New AES software speed records

Daniel J. Bernstein and Peter Schwabe

16.12.2008

Indocrypt 2008

The eSTREAM project

- ECRYPT Stream Cipher Project
- Running from 2004 to 2008 to identify promising new stream ciphers
- Established an open benchmarking suite for stream ciphers
- For comparison contains 128-bit AES in counter mode
- Reference implementation by Gladman
- Other authors added faster implementations
- This talk describes our new implementation

One round of AES in C

Input and Output

Input: 16-byte state given in 4 32-bit integers y0, y1, y2, y3
Output: 16-byte state given in 4 32-bit integers z0, z1, z2, z3

Tables and keys

```
uint32 rk[44], T0[256], T1[256], T2[256], T3[256];
```

The first round of AES

```
z0 = T0[ y0 >> 24        ] ^ T1[(y1 >> 16) & 0xff] \
   ^ T2[(y2 >>  8) & 0xff] ^ T3[ y3        & 0xff] ^ rk[4];
z1 = T0[ y1 >> 24        ] ^ T1[(y2 >> 16) & 0xff] \
   ^ T2[(y3 >>  8) & 0xff] ^ T3[ y0        & 0xff] ^ rk[5];
z2 = T0[ y2 >> 24        ] ^ T1[(y3 >> 16) & 0xff] \
   ^ T2[(y0 >>  8) & 0xff] ^ T3[ y1        & 0xff] ^ rk[6];
z3 = T0[ y3 >> 24        ] ^ T1[(y0 >> 16) & 0xff] \
   ^ T2[(y1 >>  8) & 0xff] ^ T3[ y2        & 0xff] ^ rk[7];
```

What a machine is really doing

Tables and keys

```
unsigned char rk[176], T0[1024], T1[1024], T2[1024], T3[1024];
```

The first round of AES

```
z0 = *(uint32 *)(rk + 16);
z1 = *(uint32 *)(rk + 20);
z2 = *(uint32 *)(rk + 24);
z3 = *(uint32 *)(rk + 28);

z0 ^= *(uint32 *) (T0 + ((y0 >> 22) & 0x3fc)) ^ *(uint32 *) (T1 + ((y1 >> 14) & 0x3fc)) \
   ^  *(uint32 *) (T2 + ((y2 >>  6) & 0x3fc)) ^ *(uint32 *) (T3 + ((y3 <<  2) & 0x3fc));
z1 ^= *(uint32 *) (T0 + ((y1 >> 22) & 0x3fc)) ^ *(uint32 *) (T1 + ((y2 >> 14) & 0x3fc)) \
   ^  *(uint32 *) (T2 + ((y3 >>  6) & 0x3fc)) ^ *(uint32 *) (T3 + ((y0 <<  2) & 0x3fc));
z2 ^= *(uint32 *) (T0 + ((y2 >> 22) & 0x3fc)) ^ *(uint32 *) (T1 + ((y3 >> 14) & 0x3fc)) \
   ^  *(uint32 *) (T2 + ((y0 >>  6) & 0x3fc)) ^ *(uint32 *) (T3 + ((y1 <<  2) & 0x3fc));
z3 ^= *(uint32 *) (T0 + ((y3 >> 22) & 0x3fc)) ^ *(uint32 *) (T1 + ((y0 >> 14) & 0x3fc)) \
   ^  *(uint32 *) (T2 + ((y1 >>  6) & 0x3fc)) ^ *(uint32 *) (T3 + ((y2 <<  2) & 0x3fc));
```

Interleaving tables

Tables and keys

```
unsigned char rk[176], tables[4096];
#define T0 (&tables[0])
#define T1 (&tables[4])
#define T2 (&tables[2048])
#define T3 (&tables[2052])
```

