IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 8 August 2022, accepted 6 September 2022, date of publication 12 September 2022, date of current version 20 September 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3205617*

RESEARCH ARTICLE

# Optimized Implementation of PIPO Block Cipher on 32-Bit ARM and RISC-V Processors

**YOUNGBEOM KIM, (Student Member, IEEE), AND SEOG CHUNG SEO, (Member, IEEE)**
Department of Financial Information Security, Kookmin University, Seoul 02707, South Korea

Corresponding author: Seog Chung Seo (scseo@kookmin.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) Grant through the Korea Government [Ministry of Science and ICT (MIST)] under Grant 2022R1C1C1013368.

**ABSTRACT** A lightweight block cipher PIPO-64/128 was presented in ICISC'2020. PIPO of the 8-bit unit using an unbalanced-bridge S-box showed better performance than other lightweight block cipher algorithms on an 8-bit AVR environment. So far, optimization methods for implementing PIPO have been proposed in various environments; however, no optimization research has been conducted for two popular 32-bit based processors: ARM Cortex-M4 and RISC-V. Since RISC-V and ARM Cortex-M series platforms do not support bit-based Single Instruction Multiple Data (SIMD) instructions, several aspects should be considered to apply a forced parallelization strategy. In this article, we discuss the implementation methodology of PIPO for 32-bit RISC-V and ARM Cortex-M4 environments. We optimize the performance of S-Layer via proposed register-scheduling and masking technique while we maintain parallelism to the R-Layer implementation. Moreover, we propose an on-the-fly key scheduling technique for further performance improvement. Finally, compared to the existing reference implementations in RISC-V and ARM Cortex-M4 platforms, when 4 plaintext encrypted simultaneously, our software achieved performance of 229% and 370%, respectively.

**INDEX TERMS** Pipo, arm-cortex m4, risc-v, efficient implementation, software optimization, internet of things, embedded security.

## I. INTRODUCTION

As industrial technology becomes common, various embedded devices are being actively used in daily life and industry. Embedded devices mainly used for sensor nodes are called Internet of Things (IoT) devices, and IoT devices are equipped with hardware for application and communication. Advances in application services have accelerated the importance of data confidentiality, integrity, and authenticity. To solve this issue, a block cipher operation mode Galois-Counter Mode/CBC-MAC with Counter (GCM/CCM) which is an authenticated encryption method is proposed, but there was a limitation in applying GCM/CCM on constrained embedded devices. In response, the development of lightweight ciphers has been continuously made in terms of memory usage, speed, code size, and low-power consumption; recently, Lightweight Cryptography (LWC) competition for lightweight Authenticated Encryption with Associated Data (AEAD) was held by the National Institute of Standards and Technology (NIST). Since the AEAD algorithms submitted for the LWC competition uses lightweight cryptography as a core premetive, the performance of lightweight cryptography is one of the important evaluation criteria. However, since IoT devices are relatively vulnerable to side-channel attacks, whether side-channel countermeasures can be applied in the core primitive and performance evaluation of countermeasures are also important considerations.

Block cipher PIPO is lightweight cryptography proposed by ICISC'20 [1]. PIPO's S-Box is designed through an unbalanced-bridge structure, and the permutation process, R-Layer, is designed through a simple shift operation. Due to the specificity of S-Box, PIPO is oriented towards a bit-slicing implementation different from other lightweight

The associate editor coordinating the review of this manuscript and approving it for publication was Junggab Son.

ciphers such as GIFT and has the advantage of having a small number of encryption/decryption rounds [2]. In addition, due to the above characteristics, the countermeasure of side-channel attack can be applied with fewer calculations compared to other block ciphers. In particular, in the 8-bit AVR environment, PIPO achieves high speed, small code size, and low memory requirements compared to other lightweight block ciphers. Accordingly, various studies are being actively conducted to evaluate the performance of PIPO from the perspective of implementation in various environments. Recently, research on the optimized implementation of PIPO equipped with countermeasures of side-channel attack in the 8-bit AVR environment and parallel implementation of PIPO equipped with countermeasures of fault attack in the 64-bit ARM Cortex-A series were conducted [3], [4].

However, research on the most used 32 bits-based processor among sensor nodes has not been conducted so far. ARM Cortex-M4 is currently a popular-device in the IoT industry; and, optimized implementations of various lightweight cryptographic algorithms have been benchmarked on the ARM Cortex-M4 [5], [6], [7], [8], [9]. In addition, the implementation result of PIPO for RISC-V, an embedded device that has recently been spotlighted, is still not clear. Therefore, in this article, we present the first optimized implementation of PIPO for 32 bits-based RISC-V and ARM-Cortex M4 environments.

Our contributions in this work can be summarized as below:

1) *Presenting First Optimized software of PIPO on 32-bit RISC-V/ARM Cortex-M*
   In this article, we present an optimization methodology for PIPO-64/128 block cipher in a 32 bits-constrained environment. Based on the proposed method, our re-designed PIPO software achieves fast execution time via parallel-implementation logic. It also provides flexibility by providing two versions of the implementation. Compared to the existing reference implementations in RISC-V and ARM Cortex-M4 platforms, when 4 plaintext encrypted simultaneously,the optimized PIPO implementation achieved performance of 229% and 370%, respectively. As far as we know, our software is the first PIPO implementation in 32-bit platforms.
2) *Proposing Parallel Computation Logic for Performance Efficiency*
   Our PIPO implementation is efficiently compressed through three techniques. The first is to use a forced parallel technique. we design and apply parallel logic suitable for PIPO block cipher by utilizing efficient register-scheduling and RISC/ARM instruction set. This allows simultaneous encryption of 4 plaintexts. The second is a rotate-shift technique based on bit-masking. With a masking operation that avoids bit interference, we implement an R-Layer with minimal cost. The third is a combined-AddRoundKey operation. The AddRoundkey operation is implicitly implemented in R-Layer with round keys held in general-purpose registers. Finally, our implementation is further optimized through the hand-written assembly.
3) *Presenting Extensive Performance Analysis*
   We compare our PIPO block cipher implementation in detail with various lightweight/general block ciphers in RISC-V and ARM Cortex-M4 environments. We evaluate the practical applicability of PIPO block cipher through a detailed comparison based on RAM usage, code size, and Clock cycles Per Byte (CPB). Finally, we show that our optimized PIPO software is sufficiently competitive.

### A. CODES

Our implementations are Open Source and are available at **https://github.com/Youngbeom94/PIPO_RISC-ARM**

### B. EXTENDED VERSION OF WISA'21

In this article, we expand on previous our work published in WISA'21 [10]. In WISA'21, page limitations made it difficult to describe our optimization technique in detail; therefore, in this article, we describe the optimization methods in detail and additionally present implementation techniques for the ARM Cortex-M4 device to prove the expandability of our optimization methodology.

