# Implementing AES 2000-2010: performance and security challenges

Emilia Käsper

Katholieke Universiteit Leuven

SPEED-CC
Berlin, October 2009

# The Advanced Encryption Standard

- Rijndael proposed by Rijmen, Daemen in 1998
- Selected as AES in October 2000
- Key size 128/192/256 bits (resp. 10/12/14 rounds)
- Software performance a key advantage
  - Runner-up Serpent arguably "more secure", but over 2x slower
- AES in OpenSSL — implementation by Rijmen, Bosselaers, Barreto from 2000
- AES-128 at around 18 cycles/byte = 110 MB/s @ 2GHz

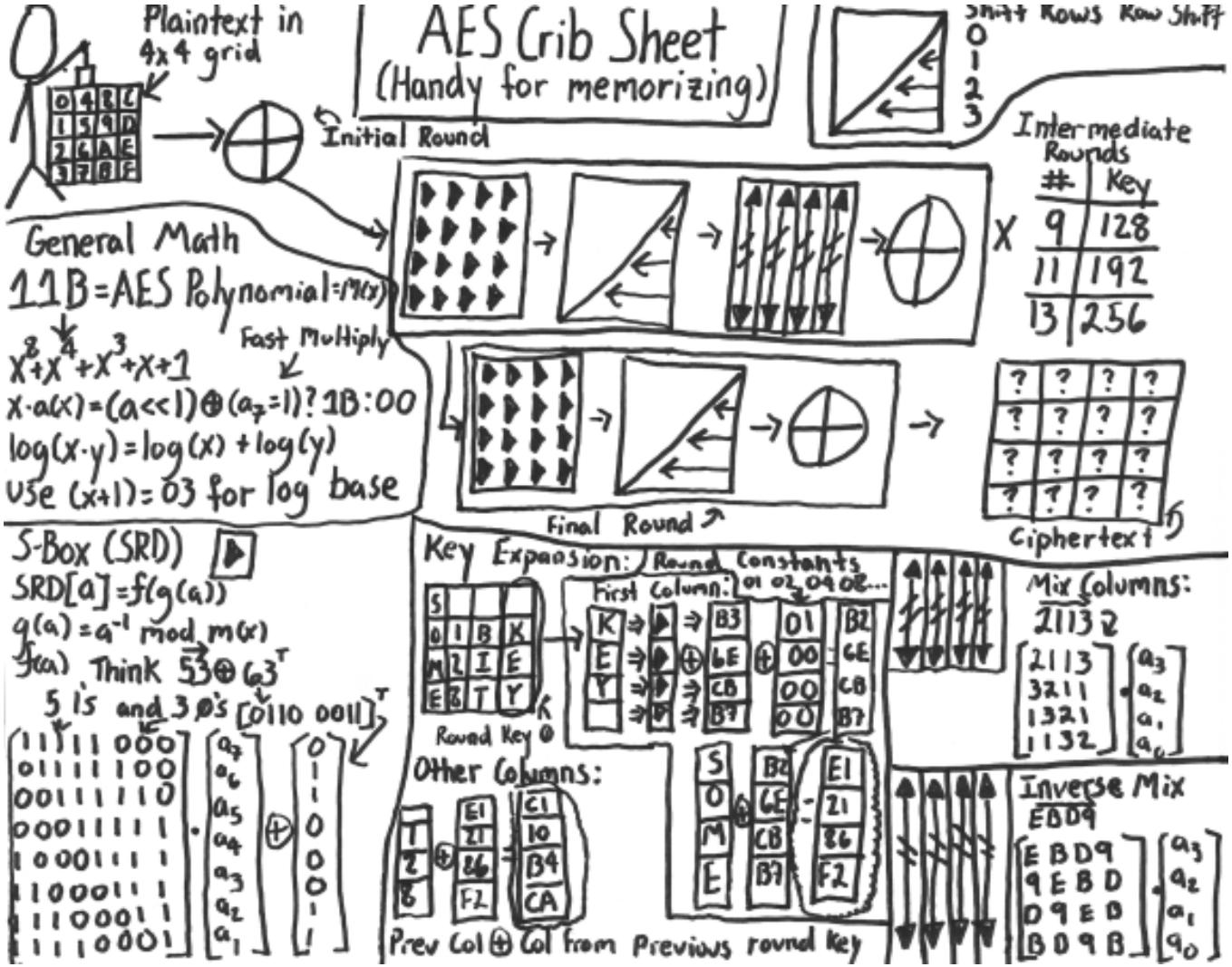
# The AES performance challenge

- Is 110 MB/s fast enough?
- Popular example: Truecrypt transparent disk encryption
- Truecrypt only supports AES-256, so make that 80 MB/s
- At the same time, consumer (solid state) hard drives can read at over 200 MB/s
- Encryption becomes performance bottleneck
- Since March 2008, Truecrypt includes an optimized assembly implementation of AES

