Accelerating Authenticated Block Ciphers via RISC-V Custom Cryptography Instructions

Yuhang Qiu[1,2], Wenming Li[1], Tianyu Liu[1,2], Zhen Wang[1,2], Zhiyuan Zhang[1,2], Zhihua Fan[1,*], Xiaochun Ye[1], Dongrui Fan[1], Zhimin Tang[1]
[1]State Key Lab of Processors, Institute of Computing Technology, CAS, Beijing, China
[2]School of Computer Science and Technology, UCAS, Beijing, China
{qiuyuhang21b, liwenming, liutianyu, wangzhen21s, zhangzhiyuan, fanzhihua, yexiaochun, fandr, tang}@ict.ac.cn

***Abstract*—As one of the standardized encryption algorithms, authenticated block ciphers based on Galois/Counter Mode (GCM) is a widely-used method to guarantee the accuracy and reliability in data transmission. The profiling work demonstrates that across the execution process of GCM mode, the authentication operation is the main performance bottleneck because it introduces operations in high-dimensional Galois field (GF), which could not be efficiently executed via existing ISA. To overcome this problem, we propose a custom ISA extension and cooperate it with RISC-V cryptography extension to accelerate the whole process of authenticated block ciphers. Besides, we design a specific crypto core including a fully-pipelined GF(2^{128}) multiplier to support the extended instructions and integrate it into the multi-issue out-of-order core XT910 without introducing any clock frequency overhead. The proposed design significantly reduces the the number of instructions required in the main operations of authenticated block ciphers. We compare the performance of our designs to other existing acceleration scheme based on RISC-V ISA extension. Experimental result shows that our design outperforms other related work and achieves up to 17× speedup with a lightweight hardware overhead.**

***Index Terms*—Block ciphers, Galois/Counter Mode, RISC-V, Instruction set extension**

I. Introduction

With the rapid development of information technologies, different types of cryptography algorithm are essential for providing basic security services such as confidentiality and integrity. Block cipher algorithms, such as AES and SM4, has been popular for decades. They divide the text into data blocks of equal length and perform encryption or decryption operations in different blocks using a same key, which shows a great convenience. However, the fixed key and blocking method makes it vulnerable to different types of attacks. Hence, different approaches have been proposed to enhance the security of the block cipher algorithms. For instance, masking [1] [2] is a popular technique against side-channel attacks. In addition, different mode of block ciphers like Cipher Block Chaining (CBC), Counter (CTR) and Galois/Counter (GCM) could be applied to protect the algorithm against the modification attack and the bit-flipping attack.

GCM mode was proposed by the National Institute of Standard and Technology (NIST) as the SP 800-38D standard.

* Corresponding author

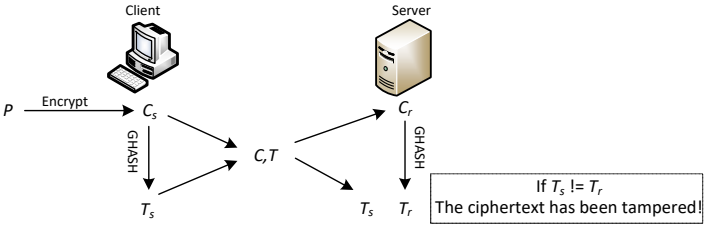

Fig. 1. The workflow of authenticated block cipher.

Compared with other block cipher mode, the introduction of authentication operations brings additional protection to data security besides encryption operation. Therefore, block ciphers in GCM mode are generally referred to as authenticated block ciphers. As is shown in Fig. 1, besides the encryption operation, GCM mode makes use of an authentication operation called GHASH to provide an authentication tag along with the ciphertext to the receiver to verify whether the ciphertext has been tampered, which ensures the confidentiality and integrity of message during transmission. Authenticated block ciphers has been popular in many applications, such as IOT [3] and IPsec [4].

The wide ranges of applications urge the need of efficient implementation of authenticated block ciphers. Accelerating basic block ciphers using ISA extension is a widely-used method because it could achieve the high performance of hardware implementation while maintaining a flexible and well-known programming model. For instance, Intel proposed AES-NI [5] to accelerate AES for X86 architecture. As for the open-source instruction set architecture RISC-V, the community proposed a scalar cryptography extension(K) [6] that accelerates several block ciphers including AES and SM4, which reduces the execution latency of up to 90%. However, this extension aims at accelerating the basic encryption process, and if extra optimizations like masking is applied to enhance the security, it would bring bottlenecks the K extension could not handle. In this paper, the profiling work demonstrates the authentication process would take up the majority of execution latency while running block ciphers in GCM mode under K-extended RISC-V ISA, because it brings large number of operations in 128-dimensional Galios field ($GF(2^{128})$). Hence, to maximum the potential of RISC-V K extension and meet the performance

978-3-9826741-0-0/DATE25/© 2025 EDAA

requirement of different applications, there is an urgent need to find the suitable method to accelerate the whole execution process of authenticated block ciphers.

