A Lightweight ISA Extension for AES and SM4

Markku-Juhani O. Saarinen

PQShield Ltd.
Oxford, United Kingdom
mjos@pqshield.com

***Abstract*—We describe a minimum baseline ISA extension for AES and SM4 block ciphers. The proposal requires 16 register-to-register instructions per round (+ key schedule) for AES and 6.5 instructions per step for SM4 on an RV32 target, assuming no other extensions. Having only one S-box the extension is optimized for minimal hardware area and is thus suitable for ultra-low power targets. Performance improvement is expected to be at least 500% over the standard instruction set. Perhaps even more importantly the ISA extension helps to protect against timing side-channel attacks. AES and SM4 implementations using ISA extension also have a much-reduced software footprint. The AES and SM4 instances can share the same data paths but are independent in the sense that a chip designer can implement SM4 without AES and vice versa. Full HDL source code for the instruction's combinatorial logic and C code for emulation is provided for the community under a permissive open source license. The implementation contains a depth- and size-optimized joint AES/SM4 S-Box logic based on the Boyar-Peralta construction with a shared non-linear middle layer, demonstrating additional avenues for logic optimization. The instruction logic has been experimentally integrated into the single-cycle execution path of the "Pluto" RV32 core and has been tested on FPGA.**

***Index Terms*—RISC-V, AES, SM4, Cryptographic ISA Extension, Lightweight Cryptography**

I. Introduction

The Advanced Encryption Standard (AES) is a 128-bit block cipher with 128/192/256 - bit key, defined in the FIPS 197 standard [1]. AES is a mandatory building block of the TLS 1.3 [2] security protocol and is widely used for storage encryption, shared-secret authentication, cryptographic random number generation, and in many other applications.

The SM4 block cipher [3] fulfills a similar role to AES in the Chinese market and is the main block cipher recommended for use in China. SM4 has a 128-bit block size, but only one key size, 128 bits. Even though its high-level structure differs completely from AES, the two share significant similarities in their sole nonlinear component, which is a single 8×8-bit "S-Box" substitution table in both cases.

Cache timing attacks on AES became well known after the mid-2000s when it was demonstrated that common table-based implementations can be exploited even remotely [4], [5]; very similar issues also affect SM4. In presence of a cache, the only way to make the execution time of these ciphers fully independent of secret data is to eliminate the table lookup either by implementing it as bitsliced Boolean logic or by providing a specific ISA extension for the S-Box lookup.

This work was supported by Innovate UK (R&D Project Ref. 104423.)

Consumer CPUs have had instructions to support AES for almost a decade via the Intel AES-NI in x86 [6] and ARMv8-A cryptographic extensions [7]; these are almost universally available in PCs and higher-end mobile devices such as phones. ARM also supports SM4 via the ARMv8.2-SM extension. The AES instructions have been shown to make AES less of a throughput bottleneck for high-speed TLS communication (servers) and storage encryption (mobile devices), thereby also extending battery life in the latter. Both Intel and ARM cryptographic ISAs require 128-bit (SIMD) registers, and are not available on lower-end CPUs.

In this work, we show that it is possible to create a simple AES and SM4 ISA extension that offers a significant performance improvement and timing side-channel resistance with a minimally increased hardware footprint. It is especially suitable for lightweight RV32 targets.

II. A Lightweight AES and SM4 ISA Extension

The ISA extension operates on the main register file only, using two source registers, one destination register, and a 5-bit field `fn[4:0]` which can be seen either as an "immediate constant" or just code points in instruction encoding. In either case, the interface to the (reference) combinatorial logic is:

```
module enc1s(
  output [31:0] rd,    // to output register
  input  [31:0] rs1,   // input register 1
  input  [31:0] rs2,   // input register 2
  input  [4:0]  fn     // 5-bit func specifier
);
```

See Section IV-B for encoding details of ENC1S as an RV32 R-type custom instruction for testing purposes. For RV64 the words are simply truncated or zero-extended.