# Optimized implementations on Intel processors

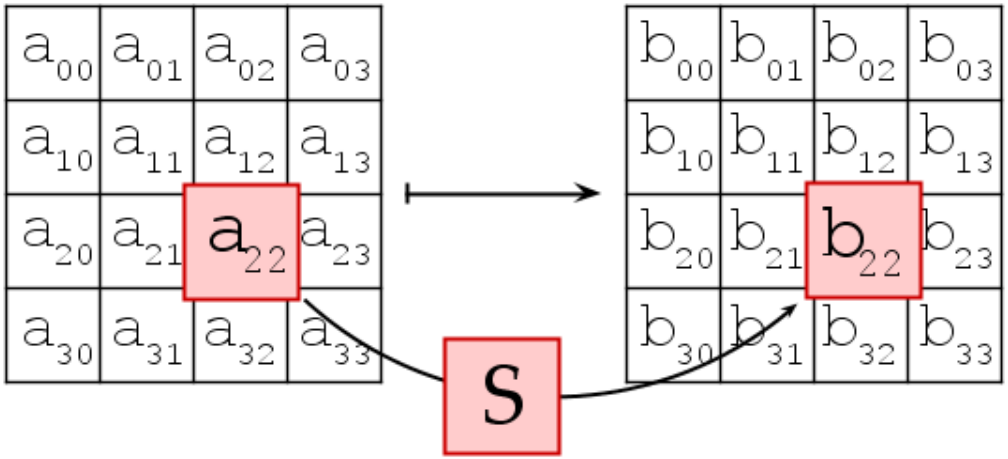
- 2000: Aoki and Lipmaa report 14.8 cycles/byte on Pentium II
- ...
- 2007: Matsui and Nakajima report 9.2 cycles/byte for AES-CTR on Core 2
  - Assuming data is processed in 2 KB blocks
  - Compatibility with existing implementations via an extra input/output transform
- 2008: Bernstein-Schwabe report 10.57 cycles/byte for AES-CTR on Core 2
- 2009: Käsper-Schwabe report 7.59 cycles/byte for AES-CTR on Core 2

# Inside AES

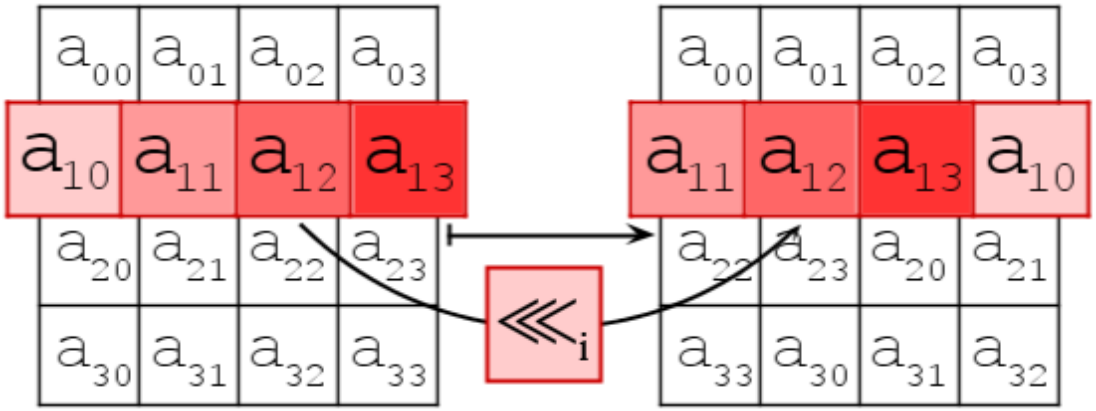

AES Crib Sheet
(Handy for memorizing)
Plaintext in 4x4 grid
Initial Round
Intermediate Rounds
# Key
9 128
11 192
13 256
Final Round
Ciphertext
General Math
11B = AES Polynomial = m(x)
$x^8+x^4+x^3+x+1$
Fast Multiply
$x \cdot a(x) = (a<<1) \oplus (a_7 = 1)\ ?\ 1B : 00$
$\log(x \cdot y) = \log(x) + \log(y)$
use (x+1) = 03 for log base
S-Box (SRD)
SRD[a] = f(g(a))
$g(a) = a^{-1} \bmod m(x)$
f(a) Think $53 \oplus 63^T$
5 1's and 3 0's [0110 0011]T
Key Expansion:
Round Constants
01 02 04 08...
First Column:
Round Key
Other Columns:
Prev Col ⊕ Col from Previous round key
Mix Columns:
2113
Inverse Mix
EBD9

