Remote Cache Timing Attack on Advanced Encryption Standard and Countermeasures

Darshana Jayasinghe, Jayani Fernando, Ranil Herath, Roshan Ragel

Department of Computer Engineering, University of Peradeniya, Peradeniya, 20400, Sri Lanka.
Advanced Encryption Standard (AES)?

Cipher(\text{bytein}[4 \times Nb], \text{byteout}[4 \times Nb], \text{wordw}[Nb \times (Nr + 1)])
begin
bytestate[4, Nb]
state = in
AddRoundKey(state, w[0, Nb − 1])
for round = 1 to Nr − 1 do
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round \times Nb, (round + 1) \times Nb − 1])
end for
SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[Nr \times Nb, (Nr + 1) \times Nb − 1])
out = state
end
Become as U.S. FIPS in November 26 2001, after a 5 year standardization process
• Become as U.S. FIPS in November 26 2001, after a 5 year standardization process
• Involved lot of computing (mathematical calculations)
AES?

- Become as U.S. FIPS in November 26 2001, after a 5 year standardization process
- Involved lot of computing (mathematical calculations)

\[
\text{ShiftRows}(\text{state}) \implies S_{r,c} = S_{r,(\text{shift}(r,Nb)) \mod Nb} \\
\text{for } 0 < r < 4 \text{ and } 0 \leq c \leq Nb
\]
• Become as U.S. FIPS in November 26 2001, after a 5 year standardization process
• Involved lot of computing (mathematical calculations)

\[ \text{ShiftRows}(\text{state}) \implies S_{r,c} = S_{r, (\text{shift}(r,Nb)) \mod Nb} \]

for \( 0 < r < 4 \) and \( 0 \leq c \leq Nb \)

• In the software implementation \([\text{subBytes, shiftRows, mixColumns}] \implies T \text{ boxes}\)
A side channel is any observable information emitted as a by product of the physical implementation of the cryptosystem.
A side channel is any observable information emitted as a by product of the physical implementation of the cryptosystem.
Bernstein’s Attack Overview

- In 2005, Daniel Bernstein demonstrated a remote cache timing attack
- At the encryption, cache hits/misses cause timing differences in providing a sufficient side channel
- Used a client server architecture to demonstrate

**Steps**

- Collect data under a known key
- Collect data for the unknown key
- Key deduction
  - Correlation
  - Key search
Bernstein’s Attack in Details

- Collect data under a known key
Bernstein’s Attack in Details

- Collect data under a known key
- Collect data for the unknown key
Bernstein’s Attack in Details

- Collect data under a known key
- Collect data for the unknown key
- Key deduction
Bernstein’s Attack in Details

- Collect data under a known key
- Collect data for the unknown key
- Key deduction
  - correlation

| 8 0 | 35 31 37 32 34 33 30 36 |
| 8 1 | 4d 4b 4e 49 4c 48 4a 4f |
| 2 2 | cc cd |
| 8 3 | 81 87 84 82 85 80 83 86 |
| 1 4 | b3 |
| 8 5 | 83 82 80 84 81 85 86 87 |
| 4 6 | 46 40 42 45 |
| 8 7 | 01 07 02 04 00 03 06 05 |
| 1 8 | 89 |
| 8 9 | c6 c3 c0 c2 c4 c7 c5 c1 |
| 8 10 | e1 e3 e2 e0 e5 e6 e7 e4 |
| 35 11 | 63 60 64 29 2e 2f 57 7f |
|        | 61 59 7e 2b 28 5a 42 7b |
|        | 5c 7c 50 47 2a 65 44 78 |
|        | 79 4b 5b 46 41 7a 55 62 |
|        | 67 4d 2d |
| 1 12 | 1b |
| 8 13 | ba b8 bc be bf bd bb b9 |
| 7 14 | d0 d3 d6 d2 d4 d1 d5 |
| 8 15 | 89 88 8a 8d 8f 8c 8b 8e |
Bernstein’s Attack in Details