### C. OUTLINE

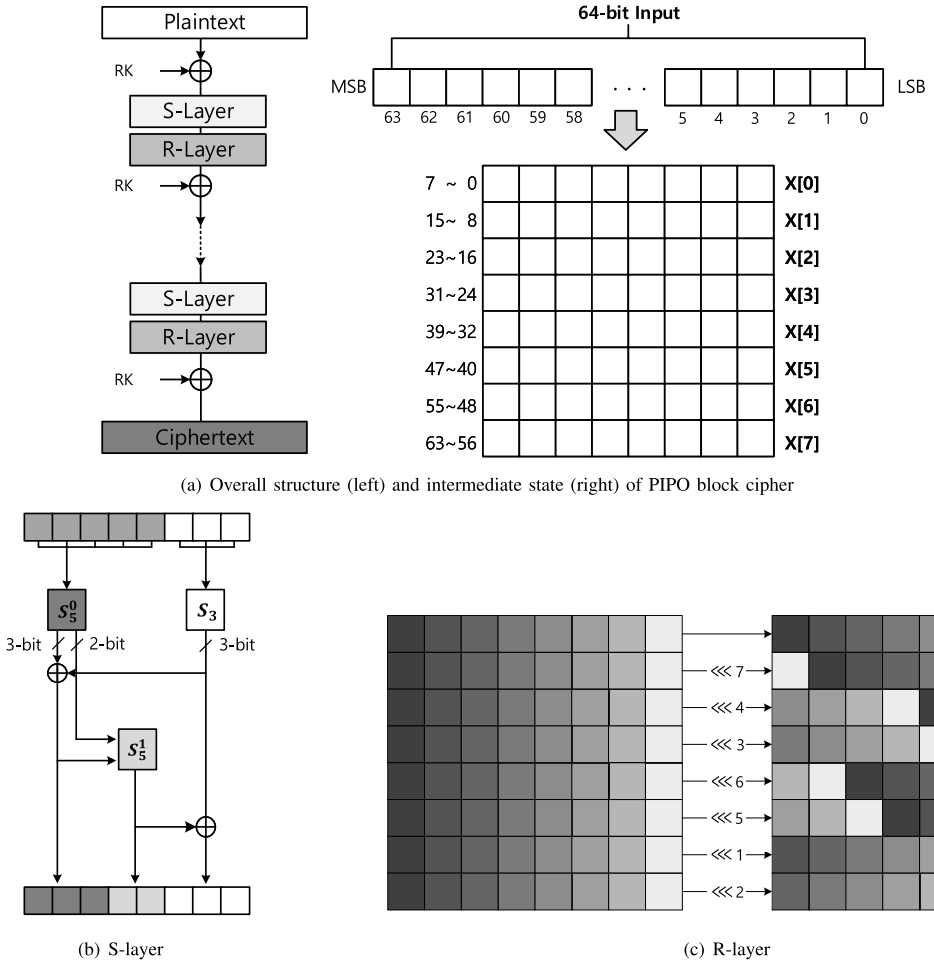
The rest of this article is structured as follows: Section II introduces block cipher PIPO and our target platforms. Section III reviews implementations of cryptographic algorithms on RISC-V and ARM Cortex-M series. An introduction into our main idea for implementing PIPO is provided in Section IV. Results of our implementations are presented in Section V, before we conclude the article in Section VI.

## II. PRELIMINARY

In this section, we describe in detail of essential to the implementation of PIPO block cipher and discuss what should be considered.

### A. PIPO: LIGHTWEIGHT BLOCK CIPHER

Lightweight ciphers that have been proposed so far, either implementation-friendly only in the SW environment [11], [12], [13], [14], [15], or friendly only in the HW environment existed [2], [16], [17], [18]. Therefore, when designing the existing lightweight cipher, the side-channel attack on point of view was not a major consideration; however, the importance of mounting countermeasures which against side-channel attacks in the IoT environment is increasing due to various side-channel attacks studies. The PIPO block cipher presented in ICISC'20 is a lightweight cipher that is friendly to SW/HW implementation and countermeasures for side-channel attacks [1]. The S-Layer is configured via an Unbalanced-Bridge structure based on a few bit-operations, which has the advantage of having some rounds compared to other lightweight block ciphers. In ICISC'20, the




(a) Overall structure (left) and intermediate state (right) of PIPO block cipher

(b) S-layer

(c) R-layer

**FIGURE 1. Overview of PIPO block cipher [1].**

performance evaluation was conducted in an 8-bit AVR environment, which is known as the most constrained embedded device, and proved its superiority. However, since the performance of the actual cryptography algorithm differs depending on the board of devices, performance evaluation in the most used 32 bits-based devices should be considered in the future. The overview structure of PIPO block cipher is shown in Figure 1(a). A round of PIPO consists of S-Layer, R-Layer, and AddRoundKey processes. The 64-bit plaintext is stored sequentially in an array when implemented, similarly to a general lightweight cipher. PIPO uses 128-bit and 256-bit keys and it performs 13 and 17 rounds of encryption, respectively. Unlike other SPN-based block ciphers, the key schedule process consists of a simple structure in which the value of the master key itself is maintained. The pseudo-code for the detailed key schedule of the PIPO block cipher is shown in Algorithm 1. The 64-bit plaintext is sequentially placed in `X[0]` to `X[7]` of 8-bit unit in the reference software. Since PIPO's S-Layer is composed as shown in Figure 1(b), it performs the substitution process with bit operations through `X[i]`, where $i \in [0, 7]$. If it is not implemented in a bit-slice fashion, the substitution process must be implemented in such a way that the bit-order is rearranged in the array and referenced to the S-Box. PIPO's R-Layer is configured as shown in Figure 1(c) and performs a simple rotate-shift operation. In the reference implementation, it is implemented through macros.

Table 1 shows the performance results of PIPO implementation, experimented in ICISC'20, in the 8-bit AVR environment. In ICISC'20, performance evaluation was made based on the RANK system introduced in [19]. The measurement of RANK is as follows:

$$RANK = (10^6/CPB)/(ROM + 2 \times RAM) \quad (1)$$

In the case of the 8-bit Atiny platform, there are devices that support byte unit Flash memory and SRAM; therefore, it is reasonable to evaluate the performance of the software through the RANK system in the AVR environment. In evaluation, PIPO has achieved a high RANK compared to the existing lightweight block cipher. However, in relation to the most popular 32-bit or higher devices, since SRAM and Flash memory in KB units is used, the code size point of view may not be important except for specific applications; therefore, in order to mount PIPO block cipher on a 32-bit processor, achieving high speed through optimization methodology will be the most important consideration. In Section IV,

**TABLE 1.** Performance comparison of 64-bit block ciphers on 8-bit AVR* [1].

| security | Block cipher | Code size (bytes) | RAM usage (bytes) | Execution time (cycles per byte) | RANK [19] |
|---|---|---|---|---|---|
| 128-bit | PIPO [1] | 320 | 31 | 197 | **13.31** |
| | SIMON [19] | 290 | 24 | 253 | 11.69 |
| | RoadRunneR [20] | 196 | 24 | 477 | 8.59 |
| | RECTANGLE [21] | 466 | 204 | 403 | 2.84 |
| | PRIDE [21] | 650 | 47 | 969 | 1.39 |
| | SKINNY [21] | 502 | 187 | 877 | 1.30 |
| | PRESENT [21] | 660 | 280 | 1,349 | 0.61 |
| | CRAFT [22] | 894 | 243 | 1,504 | 0.48 |
| 256-bit | PIPO [1] | 320 | 47 | 253 | **9.54** |

*The code size represents ROM, and RAM metric includes STACK

we introduce strategies to accelerate the speed of PIPO from a parallel perspective, and we present detailed implementation strategies for each 32-bit platform.

### B. RISC-V

RISC-V is Reduced Instruction Set Computer (RISC)-based an open-source standard Instruction Set Architecture (ISA) that is freely available to all developers. The basic RISC-V's ISAs are RV32I for 32-bit and RV64I for 64-bit, which consist of minimal instructions including bit-based arithmetic/logical/shift operations and memory access to RAM. With the investment of various companies, the basic ISA of the 8/16/128-bit unit is currently under development. Also, RISC-V has a various tool-chain for developers. Unlike ARM processors, RISC-V has a different cycle per instructions for each CPU design, therefore, the implementation results vary by board. Optimization research of cryptographic algorithms is mainly conducted on the E31 RISC-V core; therefore, we choose E31 core as our target platform. 32-bit RISC-V has 32 general-purpose registers including program counter and stack-pointer. Unlike the ARM Cortex-M4, all but one register can be assigned data. Since the basic ISA does not have an instruction to specify a state flag, there is a drawback that the radix-representation should be used in the specific algorithm, differently than ARM processors. Also, RISC-V does not support barrel-shifter, all implementations are represented as a simple list of instructions. However, research is underway on the BitManip extension [23] for logical/rotation and the cryptographic instruction set extension [24] for RISC-V, and a vector ISA set [25] for SIMD instructions is currently under development. These studies will facilitate research on the implementation of cryptographic algorithms in RISC-V in the future.

In this article, we implement PIPO block cipher using only the basic ISA, without any extended instruction set. Compared to ARM processors, RISC-V has about twice as many general-purpose registers, it is easy to store temporary variables. Therefore, for the optimization methodology of PIPO block cipher on RISC-V, it is important to make the best use of basic ISA and general-purpose registers to establish forced parallelization.