# AES round structure

- SubBytes is an S-Box acting on individual bytes

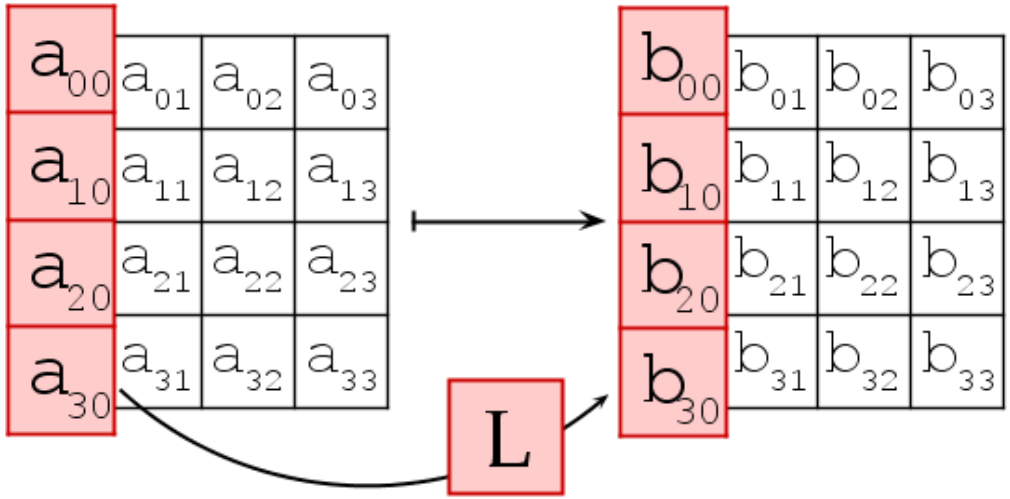



- ShiftRows rotates each row by a different amount

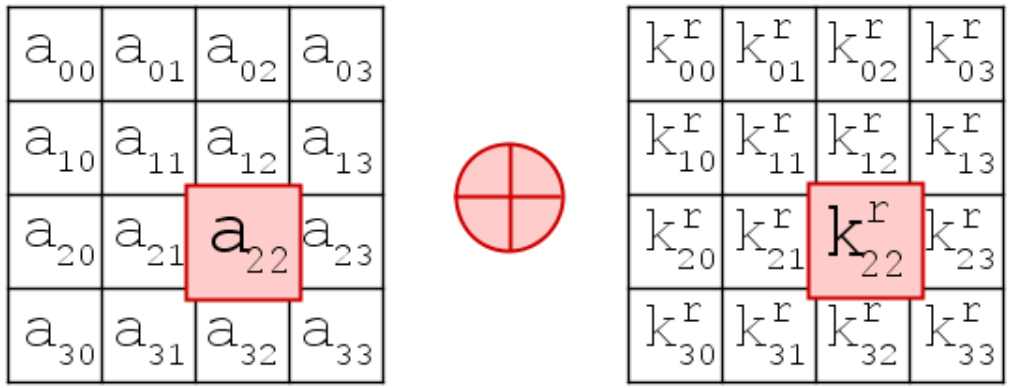


# AES round structure (cont.)

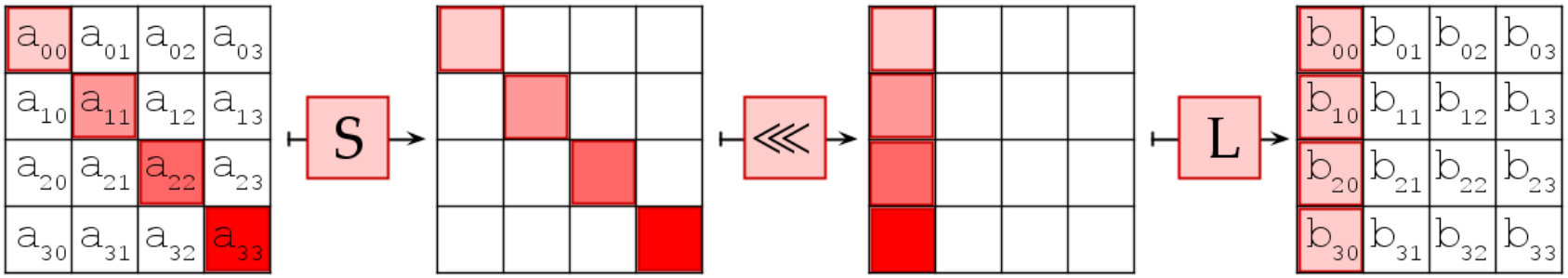
- MixColumns is a linear transformation on columns




- AddRoundKey XORs the 128-bit round key to the state



# Implementing an AES round

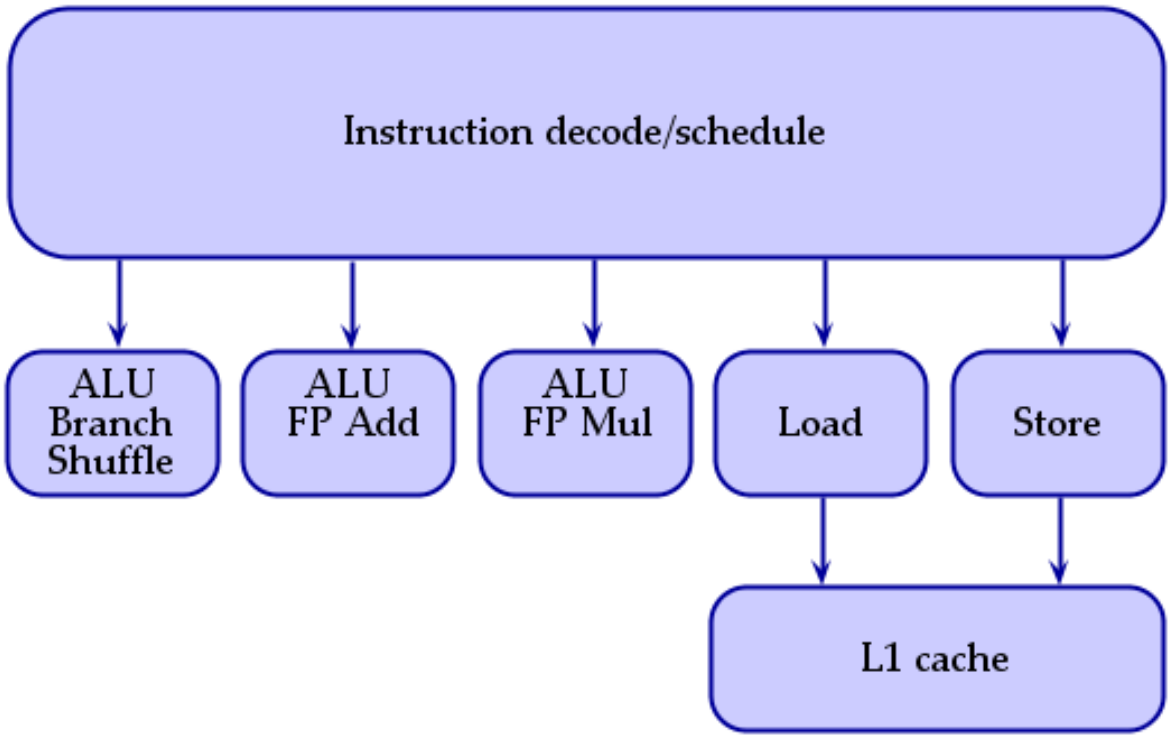



- Store AES state in 4 column vectors
- Combine SubBytes, ShiftRows and MixColumns:
- Each column vector depends on 4 bytes
- Do 4 8-to-32-bit table lookups and combine using XOR

```
b0 = T0[ a0 >> 24          ] ^ T1[(a1 >> 16) & 0xff]
   ^ T2[(a2 >>  8) & 0xff] ^ T3[ a3         & 0xff] ^ rk[4];
b1 = T0[ a1 >> 24          ] ^ T1[(a2 >> 16) & 0xff]
   ^ T2[(a3 >>  8) & 0xff] ^ T3[ a0         & 0xff] ^ rk[5];
b2 = T0[ a2 >> 24          ] ^ T1[(a3 >> 16) & 0xff]
   ^ T2[(a0 >>  8) & 0xff] ^ T3[ a1         & 0xff] ^ rk[6];
b3 = T0[ a3 >> 24          ] ^ T1[(a0 >> 16) & 0xff]
   ^ T2[(a1 >>  8) & 0xff] ^ T3[ a2         & 0xff] ^ rk[7];
```