- Collect data under a known key
- Collect data for the unknown key
- Key deduction
  - correlation
  - key search

Combinations = \(8 \times 8 \times 2 \times 8 \times 1 \times 8 \times 4 \times 8 \times 1 \times 8 \times 8 \times 35 \times 1 \times 8 \times 7 \times 8\)
Bernstein’s Attack in Details

- Collect data under a known key
- Collect data for the unknown key
- Key deduction
  - correlation
  - key search

Combinations = \(8 \times 8 \times 2 \times 8 \times 1\)
\(\times 8 \times 4 \times 8 \times 1 \times 8\)
\(\times 8 \times 35 \times 1 \times 8 \times 7\)
\(\times 8\)
= \(2.63E+11\)
Bernstein’s Attack in Details

- Collect data under a known key
- Collect data for the unknown key
- Key deduction
  - correlation
  - key search

Combinations = \(8 \times 8 \times 2 \times 8 \times 1\)
\(\times 8 \times 4 \times 8 \times 1 \times 8\)
\(\times 8 \times 35 \times 1 \times 8 \times 7\)
\(\times 8\)
\(= 2.63E+11\)

Can use brute force
Bernstein’s Attack in Details

- Collect data under a known key
- Collect data for the unknown key
- Key deduction
  - correlation
  - key search

Combinations = $8 \times 8 \times 2 \times 8 \times 1$
$\times 8 \times 4 \times 8 \times 1 \times 8$
$\times 8 \times 35 \times 1 \times 8 \times 7$
$\times 8$
$= 2.63E+11$

Can use brute force compared to $2^{128} = 3.40E+38$
Optimal Number of Packets
Optimal Number of Packets

![Graph showing the optimal number of packets for 400 byte and 800 byte data packets. The graph plots the number of key combinations (log$_{10}$ scale) against the number of packets (2$^n$).]
## Countermeasures

<table>
<thead>
<tr>
<th>SW</th>
<th>OS</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminate T boxes ✓</td>
<td>Disabling the cache</td>
<td>Place look up tables in register</td>
</tr>
<tr>
<td>Masking timing data evicted from the cache ✓</td>
<td>Cache partitioning ✓</td>
<td>Hardware encryption</td>
</tr>
<tr>
<td>Smaller tables ✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prefetching ??</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Countermeasures Experimented

- Arithmetic operations
  - Manual calculation of T boxes (subBytes, shiftRows, mixColumns)
Some Countermeasures Proposed So Far

Countermeasures Experimented

- Arithmetic operations
  - Manual calculation of T boxes (subBytes, shiftRows, mixColumns)
  - $\approx 31$ times slower
Countermeasures Experimented-2

- Bitwise operations
  \[ t_0 = (s_0 \gg 24 \oplus t_0) \ll 24 \oplus (s_1 \oplus 24 \oplus t_0) \ll 16 \oplus (s_2 \gg 24 \oplus t_0) \ll 8 \oplus (s_3 \gg 24 \oplus t_0) ; \]
  \[ t_1 = (s_1 \& 0xff \oplus t_1) \ll 24 \oplus (s_2 \& 0xff \oplus t_1) \ll 16 \oplus (s_3 \& 0xff \oplus t_1) \ll 8 \oplus (s_0 \& 0xff \oplus t_1) ; \]
  \[ t_2 = (s_2 \gg 8 \oplus t_2) \ll 24 \oplus (s_3 \gg 8 \oplus t_2) \ll 16 \oplus (s_0 \gg 8 \oplus t_2) \ll 8 \oplus (s_1 \gg 8 \oplus t_2) ; \]
  \[ t_3 = (s_3 \gg 16 \oplus t_3) \ll 24 \oplus (s_0 \gg 16 \oplus t_3) \ll 16 \oplus (s_1 \gg 16 \oplus t_3) \ll 8 \oplus (s_2 \gg 16 \oplus t_3) ; \]