For emulation, the instructions are encapsulated in C as:

```
uint32_t enc1s(uint32_t rs1, uint32_t rs2,
  int fn);              // ENC1Sfn rd, rs1, rs2
```

The five bits of fn cover encryption, decryption, and key schedule for both algorithms. Bits `fn[1:0]` first select a single byte from `rs1`. Two bits `fn[4:3]` indicate which $8 \rightarrow 8$ - bit S-Box is used (AES, AES^{-1}, or SM4), and additionally `fn[4:2]` specifies a $8 \rightarrow 32$ - bit linear expansion transformation (each of three S-Boxes has two alternative linear transforms, indicated by `fn[2]`). The expanded 32-bit

TABLE I
HIGH-LEVEL IDENTIFIERS (PSEUDO INSTRUCTIONS) FOR FN[4:2].

Identifier	**fn[4:2]**	**Description or Use**
`AES_FN_ENC`	`3'b000`	AES Encrypt round.
`AES_FN_FWD`	`3'b001`	AES Final / Key sched.
`AES_FN_DEC`	`3'b010`	AES Decrypt round.
`AES_FN_REV`	`3'b011`	AES Decrypt final.
`SM4_FN_ENC`	`3'b100`	SM4 Encrypt and Decrypt.
`SM4_FN_KEY`	`3'b101`	SM4 Key Schedule.
Unused	`3'b11x`	($4 \times 6 = 24$ points used.)

value is rotated back by $\{0, 8, 16, 24\}$ bits based on `fn[1:0]`. The result is XORed with `rs2` and written to `rd`.

Table I contains the identifiers (pseudo instructions) that we currently use for bits `fn[4:2]`. For all `fn` the second source operand `rs2` is XORed with a function of `rs1` to produce output. Often we can arrange computation so that `rd` = `rs2`, making a two-operand "compressed" encoding possible.

For AES the instruction selects a byte from `rs1`, performs a single S-box lookup (*SubBytes* or its inverse), evaluates a part of the MDS matrix (*MixColumns*) if that linear expansion is step selected, rotates the result by a multiple of 8 bits (*ShiftRows*), and XORs the result with `rs2` (*AddRoundKey*). There is no need for separate instructions for individual steps of AES as small parts of each of them have been incorporated into a single instruction. We've found that each one of these substeps requires surprisingly little additional logic.

For SM4 the instruction has the same data path with byte selection, S-Box lookup, and two different linear operations, depending on whether encryption/decryption or key scheduling task is being performed.

Both AES [1] and SM4 [3] specifications are written using big-endian notation while RISC-V uses primarily little-endian convention [8]. To avoid endianness conversion the linear expansion step outputs have a flipped byte order. This is less noticeable with AES, but the 32-bit word rotations of SM4 become less intuitive to describe (while wiring is equivalent).

We refer to the concise reference implementation discussed in Section IV for details about specific logic operations required to implement the ISA extension, and for unit tests and intermediate values.

III. USING THE AES AND SM4 INSTRUCTIONS

AES and SM4 were originally designed primarily for 32-bit software implementation. The ECN1S adopts this "intended" 32-bit implementation logic but removes the table lookup and rolls several individual steps into the same instruction.

A. AES Computation and Key Schedule

The structure of an AES implementation is equivalent to a "T-Table" implementation, with sixteen invocations of `AES_FN_ENC` per round and not much else (apart from fetching the round subkeys). In practice, two sets of four registers are used to store the state, with one set being used to rewrite the other, depending on whether an odd or even-numbered round is being processed. AES has $r \in \{10, 12, 14\}$ rounds, depending on the key size (on of $\{128, 192, 255\}$, respectively). The final round requires sixteen invocations of `AES_FN_FWD`. The same instructions are also used in the key schedule which expands the secret key to $4r+4$ subkey words.

The inverse AES operation is structured similarly, with `AES_FN_DEC` being used for main body rounds and `AES_FN_REV` for final output round. These instructions are also utilized for the reverse key schedule.

Four precomputed subkey words must be fetched in each round, requiring four loads (lw instructions) in addition to their address calculation (typically every other round). There is no need for separate *AddRoundKey* XORs as the subkeys simply initialize either one of the four-register sets used to store the state. It is also possible to compute the round keys "on the fly" without committing them to RAM. This may be helpful in some types of security applications. The overhead is roughly 30%. However, if the load operation is much slower than register-to-register arithmetic, the overhead of on-the-fly subkey computation can become negligible.