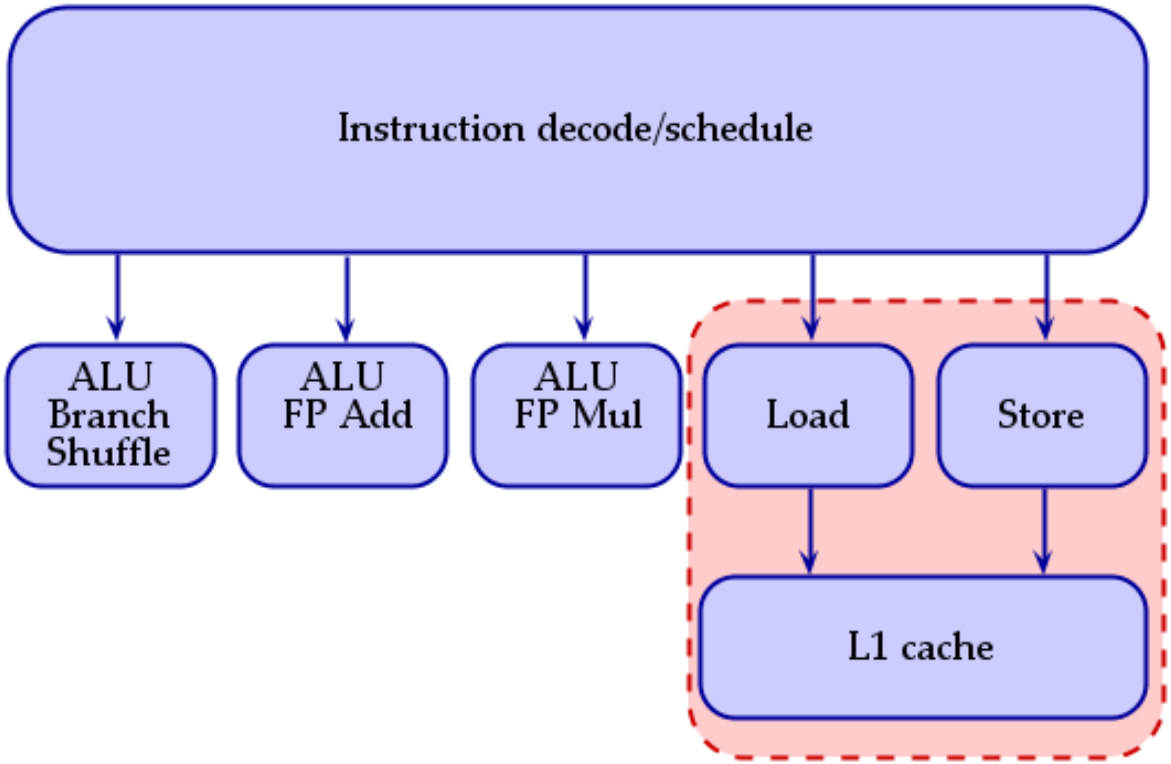
# AES performance on Core 2

## The Core 2 execution units



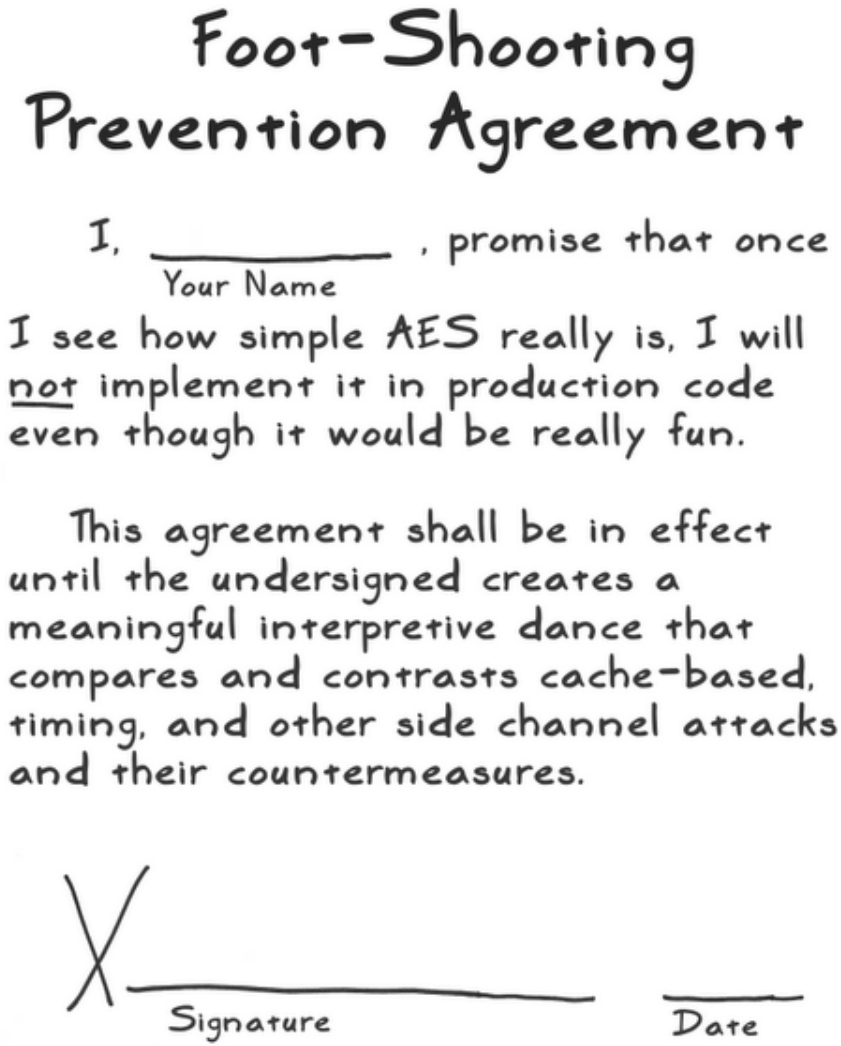
# AES performance on Core 2

## The Core 2 execution units




- The Core 2 can do one load per clock cycle
- AES-128 needs 160 table lookups to encrypt 16 bytes
- 10 cycles/byte barrier using this technique

# The AES security challenge



Foot-Shooting Prevention Agreement

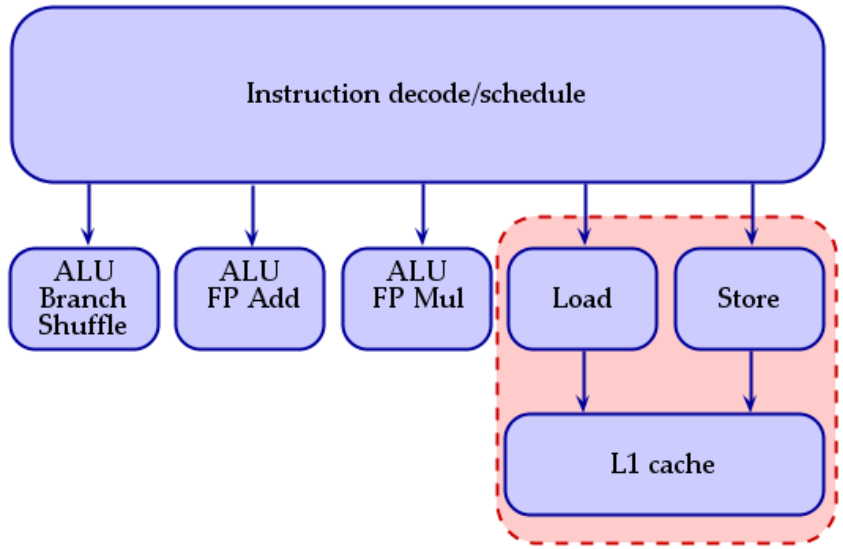
I, _____ , promise that once
Your Name
I see how simple AES really is, I will not implement it in production code even though it would be really fun.

This agreement shall be in effect until the undersigned creates a meaningful interpretive dance that compares and contrasts cache-based, timing, and other side channel attacks and their countermeasures.

X _____ _____
Signature Date

# Cache attacks on AES implementations

- Core idea (Kocher, 1996): variable-time instructions manipulating the secret key leak information about key bits
- Table lookups take different time depending on whether the value was retrieved from cache or memory
  - The case of AES: lookup table indices directly depend on the secret key
  - First round of AES: `T[plaintext` $\oplus$ `roundkey]`
  - Knowing which part of the table was accessed leaks key bits



# Cache attacks (cont.)

- A variety of attack models
  - Active cache manipulation via user processes — preload cache with known values and observe via timing if the cache was hit
  - Passive (remote) timing of cache "hits" and "misses" — shorter encryption time implies collisions in lookups
  - Power traces
- Example: passive timing attack (Bonneau, Mironov 2006)
  - Attacker runs timing code on target machine
  - Obtain timing data from $2^{14}$ random encryptions
  - Deduce when first-round collisions occur to recover 5 bits of each key byte (assuming 32-byte cache line)
  - Can be improved to recover the whole key by considering second/last round

# Countermeasures against cache attacks

- Protecting vulnerable cipher parts (e.g., first and last round) in software — only thwarts current attacks
- Add variable-time dummy instructions — attacks still work with more data
- Cache warming (preload some values) — for 32-byte cache line, $\frac{4 \cdot 256}{8} = 128$ instructions to preload all tables
- Force all operations to take constant time — as good as having no cache
- Algorithm-specific constant-time implementations

# Bitslicing AES

- Bitslicing (Biham, 1997): instead of using lookup tables, evaluate S-Boxes on the fly using their Boolean form
- Efficient if multiple S-boxes can be computed in parallel
- Serpent: bitsliced design, 32 4 $\times$ 4-bit S-boxes in each round
- AES 8 $\times$ 8 S-box based on Galois field inversion, matrix multiplication: ???
  - 2007: Matsui shows an efficient implementation using 128 parallel blocks
  - 2008: Könighofer's implementation on 64-bit processors, 4 parallel blocks, $<$ 20 cycles/byte

