

How to Make AES run Faster

[^]
much

Emily Stark Mike Hamburg Dan Boneh

Stanford University

What is AES?

Advanced Encryption Standard

- Standardized in 2001
- Block cipher of choice for new designs
- Design components are widely reused

Design is a “substitution-permutation network”

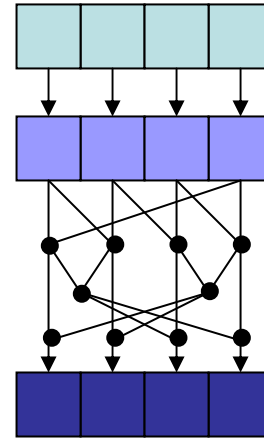
- Do table lookups
- Mix together results
- Add in key
- Repeat

AES Implementation Techniques

Small Tables

(reference, smart cards)

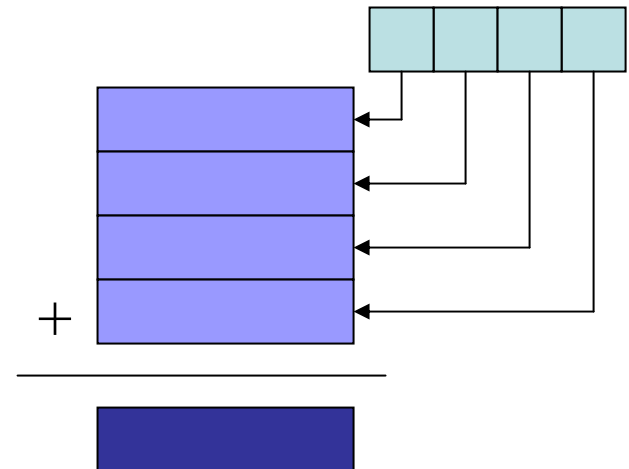
- Straightforward from spec
- 256B tables
- Mixing is complicated
- Relatively slow



Big Tables

(openssl)

- Mixing built into tables
- ~4x faster
- 4KB tables
- Opens up cache-timing attack

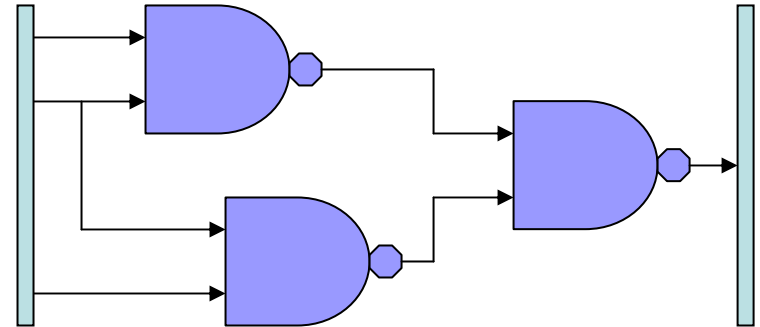


AES Implementation Techniques (cont.)

No tables

(hardware)

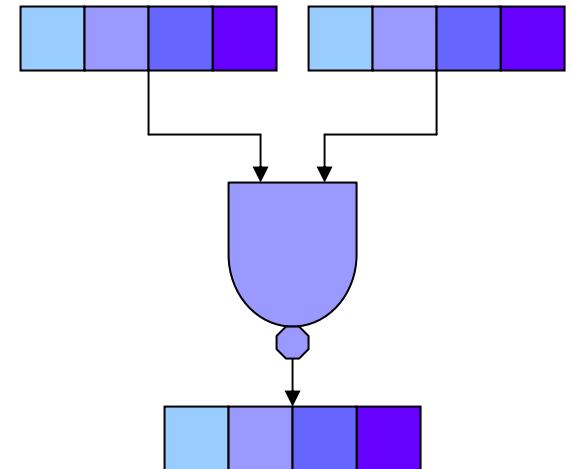
- Very slow in software
- Hard to avoid timing attacks
- Good in hardware



Bitslicing

(bulk encryption)

- 4-128 parallel “circuits” in software
- Can be very fast
- Parallel modes only
- Avoids timing attack



Vectorizing AES

Vectorization makes everything faster! TM

- Whole 128-bit block at a time
- But how do we do the table lookups?

