Fast Software Impl. of Crypto Primitives

A Survey Of AES & SM4 Impl.

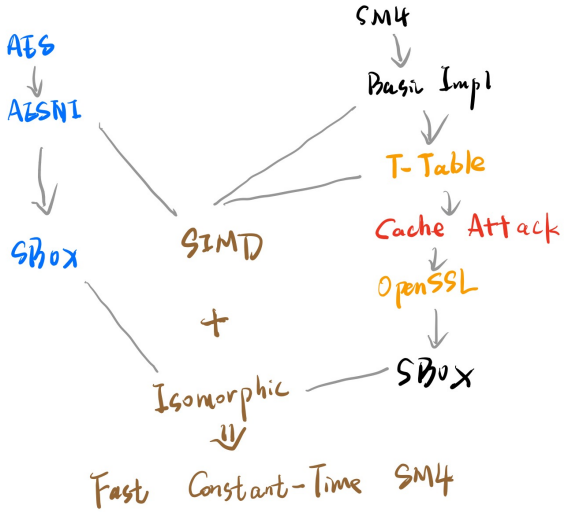
Long Wen
20250707 @ Qing Dao
longwen6@gmail.com

1

OVERVIEW

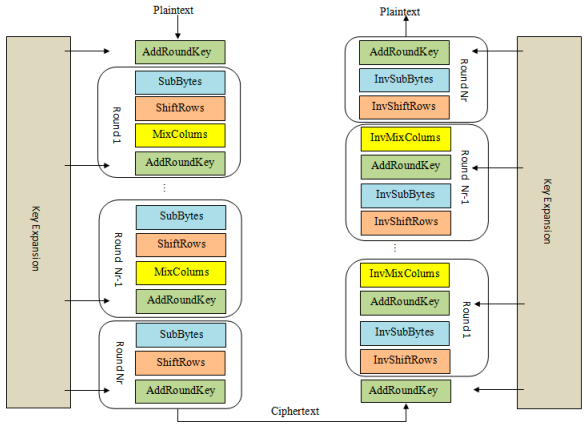
- Introduction To AES
- AES implementation via AES-NI instruction
- AES implementation via T-Table
- ~~AES implementation via subfield~~
- Introduction To SM4
- SM4 Basic Implementation & SIMD
- SM4 implementation with T-Table Method
- Cache attack against T-Table method
- Some Protection, The OpenSSL Way
- Internals of AES/SM4 SBOX
- Speed up SM4 with AES-NI
- Faster SM4 with AES-NI
- ~~What if AES-NI is unavailable?~~

2



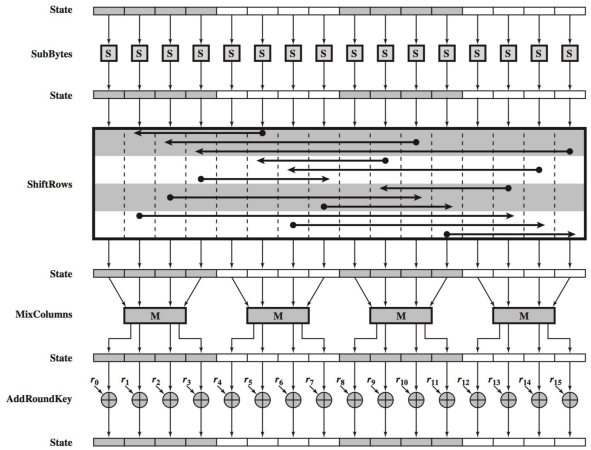
3

Introduction To AES



4

AES Round Function



5

AES Impl. With AES-NI

- Intel Advanced Encryption Standard New Instructions
- Extension to the x86 Instruction Set Architecture
- Proposed by Intel in 2008, now in Intel, AMD and ARM CPUs
- The magic part of the super speed of AES

- Six instructions
  - _mm_aesenc_si128
  - _mm_aesenclast_si128
  - _mm_aesdec_si128
  - _mm_aesdeclast_si128
  - _mm_aesimc_si128
  - _mm_aeskeygenassist_si128

Operation

```
state := a
a[127:0] := ShiftRows(a[127:0])
a[127:0] := SubBytes(a[127:0])
dst[127:0] := a[127:0] XOR RoundKey[127:0]
```

Performance

Architecture	Latency	Throughput (CPI)
Skylake	4	1
Broadwell	7	1
Haswell	7	1
Ivy Bridge	8	1