# Bitslicing AES on Core 2 (2009)

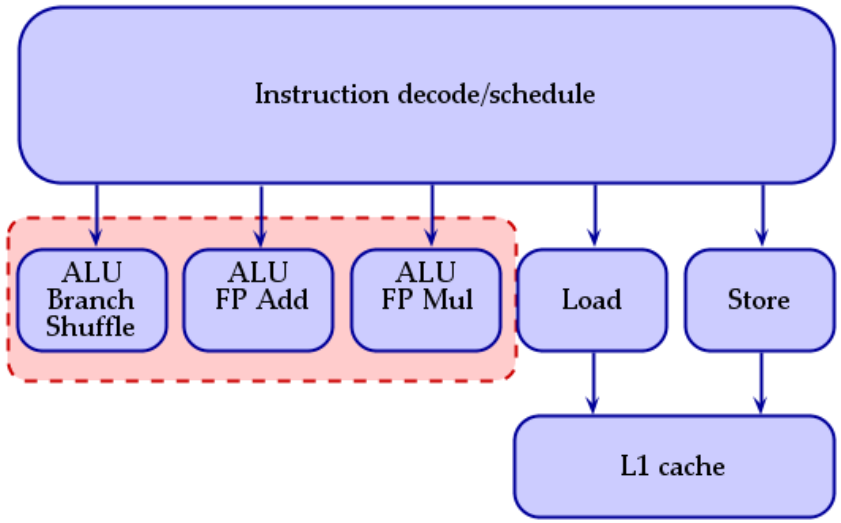
- Implementation of AES in counter mode
- Applicable to any other parallel mode
  - Counter mode particularly handy, as no need to implement decryption
- Hand-coded in GNU assembly/qhasm
- Constant-time, immune to **all** timing attacks

**New speed record**

7.59 cycles/byte for large blocks

- Also fast for packet encryption

# Making the most out of Core 2




- 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 ALU units – up to 3 bit-logical instructions per cycle