In this paper, we try to cope this problem by proposing custom instructions and integrated hardware design for authenticated block ciphers. The proposed custom instructions could be used along with K extension to execute block cipher algorithms in GCM mode efficiently because it fills the lack of acceleration of authentication operation in other work. Experimental results demonstrate that our work has a great trade-off between performance, power and area.

The main contributions of this work are as following:

- We perform profiling by executing authenticated block ciphers under RV64 architecture with and without K extension to identify the performance bottlenecks.
- We propose custom instructions for 64-bit RISC-V architecture to accelerate multiplication in $GF(2^{128})$ in GHASH, which is the major bottleneck of block ciphers executed in GCM mode. Hardware designs are also implemented to support the custom instructions.
- We incorporate our instructions along with K extension and hardware design into the out-of-order core XT910 and implement the design in RTL.
- We evaluate the performance of our design by executing various block ciphers in GCM mode. Experimental results under 12nm technology demonstrate that the area and power overhead are relatively low compared to the achieved speedup.

The rest paper is organized as follows. Section II discusses the background and previous related works, the challenges of the current research are also provided. Section III elaborates on the proposed custom instructions and hardware design of this work. Section IV shows the evaluation and experimental result of this work. Finally, Section V concludes this work.

II. PRELIMINARIES

In this section, we produce a detailed description of the GCM mode of block ciphers and RISC-V ISA extension. The profiling of the execution of the algorithm with and without RISC-V K extension is also presented.

A. Authenticated block ciphers

Block ciphers divide the message into blocks of equal length, each of which is encrypted or decrypted separately using a determined algorithm and symmetric key. GCM is a block cipher mode specifically targeting at 128-bit block ciphers like AES. It makes use of operation that uses universal hashing over a binary Galois field to provide authenticated encryption. For convenience, in the rest of the paper we assume that all the variables included in the process of authenticated encryption could be divided into full 128-bit blocks except for some special cases. As is shown in Algorithm 1, in GCM mode of block ciphers, the initial counter Y_0 and an extra additional authentication tag T is obtained by a pseudo-random function called GHASH besides the basic encryption operations(lines 2-6 and line 13).

Algorithm 1: Authenticated block ciphers

Input: Plaintext P, authenticated data A, initialization vector V, key K
Output: Ciphertext C, autenticated tag T
Note: E(K, P) means a block cipher function (e.g., AES) with a key K, incr() means incrementing the rightmost 32bits modulo 2^{32}.

1 H = E$(K, 0)$;
2 **if** *len(IV)=96* **then**
3 $\quad Y_0 \leftarrow IV \| 0^{31} \| 1$;
4 **end**
5 **else**
6 $\quad Y_0 \leftarrow$ GHASH$(H, 0^{128}, IV)$;
7 **end**
8 $s \leftarrow \lceil \text{len}(C)/128 \rceil$;
9 **for** *i=1 to s* **do**
10 $\quad Y_i \leftarrow \text{incr}(Y_{i-1})$;
11 $\quad C_i \leftarrow P_i \oplus \text{E}(K, Y_i)$;
12 **end**
13 $T \leftarrow$ GHASH$(H, A, C) \oplus$ E(K, Y_0) ;
14 return C,T

GHASH is a hash function defined over $GF(2^{128})$, in which the addition operations (equivalent to bitwise XOR) could be efficiently executed with basic ISA. However, multiplication in $GF(2^{128})$ demands specific software implementation which brings considerable overheads. GHASH is defined by GHASH$(H, A, C) = X_{m+n+1}$, Where A and C are divided into m and n groups of 128-bit blocks, respectively. The X_i is obtained through equation (1), where all operators represent operations in $GF(2^{128})$.

$$X_i = \begin{cases} 0 & i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & i = 1, ..., m \\ (X_{i-1} \oplus C_{i-m}) \cdot H & i = m+1, ..., m+n \\ (X_{i-1} \oplus (len(A) \| len(C)) \cdot H & i = m+n+1 \end{cases} \tag{1}$$

B. RISC-V ISA extension

RISC-V is an open and freely accessible ISA, which shows better convenience and availability over other ISA like ARM and X86. The RISC-V ISA could be separated into a basic ISA and optional extensions. In addition to the standardized extensions, the RISC-V ISA reserves space for designer to introduce custom instructions to application-specific tasks. In recent years, many companies and researchers are developing ISA extensions for various applications, such as artificial intelligence [7] [8], Internet of things [9] [10] and cryptographic computation [11] [12].

