing key search by constructing a probability table: 63s
- Key search: 30-300s
  - 3 minutes to find the key
  - 60kB to transfer for post-processing
Countermeasures (general)

- Don't use the cache
  - Not possible in real life
- Don't let process access high-res timers
  - A lot of legitimate apps use it
- Cache preloading by the OS
- Mark table as uncachable
- Limit the minimum time between context switch
Contermeasures (AES)

- Use more efficient instructions to reduce table size
- Use hardware-supported encryption (Intel AES-NI)
References