# 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 |

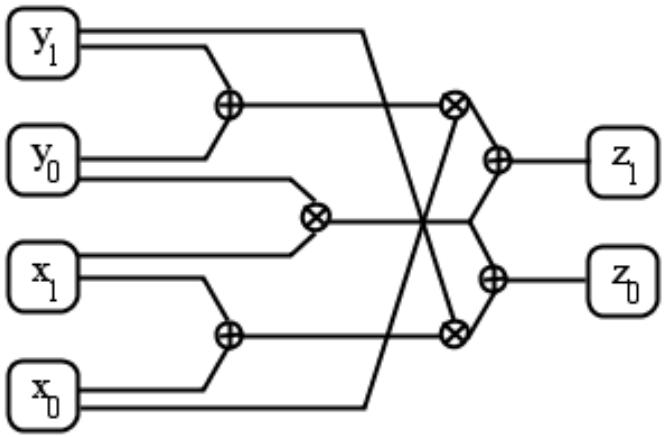
# Hardware vs software

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

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

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

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




```
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 - $8 \times 128$ bits per round, key expansion slower
  - SSSE3 available only on Intel, not on AMD processors

# Putting it all together

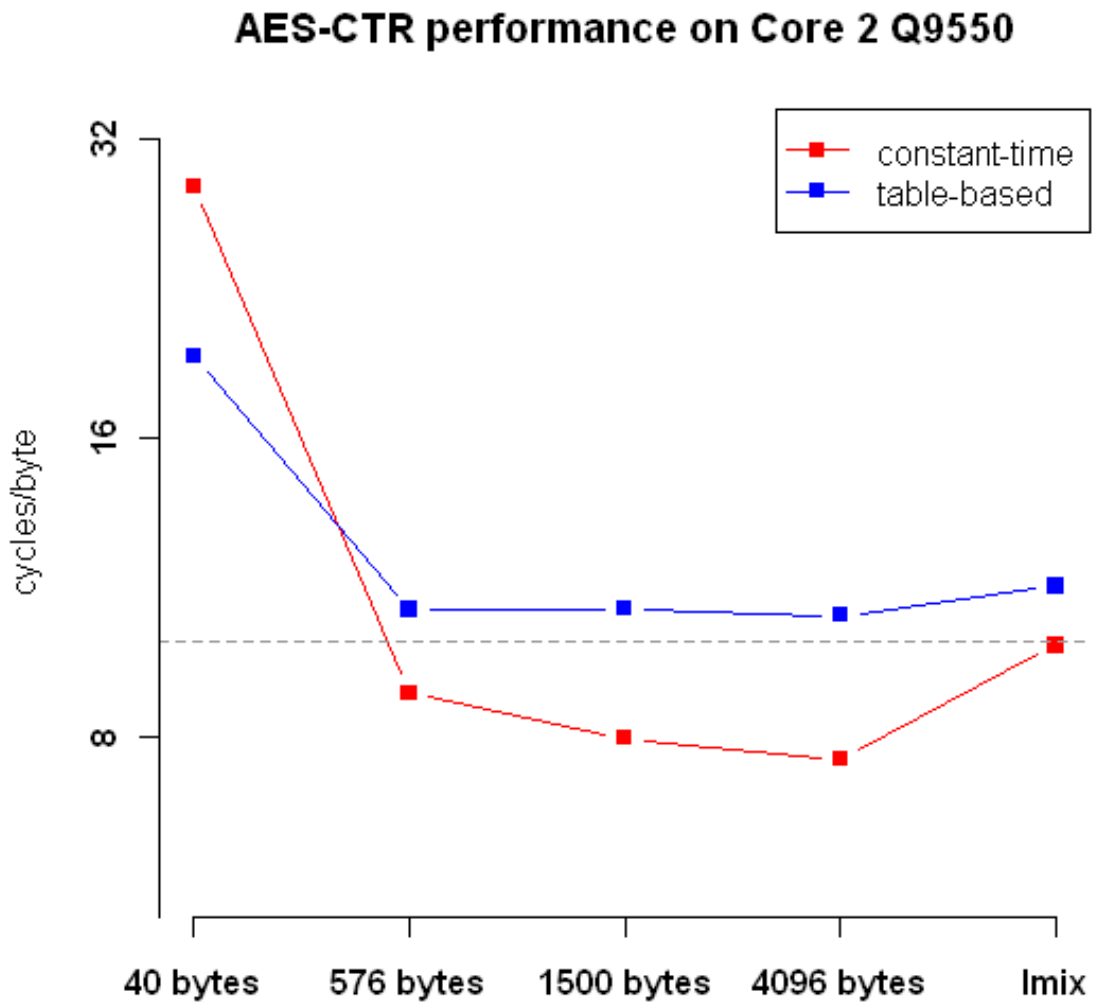
| | xor/and/or | pshufb/d | xor (mem-reg) | mov (reg-reg) | TOTAL |
|---|---|---|---|---|---|
| SubBytes | 128 | – | – | 35 | 163 |
| ShiftRows | – | 8 | – | – | 8 |
| MixColumns | 27 | 16 | – | – | 43 |
| AddRoundKey | – | – | 8 | – | 8 |
| TOTAL | 155 | 24 | 8 | 35 | 222 |

- One AES round requires 214 bit-logical instructions
- Last round omits MixColumns — 171 instructions
- Input/output transform 84 instructions/each
- Excluding data loading etc, we get a lower bound