The RISC-V community has proposed a scalar cryptography(K) extension to accelerate different cryptographic algorithms like AES, SHA256, SHA512, SM3 and SM4. In this work we focus on instructions targeting at AES and SM4 since they are algorithms belong to the type of block ciphers.

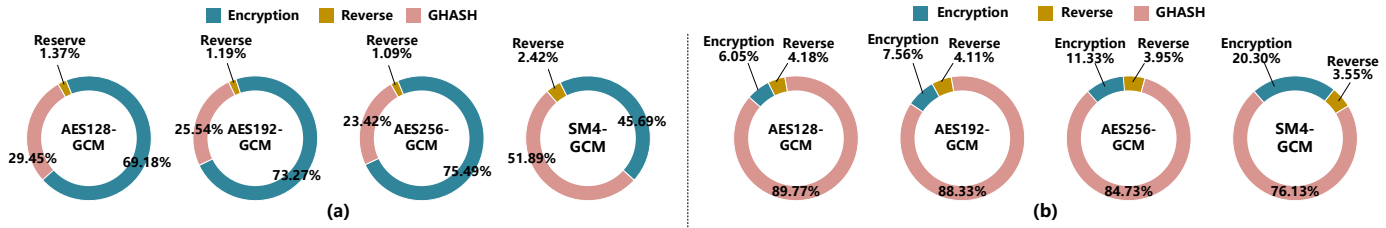

Fig. 2. The profiling result of authenticated block ciphers under (a) only base RV64 ISA (b) RV64 ISA with RV64 K extension. The scheme parameters are $len(IV) = 512, len(A) = 256, len(P) = 512$.

C. Challenges

For RISC-V cryptography extension(K), the main challenge is to accelerate operations that could not be covered by existing instructions. Taking AES-GCM as an example, the K extension introduces instructions to accelerate operations inside AES such as SubBytes, MixColumns and ShiftRows. However, these instructions fail to handle operations GCM mode brings such as GHASH and incr. This flaw would introduce new performance bottleneck and limit the speedup of K extension.

Fig. 2 shows the profiling results of AES128, AES192, AES256 and SM4 executed in GCM mode under RISC-V architecture with or without K extension, respectively. It is clear that under most cases the encryption operation incurs the most latency under the basic RISC-V ISA. On the other hand, when the K extension is applied, the encryption operation is wholly accelerated, which makes the GHASH becomes the main performance bottleneck because it introduces operations in $GF(2^{128})$. Data reversing between adjacent encryption operations is also a bottleneck operation, which brings nearly 4% of the total latency.

Over the years, in the context of various types of optimizations like GCM mode being proposed to enhance the security of block ciphers, there have been several work focusing on proposing custom instructions with better flexibility than RISC-V K extension. Kuo *et al.* [11], Cui *et al.* [12] and Choi *et al.* [13] focused on accelerating operation over $GF(2^8)$ through ISA extension or instruction overloading, but the acceleration was not considerable when masking operations were not introduced. For GCM mode of block ciphers, RISC-V community proposed published Vector Crypto Draft in 2023 [14], in which there are two vector instructions aiming at accelerating the GHASH. However, it requires the RISC-V core to have a complete vector unit, which makes it difficult to cooperate with scalar cryptography extension to accelerate the whole process of authenticated block ciphers. In this work our instruction and hardware design are designed to fit the pipeline of scalar RISC-V core to maximum the potential of RISC-V K extension.

III. System Design

In this section we elaborate on the custom instructions we proposed and the corresponding hardware design. We also discuss how they can be integrated into an out-of-order multi-issue RISC-V core.

A. Hardware design of $GF(2^{128})$ multiplier

GHASH and some other operations in block ciphers introduce the requirement of accelerating operations in $GF(2^m)$. Given the fact that the addition operation in $GF(2^m)$ could be realized by simple XOR, the main concern is how to execute multiplication in $GF(2^m)$ efficiently.

