

Received November 14, 2021, accepted December 6, 2021, date of publication December 8, 2021, date of current version December 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3133888

High-Speed Fault Attack Resistant Implementation of PIPO Block Cipher on ARM Cortex-A

JINGYO SONG^{ID}, (Student Member, IEEE), YOUNGBEOM KIM^{ID}, (Student Member, IEEE), AND SEOG CHUNG SEO^{ID}, (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 in part by the National Research Foundation of Korea (NRF) Grant by the Korean Government through the Ministry of Science and ICT (MSIT) under Grant 2019R1F1A1058494, in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant by the Korean Government through MSIT (6G autonomous security internalization-based technology research to ensure security quality at all times) under Grant A2021-0270, and in part by the Korea Evaluation Institute of Industrial Technology (KEIT) Grant by the Korean Government through the Ministry of Trade, Industry and Energy (MOTIE).

ABSTRACT In ICISC'20 conference, PIPO (Plug-In and Plug-Out) was proposed as an efficient block cipher for secure communication in IoT (Internet of Things) environment. Although PIPO equips easily high-order masking implementation because of small non-linear operations and short rounds, PIPO is still vulnerable to fault attack. For resisting the fault attack, block cipher implementation should be applied to the fault attack countermeasure. However, these techniques make the performance of computationally-intensive cryptographic algorithms slower in constrained devices. To improve this, we propose the first fast and secure software of PIPO block cipher co-designed with ARM/NEON processor for high-speed secure communication. For accelerating the performance, we present an optimal implementation of PIPO block cipher in ARM/NEON processor, respectively, and design the Interleaved way utilizing two cores. With the proposed optimal techniques, we provide the high-speed secure software. In addition, we present an interleaving random-shuffling technique, which optimizes random-shuffling by utilizing two cores. For ensuring the resistance of fault attacks, we validated the fault resistance with the computation and instruction fault model. We utilize the intra-instruction-redundancy and known ciphertext to detect them. Through the proposed contributions, the fast software for PIPO block cipher achieves the fastest performance than previous related studies. The secure software with the fault countermeasures is nearly 3 times faster than the reference implementation without any fault attack countermeasures. In addition, our secure software achieved performance improvement of 301% and 463% compared to the existing best work (HIGHT and revised CHAM). As a result, our fast and secure software for PIPO block cipher achieved the fastest performance by co-designing with two cores compared to previous work that is utilized only one core. Our software can be utilized for high-speed encrypted communication and CTR-DRBG in ARMv8-based IoT devices.

INDEX TERMS PIPO, fault attack resistance, ARMv8, ARM/NEON processor, optimization, co-designed, interleaving implementation, Internet of Things.

I. INTRODUCTION

As the explosion and network demands of IoT devices increase, the importance of secure communication for protecting transmitted data in the IoT environment is being emphasized. Since the IoT environment is basically an

environment with limited resources such as power, CPU performance, and memory, the block cipher for the IoT environment should be designed to provide high-speed encryption in S/W and H/W implementation. In addition, IoT devices are very vulnerable to side-channel attack because it exposes additional information when the cryptographic algorithm operates. Thus, block ciphers should be designed so that the countermeasures against side-channel attack can be

The associate editor coordinating the review of this manuscript and approving it for publication was Gautam Srivastava^{ID}.

easily equipped. Until now, most of the block ciphers [1]–[8] designed for IoT environment satisfy one or two requirements for IoT security requirements (high-speed encryption in S/W, H/W implementation, and side-channel countermeasure). In other words, it is challenging to design a block cipher while satisfying simultaneously the IoT security requirements. However, PIPO block cipher is both software and hardware implementation-friendly and easy to equip side-channel attack countermeasures. Although the superiority of PIPO block cipher has been verified, research on optimized implementation in IoT environment is lacking because it is the latest lightweight cipher. Moreover, since the IoT environment has different resources for each IoT device, optimization studies considering the resources of each IoT device are required rather than a simple reference to further accelerate performance.

The fault attack was first introduced in 1997 by Boneh *et al.* [9] as a method of guessing secret information by injecting a hardware fault when operating the RSA-CRT signature algorithm. Improving further the fault attack, Biham and Shamir [10] proposed that a secret key on block ciphers can be recovered with a DFA (Differential Fault Attack) combining a differential attack and an error attack. Accordingly, research on performing fault attack in block ciphers has been actively conducted, and it has been shown that key recovery is possible in many block ciphers [11]–[17]. For this reason, it is necessary to study fault attack countermeasures. However, when the fault attack countermeasures are applied, an additional computation-intensive task is generated, which degrades the performance of encryption. Thus, to provide high-speed encryption with countermeasures for secure communication, optimized implementation of fault attack countermeasures considering the specification of each IoT device is also required.