- Use bitwise operations to calculate T boxes
Countermeasures Experimented-2

- Bitwise operations
  - $t_0 = (s_0 \gg 24 \oplus t_0) \ll 24 \oplus (s_1 \oplus 24 \oplus t_0) \ll 16 \oplus (s_2 \gg 24 \oplus t_0)$
  - $\ll 8 \oplus (s_3 \gg 24 \oplus t_0)$;
  - $t_1 = (s_1 \& 0xff \oplus t_1) \ll 24 \oplus (s_2 \& 0xff \oplus t_1) \ll 16 \oplus (s_3 \& 0xff \oplus t_1)$
  - $\ll 8 \oplus (s_0 \& 0xff \oplus t_1)$;
  - $t_2 = (s_2 \gg 8 \oplus t_2) \ll 24 \oplus (s_3 \gg 8 \oplus t_2) \ll 16 \oplus (s_0 \gg 8 \oplus t_2)$
  - $\ll 8 \oplus (s_1 \gg 8 \oplus t_2)$;
  - $t_3 = (s_3 \gg 16 \oplus t_3) \ll 24 \oplus (s_0 \gg 16 \oplus t_3) \ll 16 \oplus (s_1 \gg 16 \oplus t_3)$
  - $\ll 8 \oplus (s_2 \gg 16 \oplus t_3)$;

- Use bitwise operations to calculate T boxes
- \approx 14 \text{ times slower}
Countermeasures Experimented-2

- Bitwise operations
  \[ t0 = (s0 \gg 24 \oplus t0) \ll 24 \oplus (s1 \oplus 24 \oplus t0) \ll 16 \oplus (s2 \gg 24 \oplus t0) \ll 8 \oplus (s3 \gg 24 \oplus t0) ; \]
  \[ t1 = (s1 \& 0xff \oplus t1) \ll 24 \oplus (s2 \& 0xff \oplus t1) \ll 16 \oplus (s3 \& 0xff \oplus t1) \ll 8 \oplus (s0 \& 0xff \oplus t1) ; \]
  \[ t2 = (s2 \gg 8 \oplus t2) \ll 24 \oplus (s3 \gg 8 \oplus t2) \ll 16 \oplus (s0 \gg 8 \oplus t2) \ll 8 \oplus (s1 \gg 8 \oplus t2) ; \]
  \[ t3 = (s3 \gg 16 \oplus t3) \ll 24 \oplus (s0 \gg 16 \oplus t3) \ll 16 \oplus (s1 \gg 16 \oplus t3) \ll 8 \oplus (s2 \gg 16 \oplus t3) ; \]
  - Use bitwise operations to calculate T boxes
  - \( \approx 14 \) times slower

- Cache partitioning
Countermeasures Experimented-2

- **Bitwise operations**
  
  \[ t_0 = (s_0 \gg 24 \oplus t_0) \ll 24 \oplus (s_1 \oplus 24 \oplus t_0) \ll 16 \oplus (s_2 \gg 24 \oplus t_0) \ll 8 \oplus (s_3 \gg 24 \oplus t_0) \]
  
  \[ t_1 = (s_1 \& 0xff \oplus t_1) \ll 24 \oplus (s_2 \& 0xff \oplus t_1) \ll 16 \oplus (s_3 \& 0xff \oplus t_1) \ll 8 \oplus (s_0 \& 0xff \oplus t_1) \]
  
  \[ t_2 = (s_2 \gg 8 \oplus t_2) \ll 24 \oplus (s_3 \gg 8 \oplus t_2) \ll 16 \oplus (s_0 \gg 8 \oplus t_2) \ll 8 \oplus (s_1 \gg 8 \oplus t_2) \]
  
  \[ t_3 = (s_3 \gg 16 \oplus t_3) \ll 24 \oplus (s_0 \gg 16 \oplus t_3) \ll 16 \oplus (s_1 \gg 16 \oplus t_3) \ll 8 \oplus (s_2 \gg 16 \oplus t_3) \]