In general, multiplication operations in $GF(2^m)$ could be implemented using serial approach without introducing long execution latency. However, as the value of m increases, serial implementation would spend exponentially increasing execution time and become difficult to be mapped into specific hardware. In this paper we instead focus on implementation of matrix form. The first step is the carry-less multiplication. It takes two m-bit input as polynomials of degree $(m-1)$ and the product d could be seen as a polynomial of degree $(2m-2)$. Then the next step is to carry out polynomial reduction based on an irreducible polynomial R, in which the degree of d is reduced to obtain the final result q of degree $(m-1)$. The matrix representation of reduction is described in equation (2), where the vector d and vector q are vector representations of the results of carry-less multiplication and reduction, respectively. The $m \times (m-1)$ matrix R is uniquely determined by the irreducible polynomial. Suppose the irreducible polynomial is $f(x) = f_0 + f_1 x + f_2 x^2 ... + f_{m-1} x^{m-1} (f_i = 0,1)$, the elements of the matrix R could be obtained from equation (3).

$$\begin{bmatrix} q_0 \\ q_1 \\ \vdots \\ q_{m-1} \end{bmatrix} = \begin{bmatrix} r_{0,0} & \cdots & r_{0,m-2} \\ r_{1,0} & \cdots & r_{1,m-2} \\ \vdots & \ddots & \vdots \\ r_{m-1,0} & \cdots & r_{m-1,m-2} \end{bmatrix} \begin{bmatrix} d_m \\ d_{m+1} \\ \vdots \\ d_{2m-2} \end{bmatrix} + \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{m-1} \end{bmatrix} \tag{2}$$

$$r_{i,j} = \begin{cases} f_i & i=0,...,m-1; j=0 \\ f_i \cdot r_{m-1,j-1} & i=0; j=1,...,m-2 \\ f_i \cdot r_{m-1,j-1} + r_{i-1,j-1} & i=1,...,m-1; j=1,...,m-2 \end{cases} \tag{3}$$

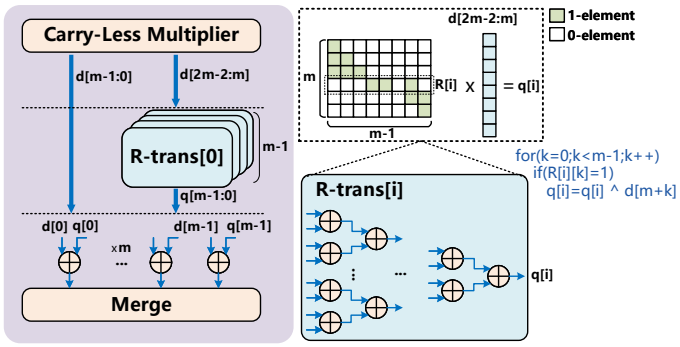

Fig. 3. The modular hardware design of $GF(2^m)$ multiplier.

Since the matrix implementation does not introduce any branch and stall operations, in this paper we explore the feasibility of performing multiplication operations in $GF(2^m)$ efficiently on hardware. Fig. 3 presents the logic diagram of $GF(2^m)$ multiplier based on matrix implementation, where the R-trans represents the implementation of the reduction operation. Once m is determined, the structure of the carry-less multiplier is determined accordingly with the latency of $\log_2 m$. In this case, the overall complexity of the multiplier is determined by the R-trans part. As is shown in Fig. 3, R-trans executes $(m-1)$ sets of linear transformations in parallel, and the complexity of each set depends on the number of 1-element in the corresponding row of the matrix R. In summary, the number of gates on the critical path of the $GF(2^m)$ multiplier can be expressed by equation (4), where H_i represents the number of 1-element in the ith row of the matrix R.

$$T_C = \log_2 m + \log_2 (\max\{H_i\}+1) \quad i=1,...,m \qquad (4)$$

In this work, we implement the $GF(2^{128})$ multiplier based on the design mentioned above with $m=128$. In context of GCM mode of block ciphers, the irreducible polynomial is given by $f(x) = 1+x+x^2+x^7+x^{128}$. The multiplier can be designed fully-pipelined to fit into the pipeline without deteriorating the critical path of the CPU core.

B. Instruction Extensions

RISC-V ISA has multiple subsets to support different architectures, and specific tasks can be accelerated by applying different extensions. In this section, we propose three custom instructions to accelerate GHASH in authenticated block ciphers. Furthermore, we propose two custom instructions as supplement of RISC-V K extension to execute block ciphers more efficiently. The descriptions of the proposed custom instructions are shown in Table. I.