**Algorithm 1** Key Scheduling of PIPO Block Cipher [1]

**Require:** $K^{128}$ or $K^{256}$
**Ensure:** RoundKey $RK_i$ where $i = 0$ to 13 or 17
1: **if** $K^{128}$ **then**
2: $K^{128} = K_1^{64}||K_0^{64}$
3: **for** $i = 0$ to 13 **do**
4: $RK_i = K_{i\ mod\ 2}^{64}$
5: **end for**
6: **else if** $K^{256}$ **then**
7: $K^{256} = K_3^{64}||K_2^{64}||K_1^{64}||K_0^{64}$
8: **for** $i = 0$ to 17 **do**
9: $RK_i = K_{i\ mod\ 4}^{64}$
10: **end for**
11: **end if**

### C. ARM-CORTEX M4

ARM Cortex-M family is the most popular 32-bit platform. So far, various bench-marking and optimization research for cryptographic-algorithm are being performed on the Cortex-M series. In particular, Cortex-M4 is a target device for the performance evaluation of algorithms submitted to Light Weight Cryptography (LWC) and Post-Quantum Cryptography (PQC) competitions held by NIST [7], [26]. The target device of this article, the Cortex M4, consists of 16 32-bit general-purpose registers. Among them, 14 registers can be used in actual implementation except for the two registers corresponding to the program counter and stack pointer. On the Cortex-M4, Bit-wise and arithmetic instructions require a single cycle, but memory access instructions require 2 cycles. Like RISC-V, all 32-bit data cannot be used for address reference, but compared to RISC-V, flexible indexing is possible when registers are used as addresses. The SIMD instruction can be used for a specific 8/16-bit unit, but it is not a consideration in the case of block ciphers and hash functions in which bit-wise shift instructions are mainly used. Similarly, the optional flag instructions are not used in this article. The most unique feature of the ARM processor is the barrel-shifter, which can perform bit-wise shift and rotate operations for almost any instruction at no additional cost.

Since the S-Layer of PIPO is designed for the bit-slice friendly, PIPO has a different slicing way from general block ciphers such as AES [6]; so, it is desirable not to apply the ARM-specific instruction scheduler presented in [27]. Therefore, it is necessary to devise a strategy to use as many barrel-shifter instructions as possible and efficiently assign registers to implement PIPO block cipher on Cortex-M4.

## III. RELATED WORK

In this section, we investigate the implementation results of PIPO block cipher in all environments. Also, we check the latest implementation results of block ciphers in our target platform, 32-bit devices, and discuss considerations for our optimization strategy.

### A. IMPLEMENTATION OF PIPO

Due to the advantages of the Unbalanced-Bridge structure-based S-Layer, various researches on countermeasures for side-channel attacks have been mainly conducted so far. [3] proposed the implementation of masked PIPO block cipher on an 8-bit AVR environment. By proposing a non-linear masked S-Layer with a minimum of arithmetic instructions, a side-channel attack countermeasure was built and its efficiency was proved. Since the masking-based implementation is a countermeasure to power analysis, designing a response to fault attacks is another study. In [28], the first attempt at fault attack on PIPO block cipher was conducted. authors emphasized the importance of designing fault attack countermeasures, along with research that accurate key recovery is possible with a 98% probability through differential fault attacks. Accordingly, research on fault attack countermeasures of PIPO block cipher was conducted in the ARMv8 Cortex-A series environment. [4] presented a random-shuffling-based countermeasure for fault attack, and also proposed parallel optimization implementation using the NEON engine. Research on PIPO implementation is not actively conducted in environments that do not support SIMD instructions such as the NEON engine.

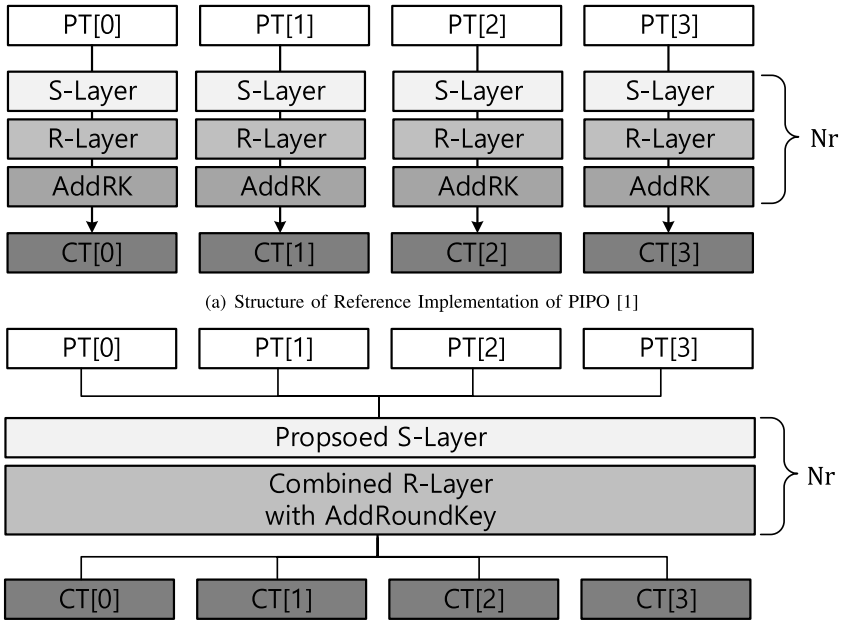
As far as we know, the only implementation research of PIPO block cipher is a bench-mark study published in [29]. Bench-marking was done in a RISC-V environment and ported as a pure assembly without adopting a parallelization strategy. As a result, PIPO implementation has significantly reduced performance in terms of speed and code size compared to general block ciphers such as AES and LEA. Unfortunately, the performance benefits of PIPO block cipher seen in Table 1 are limited to only 8-bit AVR environments. This fact ultimately suggests the limitations of implementing a simple ported PIPO block cipher on 32-bit devices. There is another study [30] on the implementation of PIPO block cipher, but it was conducted in a quantum-computing environment, so it is not of our interest.

### B. BLOCK CIPHER ON RISC-V

The study of the implementation of a block cipher on RISC-V has mainly moved toward designing an extended instruction set by the HW environment. Except for various extension sets introduced in Section II-B, [31] designed extension sets for the ARIA block cipher and [32] proposed a cryptographic extension set for LWC. However, since these are a study of finite-field operations of ARIA and instruction sets for $4 \times 4$ S-Box of LWC, it has less relevance to do with PIPO block cipher. Research on the SW implementation of block cipher using the basic ISA has also been actively conducted. In 2019, when RISC-V began to become popular, benchmark studies of AES, ChaCha20, and keccak were conducted using RV32I [33]. As a result of comparison with the implementation with Cortex-M4, RISC-V has about twice as general-purpose registers as Cortex-M4, but the efficiency of barrel-shifter is higher in terms of implementation speed for block cipher and hash function; therefore, It proved that RISC-V difficult to catch up Cortex-M4, using only 32RVI. In [34], an implementation study was conducted on the algorithm for LWC competition using RV32I. In their research, 32RVI and assembly characteristics were mainly used, not the algorithm itself. The sublation of branch instructions, loop-unrolling, and interleaved methods to prevent pipeline-stalled have been recombined to match the RV32I model. The work focused on property of block cipher is the Fixslicing work done on RISC-V and Cortex-M4. They applied Fixslicing to GIFT and SKINNY [7] and finally expanded it to AES [6]. The implication of these studies is that when only RV32I is used without barrel-shifter, the optimization strategy for block cipher should be designed to reduce memory access using as many general-purpose registers as possible.