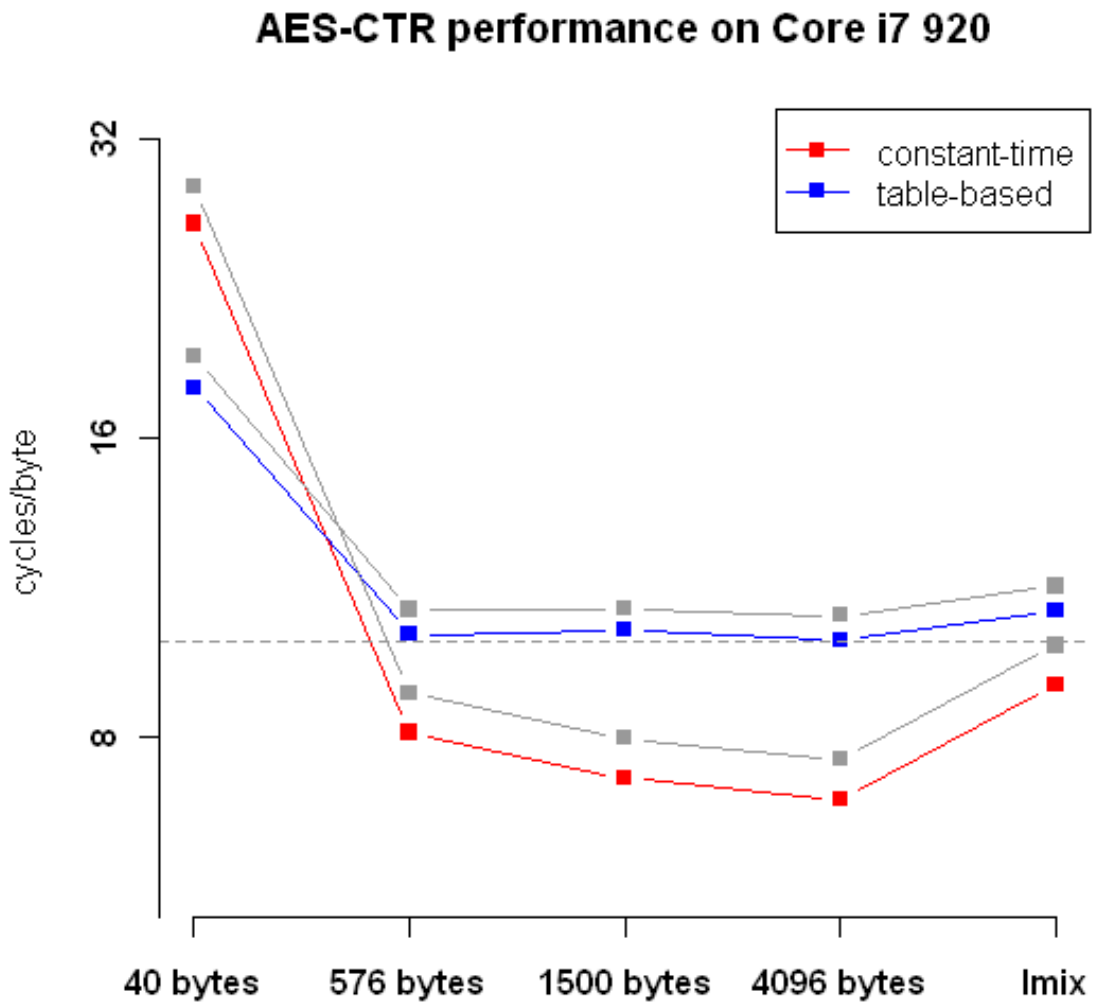
$$\frac{214 \times 9 + 171 + 2 \times 84}{3 \times 128} = 5.9 \text{ cycles/byte}$$

- Actual performance on Core 2 7.59 cycles/byte

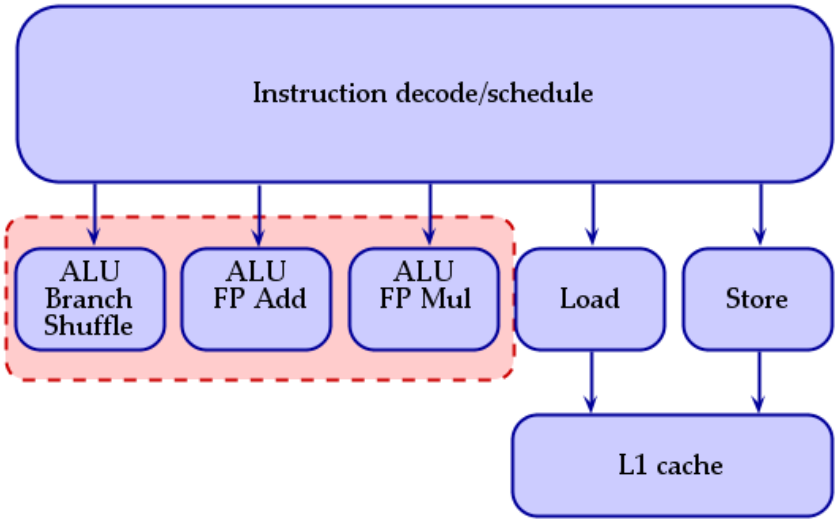
# eStream benchmarks of AES-CTR-128



# Even faster on the Core i7...

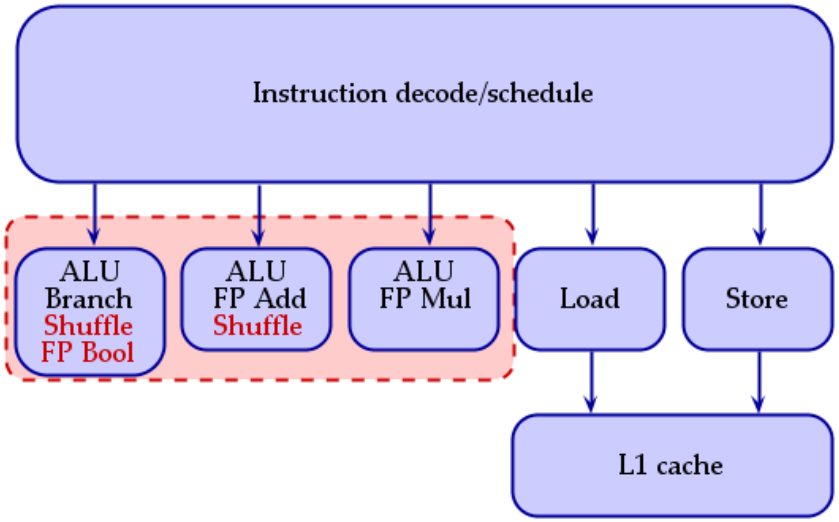


# Interlude: A little lesson...




- 3 logically equivalent instructions: `xorps`, `xorpd`, `pxor`
- On Core 2, we saw no performance difference
- On Core i7, using `xorps`/`xorpd` gave a 50% performance hit

# Interlude: A little lesson...




- 3 logically equivalent instructions: `xorps`, `xorpd`, `pxor`
- On Core 2, we saw no performance difference
- On Core i7, using `xorps`/`xorpd` gave a 50% performance hit
- The reason: only one unit in Core i7 handles fp Boolean

**Lesson**

Always use the instruction appropriate for your data type!

# Implementing AES 2010-...

- Intel has announced hardware support for AES in its next generation processors (AES-NI instruction set extension)
- Implementation simplicity:

