A Lightweight ISA Extension for AES and SM4

Markku-Juhani O. Saarinen
`mjos@pqshield.com`

PQShield Ltd.
Oxford, United Kingdom

August 23, 2020

SEC RISC-V
Workshop 2020

First International Workshop on Secure RISC-V
Architecture Design Exploration (SECRISC-V'20).

Talk Outline

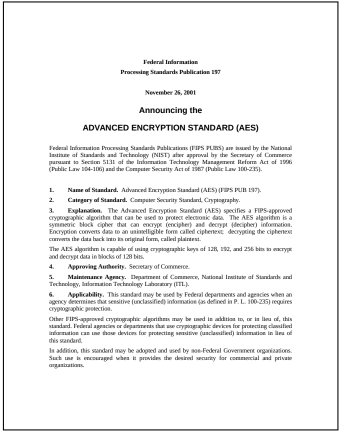
I Introduction: AES, SM4, and Crypto TG

II SAES32 and SSM4 Instruction Set Extension

III Implementation and Analysis

IV Conclusions

AES: Advanced Encryption Standard



Federal Information
Processing Standards Publication 197

November 26, 2001

Announcing the

ADVANCED ENCRYPTION STANDARD (AES)

Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology (NIST) after approval by the Secretary of Commerce pursuant to Section 5131 of the Information Technology Management Reform Act of 1996 (Public Law 104-106) and the Computer Security Act of 1987 (Public Law 100-235).

1. **Name of Standard.** Advanced Encryption Standard (AES) (FIPS PUB 197).

2. **Category of Standard.** Computer Security Standard, Cryptography.

3. **Explanation.** The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called ciphertext; decrypting the ciphertext converts the data back into its original form, called plaintext.

The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

4. **Approving Authority.** Secretary of Commerce.

5. **Maintenance Agency.** Department of Commerce, National Institute of Standards and Technology, Information Technology Laboratory (ITL).

6. **Applicability.** This standard may be used by Federal departments and agencies when an agency determines that sensitive (unclassified) information (as defined in P. L. 100-235) requires cryptographic protection.

Other FIPS-approved cryptographic algorithms may be used in addition to, or in lieu of, this standard. Federal agencies or departments that use cryptographic devices for protecting classified information can use those devices for protecting sensitive (unclassified) information in lieu of this standard.

In addition, this standard may be adopted and used by non-Federal Government organizations. Such use is encouraged when it provides the desired security for commercial and private organizations.

FIPS PUB 197

- Specified in FIPS PUB 197, International standards.
- 128-bit block size, 128 / 192 / 256-bit secret key.
- Single 8 × 8-bit S-Box, Substitution-Permutation.
- Rijndael by Joan Daemen and Vincent Rijmen (1998). Clear, open, well-understood design methodology.
- **Very common.** Hardware support saves energy in comms (TLS, IPSec, WiFi), storage (XTS), etc.
- ARMv8.0-CE (SIMD) and Intel AES-NI (SIMD) ISAs.
- Embedded often have memory mapped AES engines. No real standard for those; vendor-specific drivers.

SM4: The Chinese Standard Block Cipher

ICS 35.040
L 80
备案号：36825—2012

GM

中华人民共和国密码行业标准

GM/T 0002—2012

SM4 分组密码算法

SM4 block cipher algorithm

2012-03-21 发布　　2012-03-21 实施

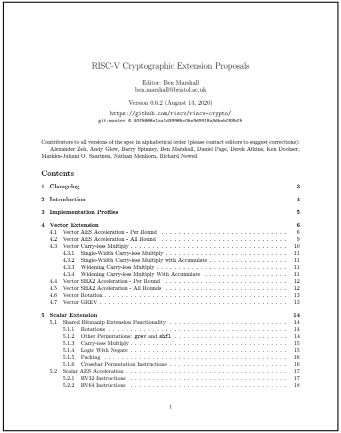
国家密码管理局　发 布

数码防伪

GM/T 0002-2012

- Specified in GM/T 0002-2012, GB/T 32907-2016, internationally ISO/IEC 18033-3:2010/DAmd 2.
- 128-bit block size, 128-bit secret key (one key size).
- Single 8 × 8-bit S-Box, Generalized Feistel Structure.
- Credited to Lu Shuwang (吕述望) et al, early 2000s. Design methodology and criteria difficult to find.
- Important to RISC-V International (due to export etc.)
- ARMv8.2-SM (SIMD) ISA has SM4 support, Intel no.
- SM4 has regulatory preference over AES in PR China.