TABLE I
INTRODUCTION OF THE CUSTOM CRYPTOGRAPHY INSTRUCTIONS

Instructions	Description
set.h	Merges the operands in registers *rs1* and *rs2* and store it in H-CSR, return the the operands in register *rs1* to *rd*.
gf128mul.l	Perform multplication of {*rs2,rs1*} and H-CSR in $GF(2^{128})$, and the least significant 64 bits are stored in *rd*
gf128mul.h	Perform multplication of {*rs2,rs1*} and H-CSR in $GF(2^{128})$, and the most significant 64 bits are stored in *rd*
greviw	Invert the bits in *rs1* according to the mask in *imm*, and the result is stored in *rd*
aes64.subh	Devide the *rs1* into eight 8-bit components and performs AES SubBytes operation on the last four components, and the result is stored in *rd*

a) Instructions for GHASH: The main performance bottleneck of the GHASH function is the multiplication operation in $GF(2^{128})$. which requires two 128-bit inputs and is hard to be divided into separate operations. Given the size of the physical registers in the RV64 architecture is 64, it makes a single RISC-V instruction fail to obtain the operands needed without utilizing the function field of the regular instruction.

However, from equation (1) we could notice that the multiplications included in GHASH shares a same operand H. Therefore, we introduce an 128-bit register H-CSR to hold this operand throughout the execution process. Correspondingly we propose custom instruction *set.h*, which merges the operands in registers *rs1* and *rs2* and stores the result in H-CSR, and the operand in *rs1* is returned to destination register *rd*.

The *gf128mul.l* and *gf128mul.h* are arithmetic instructions proposed to perform the multiplication in $GF(2^{128})$. They merge the operands in two source registers as the first operand, the value stored by H-CSR as the second operand, and store the high or low parts of the product into the destination register *rd*, respectively. Along with the instruction *set.h*, the multiplication operation in $GF(2^{128})$ and the whole process of GHASH with arbitrary input length could be completed within moderate number of instructions.

b) Application and supplement of K extension: The RISC-V K extension is proposed to accelerate various cryptographic algorithms, among which AES and SM4 belong to block ciphers. Therefore, in this work we apply instructions targeting at AES and SM4 in the RV64 K extension as a part of our design.

Moreover, we propose two custom instructions as supplement of K extension to maximum the efficiency of encryption operation. The first is *greviw*, which performs data reversing operation throughout the whole process of authenticated block ciphers. As we can see, this instruction has the same function and form as the B extension. Another is *aes64.subh*, which is proposed to accelerate the KeySchedule operations of specific round in AES256. The KeySchedule operation of AES128 and AES192 could be fully accelerated by *aes64.ks1* and *aes64.ks2* in RV64 K extension, while AES256 introduces an extra SubBytes operation. As is shown in Fig. 4, the participation of *aes64.subh* could help maximum the performance of the KeySchedule operation in AES256.

```
for(rnd=0 to 6)
      aes64.ks1i  %t[rnd+1],    %k3[rnd],     %rnd
      aes64.ks2   %k0[rnd+1],   %t[rnd+1],    %k0[rnd]
      aes64.ks2   %k1[rnd+1],   %k0[rnd+1],   %k1[rnd]
      aes64.subh  %k2[rnd+1],   %k1[rnd+1]
      aes64.ks2   %k2[rnd+1],   %k2[rnd+1],   %k2[rnd]
      aes64.ks2   %k3[rnd+1],   %k2[rnd+1],   %k3[rnd]
end
```

Fig. 4. The workflow of KeySchedule in AES256 using ISA extension.

C. Integration in Out-of-order core

In order to demonstrate the feasibility of custom instructions and hardware design, in this section we detail the integration of our design into an out-of-order multi-issue processor core. We first implement a custom crypto core to support the RV64 K extension and custom instructions targeting at AES and SM4. In addition, we include a $GF(2^{128})$ multiplier in the crypto core based on the methods proposed in Section III-A to support the *gf128mul.l* and *gf128mul.h* instructions. All the components included in the crypto core are designed to be fully-pipelined and have the same execution delay.