```
b0 = T0[ a0 >> 24        ] ^ T1[(a1 >> 16) & 0xff]
   ^ T2[(a2 >>  8) & 0xff] ^ T3[ a3        & 0xff] ^ rk[4];
b1 = T0[ a1 >> 24        ] ^ T1[(a2 >> 16) & 0xff]
   ^ T2[(a3 >>  8) & 0xff] ^ T3[ a0        & 0xff] ^ rk[5];
b2 = T0[ a2 >> 24        ] ^ T1[(a3 >> 16) & 0xff]
   ^ T2[(a0 >>  8) & 0xff] ^ T3[ a1        & 0xff] ^ rk[6];
b3 = T0[ a3 >> 24        ] ^ T1[(a0 >> 16) & 0xff]
   ^ T2[(a1 >>  8) & 0xff] ^ T3[ a2        & 0xff] ^ rk[7];
```

  becomes

```
aesenc xmm1, xmm3    % xmm1 - data, xmm3 - key
```

- Mitigates side-channel attacks
- Performance
  - Straightforward 4.4 cycles/byte
  - Parallel/optimized 1.35 cycles/byte

# Concluding remarks I

- 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 — AES was designed to be implemented with lookup tables
- In comparison: Whirlpool hash function uses an $8 \times 8$ S-Box composed of 5 $4 \times 4$ S-Boxes
- Small by design: 101 gates in hardware
- Slightly inferior security: maximum differential probability $2^{-5}$ vs $2^{-6}$ for AES
- AES requires only 16 gates more!
- But this is a result of 10 years of optimization...

# Concluding remarks II

- 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
- Next generation processors: 256-bit registers, three operand instructions
- The case of Serpent: processing 4 blocks in parallel (8 for 256-bit) could yield up to factor 4 (8) performance improvement
- In Practice: reports of factor 2.7 improvement over a previous implementation

# Links

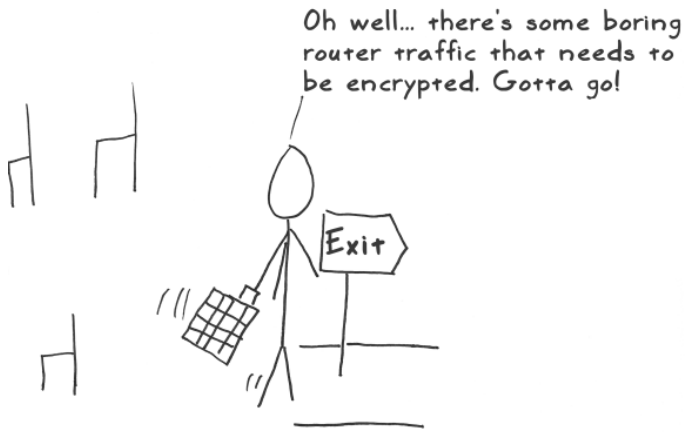
QHASM implementations of bitsliced AES:

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

GNU asm implementations:

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

A Stick Figure Guide to the Advanced Encryption Standard:




`http://www.moserware.com`