Until now, various studies have been conducted to resist fault attack and to accelerate the performance of fault attack countermeasures. In SAC'16 [18], *IIR* (Intra-Instruction-Redundancy) and *KC* slices (Known Ciphertext) were proposed to counteract computation fault and instruction fault in AES. In addition, the concept for the random-shuffling technique was introduced. In WISA'17 [19], optimization of fault attack countermeasures was proposed for LEA block cipher in NEON engine. The authors optimized *IIR* and random-shuffling by fully utilizing NEON engine. In ICISC'19 [20], an optimization of fault attack countermeasures for HIGHT cipher was proposed in ARM Cortex-M4. The authors presented *IIR* and random shuffling optimization techniques by utilizing ARM processor. In IEEE ACCESS'20 [21], the author proposed a more optimized random shuffling technique than previous work [19] by utilizing NEON engine. Most of the related studies [18]–[21] have improved the performance of fault attack countermeasures by using only one core (ARM or NEON). However, fault attack countermeasures still have computation-intensive tasks compared to the round operation of a block cipher.

Thus, in this paper, we present the first fast and secure software of PIPO block cipher co-design with two cores (ARM and NEON) to further improve the limitation of the previous works using only one core (ARM or NEON). For accelerating the performance of our software, we not only perform the maximum number of PIPO encryption on ARM/NEON processors respectively but also provide additional optimizations in ARM/NEON processors respectively. Through these optimizations, we present an optimal co-designed method of PIPO block cipher utilizing ARM/NEON processors. It interleaves each implementation so that the latencies of ARM operations are hidden into NEON overheads. We apply this optimal co-designed method to our secure software to improve performance. In addition, we improve the random-shuffling by co-designing with ARM/NEON processors. This technology has the advantage of being able to process more data in the same cycle as before because the latency for ARM operation is hidden. For providing the security against fault attack, we evaluate fault detection using the same fault model utilized in previous work [18], [22]. To validate our software, since PIPO block cipher is the latest block cipher and there is no existing related research, we compared it with the reference and the existing best work that optimize the other block cipher in ARMv8 environment. As a result, interleaved random-shuffling achieved a performance improvement of about 150% compared to the best random-shuffling work [21], and our secure software with fault attack countermeasures achieved about 3 times faster performance compared to reference implementation without fault attack countermeasures. Our software can be utilized for high-speed communication (TLS and SSL), CTR-DRBG, and fault attack resistant implementation in ARMv8-based IoT devices.

The contributions of this work can be summarized as follows.

- **First work of PIPO block cipher in IoT environment**

Although the excellence of PIPO block cipher has been verified, There are no studies on the optimization of PIPO block cipher in IoT environment yet. In addition, IoT devices are particularly vulnerable to side-channel attacks so that countermeasures should be applied when implementing cryptographic algorithms. However, applying side-channel countermeasures in constrained IoT devices incurs significant overheads. Thus, we propose the first fast and secure software for PIPO block cipher in ARMv8 platforms, which is widely utilized in IoT environments. As a result, we minimized the computation-intensive tasks on countermeasures against fault attacks through various optimization techniques in ARMv8 platforms and protected efficiently fault attacks.

- **Presenting the fast software for PIPO block cipher in ARMv8 platforms**

We propose an optimal implementation design for PIPO block cipher of ARM/NEON processor. We basically

utilize data parallelism in ARM/NEON processors respectively to process multiple encryptions at the same time. In addition, for ARM processor, the rotate shift operation is more optimized than the previous work [20]. At this time, a barrel shifter is utilized to effectively hide the clock cycles for shift operation. For NEON engine, implementations that minimize pipeline stalls and register scheduling that efficiently uses all vector registers are utilized. We not only diminish pipeline stalls by interleaving NEON instructions but also maximum encryptions are performed simultaneously. Finally, we redesign the implementation of ARM/NEON processor by interleaving each instruction. This implementation can effectively conceal the latencies of ARM computations to NEON overheads. Through these optimization techniques, the fast software for PIPO-64/128 (resp. PIPO-64/256) without countermeasures not only outperformed the reference implementation of PIPO block cipher by 8.83 (resp. 9.15) times but also achieved faster performance up to about 1.62 times and 1.48 times than the existing block cipher implementations on ARMv8 environment [21], [23].

