Faster and Timing-Attack Resistant AES-GCM

Emilia Käsper, Peter Schwabe

KU Leuven, TU Eindhoven

CHES 2009
Lausanne, September 2009

How fast is AES (on an Intel Core 2)?

AES in OpenSSL $\approx$ 18 cycles/byte (110MB/s @ 2GHz).
Can we do better?

How fast is AES (on an Intel Core 2)?

AES in OpenSSL $\approx$ 18 cycles/byte (110MB/s @ 2GHz).
Can we do better?

- One concurrent load/store per cycle
- AES needs 1 table-lookup per byte per round – 10 cycles/byte limit for AES-128
- [*BS*08]: 10.5 cycles/byte on a Core 2
  - does less than 1 lookup per byte and round by using specifics of CTR mode

How fast is AES (on an Intel Core 2)?

AES in OpenSSL $\approx$ 18 cycles/byte (110MB/s @ 2GHz).
Can we do better?

- One concurrent load/store per cycle
- AES needs 1 table-lookup per byte per round – 10 cycles/byte limit for AES-128
- [*BS*08]: 10.5 cycles/byte on a Core 2
  - does less than 1 lookup per byte and round by using specifics of CTR mode
- In this talk:

AES-CTR	7.59 cycles/byte
AES-GCM	10.68 cycles/byte

Cache Attacks on AES Implementations

- Core idea: lookup table indices dependent on secret key material
  - First round of AES: $T[\texttt{plaintext} \oplus \texttt{roundkey}]$
- Knowing which part of the table was accessed leaks key bits
- A variety of attack models
  - Active cache manipulation via user processes
  - (Remote) timing of cache "hits" and "misses"
  - Power traces
- Example countermeasures
  - Protecting vulnerable cipher parts (e.g., first and last round) in software — only thwarts current attacks
  - Hardware protection — sacrifice general CPU performance just for crypto?

Our Solution

- Implementation of AES in counter mode
  - Written in GNU assembly/qhasm using 128-bit XMM registers
  - Bitsliced implementation – constant-time, immune to **all** timing attacks
  - First bitsliced implementation that is also fast for packet encryption
- Authenticated encryption — AES in Galois/counter mode
  - Using lookup-tables: 10.68 cycles/byte
  - Constant-time: 21.99 cycles/byte

The platform: Intel Core 2 and Core i7

- 16 128-bit XMM registers
- SSE (Streaming SIMD Extension) instructions
  - followed by SSE2, SSE3, SSSE3 (Intel), SSE4 (Intel), SSE5 (AMD), AVX (Intel) etc.
- “native” 128-bit wide execution units
  - older Core Duo’s “packed” 128-bit instructions
- 3 execution units – up to 3 arithmetic and bit-logical instructions per cycle
- Instructions are two-operand

$$\texttt{xor a b} \quad \equiv \quad b = a + b$$

The Bitslicing Approach

row 0																....	row 3								
column 0				column 1				column2				column 3				....	column 0				......	column 3			
block 0	block 1	...	block 7	block 0	block 1	...	block 7	block 0	block 1	...	block 7	block 0	block 1	...	block 7	....	block 0	block 1	...	block 7	......	block 0	block 1	...	block 7

- Process 8 AES blocks (=128 bytes) in parallel
- Collect bits according to their position in the byte: i.e., the first register contains least significant bits from each byte, etc.
- AES state stored in 8 XMM registers
- Compute 128 S-Boxes in parallel, using bit-logical instructions
- For a simpler linear layer, collect the 8 bits from identical positions in each block into the same byte
- Never need to mix bits from different blocks - all instructions byte-level

Implementing the AES S-Box

- Start from the most compact hardware S-box, 117 gates [Can05, BP09]
- Use equivalent 128-bit bit-logical instructions
- Problem 1: instructions are two-operand, output overwrites one input
- Hence, sometimes need extra register-register moves to preserve input
- Problem 2: not enough free registers for intermediate values
- We recompute some values multiple times (alternative: use stack)
- Total 163 instructions — 15% shorter than previous results