6

AES Impl. With Intel's AES-NI

```
#define DO_ENC_BLOCK(m, k)                      \
    do {                                        \
        m = _mm_xor_si128(m, k[0]);             \
        m = _mm_aesenc_si128(m, k[1]);          \
        m = _mm_aesenc_si128(m, k[2]);          \
        m = _mm_aesenc_si128(m, k[3]);          \
        m = _mm_aesenc_si128(m, k[4]);          \
        m = _mm_aesenc_si128(m, k[5]);          \
        m = _mm_aesenc_si128(m, k[6]);          \
        m = _mm_aesenc_si128(m, k[7]);          \
        m = _mm_aesenc_si128(m, k[8]);          \
        m = _mm_aesenc_si128(m, k[9]);          \
        m = _mm_aesenclast_si128(m, k[10]);     \
    } while (0)
```

```
#define DO_DEC_BLOCK(m, k)                      \
    do {                                        \
        m = _mm_xor_si128(m, k[10 + 0]);        \
        m = _mm_aesdec_si128(m, k[10 + 1]);     \
        m = _mm_aesdec_si128(m, k[10 + 2]);     \
        m = _mm_aesdec_si128(m, k[10 + 3]);     \
        m = _mm_aesdec_si128(m, k[10 + 4]);     \
        m = _mm_aesdec_si128(m, k[10 + 5]);     \
        m = _mm_aesdec_si128(m, k[10 + 6]);     \
        m = _mm_aesdec_si128(m, k[10 + 7]);     \
        m = _mm_aesdec_si128(m, k[10 + 8]);     \
        m = _mm_aesdec_si128(m, k[10 + 9]);     \
        m = _mm_aesdeclast_si128(m, k[0]);      \
    } while (0)
```

7

How Fast is AES-NI? Or How To Bench?

https://github.com/google/benchmark

8

AES Impl. With ARM's AES-NI

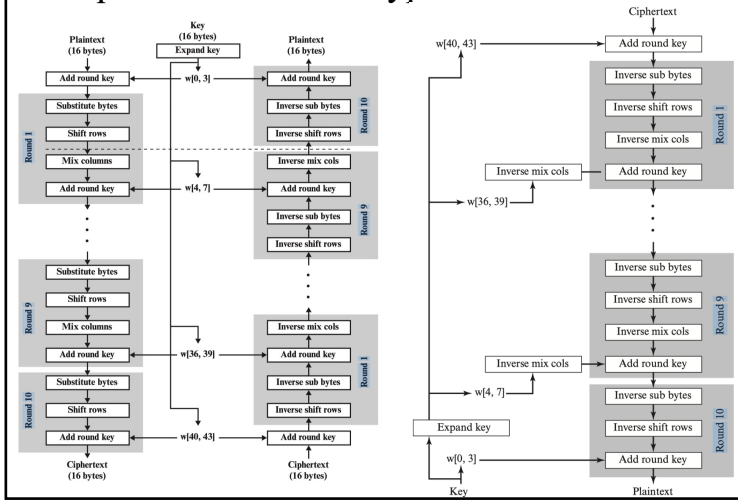
```
void aes128_enc_armv8(const uint8_t in[16], uint8_t ou[16],
                      const uint32_t rk[44]) {
    uint8x16_t block = vld1q_u8(in);

    uint8_t* p8 = (uint8_t*)rk;
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 0)));
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 1)));
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 2)));
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 3)));
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 4)));
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 5)));
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 6)));
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 7)));
    block = vaesmcq_u8(vaeseq_u8(block, vld1q_u8(p8 + 16 * 8)));
    block = vaeseq_u8(block, vld1q_u8(p8 + 16 * 9));  // final round
    block = veorq_u8(block, vld1q_u8(p8 + 16 * 10));  // final xor subkey

    vst1q_u8(ou, block);
}
```

```
void aes128_dec_armv8(const uint8_t in[16], uint8_t ou[16],
                      const uint32_t rk[44]) {
    uint8x16_t block = vld1q_u8(in);

    uint8_t* p8 = (uint8_t*)rk;
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 0)));
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 1)));
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 2)));
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 3)));
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 4)));
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 5)));
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 6)));
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 7)));
    block = vaesimcq_u8(vaesdq_u8(block, vld1q_u8(p8 + 16 * 8)));
    block = vaesdq_u8(block, vld1q_u8(p8 + 16 * 9));  // final round
    block = veorq_u8(block, vld1q_u8(p8 + 16 * 10));  // final xor subkey

    vst1q_u8(ou, block);
}
```

9

*Project: AES impl with ARM instruction

Equivalent AES Decryption



10

AES Impl – T-Table - Round Function

SubBytes	$b_{i,j} = \mathrm{S}[a_{i,j}]$
ShiftRows	$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$
MixColumns	$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$
AddRoundKey	$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$

11

AES Round Function Via T-Table

SubBytes	$b_{i,j} = \mathrm{S}[a_{i,j}]$
ShiftRows	$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$
MixColumns	$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$
AddRoundKey	$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \mathrm{S}[a_{0,j}] \\ \mathrm{S}[a_{1,j-1}] \\ \mathrm{S}[a_{2,j-2}] \\ \mathrm{S}[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

$$= \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot \mathrm{S}[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot \mathrm{S}[a_{1,j-1}] \right) \oplus \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot \mathrm{S}[a_{2,j-2}] \right)$$

$$\oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot \mathrm{S}[a_{3,j-3}] \right) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

$$T_0[x] = \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot \mathrm{S}[x] \right) \quad T_1[x] = \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot \mathrm{S}[x] \right) \quad T_2[x] = \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot \mathrm{S}[x] \right) \quad T_3[x] = \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot \mathrm{S}[x] \right)$$