Crypto TG: The RISC-V Crypto Spec



RISC-V Cryptographic Extension Proposals

Editor: Ben Marshall
ben.marshall@bristol.ac.uk

Version 0.6.2 (August 13, 2020)

`https://github.com/riscv/riscv-crypto/`
`git:master @ 40f5866e1aa1d39965c05e3d9918a3dbebf83bf5`

Contributors to all versions of the spec in alphabetical order (please contact editors to suggest corrections):
Alexander Zeh, Andy Glew, Barry Spinney, Ben Marshall, Daniel Page, Derek Atkins, Ken Dockser, Markku-Juhani O. Saarinen, Nathan Menhorn, Richard Newell

Contents

1

Crypto Spec v0.6.2, August 13, 2020

- → The RISC-V Cryptographic Extensions Task Group (Crypto TG) has been operating since 2017.
- → In late 2019 the scope was extended from Vector (RVV, SIMD-style) AES to "Scalar" RV32 and RV64.
- → I proposed the present work (as ENC1S) in Feb 2020. It was evaluated and adopted as the preferred option for RV32 some months later (as SAES32 & SSM4).
- → Evaluation (as AES "*v3*"): B. Marshall, G. R. Newell, D. Page, M.-J. O. Saarinen, and C. Wolf: *"The design of scalar AES Instruction Set Extensions for RISC-V."* `https://eprint.iacr.org/2020/930`
- → The crypto spec is going to freeze soon; you can find it at: `https://github.com/riscv/riscv-crypto`

Talk Outline

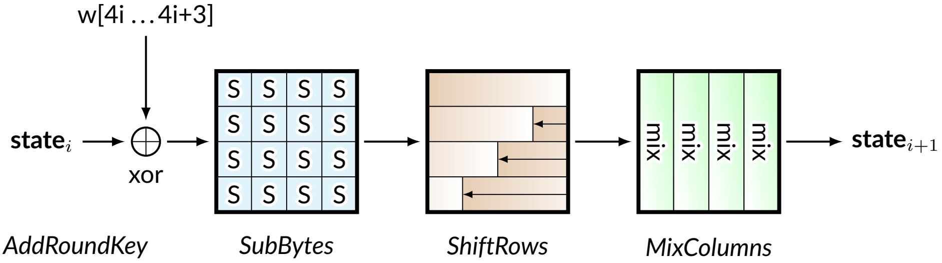
I Introduction: AES, SM4, and Crypto TG

II **SAES32 and SSM4 Instruction Set Extension**

III Implementation and Analysis

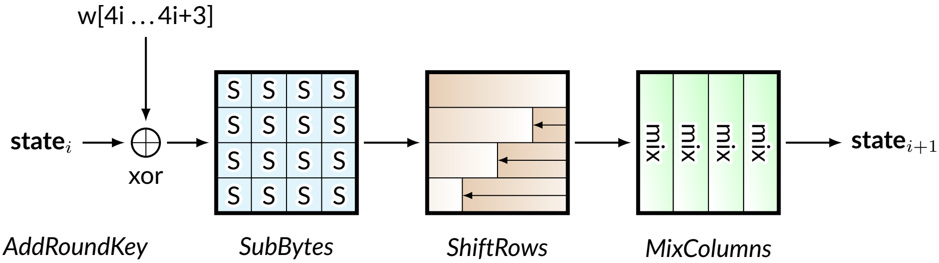
IV Conclusions

AES Steps



- → AES has $\{10, 12, 14\}$ rounds – for $\{128, 192, 256\}$ bit keys, respectively.
- → Rounds are made of: AddRoundKey, SubBytes, ShiftRows, MixColumns.
- → Contrary to Feistel ciphers – like SM4 – Inverse of Substitution-Permutation Network (SPN) like AES requires inversion of each step (inverse SB, SR, MC).