### C. BLOCK CIPHER ON CORTEX-M4

The representative runner of the conventional Cortex M series is Cortex-M3, and the researches of block cipher has also progressed a lot from Cortex-M3 [35], [36], [37]. Cortex-M4 has a very similar core to Cortex-M3, but it supports Floating Point Unit (FPU) instructions and single-cycle multiplication instructions, making it particularly efficient for implementing public-key-based cryptosystems [38]. However, as in the LWC competition held by NIST, SW/HW implementation research for performance evaluation and optimization of block cipher in Cortex-M4 is being actively conducted. In this article, we can't cover all of the vast amount of research ever conducted; therefore, we investigate a few core SW implementations on Cortex-M4. While RISC-V has not made progress in toolchain and compiler research, in Cortex-M series, tools for instruction scheduling and register allocation was proposed in [27]. These tools performed better than commercial gcc and clang, demonstrating their efficiency through the implementation of AES. In 2020, optimization study of HIGH block cipher was conducted in [5]. Since, HIGHT uses 8-bit words like PIPO block cipher, 4 words in a 32-bit register can be processed simultaneously; however, HIGHT is an Add-Rotate-XOR (ARX)-based block cipher, it differs from PIPO in that SIMD instructions can be applied. In [8], an AES implementation study was conducted. It implemented AES and CounTeR (CTR) Mode by




(a) Structure of Reference Implementation of PIPO [1]

(b) Structure of Proposed Parallel PIPO Implementation

**FIGURE 2.** The structure of the proposed PIPO implementation.

using the bit-slice technique. Later, GIFT, SKINNY, and AES using Fixslicing techniques, mentioned in Section III-B, were also implemented in Cortex-M4 [6], [7]. Since PIPO block cipher is a Substitution-Permutation Network (SPN)-based block cipher, it is not possible to apply SIMD instructions, because no arithmetic unit instructions are used to implement the S-Box. Therefore, it is important to use barrel-shifter efficiently to implement parallelized PIPO block cipher such as HIGHT.

## IV. MAIN IDEA

In this section, we consider all optimization techniques to implement PIPO block cipher in a 32-bit environment. First, we compare the two implementation method of PIPO block cipher presented in [1]; and, analyze the reason why the exquisite Fixslicing technique, which does not follow the rules of classical block cipher, cannot be applied to PIPO block cipher, unfortunately. Finally, we present three strategies suitable for 32-bit devices and discuss implementation methodologies in Cortex-M4 and RISC-V platforms. Figure 2 gives a brief overview of our optimization methodology. To encrypt 4 plaintexts in 32-bit platforms, 4 independent PIPO block ciphers should be called in the case of the reference implementation. Our software loads 4 plaintexts simultaneously for 32-bit platforms. The loaded plaintext is aligned in general-purpose registers and the optimized S-Layer, R-Layer, and AddRoundKey process are performed.

### A. FORCED PARALLELIZATION STRATEGY

According to [1], there are two implementation methods of PIPO block cipher. The first method is to sequentially store the plaintext in `X[i]`, where `i` $\in [0, 7]$ and it processes S-Layer using bitslicing technique. In this method, the most cost occurs on the S-Layer. Another method is to not follow bitslicing and use S-Box table. In order to use S-box, bits should be rearranged to enable memory access using `SWAPMOVE` technique for plaintext stored in the forward direction [39]; however, since R-Layer performs rotate-shift operations on forward array, the array should be rearranged forward after referring S-box. Therefore, in the reference code, the forward bitslicing implementation method shows much better speed than s-box method which occurs the cost of two `SWAPMOVE` for every round of the PIPO block cipher.

We examined whether Fixslicing technique introduced in [7] can be applied to PIPO block cipher. If in the S-box-based implementation, the bits are rearranged to the position of the next S-box bits instead of in the forward direction after memory reference and it uses fewer instructions than bitslicing method in each round, it can be effective enough. In other words, our goal was to eliminate the `SWAPMOVE` used in every round of PIPO block cipher. However, R-Layer consists of a bit rotate-shift operation. Unlike AES and GIFT block ciphers, there is no rule between each unit block in PIPO block cipher. In addition, when using 4 plaintexts in a 32-bit environment, the size of the S-box also increases by multiples, so it is inefficient to use this method. After all, S-Layer of the PIPO block cipher is configured based on the Unbalanced-Bridge structure, unlike GIFT and bitslicing AES, which require a realignment in the initial round; therefore, it is difficult to apply Fixslicing technique.

Therefore, we choose the forward bitslicing implementation and strategize for the forced parallel implementation. Figure 3 shows register scheduling for a 32-bit platform. In $pt_j^i$, $j$ is the index of the plaintext and $i$ is the index of the bit of the $j$-th plaintext. For bitslicing-based S-Layer, the four plaintexts have aligned 8-bits units in each register.

| | *Plaintext1* | | | *Plaintext2* | | | *Plaintext3* | | | *Plaintext4* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_0$ | $pt_1^0$ | ... | $pt_1^7$ | $pt_2^0$ | ... | $pt_2^7$ | $pt_3^0$ | ... | $pt_3^7$ | $pt_4^0$ | ... | $pt_4^7$ |
| $R_1$ | $pt_1^8$ | ... | $pt_1^{15}$ | $pt_2^8$ | ... | $pt_2^{15}$ | $pt_3^8$ | ... | $pt_3^{15}$ | $pt_4^8$ | ... | $pt_4^{15}$ |
| $R_2$ | $pt_1^{16}$ | ... | $pt_1^{23}$ | $pt_2^{16}$ | ... | $pt_2^{23}$ | $pt_3^{16}$ | ... | $pt_3^{23}$ | $pt_4^{16}$ | ... | $pt_4^{23}$ |
| $R_3$ | $pt_1^{24}$ | ... | $pt_1^{31}$ | $pt_2^{24}$ | ... | $pt_2^{31}$ | $pt_3^{24}$ | ... | $pt_3^{31}$ | $pt_4^{24}$ | ... | $pt_4^{31}$ |
| $R_4$ | $pt_1^{32}$ | ... | $pt_1^{39}$ | $pt_2^{32}$ | ... | $pt_2^{39}$ | $pt_3^{32}$ | ... | $pt_3^{39}$ | $pt_4^{32}$ | ... | $pt_4^{39}$ |
| $R_5$ | $pt_1^{40}$ | ... | $pt_1^{47}$ | $pt_2^{40}$ | ... | $pt_2^{47}$ | $pt_3^{40}$ | ... | $pt_3^{47}$ | $pt_4^{40}$ | ... | $pt_4^{47}$ |
| $R_6$ | $pt_1^{48}$ | ... | $pt_1^{55}$ | $pt_2^{48}$ | ... | $pt_2^{55}$ | $pt_3^{48}$ | ... | $pt_3^{55}$ | $pt_4^{48}$ | ... | $pt_4^{55}$ |
| $R_7$ | $pt_1^{56}$ | ... | $pt_1^{63}$ | $pt_2^{56}$ | ... | $pt_2^{63}$ | $pt_3^{56}$ | ... | $pt_3^{63}$ | $pt_4^{56}$ | ... | $pt_4^{63}$ |

**FIGURE 3.** Register scheduling of PIPO block cipher on 32-bit platforms [10].

With this, an S-Layer for 4 plaintexts can be simultaneously implemented through bitslicing. Since LoaD/STore-based instructions of Cortex-M4 do not support a barrel-shifter, on Cortex-M4 and RISC-V platforms, plaintext loading is implemented through simple bit-wise operations and bit-shift instructions.

### B. BIT-MASKING TECHNIQUE FOR R-LAYER

32-bit general-purpose registers of both RISC-V and Cortex-M4 do not support vectorization; therefore, unfortunately, rotate-shift instructions for byte/half-word/word units cannot be used. Our goal is to rotate-shift plaintext in a 32-bit register divided 4 unit with minimal use of instructions. Therefore, we use a bit-masking technique for rotate-shift for 4 8-bit units. Listing 1 shows an implementation examples of R-Layer in Cortex-M4 and RISC-V. For example, for 7-bit rotate-shift of `X[1]` in Cortex-M4, we use `0 × 01010101` and `0xFEFEFEFE` as masking value to extract 1-bit and 7-bit for each unit. Using the fact that 7-bit rotate-shift-left is the same as 1-bit rotate-shift-right, 1-bit logical-shift-right is performed on `X[1]`, the masked value is shifted and XOR(eXclusive OR)ed. Since Cortex-M4 has an advantage of the powerful barrel-shifter technology, we apply it when XORing the masked value in the last step (line 10 of Listing 1). Using a barrel-shifter can achieve the same single cycle (1 clock) as using a simple `ORR` instruction.