- Use bitwise operations to calculate T boxes
  - \( \approx 14 \) times slower

- **Cache partitioning**
  - Align the T tables into separate colors of the cache
Countermeasures Experimented-2

- Bitwise operations
  \[ t_0 = (s_0 \gg 24 \oplus t_0) \ll 24 \oplus (s_1 \oplus 24 \oplus t_0) \ll 16 \oplus (s_2 \gg 24 \oplus t_0) \ll 8 \oplus (s_3 \gg 24 \oplus t_0); \]
  \[ t_1 = (s_1 \& 0xff \oplus t_1) \ll 24 \oplus (s_2 \& 0xff \oplus t_1) \ll 16 \oplus (s_3 \& 0xff \oplus t_1) \ll 8 \oplus (s_0 \& 0xff \oplus t_1); \]
  \[ t_2 = (s_2 \gg 8 \oplus t_2) \ll 24 \oplus (s_3 \gg 8 \oplus t_2) \ll 16 \oplus (s_0 \gg 8 \oplus t_2) \ll 8 \oplus (s_1 \gg 8 \oplus t_2); \]
  \[ t_3 = (s_3 \gg 16 \oplus t_3) \ll 24 \oplus (s_0 \gg 16 \oplus t_3) \ll 16 \oplus (s_1 \gg 16 \oplus t_3) \ll 8 \oplus (s_2 \gg 16 \oplus t_3); \]
  - Use bitwise operations to calculate T boxes
  - \( \approx 14 \) times slower

- Cache partitioning
  - Align the T tables into separate colors of the cache
  - \( \approx 2.7 \) times slower
Some Countermeasures Proposed So Far

Countermeasures Experimented

- Random sleep
  
  \[ i \leftarrow \text{GCC\_random\_number} \{ i \text{ from } 0 \rightarrow 4 \} \]
  
  \text{thread.sleep}(i)

  \( \approx 1.02 \) times slower

- Random loop
  
  \[ i \leftarrow \text{GCC\_random\_number} \{ i \text{ from } 0 \rightarrow 99 \} \]
  
  \text{for } j = 1 \text{ to } i \text{ do}
  
  \text{asm("nop")}
  
  \text{end for}
Countermeasures Experimented-3

- Random sleep
  \[ i \leftarrow \text{GCC\_random\_number} \{ i \text{ from } 0 - 4 \} \]
  \[ \text{thread.sleep}(i) \]
  \[ \approx 1.02 \text{ times slower} \]

- Random loop
  \[ i \leftarrow \text{GCC\_random\_number} \{ i \text{ from } 0 - 99 \} \]
  \[ \text{for } j = 1 \text{ to } i \text{ do} \]
  \[ \text{asm(”nop”) } \]
  \[ \text{end for} \]
  \[ \approx 1.05 \text{ times slower} \]
**Conclusions**

- **Optimal number of packets**
  
  \[ \approx 2^{24} \] is the minimum number of data packets that should be used to carry out a successful attack.
Conclusions

- **Optimal number of packets**
  \[ \approx 2^{24} \] is the minimum number of data packets that should be used to carry out a successful attack.

- **Countermeasures against Bernstein’s attack**
  The random sleep or loop will be a good countermeasure.
The random sleep or loop can be vulnerable to statistical attack.
Future Work

- The random sleep or loop can be vulnerable to statistical attack
- Software pre-fetching
Future Work

- The random sleep or loop can be vulnerable to statistical attack
- Software pre-fetching
  - Can hide fetching time in arithmetic operations
Future Work

- The random sleep or loop can be vulnerable to statistical attack
- Software pre-fetching
  - Can hide fetching time in arithmetic operations
  - No time variation, encrypting packets


Thank you..