- **Presenting the secure software for PIPO block cipher in ARMv8 platforms**

We present the secure software based on Interleaved way of fault attack countermeasures for PIPO block cipher. To counteract the fault attack, we utilize the *IIR* and *KC slices* to detect computation fault and instruction fault. In addition, to make it difficult for an attacker to inject faults, we apply random shuffling in each round of PIPO block cipher. However, since random-shuffling has the highest overheads among fault attack countermeasures, crypto implementation applying random-shuffling incurs performance overheads so that execution time slows down. To minimize the overheads incurred in random-shuffling, we present an Interleaved way of fault attack countermeasure co-designed with ARM/NEON processor. Moreover, we utilize techniques applied in fast software to minimize the computation-intensive tasks incurred from fault attack countermeasure. As a result, the secure software for PIPO block cipher efficiently detects computation fault and instruction fault and achieves performance improvement of about 4.63 times and 2.11 times over previous work [21], HIGHT and revised CHAM-64/128. In addition, the secure software for PIPO-64/128 (resp. PIPO-64/256) outperforms the reference implementation without fault attack countermeasures by 3.01 times (resp. 2.96 times).

The rest of the paper is summarized as follows. Section II gives the description of ARMv8 microcontroller, PIPO block cipher, and fault attack on a block cipher. Section III presents the description of existing work related to optimized implementation and fault attack countermeasure. Section IV proposes optimized implementation and fault attack countermeasures for PIPO block cipher in ARMv8 platforms.

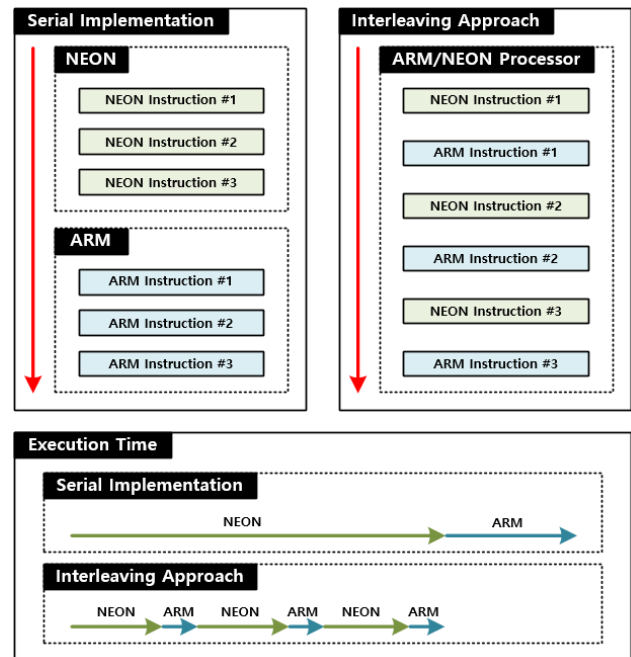


FIGURE 1. Comparison of serial implementation and interleaving approach.

Section V evaluates the performance and security of PIPO block cipher compared to the previous works in ARMv8 platforms. Finally, Section VI gives the conclusions of this paper and future work.

II. BACKGROUNDS

A. ARMv8 MICROCONTROLLER

ARM (Advanced RISC Machine) is widely utilized in the embedded industry due to supporting low power and high performance compared to previous low-end processors, AVR and MSP. ARMv8 microcontroller, the current most recent version of ARM, provides an ARM processor and NEON engine. Unlike the NEON engine, ARM processor does not support parallel processing, but it is sufficiently functional for small tasks. Moreover, ARM processor provides a barrel shifter that can hide clock cycles for shift operations in the operand, which is a very powerful technology. The register structure of the ARM processor is composed of 64-bit general-purpose registers $x0$ - $x30$, and A64 instruction set architecture [24] is provided.

NEON engine is a powerful engine that supports parallel processing and supports 128-bit vector register $v0$ - $v31$ and ASIMD (Advanced Single Instruction Multiple Data) instruction set architecture [25] for parallel processing. This parallel processing can be performed in units of 64-bit, 32-bit, 16-bit, and 8-bit within a 128-bit vector register. In addition, ARM processor and NEON engine are independent modules and perform operations independently of each other. In other words, in the case of the sequential instruction order of ARM/NEON processor, it is the sum of the execution times of ARM/NEON processor, but in the case of the Interleaving

TABLE 1. Summary of A64 and ASIMD instruction set in ARMv8 platform [24], [25].