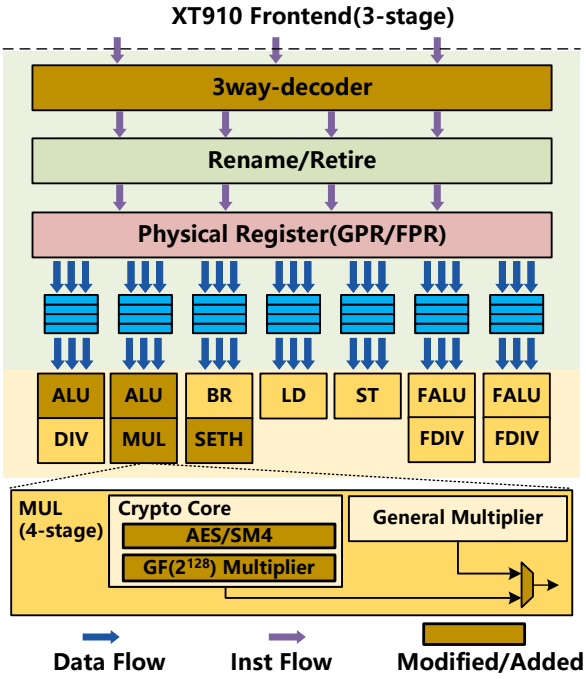

Fig. 5. The main pipeline of extended XT910 core.

We integrated our design to open-source core XT910, an out-of-order core with 3-way decoder and 7 issue queues. In the decoding phase, the decoder is modified to recognize added extended instructions. In the execution phase, the logic of *greviw* is added to all of the ALUs of XT910 due to its low complexity. The crypto core and *set.h* are added to ALU/MULT queue and branch queue, respectively. The modified XT910 pipeline is shown in Fig. 5. The allocation of the crypto core lies in that the hardware implementations of general multiplier and $GF(2^{128})$ multiplier have similar complexity, and the whole authenticated encryption process does not introduce any general multiplication. Furthermore, *set.h* involves operations towards the new control and state register H-CSR, and it needs to cooperate with *gf128mul.l* and *gf128mul.h* in a fixed order to complete the multiplication in $GF(2^{128})$, hence it would bring hazard which could not be handled by renaming technique in out-of-order core. Allocating the pipeline of *set.h* to branch queue could resolve this problem because the flushing technique inside the branch queue could be utilized to guarantee that the instructions following *set.h* would be stalled until it is committed.

IV. Evaluation

In this section, we illustrate the evaluation setting used to test our extended instructions and the performance result of GHASH and various authenticated block ciphers executed on our design. The hardware overhead of the extended CPU core is also provided.

A. Experiment settings

Methodology: We implemented our hardware design in RTL and integrated it into the out-of-order core XT910 following the method described in Section III-C. We synthesized the extended core using 12nm process, meeting timing at a 2GHz clock frequency consistent with that described in [15].

Baseline: We take the original XT910 core with RV64GC as our baseline. For GHASH, we implemented the method described in [16] for comparison. Furthermore, we implemented two extended XT910 to demonstrate the advantage of our design in authenticated block ciphers. The first supports only the RISC-V K extension. The second supports an extended instruction for $GF(2^8)$ arithmetic proposed in [11]. For fair comparison, all extended instructions are designed to be fully pipelined, and the added hardware is assured that would not deteriorate the critical path of XT910.

Benchmark: We first compare the performance of GHASH with different input lengths with or without custom instructions to verify the effectiveness of our design. Moreover, we compare the performance of multiple authenticated block ciphers, including AES128-GCM, AES192-GCM, AES256-GCM and SM4-GCM. The RISC-V GNU toolchain is modified to support all the extended instructions mentioned above and all the applications written in C with different extensions are successfully compiled. It is worth noting that since the XT910 has its own compiler whose source code is not open, we performed our test on XT910 by overwriting unused existing instructions.

B. GHASH

GHASH is the key function in the authentication encryption process, and it is also the main target of our proposed custom instructions. Hence, we compare the performance of GHASH with different scheme parameters. The comparison is done on the baseline XT910 core and our design, the state-of-art method [16] using RISC-V B extension to accelerate GHASH is also included.

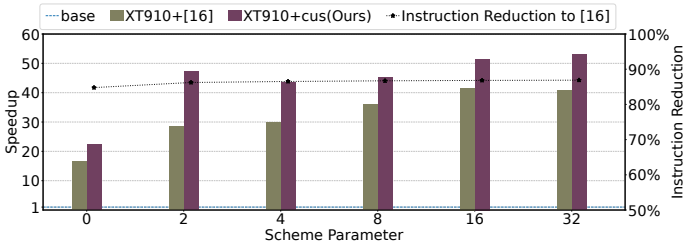

Fig. 6. Performance results of GHASH. The scheme parameters represent $len(A+C)/128$. The case '0' represents a single multiplication in $GF(2^{128})$.