AES Steps: T-table



- ShiftRows just shuffles bytes and SubBytes operates on individual bytes.
- SubBytes and Mixcolumns can be combined into 8 $\rightarrow$ 32 - bit “T table” lookups.
- MixColumns is 4 $\times$ 4 byte matrix multiplication defined in GF(2^8); it’s **linear**!

32-bit T-table

The original 1998 Rijndael Reference code targeted 32-bit systems of the day:

```
s0 =
   Te0[(t0 >> 24)       ] ^
   Te1[(t1 >> 16) & 0xff] ^
   Te2[(t2 >>  8) & 0xff] ^
   Te3[(t3      ) & 0xff] ^
   rk[0];
s1 =
   Te0[(t1 >> 24)       ] ^
   Te1[(t2 >> 16) & 0xff] ^
   Te2[(t3 >>  8) & 0xff] ^
   Te3[(t0      ) & 0xff] ^
   rk[1];
s2 =
   Te0[(t2 >> 24)       ] ^
   Te1[(t3 >> 16) & 0xff] ^
   Te2[(t0 >>  8) & 0xff] ^
   Te3[(t1      ) & 0xff] ^
   rk[2];
s3 =
   Te0[(t3 >> 24)       ] ^
   Te1[(t0 >> 16) & 0xff] ^
   Te2[(t1 >>  8) & 0xff] ^
   Te3[(t2      ) & 0xff] ^
   rk[3];
```

- For a decade, all AES Implementations looked $\approx$ like this.
- 4 input bytes $\times$ 256 S-Box entries $\times$ 32 bits = 4 kB.
- Another 4 kB for decryption, possibly 1 kB for last rounds.
- Serious cache timing attacks emerged after mid-2000s (Bernstein, Osvik, et al [2,12]). Can be exploited remotely.
- Need to replace table lookups with with straight-line logic.
- On RV32 targets such bit-sliced implementations are 2.5$\times$ slower than table-based ones (Stoffelen [15]).

ARM32: Google (Android, Chrome) tries to negoatiate ChaCha20 for TLS instead of AES on systems that do not have AES instructions. Secure AES is just too slow.

Old school: Hand-optimized RV32I T-Table AES

https://github.com/Ko-/riscvcrypto/blob/master/aes128tables/aes_asm.S

```
andi \T0, \X0, 0xff
andi \T1, \X1, 0xff
andi \T2, \X2, 0xff
andi \T3, \X3, 0xff
slli \T0, \T0, 4
slli \T1, \T1, 4
slli \T2, \T2, 4
slli \T3, \T3, 4
add  \T4, \T0, \LUT1
lw   \T0, (\T4)
add  \T4, \T1, \LUT1
lw   \T1, (\T4)
add  \T4, \T2, \LUT1
lw   \T2, (\T4)
add  \T4, \T3, \LUT1
lw   \T3, (\T4)
xor  \Y0, \Y0, \T0
xor  \Y1, \Y1, \T1
xor  \Y2, \Y2, \T2
xor  \Y3, \Y3, \T3
```