ARM A64 Instructions			
Instructions	Operand	Description	Clock Cycles
<i>EOR</i>	X_n, X_m, X_d	XOR operation, $X_n = X_m \oplus X_d$	1
<i>ORR</i>	X_n, X_m, X_d	OR operation, $X_n = X_m \mid X_d$	1
<i>AND</i>	X_n, X_m, X_d	AND operation, $X_n = X_m \& X_d$	1
NEON ASIMD Instructions			
Instructions	Operand	Description	Clock Cycles
<i>LD4, ST4</i>	$\{V_m-V_n\}, [X_d]$	Loading and Storing multiple 4-element structures to four vector registers from X_d address register, $[X_d] \rightarrow \{V_m-V_n\}$	4
<i>TBL</i>	$V_m, \{V_n\}, V_d$	Looking up the vector table (V_n), $V_d \rightarrow \{V_n\} \rightarrow V_m$	2
<i>TRN</i>	V_m, V_n, V_d	Transpose vector operation (primary: <i>TRN1</i> , secondary: <i>TRN2</i>)	1
<i>USHL</i>	$V_m, V_n, \#n$	Unsigned shift left operation ($V_n, \#n \rightarrow V_m$)	1
<i>SRI</i>	$V_m, V_n, \#n$	Shift right and insert operation, ($V_n, \#n \rightarrow V_m$)	2
<i>DUP</i>	V_m, X_m	Duplicating the general-purpose register to vector register, $X_m \rightarrow V_m$	1
<i>EOR, ORR, AND</i>	V_m, V_n, V_d	Logical operations (XOR, OR, AND), $V_m = V_n \oplus V_d, V_m = V_n \mid V_d, V_m = V_n \& V_d$	1

approach, the pipeline stall of each instruction can be hidden and performance can be optimized efficiently. The comparison between serial and interleaved implementation is shown in Fig. 1.

Table 1 summarizes a description and cycle of 8-bit wise A64 and ASIMD instructions utilized in optimization of PIPO block cipher. A64 instructions required for the optimization of PIPO block cipher on ARM processor consist only of logical operations such as *OR*, *AND*, and *XOR*. ASIMD instructions required for optimization of PIPO block cipher on NEON engine are as follows. *STN* and *LDN* instructions store data from vector registers to memory and load them from memory to vector registers. At this time, the transpose operation is performed without additional costs. *TRN* and *TBL* instructions efficiently process the transpose process required for data parallelism, and *SHL* and *SRI* instructions are commonly used methods to implement a rotate shift operation in parallel in NEON engine. *DUP* instruction copies some elements of the vector register and expands it into a single vector register. The rest of the instructions are composed of logical operators such as *OR*, *AND*, and *XOR*. Furthermore, through the clock cycle of each instruction, it can be seen that memory access requires more cycles than simple register operations.

B. PIPO BLOCK CIPHER

In ICISC'20 [26], PIPO block cipher was proposed for secure communication in the IoT environment. In the IoT environment with limited resources for secure communication, it is necessary to satisfy all considerations such as the S/W implementation, H/W implementation, and countermeasure against side-channel attack at the same time. However, most of the previous lightweight ciphers were difficult to satisfy all of these considerations at the same time. Most satisfy one consideration or up to two considerations at the same time.

For example, PICCOLO [4], MIDORI [5], PRINCE [27], PRESENT [7] and HIGHT [28] lightweight block cipher were only suitable for H/W implementation. RoadRunner [8] and Zorro [6] were friendly only for S/W implementation, and GIFT [2], SPECK [1], SIMON [1], SKINNY [3], and LED [29] were friendly for both H/W and S/W implementation. In other words, it is very challenging to design a lightweight block cipher that satisfies simultaneously all considerations required in an IoT environment.

However, PIPO block cipher is more competitive than other lightweight block ciphers in that it satisfies simultaneously all the considerations required in the IoT environment. In the perspective of side-channel attacks, the nonlinear lay of PIPO block cipher consists of only 88 nonlinear bit operations (*AND*, *OR*). In addition, with the advantage of fewer rounds, PIPO-64/128 (resp. PIPO-64/256) only performs 13 rounds (resp. 17). This enables efficient high-order masking implementation compared to other lightweight 64-bit block ciphers. In the perspectives of S/W and H/W implementation, it outperforms the existing 64-bit lightweight block ciphers with other 128-bit key, such as SIMON [1], PRESENT [7], SKINNY [3], etc. in 8-bit AVR and hardware.