In the case of RISC-V, the maximum value for the immediate value of bit-operation instructions (`ANDI`, `XORI`, `ORI`) is 12-bit due to the characteristic of the instruction format of RISC-V [40]. Therefore, it is not possible to mask 32-bit data using single-bit instructions like the Cortex-M4 (like a line 7-8 of Listing 1). To solve this, a commonly used method is to use the `LI` instruction to immediate-load the masking values, for 7-bit rotate-shift, `0 × 01010101` and `0xFEFEFEFE`. In this case, 28 (2(`LI`) × 2(masking) × 7(PT)) clock cycles occur per round to load the masking value whenever R-Layer operation is performed once. In other words, an additional 364 (28 × 13) clock cycles are required during the 13 rounds of the PIPO-64/128. Therefor, we propose a masking table of 32 bytes for optimization of R-Layer. Listing 1 has an example of our R-Layer implementation for RISC-V. We observe that the XOR value of two 32-bit data pairs required for masking is `0xFFFFFFFF`. For example, we XOR-operate `0 × 01010101` and `0xFFFFFFFF` to produce `0xFEFEFEFE`, which is the masking pair of `0 × 01010101`. `0xFFFFFFFF` data can only be loaded once at the beginning of the R-Layer. By using this technique, it costs 18 (2(`LW`) × 1(masking) × 7(PT) + 2(`LA`) + 2(`LW`)) clock cycle per round, 13 rounds of for PIPO-64/128 It costs 234 (18 × 13) clock cycles. This means this method can achieve a bit-masking more efficiently than another method using `LI` instructions.

### C. COMBINED ADDROUNDKEY

In the case of Cortex-M4, a powerful technology called barrel-shifter exists, but unfortunately, the RISC-V platform has fewer effective instructions for block ciphers implementation than Cortex-M4 platform. However, as introduced in Section II-B, RISC-V device has twice as many general-purpose registers as Cortex-M4; therefore, we focused on how to make the most of this characteristic. In this section, we propose a methodology to reduce the memory access cost by holding the master key in registers of RISC-V environments. Detailed register scheduling of RISC-V platform is shown in Figure 4. The shaded areas of Figure 4 are registers that are not used by the actual implementation. Our implementation divides and stores the 128-bit master key in 4 of the 32 general-purpose registers of RISC-V (`x14` to `x17`). If PIPO block cipher use a 256-bit security level, we can still store the entire 256-bit master key by additionally using `x2`, `x7`, `x12`, and `x13` registers. Since the master key is stored in several registers, there is no need to load the master key every round. The only cost in AddRoundKey process is to unpack and expand the master key for forced parallelism. Of course, there is another method to optimize AddRoundKey process via simple loading by operating all round keys in advance, but since it needs to be extended to 32-bit and stored, an additional memory space

```
// Input : X[i], where i in [0, 7]
// Input : T[j], where j in [0, 3]
// Input : A[0] (address)

//*Cortex-M4*//
// 7-bit left rotate-shift
AND T[0], X[1], #0x01010101
AND X[1], X[1], #0xFEFEFEFE
LSR X[1], X[1], #1
ORR X[1], X[1], T[0], LSL #7

//*RISC-V*//
MASKING_TABLE :
    .word 0x01010101, 0x0F0F0F0F
    .word 0x1F1F1F1F, 0x03030303
    .word 0x07070707, 0x7F7F7F7F
    .word 0x3F3F3F3F, 0xFFFFFFFF

// Set Address
LA A[0], MASKING_TABLE
LW T[3], 28(A[0]) // 0xFFFFFFFF

// 7-bit left rotate-shift
LW T[0], A[0]
AND T[1], X[1], T[0]
SLLI T[1], T[1], 7
XOR T[2], T[0], T[3]
AND T[2], X[1], T[2]
SRLI T[2], T[2], 1
OR X[1], T[1], T[2]
```

**Listing. 1.** R-layer of Cortex-M4 and RISC-V.

of at least 64 bytes is required based on 128-bit security level. Therefore, if all round keys are stored in RAM or Flash memory, 832/1088 bytes are required based on 128/256-bit security level. Since the PIPO block cipher is a lightweight cipher, RAM should be used to a minimum according to its characteristics. The method of storing the round key in Flash memory can cause other security issues; therefore, this method is not very effective except for devices with completely physical security countermeasures applied. Also, it should be considered that encrypting small-sized plaintexts as reported in [6]. Therefore, pre-computing all round keys may not be a flexible solution in all application situations. Hence, we believe that the best implementation way of the PIPO block cipher in RISC-V is to hold the master key in a registers, unpack it, and apply it to forced parallelism. Our methodology can accelerate performance while preserving parallelism without additional memory usage. Finally, AddRoundKey process is merged with R-Layer process.

## V. RESULT

This section shows our software results on the RISC-V and Cortex-M4. We first describe our target platforms and setup and then present the evaluation in Section V-B and Section V-C. Section V-D analyzes PIPO block cipher performance in RISC-V and Cortex-M4 platforms. We have made every effort to ensure that performance measurements are performed as fair and accurate as possible. Our implementation is available in two versions. The first implementation (marked by †) is the software of encrypting one plaintext, and the reference implementation methodology is ported to software as a handwritten assembly suitable for RISC-V and Cortex-M4 environments. The second implementation (marked by ‡) is the code to which the methodology we propose is applied, and it is a software that encrypts 4 plaintexts together with the assembly instructions in parallel. We evaluate our performance based on the performance improvement of the second software. The reference implementation of PIPO block cipher presented in [1] was measured by compiling on Cortex-M4 and RISC-V, respectively.

### A. SETUP TARGET DEVICE

#### 1) RISC-V: HiFiveRevB

Our RISC-V target platform is the HiFiveRevB board containing a SiFive's 32-bit RISC-V processor with 16 kB of SRAM, 32 MB of flash, and 320 MHz of frequency. The integrated development environment is FreedomStudio (ver. 2020) of SiFive, our work is handwritten assembly PIPO block cipher code. For building code, we use the gcc (ver. 10.1.0) with -O3 option. Benchmarking was conducted in the same method as in the RISC-V platforms. For benchmarking, 32 bytes of plaintext are encrypted 10,000 times and averaged. To ensure that no instructions and data remain in cache memory, a dummy operation is performed after one PIPO block cipher is executed. Dummy operations are not counted in the average of clock cycles. The process of loading and aligning the 4 plaintexts and the process of rearranging and writing out the ciphertext aligned in the general-purpose registers are also counted by clock cycles.

#### 2) CORTEX-M4: STM32F407-DISOVERY

We target the STM32F407-DISOVERY board, featuring a STM32F407VG Cortex-M4 microcontroller with 196 kB of SRAM, 1 MB of flash, and 168 MHz of frequency. The integrated development environment is Keil-uVision5, our work is handwritten assembly PIPO block cipher code. For building code, we use the Keil ARM Compiler with -O3 option. Benchmarking was conducted in the same method as in the RISC-V platforms.