$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

12

SM2/3/4/9 & ZUC

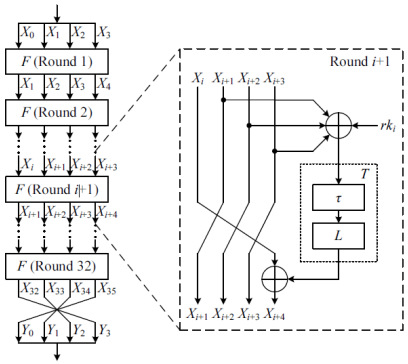
- SM2: Elliptic Curve over prime field
  - GM/T 0003-2012; GB/T 32918-2016
  - Digital signature algorithm/key exchange protocol/public key encryption
  - FIPS 186-4: NIST P-256 (aka secp256r1 or prime256v1)
- SM3: hash function with Merkle–Damgård constructuion
  - GM/T 0004-2012; GB/T 32905-2016
  - FIPS 180-4: SHA-256
- SM4: block cipher with Unbalanced Feistel structure
  - GM/T 0002-2012; GB/T 32907-2016
  - FIPS 197: AES
- SM9: identity-based cryptosystem with bilinear pairing
  - GM/T 0044-2016
  - Digital signature algorithm/key exchange protocol/public key encryption
  - RFC 5091: Boneh-Franklin BF, Boneh-Boyen BB1
- ZUC: word-oriented stream cipher
  - GM/T 0001-2012; GB/T 33133-2016
  - 3GPP LTE: ZUC128-EEA3 & ZUC128-EIA3
  - RFC 7539: Chacha20

13

Introduction To SM4

- Published in 2006
- Initially used in WAPI
- GM/T002-2012
- GB/T 32907-2016

- 128-bit block
- 128-bit key
- Unbalanced Feistel
- 32 rounds
- 32-bit round key
- xor, rotation, sbox



14

SM4 Round Function

- Round Function $F(\cdot)$

$$F(X_0, X_1, X_2, X_3, rk) = X_0 \oplus T(X_1 \oplus X_2 \oplus X_3 \oplus rk)$$
$$X_0, X_1, X_2, X_3, rk \in Z_2^{32}$$

- Composed Transformation $T(\cdot)$

$$T(\cdot) = L(\tau(\cdot))$$

- Non-Linear Transformation $\tau(\cdot)$

$$B = \tau(A) = \big(Sbox(x_0), Sbox(x_1), Sbox(x_2), Sbox(x_3)\big)$$
$$A = (x_0, x_1, x_2, x_3), B = (y_0, y_1, y_2, y_3),$$
$$x_0, x_1, x_2, x_3 \in Z_2^8, y_0, y_1, y_2, y_3 \in Z_2^8$$

- Linear Transformation $L(\cdot)$

$$L(B) = B \oplus (B \lll 2) \oplus (B \lll 10) \oplus (B \lll 18) \oplus (B \lll 24)$$
$$L'(B) = B \oplus (B \lll 13) \oplus (B \lll 22)$$

15

Random Thoughts on SM4 Software Impl.

- Eliminate Loops: unroll loops with macros of C/C++
- Avoid Moving Data Inside Memory
  - Branch rotation costs nothing
- Know Your CPU: Cache System, Intrinsic Instructions, …
- Try Different Compilers with Different Optimization Options
- Parallel at Different Levels
- Equivalent Transformation of SM4 Algorithm
- Learning From Hardware Implementation
- Security Side: Constant Time ~~& Key Protection~~

16

SM4 Impl. -- Basic

```
#include <climits>
#define bitsof(x) (CHAR_BIT * sizeof(x))
#define rol(x, n) (((x) << (n)) | ((x) >> ((bitsof(x) - (n)))))
#define ror(x, n) (((x) >> (n)) | ((x) << ((bitsof(x) - (n)))))


#define SM4_CORE_4R(rk0, rk1, rk2, rk3)                        \
  do {                                                         \
        tmp = m[1] ^ m[2] ^ m[3] ^ rk0, m[0] ^= sm4_t_sub(tmp); \
        tmp = m[2] ^ m[3] ^ m[0] ^ rk1, m[1] ^= sm4_t_sub(tmp); \
        tmp = m[3] ^ m[0] ^ m[1] ^ rk2, m[2] ^= sm4_t_sub(tmp); \
        tmp = m[0] ^ m[1] ^ m[2] ^ rk3, m[3] ^= sm4_t_sub(tmp); \
  } while (0)
```

17

SM4 Impl. -- Basic

```
void sm4_enc_core(u32t *m, const u32t *rk) {
        u32t tmp;
        SM4_CORE_4R(rk[0], rk[1], rk[2], rk[3]);
        SM4_CORE_4R(rk[4], rk[5], rk[6], rk[7]);
        SM4_CORE_4R(rk[8], rk[9], rk[10], rk[11]);
        SM4_CORE_4R(rk[12], rk[13], rk[14], rk[15]);
        SM4_CORE_4R(rk[16], rk[17], rk[18], rk[19]);
        SM4_CORE_4R(rk[20], rk[21], rk[22], rk[23]);
        SM4_CORE_4R(rk[24], rk[25], rk[26], rk[27]);
        SM4_CORE_4R(rk[28], rk[29], rk[30], rk[31]);
        tmp = m[0], m[0] = m[3], m[3] = tmp;
        tmp = m[1], m[1] = m[2], m[2] = tmp;
}
```