The first round of AES

```
z0 = *(uint32 *)(rk + 16);
z1 = *(uint32 *)(rk + 20);
z2 = *(uint32 *)(rk + 24);
z3 = *(uint32 *)(rk + 28);

z0 ^= *(uint32 *) (T0 + ((y0 >> 21) & 0x7f8)) ^ *(uint32 *) (T1 + ((y1 >> 13) & 0x7f8)) \
   ^  *(uint32 *) (T2 + ((y2 >>  5) & 0x7f8)) ^ *(uint32 *) (T3 + ((y3 <<  3) & 0x7f8));
z1 ^= *(uint32 *) (T0 + ((y1 >> 21) & 0x7f8)) ^ *(uint32 *) (T1 + ((y2 >> 13) & 0x7f8)) \
   ^  *(uint32 *) (T2 + ((y3 >>  5) & 0x7f8)) ^ *(uint32 *) (T3 + ((y0 <<  3) & 0x7f8));
z2 ^= *(uint32 *) (T0 + ((y2 >> 21) & 0x7f8)) ^ *(uint32 *) (T1 + ((y3 >> 13) & 0x7f8)) \
   ^  *(uint32 *) (T2 + ((y0 >>  5) & 0x7f8)) ^ *(uint32 *) (T3 + ((y1 <<  3) & 0x7f8));
z3 ^= *(uint32 *) (T0 + ((y3 >> 21) & 0x7f8)) ^ *(uint32 *) (T1 + ((y0 >> 13) & 0x7f8)) \
   ^  *(uint32 *) (T2 + ((y1 >>  5) & 0x7f8)) ^ *(uint32 *) (T3 + ((y2 <<  3) & 0x7f8));
```

```
z0 = *(uint32 *)(rk + 16);
z1 = *(uint32 *)(rk + 20);
z2 = *(uint32 *)(rk + 24);
z3 = *(uint32 *)(rk + 28);

z0 ^= *(uint32 *) (T0 + ((y0 >> 21) & 0x7f8)) \
   ^  *(uint32 *) (T1 + ((y1 >> 13) & 0x7f8)) \
   ^  *(uint32 *) (T2 + ((y2 >>  5) & 0x7f8)) \
   ^  *(uint32 *) (T3 + ((y3 <<  3) & 0x7f8));
z1 ^= *(uint32 *) (T0 + ((y1 >> 21) & 0x7f8)) \
   ^  *(uint32 *) (T1 + ((y2 >> 13) & 0x7f8)) \
   ^  *(uint32 *) (T2 + ((y3 >>  5) & 0x7f8)) \
   ^  *(uint32 *) (T3 + ((y0 <<  3) & 0x7f8));
z2 ^= *(uint32 *) (T0 + ((y2 >> 21) & 0x7f8)) \
   ^  *(uint32 *) (T1 + ((y3 >> 13) & 0x7f8)) \
   ^  *(uint32 *) (T2 + ((y0 >>  5) & 0x7f8)) \
   ^  *(uint32 *) (T3 + ((y1 <<  3) & 0x7f8));
z3 ^= *(uint32 *) (T0 + ((y3 >> 21) & 0x7f8)) \
   ^  *(uint32 *) (T1 + ((y0 >> 13) & 0x7f8)) \
   ^  *(uint32 *) (T2 + ((y1 >>  5) & 0x7f8)) \
   ^  *(uint32 *) (T3 + ((y2 <<  3) & 0x7f8));
```

Baseline for all architectures

- Each round has 20 loads, 16 shifts, 16 masks and 16 xors

Baseline for all architectures

- Each round has 20 loads, 16 shifts, 16 masks and 16 xors
- Last round is slightly different: Needs 16 more mask instructions
- Four load instructions to load input, four xors with key stream, four stores for output
- . . . some more overhead
- Results in 720 instructions needed to encrypt a block of 16 bytes
- Specifically: 208 loads, 4 stores, 508 integer instructions

Baseline for all architectures

- Each round has 20 loads, 16 shifts, 16 masks and 16 xors
- Last round is slightly different: Needs 16 more mask instructions
- Four load instructions to load input, four xors with key stream, four stores for output
- . . . some more overhead
- Results in 720 instructions needed to encrypt a block of 16 bytes
- Specifically: 208 loads, 4 stores, 508 integer instructions

Two strategies to make these instructions execute fast:

- Reduce the number of instructions
- Make the machine execute the instructions as fast as possible

UltraSPARC

(Not the most common architecture but an easy-to-understand example)

UltraSPARC

(Not the most common architecture but an easy-to-understand example)

- 64-bit architecture
- In-order execution
- Up to 4 instructions per cycle
- At most 2 integer instructions per cycle
- At most 1 load/store instruction per cycle
- Instructions have latencies
- 24 integer registers available

Fast AES on the UltraSPARC

Reminder: 208 loads, 4 stores, 508 integer instructions

- Only one load or store per cycle ($\Rightarrow$ at least 212 cycles)
- Only 2 integer instructions per cycle ($\Rightarrow$ at least 254 cycles)
- Make sure to have 2 integer instructions each cycle
- Make sure to not have extra cycles for load/store instructions
- Result: 254 cycles/block
- 15.98 cycles/byte in the eSTREAM benchmarking framework for encryption of 4096 bytes