### B. RESULT OF RISC-V

We evaluate our optimization strategy in a RISC-V environment through the set HiFiveRevB board. Table 2 shows the performance of the implementation of various cryptographic algorithms and our software in a RISC-V platform. Unfortunately, research on lightweight block ciphers of the RISC-V environment is relatively small compared to Cortex-M3/M4 platforms; therefore, we investigate all the latest cryptographic software implemented in RISC-V platforms. Fix-GIFT and Fix-AES are implementations to which Fixslicing technology proposed in [7] and [6] is applied.

| *RISC-V* | | | | *Cortex-M4* | |
|---|---|---|---|---|---|
| *Register* | *Used for* | *Register* | *Used for* | *Register* | *Used for* |
| x0 | *Hard-wired zero* | x16 | $mk^{64\sim95}$ | x0 | *pt address* |
| x1 | *Return address* | x17 | $mk^{96\sim127}$ | x1 | *mk address* |
| x2 | *Stack pointer* | x18 | $pt_i^{0\sim7}, i \in [0,3]$ | x2 | $pt_i^{0\sim7}, i \in [0,3]$ |
| x3 | *Global pointer* | x19 | $pt_i^{8\sim15}, i \in [0,3]$ | x3 | $pt_i^{8\sim15}, i \in [0,3]$ |
| x4 | *Thread pointer* | x20 | $pt_i^{16\sim23}, i \in [0,3]$ | x4 | $pt_i^{16\sim23}, i \in [0,3]$ |
| x5 | 0x08 *(shift temp)* | x21 | $pt_i^{24\sim31}, i \in [0,3]$ | x5 | $pt_i^{24\sim31}, i \in [0,3]$ |
| x6 | 0xFF *(mask temp)* | x22 | $pt_i^{32\sim39}, i \in [0,3]$ | x6 | $pt_i^{32\sim39}, i \in [0,3]$ |
| x7 | - | x23 | $pt_i^{40\sim47}, i \in [0,3]$ | x7 | $pt_i^{40\sim47}, i \in [0,3]$ |
| x8 | *Frame pointer* | x24 | $pt_i^{48\sim55}, i \in [0,3]$ | x8 | $pt_i^{48\sim55}, i \in [0,3]$ |
| x9 | *Saved register* | x25 | $pt_i^{56\sim63}, i \in [0,3]$ | x9 | $pt_i^{56\sim63}, i \in [0,3]$ |
| x10 | *pt address* | x26 | 0x01 *(loop + 1)* | x10 | *Temp register* |
| x11 | *mk address* | x27 | *Rcon (for round #)* | x11 | *Temp register* |
| x12 | - | x28 | *Temp register* | x12 | *Temp register* |
| x13 | - | x29 | *Temp register* | x13 | *Stack pointer* |
| x14 | $mk^{0\sim31}$ | x30 | *Temp register* | x14 | *Link register* |
| x15 | $mk^{32\sim63}$ | x31 | *Temp register* | x15 | *Program counter* |

**FIGURE 4.** Details of register scheduling on RISC-V and cortex-M4.

The implementation of Fix-GIFT in RISC-V was done in [41]. bench-ARIA and bench-LEA are benchmark reports that are simply ported from [29] using assembly instructions of RISC-V. We also compare the various cryptographic functions in the RISC-V platform reported in [33]. Compared to the reference implementation of PIPO block cipher, our PIPO$^{\dagger}$ software without parallelization strategy achieved 260 CPB (a performance improvement of about 50 %). PIPO$^{\dagger}$, applied the forced parallelization strategy, achieved 121 CPB (approximately 229 % performance improvement). PIPO$^{\dagger}$ software requires a higher cost than Fix-GIFT implementation, while PIPO$^{\dagger}$ software requires a lower CPB than Fix-GIFT due to our parallelization strategy. Unfortunately, we report that our code shows lower performance than LEA and AES in RISC-V. The implementation of PIPO$^{\dagger}$ with our optimization strategy shows that PIPO in the RISC-V environment is superior to the software of GIFT and ARIA, purely in terms of performance.

### C. RESULT OF ARM CORTEX-M4

As in the previous section, we evaluate our software through the set STM32F407-DISCOVERY board. Table 3 shows the performance of the implementation of various cryptographic algorithms and our software in a Cortex-M4. Lightweight and general block cipher implementations were evenly implemented in Cortex-M3 and M4. We evaluate software based on the performance of implementation reports for the Cortex-M3 and M4 platforms. The biggest difference between Cortex-M3 and Cortex-M4 is the multiplication instruction and the Floating-Point Unit instruction, so it does not significantly affect the performance evaluation of the actual block cipher implementation. For that reason, we refer to [6], Fix-AES performance report, which shows the fact that the performance did not change significantly in the Cortex-M3 and Cortex-M4 environments. Compared to the reference implementation of PIPO block cipher, our PIPO$^{\dagger}$ software without parallelization strategy achieved 171 CPB (a performance improvement of about 81 %). PIPO$^{\ddagger}$, applied the forced parallelization strategy, achieved 66 CPB (approximately 370 % performance improvement). Comparing other lightweight block ciphers with our PIPO software, PIPO$^{\ddagger}$ achieves better performance than the software of Fix-GIFT, PRESENT, RECTANGLE, and SIMON. For HIGHT software presented in [5], it achieves the 56 CPB during the encryption process, but, for key scheduling process, it requires an additional 49 CPB. Since our software includes the key scheduling inside the encryption process, we can show that our PIPO$^{\ddagger}$ case achieved faster results than HIGHT implementation, considering all the clocks required for the actual encryption process. In addition, our implementation achieves lower RAM usage and higher performance than general block ciphers with 128-bit plaintext lengths such as Fix-AES and ARIA. Unfortunately, similar to what was reported in the previous section, PIPO block cipher also

**TABLE 2.** Constant-time implementation results on RISC-V for various versions of PIPO block cipher, as well as other block ciphers. For encryption routines, speed is expressed in Cycles Per Bytes*.

| security | Block cipher | PlainText size (bits) | Code size (bytes) | RAM usage (bytes) | Execution time (cycles per byte) |
|---|---|---|---|---|---|
| 128-bit | PIPO [1] | 64 | 3,288 | 31 | **392** |
| | Fix-GIFT [41] | 64 | 762 | 176 | 197 |
| | bench-LEA [29] | 128 | 2,432 | 32 | 46 |
| | bench-ARIA [29] | 128 | 8,192 | 32 | 291 |
| | Fix-AES [6] | 128 | 10,652 | 72 | 87 |
| | Table-AES [33] | 128 | 4,352 | - | 57 |
| | ChaCha20 enc [33] | 512 | 12,960 | - | 28 |
| | Keccack 1600 [33] | 1600 | 7,936 | - | 69 |
| 128-bit | (our work) PIPO† | 64 | 10,816 | 24 | 260 |
| | (our work) PIPO‡ | 64 | 12,352 | 48 | **121** |

**TABLE 3.** Constant-time implementation results on cortex-M4 for various versions of PIPO block cipher, as well as other block ciphers. For encryption routines, speed is expressed in Cycles Per Bytes*.

| Platform with (128 security) | Block cipher | PlainText size (bits) | Code size (bytes) | RAM usage (bytes) | Execution time (cycles per byte) |
|---|---|---|---|---|---|
| Cortex-M4 | PIPO [1] | 64 | 3,288 | 31 | **310** |
| | Fix-GIFT [7] | 64 | 762 | 176 | 197 |
| | HIGHT [5] | 64 | 592 | 188 | 56 |
| | PRESENT [42] | 64 | 2,476 | - | 100 |
| | Fix-AES [6] | 128 | 9,184 | 112 | 80 |
| | Fix-SKINNY [6] | 128 | 1,620 | 60 | 117 |
| | Table-AES [33] | 128 | 3,613 | - | 40 |
| | ChaCha20 enc [33] | 512 | 10,112 | - | 14 |
| Cortex-M3 | Fix-AES [6] | 128 | 9,184 | 112 | 79 |
| | RECTANGLE [43] | 64 | 800 | 76 | 107 |
| | SIMON [43] | 64 | 456 | 48 | 81 |
| | SPECK [43] | 64 | 628 | 36 | 24 |
| | LEA [37] | 128 | 1,532 | 228 | 34 |
| | ARIA [9] | 128 | 10,636 | 224 | 160 |
| Cortex-M4 | (our work) PIPO† | 64 | 8,348 | 24 | 171 |
| | (our work) PIPO‡ | 64 | 10,752 | 48 | **66** |

*The code size represents ROM, and RAM metric includes STACK

† single plaintext encryption includes key schedule process

‡ 4 plaintext encryption includes key schedule process

shows lower performance than the LEA in the Cortex-M4 environment.