18

SM4 Impl. & Basic Parallelization

- We are working on 64-bit system
- We have 64-bit register, why just working with 32-bit variables?
- Processing two blocks in parallel with 64-bit register

- Expecting a 2x speed up?
- Usually NOT that Lucky
  - Formatting input and output messages
  - SBox substitution must be done in a serialized way, e.g. one by one
- But we do gain speed improvement:

  30.7 cycles/byte → 23.5 cycles/byte
- What if we have 128-bit, 256-bit or even 512-bit register?

19

SIMD

- Single Instruction Multiple Data
- SSE: Streaming SIMD Extensions, 128-bit register, 16 XMM
  - SIMD instruction set extension to x86 architecture
  - Designed by Intel in 1999 in Pentium III (after AMD's 3DNow!)
  - Subsequently expanded by Intel to SSE2, SSE3, SSSE3, SSE4
- AVX: Advanced Vector Extensions, 256-bit register, 16 YMM
  - AVX: also known as Sandy Bridge New Extensions
  - released in Q1 2011 by Intel and Q3 by AMD
  - AVX2: also known as Haswell New Instructions
  - Introduced into OpenSSL since version 1.0.2 (2015)
- AVX512: 512-bit register, 32 ZMM
  - Proposed by Intel in July 2013
  - Shipped in 2016 with Knights Landing processor
  - Multiple extensions: Galois Field New Instructions, Bit Algorithms, …

20

SM4 Impl. T-Table [Simple Trans.]

- Composed Transformation: $T(\cdot) = L(\tau(\cdot))$
- SBox is followed by linear transformation
- Fold the linear transformation into SBox
  - Incurs (almost) no extra cost on SBox Operation
  - Eliminate the linear transformation saves 4 rotations per round
    $L(B) = B \oplus (B \lll 2) \oplus (B \lll 10) \oplus (B \lll 18) \oplus (B \lll 24)$
- Of Course, SBox is now different
  - One 8x8 SBox to Four 8x32 SBox (4KB in total)
  - $L(\tau(\cdot)) \rightarrow Sbox_0(x_0) \oplus Sbox_1(x_1) \oplus Sbox_2(x_2) \oplus Sbox_3(x_3)$

21

SM4 Impl. T-Table & Parallel

- Composed Transformation: $T(\cdot) = L(\tau(\cdot))$
- We are working on 64-bit system
- We have 128-bit, 256-bit registers
- Processing 2/4/8 blocks in parallel

- Expecting a 2x, 4x, 8x speed up?
- NOT that Lucky
  - Formatting input and output messages
  - SBox substitution must be done in a serialized way, e.g. one by one
- We do gain speed improvement:

  32 → 64 → 128 → 256

  19.9 →13.8 → 11.2 → 9.6

22

The Cache Attack

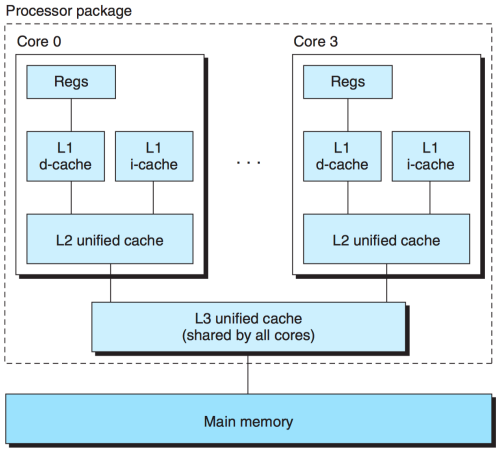
- Single Instruction Multiple Data
- Similar technique has been applied to AES software impl.
- Which is then introduced into OpenSSL
- Big table lead to non-constant encryption time due to cache miss …

Daniel J. Bernstein (DJB) in 2005:

demonstrated complete AES key recovery from known-plaintext timings of a network server on another computer

23

Intel Core i7 Cache Hierarchy



24

Some Protection, The OpenSSL Way

An implementation of the Chinese SM4 block cipher #4552

Closed ronaldtse wants to merge 9 commits into openssl:master from riboseinc:add-sm4

Conversation 126 · Commits 9 · Checks 0 · Files changed 22 · +752 −14

ronaldtse commented on 18 Oct 2017 • edited — Contributor

This PR introduces the Chinese SM4 algorithm which is a 128-bit block cipher.

It was originally ported from Ribose's contribution to Botan, but has received further updates to make its performance on par with the SM4 implementation there.

Currently `openssl speed -evp sm4-ecb` reports:

- ~150 MB/s on @randombit 's machine
- ~110 MB/s on @ronaldtse 's macOS Macbook Pro 3.1 GHz

This is a contribution by Ribose Inc. and was done with knowledge of @richsalz, @levitte, @InfoHunter.