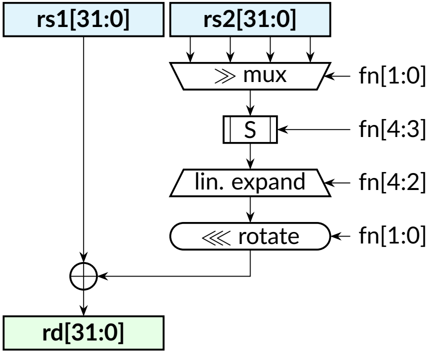
```
srli \X0, \X0, 4
srli \X1, \X1, 4
srli \X2, \X2, 4
srli \X3, \X3, 4
and  \T0, \X1, \C
and  \T1, \X2, \C
and  \T2, \X3, \C
and  \T3, \X0, \C
add  \T4, \T0, \LUT3
lw   \T0, (\T4)
add  \T4, \T1, \LUT3
lw   \T1, (\T4)
add  \T4, \T2, \LUT3
lw   \T2, (\T4)
add  \T4, \T3, \LUT3
lw   \T3, (\T4)
xor  \Y0, \Y0, \T0
xor  \Y1, \Y1, \T1
xor  \Y2, \Y2, \T2
xor  \Y3, \Y3, \T3
```

```
srli \X0, \X0, 8
srli \X1, \X1, 8
srli \X2, \X2, 8
srli \X3, \X3, 8
and  \T0, \X2, \C
and  \T1, \X3, \C
and  \T2, \X0, \C
and  \T3, \X1, \C
add  \T4, \T0, \LUT0
lw   \T0, (\T4)
add  \T4, \T1, \LUT0
lw   \T1, (\T4)
add  \T4, \T2, \LUT0
lw   \T2, (\T4)
add  \T4, \T3, \LUT0
lw   \T3, (\T4)
xor  \Y0, \Y0, \T0
xor  \Y1, \Y1, \T1
xor  \Y2, \Y2, \T2
xor  \Y3, \Y3, \T3
```

```
srli \X0, \X0, 8
srli \X1, \X1, 8
srli \X2, \X2, 8
srli \X3, \X3, 8
and  \T0, \X3, \C
and  \T1, \X0, \C
and  \T2, \X1, \C
and  \T3, \X2, \C
add  \T4, \T0, \LUT2
lw   \T0, (\T4)
add  \T4, \T1, \LUT2
lw   \T1, (\T4)
add  \T4, \T2, \LUT2
lw   \T2, (\T4)
add  \T4, \T3, \LUT2
lw   \T3, (\T4)
xor  \Y0, \Y0, \T0
xor  \Y1, \Y1, \T1
xor  \Y2, \Y2, \T2
xor  \Y3, \Y3, \T3
```

$4 \times 20 = 80$ instructions (+ key fetch) per round.

Approach: Roll T-Table Operations into a Single Instruction



SAES32 & SSM4: Scalar RV32 AES,SM4

```
saes32.encsm   rd, rs1, rs2, bs
saes32.encs    rd, rs1, rs2, bs
saes32.decsm   rd, rs1, rs2, bs
saes32.decs    rd, rs1, rs2, bs
ssm4.ed        rd, rs1, rs2, bs
ssm4.ks        rd, rs1, rs2, bs
```

- → `encs` and `decs` lack MixColumns. Used for final round, key schedule.
- → R-type, immediate bs $\in \{0, 1, 2, 3\}$: $4 \times 6 = 24$ code points total.
- → The same logic also supports SM4.

SAES32 AES

`https://github.com/mjosaarinen/lwaes_isa/blob/master/asm/saes32_enc.S`

```
saes32.encsm  t4, t4, t0, 0
saes32.encsm  t4, t4, t1, 1
saes32.encsm  t4, t4, t2, 2
saes32.encsm  t4, t4, t3, 3

saes32.encsm  t5, t5, t1, 0
saes32.encsm  t5, t5, t2, 1
saes32.encsm  t5, t5, t3, 2
saes32.encsm  t5, t5, t0, 3

saes32.encsm  t6, t6, t2, 0
saes32.encsm  t6, t6, t3, 1
saes32.encsm  t6, t6, t0, 2
saes32.encsm  t6, t6, t1, 3

saes32.encsm  a7, a7, t3, 0
saes32.encsm  a7, a7, t0, 1
saes32.encsm  a7, a7, t1, 2
saes32.encsm  a7, a7, t2, 3
```

$4 \times 4 = 16$ (+ key fetch).

- From 80 instrs to 16× `saes32.encsm` for main rounds, 16× `saes32.encs` for final.
- Same for decryption, with `saes32.decs[m]`.
- No table lookups, which often require multiple cycles. Timing-attack secure.
- Key schedule uses the same instructions.
- It would be possible to reduce insn count to 12 by having four parallel *S*-boxes, but that has $> 3\times$ implementation size, more energy.
- $\approx 5\times$ faster than table-based (insecure), $> 10\times$ faster than constant-time.