B. SM4 Computation and Key Schedule

SM4 has an unbalanced Feistel structure. The inverse cipher is equivalent to the forward cipher with a reversed subkey order. There is only one key size, 128 bits, and 32 steps which are typically organized into 8 rounds of 4 steps each. Each step uses all four state words and a single subkey word as inputs, overwriting one state word. Since the extensive input mixing is entirely linear some temporary XOR values are shared between steps. Each round requires 10 additional XORs, four for subkeys, and six for mixing the input words in addition to sixteen `SM4_FN_ENC` invocations of ENC1S. Therefore the SM4 performance is slightly lower than that of AES despite having fewer rounds.

The key schedule similarly requires 16 invocations of `SM4_FN_KEY` and 10 XORs to produce a block of four subkey words. The key schedule uses 32 "CK" round constants which can be either fetched from a table or computed with 8-bit addition operations.

For SM4 each block of four consecutive invocations of `SM4_FN_ENC` and `SM4_FN_KEY` share the same source and destination registers, differing only in `fn[1:0]` which steps through $\{0, 1, 2, 3\}$. We denote such a four-ENC1S block as pseudo instruction ENC4S. One can reduce the per-round instruction count of SM4 from 26 (+4 lw) to 14 (+4 lw) by implementing it as a "real" instruction that is four times larger than ENC1S. Note that AES does *not* benefit from ENC4S in encryption or decryption, only in key schedule.

IV. REFERENCE IMPLEMENTATION

An open-source reference implementation is available[1]. The initial distribution contains a C-language "emulator" code for the instruction and mock implementations for full AES-128/192/256 and SM4-128, together with essential unit tests. The distribution also has Verilog source code for combinatorial

[1]AES/SM4 ISA Extension: https://github.com/mjosaarinen/lwaes_isa

TABLE II
ALGEBRAIC GATE COUNTS FOR A BOYAR-PERALTA TYPE LOW-DEPTH S-BOXES THAT IMPLEMENT SM4 IN ADDITION TO AES AND AES^{-1}.

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

logic implementing the instruction (including the S-Boxes). This distribution is intended for deriving instruction counts and obtaining intermediate values for debugging purposes, but can be rapidly integrated into many RISC-V cores. Converting the mock C code to machine language implementing full AES or SM4 is easy, but we do not yet publicly provide source code for that since non-standard opcodes are being used.

A. About the AES, SM4 S-Boxes

AES and SM4 can share data paths so it makes sense to explore their additional structural similarities and differences. Both SM4 and AES S-Boxes are constructed from finite field inversion x^{-1} in $\mathsf{GF}(2^8)$ together with a linear (affine) transformations on input and/or output. The inversion makes them "Nyberg S-Boxes" [9] with desirable properties against differential and linear cryptanalysis, while the linear mixing steps are intended to break the bytewise algebraic structure.

Since x^{-1} is self-inverse (an involution) and affine isomorphic on all polynomial basis (SM4 and AES use different basis), AES, AES^{-1}, and SM4 S-Boxes differ only in their linear components. Note that due to its Feistel-like structure SM4 does not require an inverse S-Box for decryption like AES, which is a substitution-permutation network (SPN).

Boyar and Peralta [10] show how to build low-depth circuits for AES that are composed of a linear top and bottom layers and a shared nonlinear middle stage. We created additional top and bottom layers for SM4 specifically for this project that use the same the middle nonlinear layer as AES and AES^{-1}.

Here "linear" can be understood to mean NOT, XOR, and XNOR gates and the shared nonlinear layer consists of XOR and AND gates only. Each S-Box expands an 8-bit input to 21 bits in a linear inner ("top") layer, uses a shared nonlinear 21-to-18 bit mapping as a middle layer, and again compresses 18 bits to 8 bits in the outer ("bottom") layer. Table II gives the individual gate counts to each layer; summing up top, middle, and bottom gives the total S-Box gate count (≈ 128).