There might be some stuff missing but please do bring on the review process 😉 .

cc: @randombit @dewyatt @erikbor

This is a contribution from Ribose Inc..

Checklist

- documentation is added or updated
- tests are added or updated

Reviewers: liuqun, InfoHunter, mattcaswell, dot-asm, erikbor, levitte, paulidale

Assignees: No one assigned

Labels: None yet

Projects: None yet

Milestone

25

Some Protection, The OpenSSL Way

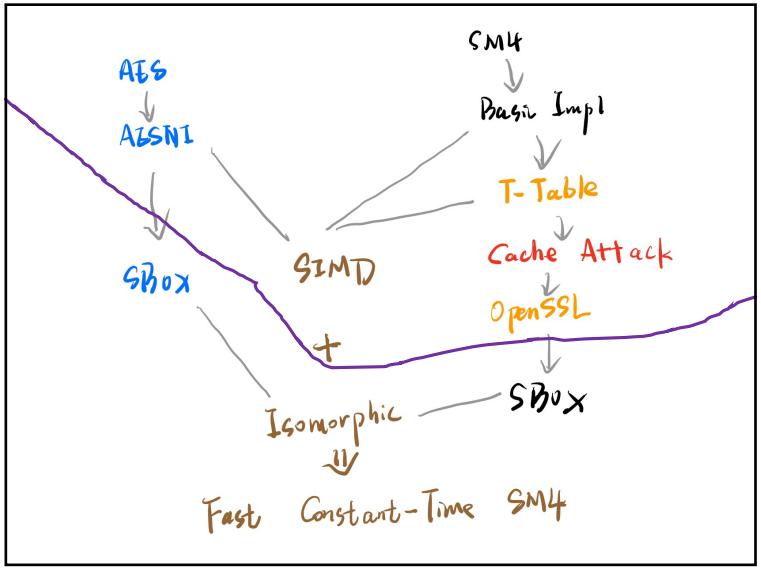
```
109 static ossl_inline uint32_t SM4_T_slow(uint32_t X)
110 {
111     uint32_t t = 0;
112
113     t |= ((uint32_t)SM4_S[(uint8_t)(X >> 24)]) << 24;
114     t |= ((uint32_t)SM4_S[(uint8_t)(X >> 16)]) << 16;
115     t |= ((uint32_t)SM4_S[(uint8_t)(X >> 8)]) << 8;
116     t |= SM4_S[(uint8_t)X];
117
118     /*
119      * L linear transform
120      */
121     return t ^ rotl(t, 2) ^ rotl(t, 10) ^ rotl(t, 18) ^ rotl(t, 24);
122 }
123
124 static ossl_inline uint32_t SM4_T(uint32_t X)
125 {
126     return SM4_SBOX_T[(uint8_t)(X >> 24)] ^
127            rotl(SM4_SBOX_T[(uint8_t)(X >> 16)], 24) ^
128            rotl(SM4_SBOX_T[(uint8_t)(X >> 8)], 16) ^
129            rotl(SM4_SBOX_T[(uint8_t)X], 8);
130 }
```

26

Some Protection, The OpenSSL Way

```
194     /*
195      * Uses byte-wise sbox in the first and last rounds to provide some
196      * protection from cache based side channels.
197      */
198     SM4_RNDS( 0,  1,  2,  3, SM4_T_slow);
199     SM4_RNDS( 4,  5,  6,  7, SM4_T);
200     SM4_RNDS( 8,  9, 10, 11, SM4_T);
201     SM4_RNDS(12, 13, 14, 15, SM4_T);
202     SM4_RNDS(16, 17, 18, 19, SM4_T);
203     SM4_RNDS(20, 21, 22, 23, SM4_T);
204     SM4_RNDS(24, 25, 26, 27, SM4_T);
205     SM4_RNDS(28, 29, 30, 31, SM4_T_slow);
206
207     store_u32_be(B3, out);
208     store_u32_be(B2, out + 4);
209     store_u32_be(B1, out + 8);
210     store_u32_be(B0, out + 12);
```

27



28

Revisit AES SBox

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

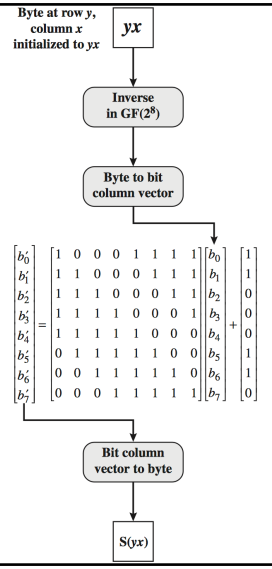
Figure 7. S-box: substitution values for the byte xy (in hexadecimal format).