Padded registers (saving 80 instructions)

Idea: The UltraSPARC is a 64-bit architecture, pad 32-bit values with zeros, i.e. 0xc66363a5 becomes 0x0c60063006300a50
Do that consistently for values in registers, the tables and the round keys, then:

Without padded registers

```
p00 = (uint32) y0 >> 21
p01 = (uint32) y0 >> 13
p02 = (uint32) y0 >> 5
p03 = (uint32) y0 << 3
p00 &= 0x7f8
p01 &= 0x7f8
p02 &= 0x7f8
p03 &= 0x7f8
```

With padded registers

```
p00 = (uint64) y0 >> 48
p01 = (uint64) y0 >> 32
p02 = (uint64) y0 >> 16
p01 &= 0xff0
p02 &= 0xff0
p03 = y0 & 0xff0
```

Padded registers (saving 120 instructions)

Idea: The UltraSPARC is a 64-bit architecture, pad 32-bit values with zeros, i.e. 0xc66363a5 becomes 0x0c60063006300a50
Do that consistently for values in registers, the tables and the round keys, then:

Without padded registers

```
p00 = (uint32) y0 >> 21
p01 = (uint32) y0 >> 13
p02 = (uint32) y0 >> 5
p03 = (uint32) y0 << 3
p00 &= 0x7f8
p01 &= 0x7f8
p02 &= 0x7f8
p03 &= 0x7f8
```

Padded registers and 32-bit shift

```
p00 = (uint64) y0 >> 48
p01 = (uint64) y0 >> 32
p02 = (uint32) y0 >> 16
p01 &= 0xff0
p03 = y0 & 0xff0
```

Counter-mode caching (saving about 100 instructions)

Idea: Make use of the fact that AES is used in CTR mode

- A 16-byte counter is increased by 1 for every block
- 15 bytes of the counter remain constant for 256 blocks
- In the first round most computations only depend on these 15 bytes
- 12 bytes of the output of the first round remain constant for 256 blocks
- In the second round most computations only depend on these 12 bytes

Counter-mode caching (saving about 100 instructions)

Idea: Make use of the fact that AES is used in CTR mode

- A 16-byte counter is increased by 1 for every block
- 15 bytes of the counter remain constant for 256 blocks
- In the first round most computations only depend on these 15 bytes
- 12 bytes of the output of the first round remain constant for 256 blocks
- In the second round most computations only depend on these 12 bytes

Do all these computations only once for 256 subsequent blocks and cache the result.

Results

Encryption speed given in cycles/byte for encrypting a block of 4096 bytes

	Verifiable eSTREAM results		Unverifiable claims
Architecture	This paper	Best previous	Unpublished code
UltraSPARC III	12.06	20.75 (Bernstein)	16.875 (Lipmaa)
Power PC G4	14.57	16.26 (Wu)	25.06 (Ahrens)
Pentium 4 f12	14.15	16.97 (Bernstein)	15 (Osvik)
Core 2 6fb	10.57	12.27 (Wu)	14.5 (Matsui & Nakajima)
Athlon 64	10.43	13.32 (Wu)	10.62 (Matsui)

Results

Encryption speed given in cycles/byte for encrypting a block of 4096 bytes

	Verifiable eSTREAM results		Unverifiable claims
Architecture	This paper	Best previous	Unpublished code
UltraSPARC III	12.06	20.75 (Bernstein)	16.875 (Lipmaa)
Power PC G4	14.57	16.26 (Wu)	25.06 (Ahrens)
Pentium 4 f12	14.15	16.97 (Bernstein)	15 (Osvik)
Core 2 6fb	10.57	12.27 (Wu)	14.5 (Matsui & Nakajima)
Athlon 64	10.43	13.32 (Wu)	10.62 (Matsui)

Some post-paper results

- Measurements within the eBACS project on a Core 2 10676: 9.31 cycles/byte
- Young reports 9.12 cycles/byte on a Core 2 6fb after integrating optimizations described in this paper

Further information

Paper and Software

`http://cr.yp.to/aes-speed.html`

eSTREAM and eBACS

`http://www.ecrypt.eu.org/stream/`
`http://bench.cr.yp.to/`

qhasm

`http://cr.yp.to/qhasm.html`
`http://cryptojedi.org/programming/qhasm-doc/`