Use vector permute instruction

- Available on PowerPC-like processors
 - G4, G5, e600, Cell, Xenon and AMD “Bulldozer”
 - Weaker version in SSE3
- 16 parallel table lookups!
- ... but the table is small

Structure of AES lookup table

AES table is defined mathematically

- Inversion over $GF(2^8)$
- Still too complicated

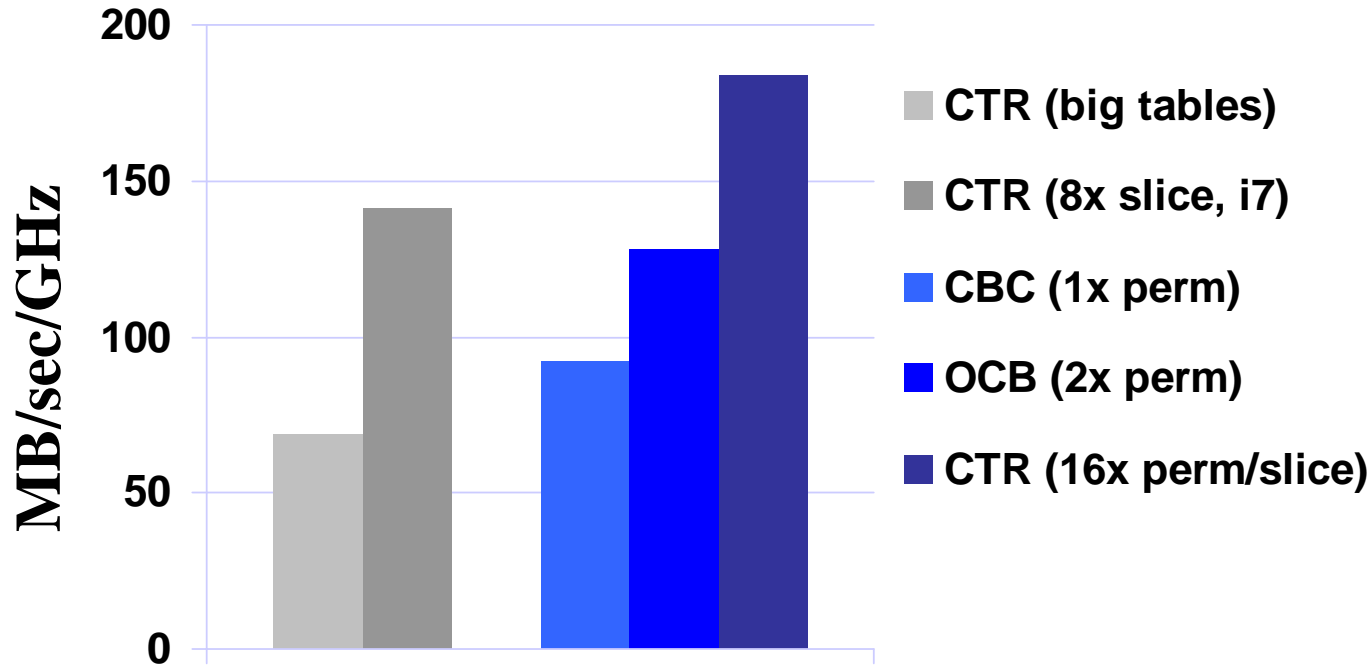
Split the field

- Write $GF(2^8)$ as $GF(2^4)^2$
- Used in hardware for several years

Multiply using log tables

- “Packed add signed bytes with saturation”

Performance (PowerPC G4e)



Faster per cycle
than state-of-the-art code (< 1 mo)
on state-of-the-art processor (< 1 yr)
in serial or parallel mode

Future Work

Implement on x86

- Fewer registers
- Weaker vector shuffle (except on Bulldozer)
- CISC-style 2-argument instructions
- Dual shuffle units on i7
- Memory operands (backwards!)
- Different pipeline
- Should still be pretty fast for block-at-a-time!

Implement on Cell?