29

Inside AES SBox

- Single Instruction Multiple Data
- $SBox_{AES}(x) = A \cdot x^{-1} + 0x63,$
- $x \in GF(2^8), f(x) = x^8 + x^4 + x^3 + x + 1$
- Gen AES SBox with Sage

```
#generate aes's sbox
for v in range(0, 0x100):
  v_inv = aes_field_inv(v)

  v_inv_list = [int(d) for d in format(v_inv, "08b")]
  v_inv_list.reverse() # lower bit comes first
  v_inv_vec = matrix(GF(2),8,1, v_inv_list)

  res_matrix = A * v_inv_vec + C
  res = ZZ(res_matrix.list(), base=2)

  print "%02x" % res,
  if (v+1) % 16 == 0:
    print
```



30

Revisit SM4 SBox

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D6	90	E9	FE	CC	E1	3D	B7	16	B6	14	C2	28	FB	2C	05
1	2B	67	9A	76	2A	BE	04	C3	AA	44	13	26	49	86	06	99
2	9C	42	50	F4	91	EF	98	7A	33	54	0B	43	ED	CF	AC	62
3	E4	B3	1C	A9	C9	08	E8	95	80	DF	94	FA	75	8F	3F	A6
4	47	07	A7	FC	F3	73	17	BA	83	59	3C	19	E6	85	4F	A8
5	68	6B	81	B2	71	64	DA	8B	F8	EB	0F	4B	70	56	9D	35
6	1E	24	0E	5E	63	58	D1	A2	25	22	7C	3B	01	21	78	87
7	D4	00	46	57	9F	D3	27	52	4C	36	02	E7	A0	C4	C8	9E
8	EA	BF	8A	D2	40	C7	38	B5	A3	F7	F2	CE	F9	61	15	A1
9	E0	AE	5D	A4	9B	34	1A	55	AD	93	32	30	F5	8C	B1	E3
A	1D	F6	E2	2E	82	66	CA	60	C0	29	23	AB	0D	53	4E	6F
B	D5	DB	37	45	DE	FD	8E	2F	03	FF	6A	72	6D	6C	5B	51
C	8D	1B	AF	92	BB	DD	BC	7F	11	D9	5C	41	1F	10	5A	D8
D	0A	C1	31	88	A5	CD	7B	BD	2D	74	D0	12	B8	E5	B4	B0
E	89	69	97	4A	0C	96	77	7E	65	B9	F1	09	C5	6E	C6	84
F	18	F0	7D	EC	3A	DC	4D	20	79	EE	5F	3E	D7	CB	39	48

注：输入'EF'，则经 S 盒后的值为表中第 E 行和第 F 列的值，Sbox(EF)＝84。

31

Inside SM4 SBox

- Specification contains no info. how the SBox is constructed
- Internal structure of the SBox was reverse engineered later
  - Analysis of the SMS4 Block Cipher @ ACISP'07 by Liu et al.
    $SBox_{SM4}(x) = (x \cdot A_1 + C_1)^{-1} \cdot A_2 + C_2$
  - Algebraic Cryptanalysis of SMS4: Grobner Basis Attack and SAT Attack Compared @ ICISC'09 by Erickson et al.
    $SBox_{SM4}(x) = (x \cdot A + C)^{-1} \cdot A + C$

- $SBox_{SM4}(x) = \boldsymbol{Inv}(x \cdot A_1 + C_1) \cdot A_2 + C_2,$
- $x \in GF(2^8), g(x) = x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1$

32

Field Isomorphism: $GF_{AES}(2^8)$ & $GF_{SM4}(2^8)$

- Both SBoxes are 8 in 8 out, defined over $GF(2^8)$
  $GF_{SM4}(2^8) \cong GF_{AES}(2^8)$
- Inversion in $GF_{SM4}(2^8)$ can be done in $GF_{AES}(2^8)$

- This is where AESNI instruction come into play
  - Last round AES: AESENCLAST or AESDECLAST
  - SubstituteBytes, ShiftRows, AddRoundKey, no MixColumn
  - $SBox_{AES}(x) = A \cdot x^{-1} + 0x63$
  - $SBox_{SM4}(x) = (x \cdot A_1 + C_1)^{-1} \cdot A_2 + C_2$
  - Separate the inversion out of AESENCLAST/AESDECLAST