### D. PERFORMANCE ANALYSIS

To achieve the forced parallelization methodology, the linear and non-linear layer operations for four plaintexts should theoretically be reduced by a factor of four. In the case of S-Layer, it is natural that the performance improvement is 4 times compared to the single plaintext implementation; however, the costs of loading bit masking in R-Layer and expanding the round key kept in the register are required additional clock cycles. Our ideal goal is best speed performance based on minimal RAM usage. As discussed in Section II-A, code size, measured to Flash memory, is not a major limit point for 32-bit environments. Our implementation is fully unrolled to achieve top speed. Also, as reported in [6], Since different keys may be used based on a small size of plaintext depending on the application situation, we do not pre-compute all round keys, using minimal RAM usage. Our choice makes our software flexible in that it can be used fluidly for all block cipher operation modes and all platforms and application situations. The additional overhead incurred in the process of aligning plaintext, loading bit-masking value, and expanding round keys for forced parallelization strategy is offset to an extent by architectural characteristics. In the case of RISC-V, the process of loading the round key for each round of PIPO block cipher is omitted by holding the key in the general-purpose register. In the case of Cortex-M4, By using a barrel-shifter, the clocks of bit-masking and round key processes are compactly

compressed. Finally, we report that our PIPO software generally achieved higher performance in RISC-V and Cortex-M4 platforms than other lightweight and common block ciphers except for LEA block cipher.

However, our PIPO‡ software still requires a higher CPB than the implementation of the LEA block cipher in both RISC-V and Cortex-M4. Block cipher LEA is an ARX-based block cipher consisting of bit and arithmetic operations; and, its optimization method used in 32-bit units has been proposed. The 32-bit LEA implementation has the advantage that bit and arithmetic operations do not need to consider the bit-masking technique for rotate-shifts and carry for module addition. Therefore, in 32-bit devices, computing plaintext packed in 32-bit is usually a more efficient implementation strategy for LEA block cipher than parallelization tasks. To determine if the performance of the LEA block cipher depend on the 32-bit platform, we also research LEA implementations in the AVR environment. The latest optimized LEA implementation results in an 8-bit AVR platforms were presented in [44], and performance was reported at 167 CPB. As can be seen in Table 1, the reported PIPO reference Implementation is 197 CPB, which is basically lower than the LEA block cipher. Therefore, it can be considered algorithm-dependent results that our PIPO implementation performs lower than implementation of LEA block cipher on a 32-bit platform.

## VI. CONCLUSION

In this article, we applied a forced parallelization strategy to PIPO block cipher to push the performance of PIPO block cipher to the limit on the 32-bit platform. To this end, we proposed a bit mask technique consisting of minimal instructions. In addition, we took full advantage of the architectural characteristics to reduce the additional overhead incurred in the forced parallelization strategy. Depending on the characteristics of the architecture, for Cortex-M4 platform, we accelerated performance via barrel-shifter; and, for RISC-V platform, we took full advantage of the registers to keep the master key in the registers. We aimed to achieve high performance with minimal RAM usage to match the characteristics of resource-constrained devices and lightweight block cipher PIPO, as a result, our PIPO-64/128 software achieved 121 CPB and 66 CPB in RISC-V and Cortex-M4, respectively. In addition, our fair and detailed performance analysis shows that PIPO block cipher is sufficiently competitive compared to other lightweight block ciphers in a 32-bit implementation environment, which contributes to PIPO's performance evaluation. While our constant implementation was targeted at 32-bit platforms, our forced parallelization methodology can be extended to other architectures, such as the 16 bits-based MSP430 and Intel CPUs that can use 16/32-bit data types.

## CONFLICT OF INTEREST

All authors have no conflict of interest

## ACKNOWLEDGMENT

YoungBeom Kim wrote the original draft, and Seog Chung Seo wrote, reviewed, and edited the article. The authors have read and agreed to the published version of the manuscript. They would like to thank the anonymous reviewers for their helpful comments.

## REFERENCES

[1] H. Kim, Y. Jeon, G. Kim, J. Kim, B.-Y. Sim, D.-G. Han, H. Seo, S. Kim, S. Hong, J. Sung, and D. Hong, "PIPO: A lightweight block cipher with efficient higher-order masking software implementations," in *Information Security and Cryptology–(ICISC)*, D. Hong, Ed. Cham, Switzerland: Springer, 2021, pp. 99–122.

[2] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A small Present-towards reaching the limit of lightweight encryption," in *Proc. 19th Int. Conf. Cryptograph. Hardw. Embedded Syst. (CHES)* (Lecture Notes in Computer Science), vol. 10529, W. Fischer and N. Homma, Eds. Taiwan: Springer, 2017, pp. 321–345.

[3] H. Kim, M. Sim, S. Eum, K. Jang, G. Song, H. Kim, H. Kwon, W. Lee, and H. Seo, "Masked implementation of PIPO block cipher on 8-bit AVR microcontrollers," in *Proc. 22nd Int. Conf. Inf. Secur. Appl. (WISA)* (Lecture Notes in Computer Science), vol. 13009, H. Kim, Ed. Jeju Island, South Korea: Springer, Aug. 2021, pp. 171–182.

[4] J. Song, Y. Kim, and S. C. Seo, "High-speed fault attack resistant implementation of PIPO block cipher on ARM cortex—A," *IEEE Access*, vol. 9, pp. 162893–162908, 2021.

[5] H. Seo and Z. Liu, "All the hight you need on cortex–M4," in *Proc. 22nd Int. Conf. Inf. Secur. Cryptol. (ICISC)* (Lecture Notes in Computer Science), vol. 11975, J. H. Seo, Ed. Seoul, South Korea: Springer, 2019, pp. 70–83.

[6] A. Adomnicai and T. Peyrin, "Fixslicing AES-like ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V," *IACR Trans. Cryptogr. Hardw. Embedded Syst.*, vol. 2021, no. 1, pp. 402–425, 2021.

[7] A. Adomnicai, Z. Najm, and T. Peyrin, "Fixslicing: A new GIFT representation: Fast constant-time implementations of GIFT and GIFT-COFB on ARM Cortex-M," *IACR Trans. Cryptogr. Hardw. Embedded Syst.*, vol. 2020, no. 3, pp. 402–427, 2020.

[8] P. Schwabe and K. Stoffelen, "All the AES you need on Cortex-M3 and M4," in *Proc. 23rd Int. Conf. Sel. Areas Cryptogr. (SAC)* (Lecture Notes in Computer Science), vol. 10532, R. Avanzi and H. M. Heys, Eds. St. John's, NL, Canada: Springer, 2016, pp. 180–194.

[9] H. Seo, H. Kim, K. Jang, H. Kwon, M. Sim, G. Song, and S. Uhm, "Compact implementation of ARIA on 16-bit MSP430 and 32-bit ARM cortex-M3 microcontrollers," *Electronics*, vol. 10, no. 8, p. 908, Apr. 2021.

[10] Y. Kwak, Y. Kim, and S. C. Seo, "Parallel implementation of PIPO block cipher on 32-bit RISC-V processor," in *Proc. 22nd Int. Conf. Inf. Secur. Appl. (WISA)* (Lecture Notes in Computer Science), vol. 13009, H. Kim, Ed. Jeju Island, South Korea: Springer, 2021, pp. 183–193.

[11] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: An ultra-lightweight blockcipher," in *Proc. 13th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)* (Lecture Notes in Computer Science), vol. 6917, B. Preneel and T. Takagi, Eds. Nara, Japan: Springer, 2011, pp. 342–357.

[12] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," in *Proc. 9th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)* (Lecture Notes in Computer Science), vol. 4727, P. Paillier and I. Verbauwhede, Eds. Vienna, Austria: Springer, 2007, pp. 450–466.

[13] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, "Midori: A block cipher for low energy," in *Proc. 21st Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)* (Lecture Notes in Computer Science), vol. 9453, T. Iwata and J. H. Cheon, Eds. Auckland, New Zealand: Springer, 2015, pp. 411–436.

[14] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee, "HIGHT: A new block cipher suitable for low-resource device," in *Proc. 8th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)* (Lecture Notes in Computer Science), vol. 4249, L. Goubin and M. Matsui, Eds. Yokohama, Japan: Springer, 2006, pp. 46–59.

[15] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "PRINCE—A low-latency block cipher for pervasive computing applications," in *Proc. 18th Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)* (Lecture Notes in Computer Science), vol. 7658, X. Wang and K. Sako, Eds. Beijing, China, Springer, 2012, pp. 208–225.

