“A wider variant of AES”
Rijndael-256 and RISC-V Crypto ISA

Markku-Juhani O. Saarinen
<markku-juhani.saarinen@tuni.fi>

January 16, 2025
RISC-V Cryptography SIG

https://csrc.nist.gov/pubs/sp/800/197/iprd

NIST SP 800-197 (Initial Preliminary Draft)

PRE-DRAFT Call for Comments: NIST Proposes to Standardize a Wider Variant of AES

Date Published: December 23, 2024
Comments Due: June 25, 2025
Email Comments to: ciphermodes@nist.gov

Author(s)

National Institute of Standards and Technology

Announcement

The Advanced Encryption Standard (AES) specifies a subset of the Rijndael block cipher family with 128-bit blocks that was submitted to the NIST AES development effort. While this block size remains sufficient for many applications, the increasing demand for processing large volumes of data highlights the potential advantages of a larger block size. This need was pointed out in the public comments received for (Special Publication) SP 800-38A, acknowledged in NIST Internal Report 8459, and further reinforced during two NIST public workshops on block cipher modes of operation.

In August 2024, NIST indicated its interest in vetting another Rijndael variant for approval: Rijndael with 256-bit blocks (i.e., Rijndael-256) with a single key size of 256-bits. NIST plans to develop a draft standard for Rijndael-256 over the next year and requests public comments on this plan by **June 25, 2025**, especially for the following categories:

- **Security analysis**, including any new cryptanalytic results related to the 256-bit block size
- **Performance and efficiency**, particularly in environments with hardware support for AES

DOCUMENTATION

Publication:
No Download Available

Supplemental Material:
None available

Document History:
12/23/24: SP 800-197 (Draft)

TOPICS

Security and Privacy
encryption

Activities and Products
standards development

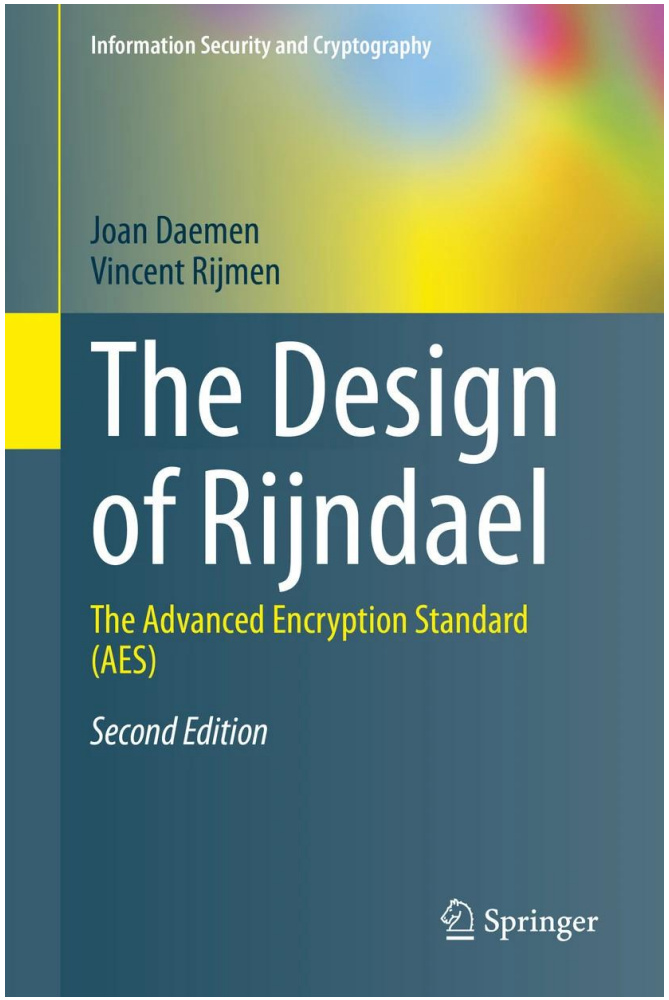
Rijndael-256 Specs & Implementation

- The original 1990s Rijndael proposal allowed block size and key to independently be { 128, 192, 256 } bits.

- The non-AES variants were largely ignored for over 20 years. Appendices B.3 and C of the Rijndael book (👉) has some test vectors and reference code for them.

- I used these to verify that my RISC-V implementation with current **Zvkned** should be correct. See:

  https://github.com/mjosaarinen/rij256-rv/



Rijndael-256 vs AES-256

- **Rijndael-256 has 14 rounds** and 14+1=15 round keys (same as AES-256).

- **The key schedule of Rijndael-256 is the same as AES-256**, except that more round constants are used. For a given key K, AES-256 round keys 1..14 match Rijndael-256 expanded key material for rounds 1..7.

Only Rijndael-256 ShiftRows() differs from 2 × AES parallel round steps:

- **SubBytes()**: 32 S-Box byte substitutions, independent of each others.
- **ShiftRows()**: 4 rows of 8 bytes, rotated left by { 0, 1, 3, 4 } positions.
- **MixColumns()**: 8 columns (32-bit chunks). Same linear operation as in AES.
- **AddRoundKey()**: A 32 byte-XOR with the round kys.