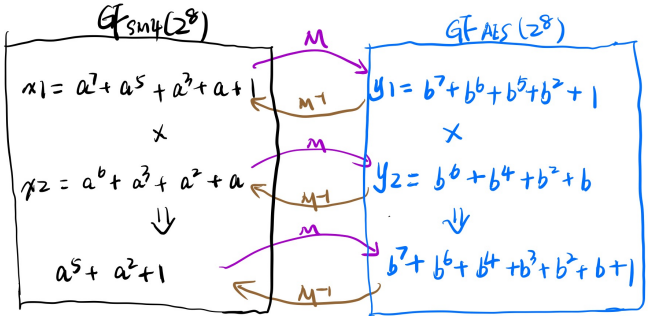
33

Mapping Matrix

[1 0 1 1 0 0 0 0] [1 1 1 0 0 1 0 0] [1 1 1 0 1 1 1 0] [1 0 1 0 1 0 0 0] [1 0 1 0 0 1 0 0] [1 1 0 1 1 0 0 0] [0 1 1 1 0 0 1 0] [1 1 0 0 0 1 1 1]	[1 0 0 0 1 0 0 0] [0 0 1 1 1 0 0 0] [1 1 0 1 0 1 1 0] [1 1 0 0 0 0 1 0] [0 0 1 1 0 0 1 0] [1 1 1 0 0 0 1 0] [0 0 0 0 1 1 1 0] [0 0 1 1 1 1 0 1]	[1 0 0 0 0 1 0 0] [1 1 1 0 0 1 1 0] [1 1 0 1 1 0 1 0] [1 0 1 1 1 1 0 0] [0 0 1 1 0 0 0 0] [1 0 1 1 1 1 1 0] [1 1 1 1 1 1 0 0] [0 0 0 0 1 0 1 1]	[0 1 0 1 0 1 0 0] [0 1 0 0 1 0 1 0] [0 0 0 0 1 0 1 0] [1 1 0 0 0 0 0 0] [0 0 0 1 0 0 1 0] [1 0 0 1 1 1 0 0] [1 1 1 1 1 1 0 0] [1 0 0 0 1 0 1 1]
[0 0 0 1 1 1 1 0] [0 0 0 1 1 0 0 0] [0 1 0 0 0 0 0 0] [0 0 0 0 1 0 0 0] [1 1 0 1 0 1 0 0] [1 1 1 1 0 1 1 0] [1 1 0 1 1 1 1 0] [0 0 0 0 0 0 0 1]	[0 0 0 0 0 1 1 0] [1 0 0 1 0 1 0 0] [0 1 0 1 1 0 0 0] [1 1 1 0 0 0 0 0] [0 1 0 0 1 1 1 0] [1 0 0 1 1 1 1 0] [0 0 0 0 1 1 1 0] [0 0 0 1 0 0 0 1]	[1 0 0 1 0 0 1 0] [0 0 0 1 0 1 1 0] [1 1 0 0 1 1 0 0] [0 1 1 1 1 0 0 0] [0 0 1 0 1 0 1 0] [0 1 0 1 0 1 1 0] [0 1 1 1 0 0 1 0] [0 1 1 0 0 1 0 1]	[0 1 1 0 0 0 1 0] [1 1 1 0 1 0 1 0] [0 0 1 1 1 1 0 0] [1 0 0 0 0 1 1 0] [0 0 0 0 0 1 0 0] [0 0 1 0 0 1 1 0] [1 1 0 1 1 1 1 0] [0 1 0 1 0 0 0 1]

34

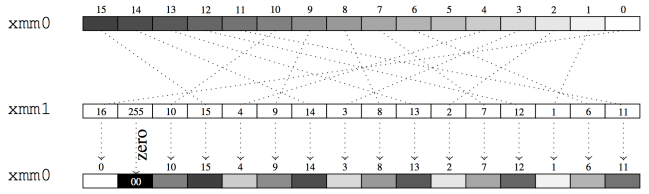
Isomorphic Demo



35

Towards Runnable Codes: Core Instruction

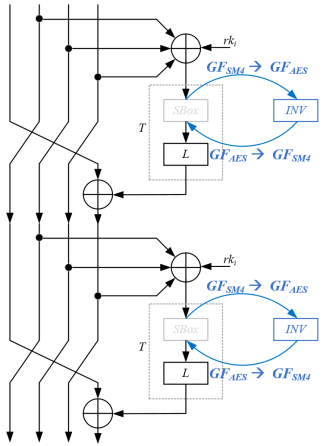
PSHUFB XMM0, XMM1



Matrix-Vector can be done within SIX instructions

36

The Whole Picture



37

US 20150341168A1

(19) **United States**

(12) **Patent Application Publication**

GUERON

(10) **Pub. No.: US 2015/0341168 A1**

(43) **Pub. Date: Nov. 26, 2015**

(54) **TECHNOLOGIES FOR MODIFYING A FIRST CRYPTOGRAPHIC CIPHER WITH OPERATIONS OF A SECOND CRYPTOGRAPHIC CIPHER**

(71) Applicant: **Intel Corporation**, Santa Clara, CA (US)

(72) Inventor: **SHAY GUERON**, Haifa (IL)

(73) Assignee: **Intel Corporation**, Santa Clara, CA (US)

(21) Appl. No.: **14/283,955**

(22) Filed: **May 21, 2014**

Publication Classification

(51) **Int. Cl.**
H04L 9/08 (2006.01)
H04L 9/14 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 9/0838*** (2013.01); ***H04L 9/14*** (2013.01); *H04L 2209/24* (2013.01)