[16] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK lightweight block ciphers," in *Proc. 52nd Annu. Design Automat. Conf.*, San Francisco, CA, USA, Jun. 2015, p. 175.

[17] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "SKINNY-AEAD and skinny-hash," *IACR Trans. Symmetric Cryptol.*, vol. 2020, no. S1, pp. 88–131, 2020.

[18] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw, "The LED block cipher," in *Proc. 13th Int. Workshop (CHES)* (Lecture Notes in Computer Science), B. Preneel and T. Takagi, Eds. Nara, Japan: Springer, 2011, pp. 326–341.

[19] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK block ciphers on AVR 8-bit microcontrollers," in *Proc. 3rd Int. Workshop Lightweight Cryptogr. Secur. Privacy* (Lecture Notes in Computer Science), vol. 8898, T. Eisenbarth and E. Öztürk, Eds. Istanbul, Turkey: Springer, 2014, pp. 3–20.

[20] A. Baysal and S. Sahin, "RoadRunneR: A small and fast bitslice block cipher for low cost 8-bit processors," in *Proc. 4th Lightweight Cryptogr. Secur. Privacy* (Lecture Notes in Computer Science), vol. 9542, T. Güneysu, G. Leander, and A. Moradi, Eds. Bochum, Germany: Springer, 2015, pp. 58–76.

[21] D. D. Dinu, A. Biryukov, J. Groszschädl, D. Khovratovich, Y. L. Corre, and L. Perrin, "FELICS–fair evaluation of lightweight cryptographic systems," in *Proc. NIST Workshop Lightweight Cryptogr.*, 2015, pp. 1–38.

[22] C. Beierle, G. Leander, A. Moradi, and S. Rasoolzadeh, "CRAFT: Lightweight tweakable block cipher with efficient protection against DFA attacks," *IACR Trans. Symmetric Cryptol.*, vol. 2019, no. 1, pp. 5–45, 2019.

[23] RISC-V. (2021). *RISC-V Bitmanip Extension*. [Online]. Available: https://github.com/riscv/riscv-bitmanip

[24] B. Marshall, G. R. Newell, D. Page, M. O. Saarinen, and C. Wolf, "The design of scalar AES instruction set extensions for RISC-V," *IACR Trans. Cryptogr. Hardw. Embedded Syst.*, vol. 2021, no. 1, pp. 109–136, 2021. [Online]. Available: https://github.com/riscv/riscv-crypto

[25] RISC-V. (2021). *RISC-V Vector Extension*. [Online]. Available: https://github.com/riscv/riscv-v-spec

[26] C. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C. Shih, and B. Yang, "NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on cortex-M4 and AVX2," *IACR Trans. Cryptogr. Hardw. Embedded Syst.*, vol. 2021, no. 2, pp. 159–188, 2021.

[27] K. Stoffelen, "Instruction scheduling and register allocation on ARM cortex-M," in *Software Performance Enhancement for Encryption and Decryption, and Benchmarking–SPEED-B*. 2016. [Online]. Available: https://ko.stoffelen.nl/talks/20161020-speedb.pdf

[28] S. Lim, J. Han, T.-H. Lee, and D.-G. Han, "Differential fault attack on lightweight block cipher PIPO," Cryptol. ePrint Arch., Tech. Rep. 2021/1190, 2021. [Online]. Available: https://eprint.iacr.org/2021/1190

[29] Y. Kwak, Y. Kim, and S. C. Seo, "Benchmarking Korean block ciphers on 32-bit RISC-V processor," *J. Korea Inst. Inf. Secur. Cryptol.*, vol. 31, no. 3, pp. 331–340, 2021.

[30] K. Jang, G. Song, H. Kwon, S. Uhm, H. Kim, W.-K. Lee, and H. Seo, "Grover on PIPO," *Electronics*, vol. 10, no. 10, p. 1194, May 2021.

[31] J.-J. Lee, J.-U. Park, M.-J. Kim, and H.-W. Kim, "Efficient ARIA cryptographic extension to a RISC-V processor," *J. Korea Inst. Inf. Secur. Cryptol.*, vol. 31, no. 3, pp. 309–322, 2021.

[32] E. Tehrani, T. Graba, A. S. Merabet, and J.-L. Danger, "RISC-V extension for lightweight cryptography," in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 222–228.

[33] K. Stoffelen, "Efficient cryptography on the RISC-V architecture," in *Progress in Cryptology–(LATINCRYPT)*. P. Schwabe and N. Thériault, Eds. Cham, Switzerland: Springer, 2019, pp. 323–340.

[34] F. Campos, L. Jellema, M. Lemmen, L. Müller, D. Sprenkels, and B. Viguier, "Assembly or optimized C for lightweight cryptography on RISC-V?" in *Cryptology and Network Security* S. Krenn, H. Shulman, and S. Vaudenay, Eds. Cham, Switzerland: Springer, 2020, pp. 526–545.

[35] Y. Le Corre, J. Großschädl, and D. Dinu, "Micro-architectural power simulator for leakage assessment of cryptographic software on ARM cortex-M3 processors," in *Proc. Int. Workshop Constructive Side-Channel Anal. Secure Design*. Springer, 2018, pp. 82–98.

[36] H. Kim, K. Jang, G. Song, M. Sim, S. Eum, H. Kim, H. Kwon, W.-K. Lee, and H. Seo, "SPEEDY on cortex–M3: Efficient software implementation of SPEEDY on ARM cortex–M3," Cryptology ePrint Arch., Tech. Rep. 2021/1212, 2021. [Online]. Available: https://ia.cr/2021/1212

[37] H.-J. Seo, "High speed implementation of LEA on ARM cortex-M3 processor," *J. Korea Inst. Inf. Commun. Eng.*, vol. 22, no. 8, pp. 1133–1138, 2018.

[38] D. O. C. Greconici, M. J. Kannwischer, and D. Sprenkels, "Compact dilithium implementations on cortex-M3 and cortex-M4," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 1, pp. 1–24, Dec. 2020.

[39] L. May, L. Penna, and A. Clark, "An implementation of bitsliced DES on the Pentium MMX processor," in *Proc. Australas. Conf. Inf. Secur. Privacy*. Springer, 2000, pp. 112–122.

[40] A. Waterman1 and K. Asanovic. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA*. [Online]. Available: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

[41] G. Pojoga and K. Papagiannopoulos, "Low-latency implementation of the GIFT cipher on RISC-V architectures," in *Proc. 19th ACM Int. Conf. Comput. Frontiers*, May 2022, pp. 287–295.

[42] T. Reis, D. F. Aranha, and J. López, "PRESENT runs fast," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.* Springer, 2017, pp. 644–664.

[43] D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, "Triathlon of lightweight block ciphers for the Internet of Things," *J. Cryptogr. Eng.*, vol. 9, no. 3, pp. 283–302, Sep. 2019.

[44] H. Seo, K. An, and H. Kwon, "Compact LEA and HIGHT implementations on 8-bit AVR and 16-bit MSP processors," in *Proc. Int. Workshop Inf. Secur. Appl.* Springer, 2018, pp. 253–265.

**YOUNGBEOM KIM** (Student Member, IEEE) received the B.S. degree from the Department of Information Security, Cryptology, and Mathematics, Kookmin University, where he is currently pursuing the master's degree in financial information security. His research interests include the optimization of cryptographic algorithms, its efficient implementations on various IoT devices, and cryptographic module validation programs.

**SEOG CHUNG SEO** (Member, IEEE) received the B.S. degree in information and computer engineering from Ajou University, Suwon, South Korea, in 2005, the M.S. degree in information and communications from the Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea, in 2007, and the Ph.D. degree from Korea University, Seoul, South Korea, in 2011. He worked as a Research Staff Member of the Samsung Advanced Institute of Technology (SAIT) and the Samsung DMC Research and Development Center, from September 2011 to April 2014. He was a Senior Research Member of the Affiliated Institute of ETRI, South Korea, from 2014 to 2018. He currently works as an Associate Professor at Kookmin University, South Korea. His research interests include public-key cryptography, its efficient implementations on various IT devices, cryptographic module validation program, networks security, and data authentication algorithms.