34.3.6. vaeskf2.vi

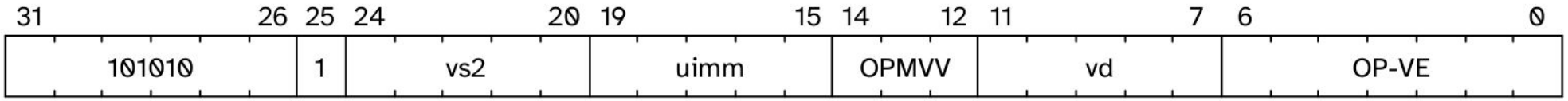
Synopsis

Vector AES-256 Forward KeySchedule generation

Mnemonic

vaeskf2.vi vd, vs2, uimm

Encoding



Reserved Encodings

- `SEW` is any value other than 32

This is fine for Rijndael-256 except that we now need more round constants:

The round number, which ranges from 2 to 14, comes from `uimm[3:0]`; `uimm[4]` is ignored. The out-of-range `uimm[3:0]` values of 0-1 and 15 are mapped to in-range values by inverting `uimm[3]`. Thus, 0-1 maps to 8-9, and 15 maps to 7.

Only ShiftRows() mixes 128-bit “lanes” (1)

(**0** 4 8 12 16 20 24 28)	*Input: Bytes 0,1,2,..., 31.*
(**1** 5 9 13 17 21 25 29)	*Bytes are ordered “column first!”*
(**2** 6 10 14 18 22 26 30)	
(**3** 7 11 15 19 23 27 31)	
(**0** 4 8 12 16 20 24 28)	*Rijndael-256 ShiftRows():*
(5 9 13 17 21 25 29 **1**)	*Entire row rotated left by 1*
(14 18 22 26 30 **2** 6 10)	*Entire row rotated left by 3*
(19 23 27 31 **3** 7 11 15)	*Entire row rotated left by 4*

Only ShiftRows() mixes 128-bit “lanes” (2)

(**0** 4 8 12 | **16** 20 24 28)	*2 × parallel AES ShiftRows():*
(5 9 13 **1** | 21 25 29 **17**)	*Half-rows rotated left by 1*
(10 14 **2** 6 | 26 30 **18** 22)	*Half-rows rotated left by 2*
(15 **3** 7 11 | 31 **19** 23 27)	*Half-rows rotated left by 3*
(**0** 4 8 12 16 20 24 28)	*Rijndael-256 ShiftRows():*
(5 9 13 17 21 25 29 **1**)	*Entire row rotated left by 1*
(14 18 22 26 30 **2** 6 10)	*Entire row rotated left by 3*
(19 23 27 31 **3** 7 11 15)	*Entire row rotated left by 4*

Zvkned: Each round vrgather.vv and vaes__.vv

- Before each round, permutate 32 bytes with vrgather.vv, indices:

```
{  0,  17, 22, 23, 4,  5,  26, 27, 8,  9,  14, 31, 12, 13, 18, 19,
   16, 1,  6,  7,  20, 21, 10, 11, 24, 25, 30, 15, 28, 29, 2,  3  };
```

- “Undoes” 2 × parallel AES-128 ShiftRows(), then the Rijndael-256 ShiftRows.

- **vaesem.vv: SubBytes() ➤ ShiftRows() ➤ MixColumns() ➤ AddRoundKey()**

- I’ve implemented and tested this; having the same round keys in encryption and decryption still works (no need to shuffle round keys.)

https://github.com/mjosaarinen/rij256-rv/

Annoying SEW toggle required each round

```
vsetvli      zero, a4, e8, m1, ta, ma  # AVL=32, SEW=8
vrgather.vv  v25, v24, v8              # shuffle bytes
vsetivli     zero, 8, e32, m1, ta, ma  # AVL=8, SEW=32
vaesem.vv    v25, v10                  # 2×128b AES

vsetvli      zero, a4, e8, m1, ta, ma  # AVL=32, SEW=8
vrgather.vv  v24, v25, v8              # shuffle bytes
vsetivli     zero, 8, e32, m1, ta, ma  # AVL=8, SEW=32
vaesem.vv    v24, v11                  # 2×128b AES
```

Encrypt and decrypt 1024 bytes this way

```
=== AES-256 ===
aes256_exp_key():  ins=    58  cyc=  ?
aes256_enc(1024):  ins=   775  cyc=  ?
aes256_dec(1024):  ins=   775  cyc=  ?

=== Rijndael-256 ===
rij256_exp_key():  ins=   123  cyc=  ?
rij256_enc(1024):  ins=  2059  cyc=  ?
rij256_dec(1024):  ins=  2059  cyc=  ?
```

This is already “constant time”

The vrgather.vv instruction is “half constant-time” in the sense that under Zvkt the latency does not depend on the data being permuted.

However latency can depend on the permutation, but that is constant.

34.2.15.12. permute

In the `.vv` and `.xv` forms of the `vrgather[ei16]` instructions, the values in `vs1` and `rs1` are used for control and therefore are exempt from DIEL.

- vrgather.v[ivx]
- vrgatherei16.vv

Some Proposals for Consideration

1. With SEW=64 the AES instructions become Rijndael-256 Instructions.

`vaesef.[vv,vs], vaesem.[vv,vs], vaesdf.[vv,vs], vaesdm.[vv,vs], vaeskf2.vi`

2. Allow the same AES instructions with SEW=8, eliminating SEW toggling.

3. Modify `vaeskf2.vi` definition to support additional round constants.

4. Insert a hint into `vrgather` instruction for the special Rijndael-256 encryption and decryption permutations and “hardwire” them.