(57) **ABSTRACT**

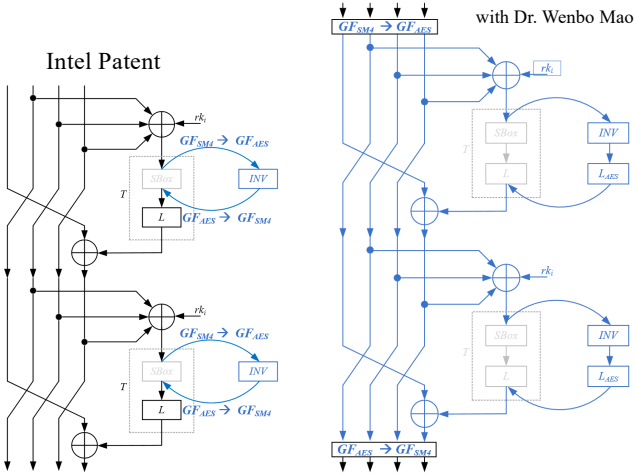
Generally, the present disclosure provides technology modifying a first cryptographic cipher with one or more operations of a second cryptographic cipher. In some embodiments the technology leverages a mathematical relationship between representations of data used in the first and second ciphers to enable the substitution of one or more operations of the first cipher with one or more operations of the second cipher. The resulting modified cipher may in some instances exhibit improved performance and or security, relative to the unmodified first cipher. Methods, computer readable media, and apparatus including or utilizing the technologies are also described.

38

Can We Do Better?

Intel Patent

with Dr. Wenbo Mao



39

Can you SPOT the PROBLEM ?

40

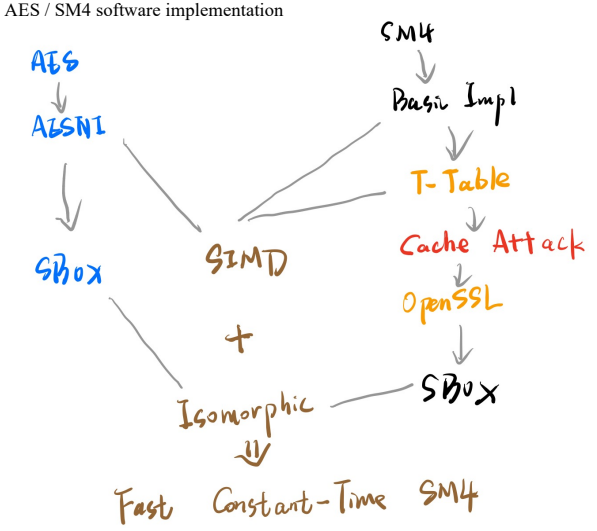
Actual Speed Measure

blocks	Basic	T-Table Cache Attack	Intel Constant-Time	With Dr. Mao Constant-Time
1	30.7	19.9		
2	23.5	13.8		
4	17.8	11.2	14.2	21.5
8	15.5	9.6	8.6	7.5
16			6.5	4.8

26%

41

*Project: AES / SM4 software implementation

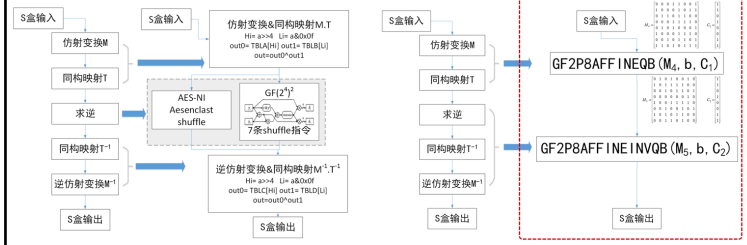


42

Better Impl. with New Instructions

① 基于GFNI的S盒实现　摒除同构映射，应用GFNI指令直接完成S盒计算

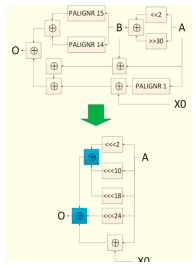
AESNI　指令数：14　本方案　指令数：2 ⬇ 86%



43

Better Impl. with New Instructions

② 基于AVX512的轮函数优化　应用VPROLD 和 VPTERNLOGD提升线性层运算速率



	AESNI方案		本方案
P	XOR(A,A1,RK[i]) XOR(B,X1,X2) XOR(B,B,X3) XOR(A,A,B)	指令数：33	XOR(A,X1, RK[i]) VPTERNLOGD(A,X2,X3,0x96)
T	H= A&0x0F L= A>>4 L=L&0xFF H= SHUFFLE(H,TBL0) L=SHUFFLE(L,TBL1) H=XOR(H,L) H=AESENCLAST(H)	A=SHULLF(H, TBL2) H= A&0x0F L= A>>4 L=L&0xFF H= SHUFFLE(H,TBL0) L=SHUFFLE(L,TBL1) A=XOR(H,L)	指令数：11 H=GF2P8AFFINE (A, M5, c1) A=GF2P8AFFINEINV (H, M4, c2) 67%指令缩减
L	C= A<<2 D= A>>30 C= C|D D= LIGN(C,1) E= LIGN(C,2) F= LIGN(A,3) XOR(A,X0,A)	XOR(E,E,F) XOR(C,C,E) XOR(A,A,C) X0=X1 x1=x2 x2=x3 x3=A	VPROLD (C,A,2) VPROLD (D,A,10) VPROLD (E,A,18) VPROLD (F, A,24) XOR(X1,X1,A) VPTERNLOGD(C,D,E,0x96) VPTERNLOGD(A,C,F,0x96)

0.66 cycles per byte @ 2024.11
(~20 cycles per byte w. basic impl.

44

Q & A

45