SSM4.ED and SSM4.KS

`https://github.com/mjosaarinen/lwaes_isa/blob/master/asm/sm4_encdec.S`

- SM4 is a Generalized Feistel; unlike AES, encryption and decryption are the same (with 32 expanded key words reversed). Separate key sched instruction.
- 4 × S-Box would probably give a bigger speed-up for SM4 than for AES.
- The linear transformation in SM4 is based on rotations. In RISC-V rotation instructions are in RV32B Bitmanip extension; those are not needed here.
- Depending on availability of rotations, the speedup is similar or much better than that of AES, without much additional area – and greatly reduced energy.
- Importantly the instruction makes SM4 *constant-time* too.

Is this new? Prior Art

No mainstream "pure scalar" 32-bit ISA currently has AES or SM4 instructions. Similar, custom instructions for "T-Table" style AES has been discussed in:

[NIK04] K. Nadehara, M. Ikekawa, and I. Kuroda. "Extended instructions for the AES cryptography and their efficient implementation." 2004 IEEE Workshop on Signal Processing Systems (SIPS), pp. 152–157, 2004. DOI:10.1109/SIPS.2004.1363041.

[BBFR06] G. Bertoni, L. Breveglieri, R. Farina, and F. Regazzoni. "Speeding up AES by extending a 32-bit processor instruction set." IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP'06) pp. 275–282, 2006. DOI:10.1109/ASAP.2006.62.

However these proposals did not roll the AddRoundKey operation into the same instruction, and apparently need 20 per round rather than 16 (plus key schedule).

Talk Outline

I Introduction: AES, SM4, and Crypto TG

II SAES32 and SSM4 Instruction Set Extension

III **Implementation and Analysis**

IV Conclusions

Custom-0 Encoding (Temporary)

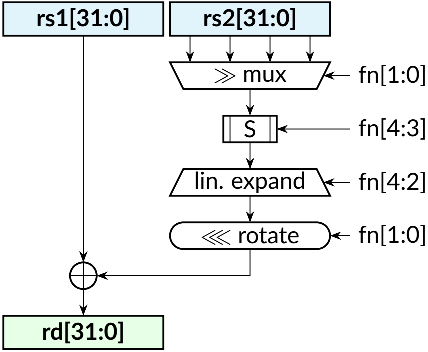
R-Type:

[31:30]	[29:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
00	`fn`	`rs2`	`rs1`	000	`rd`	0001011

Instruction	**fn[4:2]**	**Description or Use**
`saes32.encsm`	`3'b000`	AES Encrypt round.
`saes32.encs`	`3'b001`	AES Final / Key sched.
`saes32.decsm`	`3'b010`	AES Decrypt round.
`saes32.decs`	`3'b011`	AES Decrypt final.
`ssm4.ed`	`3'b100`	SM4 Encrypt and Decrypt.
`ssm4.ks`	`3'b101`	SM4 Key Schedule.
Unused	`3'b11x`	($4 \times 6 = 24$ points used.)

Hardware Implementation

https://github.com/mjosaarinen/lwaes_isa/blob/master/hdl/saes32.v



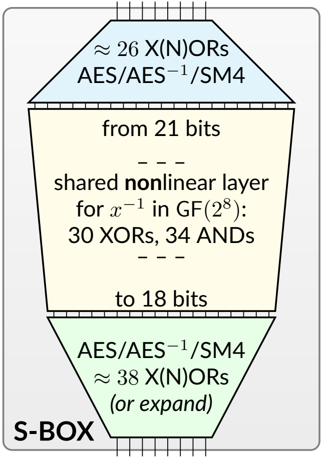
The original “reference implementation” is pure combinatorial logic, in verilog.

```
module saes32(
    output  [31:0] rd,  // not a reg
    input   [31:0] rs1, // rs1 wire
    input   [31:0] rs2, // rs2 wire
    input   [4:0]  fn   // 5-bit func
);
```

Obtained 100 MHz timing signoff on Artix-7 (old!) when inserted into 1-cycle decoding pipeline of the Pluto RV32 core.