Implementors can consider if it is beneficial to multiplex the linear layers with the shared middle layer. However the required mux logic is large and increases depth, so our current reference implementation does not share the middle layer circuitry between AES and SM4.

Despite such a strict structure and limited choice of gates (that is suboptimal for silicon but very natural to mathematics), these are some of the smallest circuits for AES known. Note

TABLE III
RV32 SOC AREA WITH AND WITHOUT ENC1S (AES, AES^{-1}, SM4); 'PLUTO" CORE ON AN ARTIX-7 FPGA. EXTAES IS A CPU-EXTERNAL MEMORY-MAPPED AES-ONLY MODULE, PRESENTED FOR COMPARISON.

Resource	**Base**	**ENC1S (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)

that it is possible to implement AES with fewer gates (113 total), but this results in 50% higher circuit depth [11].

B. Experimental Instruction Encoding and Synthesis

For prototyping we interfaced the ENC1S logic using the *custom-0* opcode and R-type instruction encoding with `fn[4:0]` occupying lower 5 bits of the funct7 field:

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

The implementation has been tested with PQShield's "Pluto" RISC-V core. We synthesized the same core on low-end Xilinx Artix-7 FPGA target (XC7A35TICSG324-1L) with and without the ENC1S (AES, SM4) instruction extension and related execution pipeline interface.

For comparison, we also measured the size of an external, memory-mapped AES module "EXTAES". This module implements AES encryption only, not inverse AES or SM4. Table III summarizes the relative area of ENC1S and EXTAES. Note that the SoC used in the synthesis has some additional logic that is not relevant to the current discussion. We estimate that our instruction proposal increases the amount of core logic (LUTs) by about 5-10% over a typical baseline RV32I – much less for more complex cores.

The external module requires a large amount of additional slice register state. Such a memory-mapped state is much more difficult to manage and share among processes (and cores) than the state of ENC1S which is entirely in the integer register file. While the EXTAES module has 16 parallel S-Boxes and executes the core AES iteration itself in about a dozen cycles, loading and storing of blocks and waiting of the operation to finish creates significant additional latency that is usually larger than the latency caused by the operation itself.

V. PERFORMANCE AND SECURITY ANALYSIS

Much of the relative performance gain over a speed-optimized table-based implementation depends on the latency of memory load operations required by the latter. Generally speaking, a single ENC1S instruction replaces a byte mask, address calculation, table lookup (load), and an XOR operation – five instructions in total. The assembler-optimized AES implementation[2] referenced in [12] requires 80 instructions for the same task that is accomplished by 16 ENC1S instructions. However, 16 of those 80 are loads, which typically require more cycles than a simple arithmetic instruction. In addition to

[2]Ko Stoffelen: "RISC-V Crypto" [12] https://github.com/Ko-/riscvcrypto

these, each round requires some instructions for loading subkeys and managing loop counters; overall the performance of ENC1S is likely to be more than 500% better (sixfold) when compared to the best speed-optimized AES implementations on RV32. Note that the availability of 64-bit registers does not help table-based implementations much.

ENC1S-based AES and SM4 implementations are inherently constant-time and resistant to timing attacks. Stoffelen [12] also presents a constant-time, bitsliced AES implementation for RISC-V which requires 2.5 times more cycles when compared to the optimized table-based implementation. So ENC1S speedup over a timing side-channel hardened base ISA implementation is expected to be roughly 15-fold.

We are not aware of any definitive assembler benchmarks for SM4 on RISC-V, but based on instruction count estimates the performance improvement can be expected to be roughly similar or more (over 500 %). Without ENC1S any SM4 software implementation would benefit greatly from rotation instructions which have been proposed in the RISC-V bit manipulation ("B") extension, but not widely implemented.

We have only discussed timing side-channel attacks. Since these instructions interact with the main register file, any electromagnetic emission countermeasures would probably have to be extended to additional parts of the CPU core.

It may be possible to address electromagnetic emissions with completely different types of "masking" instructions. We note that the low multiplicative complexity of our S-Box logic helps when building side-channel resistance beyond timing attacks. Goudarzi et al [13] found the Boyar-Peralta type S-Box to be ideal for masked implementations, a general countermeasure against emission side-channel attacks.