	`xor`	`and/or`	`mov`	TOTAL
Hardware	82	35	–	117
Software	93	35	35	163

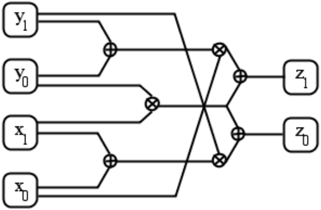
Hardware vs Software

Example: multiplication in $GF(2^2)$

$$(x_1, x_0) \otimes (y_1, y_0) \longrightarrow (z_1, z_0)$$

$$z_1 = (y_0 + y_1)x_0 + x_1 y_0$$

$$z_0 = (x_0 + x_1)y_1 + x_1 y_0$$

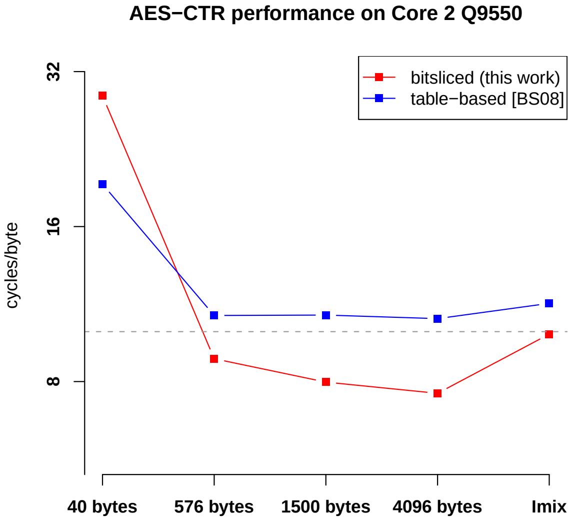



```
movdqa   \x0, \z0
movdqa   \x1, \z1
movdqa   \y0, \t0
pxor     \y1, \t0
pand     \z0, \t0
pxor     \z1, \z0
pand     \y1, \z0
pand     \y0, \z1
pxor     \z1, \z0
pxor     \t0, \z1
```

Implementing the AES Linear Layer

- Each byte in the bitsliced vector corresponds to a different byte position in the AES state
- Thus, ShiftRows is a permutation of bytes
- Use SSSE3 dedicated byte-shuffle instruction `pshufb`
- Repeat for each bit position (register) = 8 instructions
- MixColumns uses byte shuffle and XOR, total 43 instructions
- AddRoundKey also requires only 8 XORs from memory
- Some caveats:
  - Bitsliced key is larger - 8×128 bits per round, key expansion slower
  - SSSE3 available only on Intel, not on AMD processors

eStream benchmarks of AES-CTR-128



AES-GCM with Lookup Tables

- Use the fast constant-time AES
- GCM core operation: multiplication in 128-bit Galois field
- Implemented using key-dependent lookup tables
- Several choices for table sizes
- Our implementation: 32 tables, 16 128-bit values each – memory 8KB
- One multiplication with 32 loads and 84 arithmetic instructions – $\approx$ 3 cycles/byte; 10.68 cycles/byte for AES-GCM

Cache-timing vulnerabilities in GCM

- In the first multiplication $C_0 \cdot H$, the lookup indices come from a known ciphertext block C_0 — does not leak information about the key H
- Second multiplication:

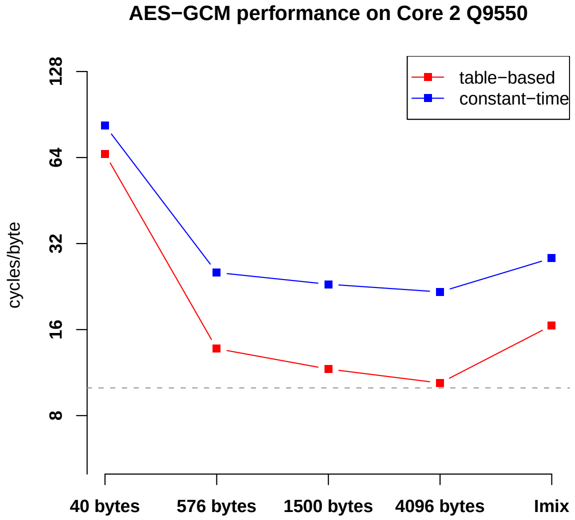
$$(C_o H + C_1) \cdot H$$

- Assuming attacker knows C_0, C_1, lookup indices from $C_0 H + C_1$ leak information about H
- No published attacks (as far as we know), but seemingly vulnerable
- Can compromise only authentication key, not encryption key

AES-GCM without lookup tables

- Key idea: use 1-bit "lookup tables"
- Use constant-time bit-logical operations to do the "lookup"
- Speed $\approx$ 14 cycles/byte (21.99 cycles/byte for AES-GCM)
- First practical constant-time implementation
- Previously reported table-free implementations over 100 cycles/byte on a Motorola G4

eStream Benchmarks of AES-GCM-128



Concluding remarks

- Breaking the 10 cycles/byte barrier: 7.59 cycles/byte for AES (from 110 MB/s in OpenSSL to 260 MB/s @ 2GHz)
- A posteriori improvement — both AES and GCM were designed to be implemented with lookup tables
- Dedicated instructions (Intel AES-NI) available soon, but...
- ...almost 10 years after standardization, 5? years to become widespread
- A general lesson: trends in processor architecture/graphics processing in favour of fast crypto
    - 256-bit registers, three operand instructions
- Bitslicing is an efficient countermeasure against cache-timing attacks on current processors

The Software

The software is available in public domain

QHASM implementations:

http://cryptojedi.org/crypto/#aesbs

GNU asm implementations:

http://homes.esat.kuleuven.be/~ekasper/#software