A Javascript Crypto Library

Goal: open source Javascript crypto library

- Clean API, “high” performance

Why? Examples:

- Mozilla Weave
- PII info collected for 3rd party
- “honest but curious” server



AES in Javascript

Dilemma: precompute or don't precompute

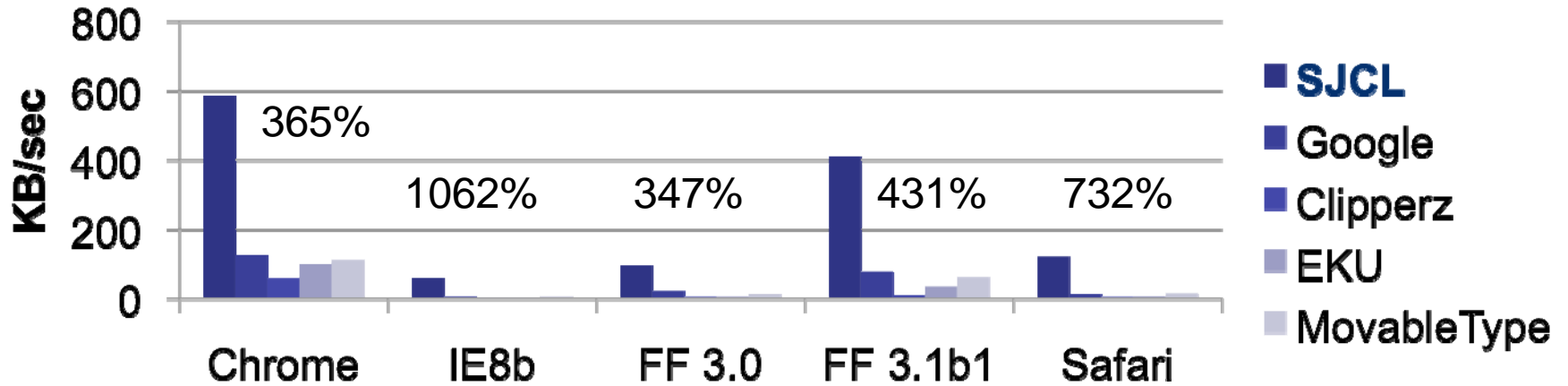
- Need speed, Javascript is slow enough as it is
- Use “big table” method
- Don't want to send two 4kB tables down the wire
- Bitslicing performs poorly

Our approach

- Keep code small: no hard coded tables
 - Ship code to compute tables
 - Pre-compute tables on browser during first call

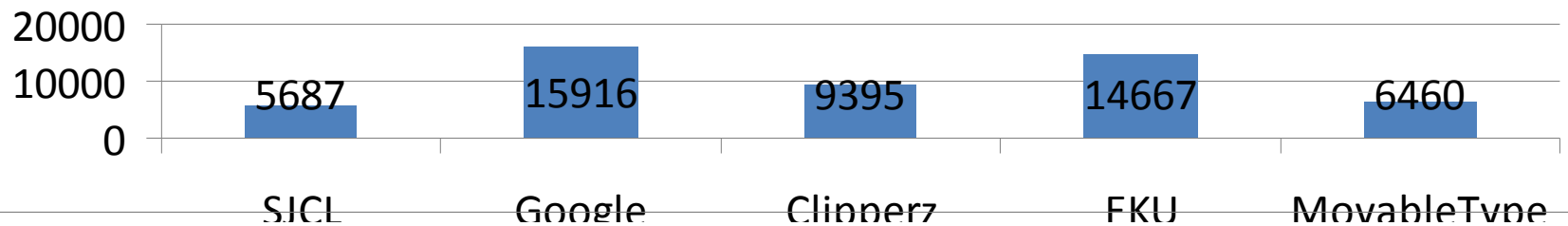
Other optimizations: loop unrolling, inlining, ...

Performance



even SJCL is 46x slower than OpenSSL

Code Size



Future Work

Implement asymmetric crypto

- RSA, DH, ECC
- PAKE

Improve JITs

- Poor Firefox performance is JIT bug
- Recognize packable tables, bit ops

That's all, folks!

Questions?