Fig. 6 presents the performance result of our design versus two baselines selected. The presented data demonstrates that our design achieves 22× speedup in $GF(2^{128})$ multiplication and 43× to 53× speedup in GHASH. With comparison to the method described in [16], our design achieves an average reduction of instructions of 86%. When executed on XT910 core, our design demonstrates up to 1.60× speedup. With the input length of GHASH increasing, the performance improvement would be better due to the increasing number of multiplication operations in $GF(2^{128})$ included in GHASH.

C. Authenticated Block ciphers

To evaluate the effectiveness of our design on different authenticated block ciphers, we compare the performance of

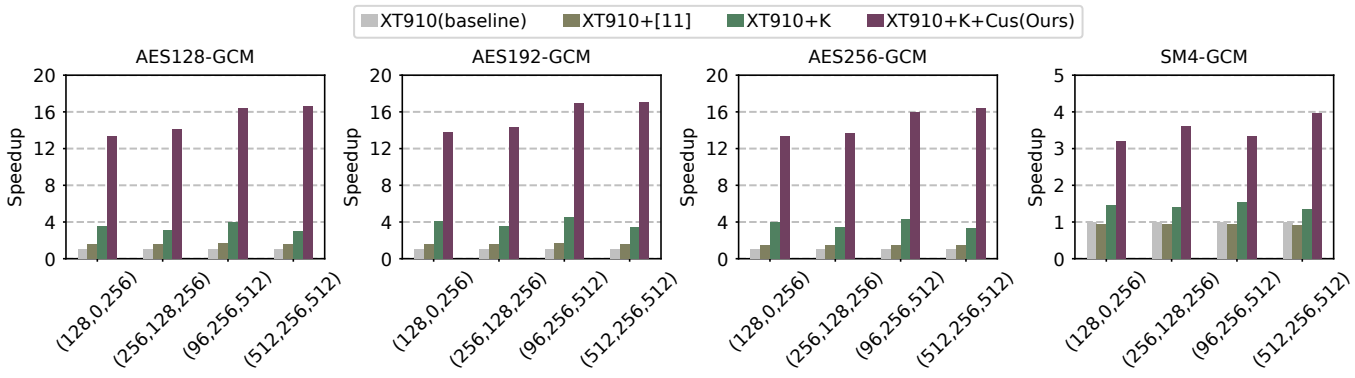

Fig. 7. Performance results of authenticated block ciphers. The scheme parameters represent $(len(A), len(IV), len(P))$.

our design and several baselines mentioned in Section IV-A. The performance results of AES128-GCM, AES192-GCM, AES256-GCM and SM4-GCM across different scheme parameters are shown in Fig. 7. The XT910 extended with $GF(2^8)$ ISA extensions proposed in [11] achieves up to 1.66× speedup compared to baseline because it managed to accelerate the SubBytes operation in all applications and the MixColumn operation in AES. Due to the absence of other optimizations like masking, the number of arithmetic operations in $GF(2^8)$ included is not large, which limits the improvement of the performance to some extent. For SM4-GCM, its performance is even worse than the baseline if lookup tables are used to complete SubBytes operation. In comparison, the K extended XT910 achieves higher speedup because it could accelerate almost the whole encryption process in authenticated block ciphers, especially in the case of $len(IV) = 96$ because there is no need to generate the initial count value through GHASH, as is shown in Algorithm 1.

The design in this work extends the XT910 with K extension and custom instructions proposed in Section III-B. Experimental result demonstrates that it outperforms other state-of-art researches on all types of authenticated block ciphers. It achieves the best acceleration in AES192, with a speedup of 13.78× to 17.00× across different scheme parameters. Among all of the applications selected, it achieves a greater speedup with the increase of arbitrary scheme parameter. This further validates the advantage of our design over state-of-art researches.

D. Hardware Overhead

The evaluation of our hardware design is performed in 12nm technology with the frequency of 2GHz, and it is confirmed that there is no decrease in frequency due to the addition of new hardware. The area and power overhead of baseline XT910 and extra hardware to support ISA extension is shown in Table II. The control overhead mainly lies in the decoding logic added towards the decoder and issue queues involved, which is negligible. The main overhead comes from the crypto core, in which the $GF(2^{128})$ multiplier occupies 77% of area and 66% of dynamic power, respectively. Overall, the hardware introduced to support ISA extension increase area by 2.17% and dynamic power by 4.89% with regard to baseline. Compared with the improvement of performance, the hardware overhead would be lightweight.