S-Boxes: Boyar-Peralta for SM4 too

`https://github.com/mjosaarinen/lwaes_isa/blob/master/hdl/sboxes.v`

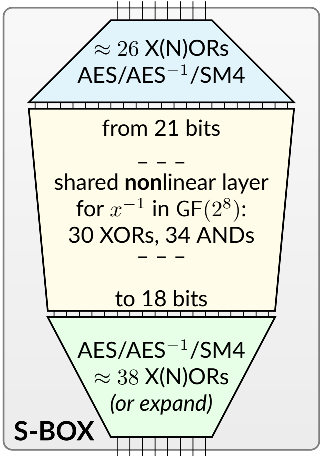


AES and SM4 S-Boxes are not "random":

- → Both are "Nyberg S-Boxes" [11] built from inversion x^{-1} in GF(2^8) and linear (XORs) input/output layers.
- → AES, AES^{-1}, and SM4 S-Boxes are "affine equivalent".
- → We expanded Boyar-Peralta [4] low-depth AES S-Box to the SM4 case by creating linear outer layers for it.
- → Non-linear middle layer could be muxed and shared; probably not worth it due to latency, small size gain.

S-Boxes: Algebraic gate counts

`https://github.com/mjosaarinen/lwaes_isa/blob/master/hdl/sboxes.v`



≈ 26 X(N)ORs
AES/AES^{-1}/SM4

from 21 bits

– – –
shared **non**linear layer
for x^{-1} in GF(2^8):
30 XORs, 34 ANDs
– – –

to 18 bits

AES/AES^{-1}/SM4
≈ 38 X(N)ORs
(or expand)

S-BOX

Low-depth S-Boxes that implement AES,AES^{-1},SM4.

Component	In, Out	XOR	XNOR	AND	**Total**
Shared middle	21 → 18	30	-	34	64
AES top	8 → 21	26	-	-	26
AES bottom	18 → 8	34	4	-	38
AES^{-1} top	8 → 21	16	10	-	26
AES^{-1} bottom	18 → 8	37	-	-	37
SM4 top	8 → 21	18	9	-	27
SM4 bottom	18 → 8	33	5	-	38

→ Each gate count $\approx$ 128, only XORs and ANDs.

→ Usually better than synthesis from a table.

FPGA Area Example

RV32 SoC area with and without SAES32 (AES, AES^{-1}, SM4).

Resource	**Base**	**SAES32** (Δ)	**EXTAES** (Δ)
Logic LUTs	7,767	8,202 (+435)	9,795 (+2,028)
Slice regs	3,319	3,342 (+23)	4,361 (+1,042)
SLICEL	1,571	1,864 (+293)	2,068 (+497)
SLICEM	734	737 (+3)	851 (+117)

- EXTAES is a CPU-external memory-mapped AES-only module (for comparison).
- “Pluto” core on an Artix-7 FPGA. Area grows by $\approx 5\%$ for this simple core.

CMOS Area Estimates

Yosys Simple CMOS Flow area estimates for SAES32 & SSM4

Target	GE (NAND2)	Transistors	LTP
AES Encrypt only	642	2,568	25
SM4 Full	767	3,066	25
AES Full	1,240	4,960	28
AES + SM4 Full	1,679	6,714	28

- → **It’s very small.** AES is a “lightweight cipher” for embedded RISC-V MCUs!
- → Not all applications need both SM4 and AES, or AES inverse (e.g. CTR, SIV).

Talk Outline

- I Introduction: AES, SM4, and Crypto TG
- II SAES32 and SSM4 Instruction Set Extension
- III Implementation and Analysis
- IV **Conclusions**

Conclusions

- **Crypto TG** is proposing three kinds of AES extensions: RV32, RV64, and RVV.
- **SAES32** – The scalar RV32 AES ISE – is *very* lightweight with only 1 S-Box. Designed primarily for timing-attack security, latency, and energy savings.
- Straight-forward design – rolling five T-table operations into one instruction: Byte select, S-Box, MixColumns, Rotate, final XOR (MC / AddRoundKey).
- **SM4**: The same "architecture" optionally also supports the Chinese Standard.
- Enables interoperable middleware for security functions in tiny RISC-V MCUs.

..Thank You!