VI. Conclusions

We have proposed a minimalistic RISC-V ISA extension for AES and SM4 block ciphers. The resulting speedup is 500% or more for both ciphers when compared to hand-crafted table-based assembler implementations.

In addition to saving energy and reducing latency in secure communications and storage encryption, the main security benefit of the instructions is their constant-time operation and resulting resistance against cache timing attacks. Such countermeasures are expensive in pure software implementations.

The instructions require logic only for a single S-Box, which is combined with additional linear layers to greatly increase code density. The hardware footprint of the instruction is very small as a result. If both AES and SM4 are implemented on the same target they can share data paths which is also helpful for hardware. However both AES and SM4 are independent options. It is also possible to implement and use the forward AES without inverse AES logic – common CTR-based modes do not require inverse cipher for decryption [14].

This proposal is targeted towards (ultra) lightweight MCUs and SoCs. A different type of ISA extension may provide additional speedups on 64-bit and vectorized platforms, but with the cost of significantly higher implementation area. Designers may still want to choose this minimal-footprint option if timing side-channel resistance is their primary concern.

References

[1] NIST, "Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication FIPS 197, November 2001.

[2] E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.3," IETF RFC 8446, August 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[3] SAC, "GB/T 32907-2016: SM4 block cipher algorithm," Cryptographic Standards Publication, original in Chinese. Also GM/T 0002-2012, August 2016. [Online]. Available: http://www.gmbz.org.cn/upload/2018-04-04/1522788048733065051.pdf

[4] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, ser. Lecture Notes in Computer Science, D. Pointcheval, Ed., vol. 3860. Springer, 2006, pp. 1–20. [Online]. Available: https://eprint.iacr.org/2005/271

[5] D. J. Bernstein, "Cache-timing attacks on AES," Web-published Manuscript, April 2005. [Online]. Available: http://cr.yp.to/papers.html#cachetiming

[6] S. Gueron, "Intel Advanced Encryption Standard (AES) new instructions set," May 2010, 323641-001. [Online]. Available: https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf

[7] ARM, "Arm A64 Instruction Set Architecture Armv8, for Armv8-A architecture profile," 2019, ARM DDI 0596 (ID 120619). [Online]. Available: https://static.docs.arm.com/ddi0595/f/SysReg_xml_v86A-2019-12.pdf

[8] A. Waterman and K. Asanović, Eds., *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213*. RISC-V Foundation, December 2019. [Online]. Available: https://riscv.org/specifications/

[9] K. Nyberg, "Differentially uniform mappings for cryptography," in *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, ser. Lecture Notes in Computer Science, T. Helleseth, Ed., vol. 765. Springer, 1993, pp. 55–64. [Online]. Available: https://doi.org/10.1007/3-540-48285-7_6

[10] J. Boyar and R. Peralta, "A small depth-16 circuit for the AES S-box," in *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings*, ser. IFIP Advances in Information and Communication Technology, D. Gritzalis, S. Furnell, and M. Theoharidou, Eds., vol. 376. Springer, 2012, pp. 287–298. [Online]. Available: https://eprint.iacr.org/2011/332

[11] ——, "A new combinational logic minimization technique with applications to cryptology," in *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings*, ser. Lecture Notes in Computer Science, P. Festa, Ed., vol. 6049. Springer, 2010, pp. 178–189. [Online]. Available: https://doi.org/10.1007/978-3-642-13193-6_16

[12] K. Stoffelen, "Efficient cryptography on the RISC-V architecture," in *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, ser. Lecture Notes in Computer Science, P. Schwabe and N. Thériault, Eds., vol. 11774. Springer, 2019, pp. 323–340. [Online]. Available: https://eprint.iacr.org/2019/794

[13] D. Goudarzi and M. Rivain, "How fast can higher-order masking be in software?" in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10210, 2017, pp. 567–597. [Online]. Available: https://eprint.iacr.org/2016/264

[14] M. Dworkin, "Recommendation for block cipher modes of operation: Methods and techniques," NIST Special Publication SP 800-38A, December 2001.