TABLE II
HARDWARE OVERHEAD OF EXTENDED XT910 CORE

Project		Area(mm^2)	Dynamic Power(mW)
XT910(baseline)		0.6	200
Crypto core	K extension accelerator	0.003	3.34
	$GF(2^{128})$ multiplier	0.010	6.45
	Total	0.013	9.79
Overhead		2.17%	4.89%

V. Conclusion

In this paper, we propose a custom cryptography RISC-V ISA extension to accelerate applications of authenticated block ciphers with the participation of RISC-V K extension. The extended instructions and corresponding hardware designs are integrated into the out-of-order core XT910. Our work explores and alleviates the performance bottleneck of the RISC-V K extension in accelerating authenticated block ciphers, and the design achieves a speedup of up to 17.00× compared to the baseline and outperforms the existing work while introducing only lightweight hardware overhead.

VI. Acknowledgments

This work was supported by National Key R&D Program of China (Grant No. 2023YFB4503500), Institute of Computing Technology, Chinese Academy of Sciences-China Mobile Communications Group Co.,Ltd. Joint Institute, Beijing Natural Science Foundation (Grant No.L234078) and Beijing Nova Program (NO. 20220484054, No.20230484420).

References

[1] J. Balasch *et al.*, "Consolidating inner product masking," in *Advances in Cryptology – ASIACRYPT 2017* (T. Takagi and T. Peyrin, eds.), (Cham), pp. 724–754, Springer International Publishing, 2017.

[2] M. Rivain and E. Prouff, "Provably secure higher-order masking of AES." Cryptology ePrint Archive, Paper 2010/441, 2010.

[3] Y. Sovyn *et al.*, "Comparison of three cpu-core families for iot applications in terms of security and performance of AES-GCM," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 339–348, 2020.

[4] W. Stallings, *Cryptography and network security - principles and practice (8. ed.)*. Prentice Hall, 2023.

[5] K. Akdemir *et al.*, "Breakthrough AES performance with intel AES new instructions." WhitePaper, 2015.

[6] A.Zeh *et al.*, "RISC-V cryptographic extension proposals - version 0.6.1." [Online], 2020. https://lists.riscv.org/g/tech-crypto-ext/attachment/259/0/riscv-crypto-spec-v0.6.1.pdf.

[7] S. Lee *et al.*, "DBPS: dynamic block size and precision scaling for efficient DNN training supported by RISC-V ISA extensions," in *60th ACM/IEEE Design Automation Conference, DAC 2023, San Francisco, CA, USA, July 9-13, 2023*, pp. 1–6, IEEE, 2023.

[8] F. Conti *et al.*, "Marsellus: A heterogeneous RISC-V AI-IoT end-node SoC with 2-8 b DNN acceleration and 30%-boost adaptive body biasing," *IEEE J. Solid State Circuits*, vol. 59, no. 1, pp. 128–142, 2024.

[9] H. B. Amor *et al.*, "A RISC-V ISA extension for ultra-low power IoT wireless signal processing," *IEEE Trans. Computers*, vol. 71, no. 4, pp. 766–778, 2022.

[10] Y. Chen *et al.*, "RISC-V custom instructions of elementary functions for IoT endpoint devices," *IEEE Trans. Computers*, vol. 73, no. 2, pp. 523–535, 2024.

[11] Y. Kuo *et al.*, "RISC-V galois field ISA extension for non-binary error-correction codes and classical and post-quantum cryptography," *IEEE Trans. Computers*, vol. 72, no. 3, pp. 682–692, 2023.

[12] S. Cui and J. Balasch, "Efficient software masking of AES through instruction set extensions," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*, pp. 1–6, IEEE, 2023.

[13] P. Choi *et al.*, "Architectural supports for block ciphers in a RISC CPU core by instruction overloading," *IEEE Trans. Computers*, vol. 71, no. 11, pp. 2844–2857, 2022.

[14] RISCVTeam, "RISC-V cryptography extensions volume II vector instructions," 2023. https://github.com/riscv/riscv-crypto/releases/tag/v1.0.0/riscv-crypto-spec-vector.pdf.

[15] C. Chen *et al.*, "Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance RISC-V processor with vector extension : Industrial product," in *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Virtual Event / Valencia, Spain, May 30 - June 3, 2020*, pp. 52–64, IEEE, 2020.

[16] B. Marshall *et al.*, "The design of scalar AES instruction set extensions for RISC-V," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 1, pp. 109–136, 2021.