PIPO block cipher is optimized for bit slicing implementation and supports parameters of 64-bit block and 128-bit, 256-bit key. The parameters of 64-bit block and 128-bit key (resp. 64-bit block and 256-bit key) finish encryption of PIPO block cipher in only 13 rounds (resp. 15 rounds), which is a relatively short number of rounds. In the key scheduling of PIPO block cipher, it is very simple because a part of the secret key is imported and used as it is. In the round function of PIPO block cipher, it consists of Addroundkey, S-Layer, and R-Layer, and the overall process of encryption is as shown in Fig. 2. Addroundkey performs *XOR* operations between the round keys and plaintexts. S-Layer is composed of an 8-bit S-Box, and S-Box is designed with an

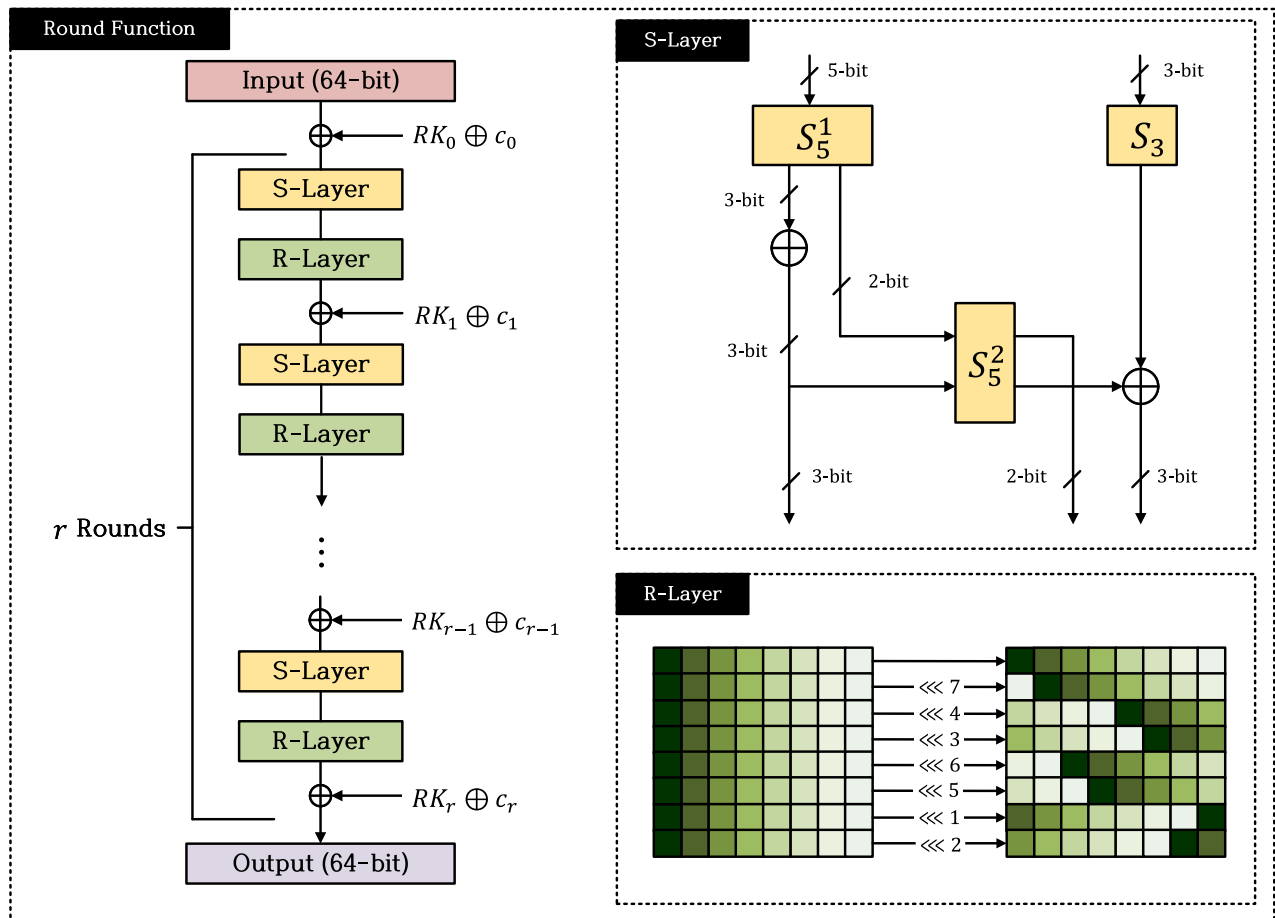


FIGURE 2. Round function of PIPO block cipher [26].

unbalanced-Bridge structure to provide efficient bit slicing implementation. R-Layer performs only of rotate shift operation for each byte unit in order to be friendly to S/W and H/W implementation. By repeating the round function for the required number of rounds, the encryption of PIPO block cipher is accomplished.

1) FAULT ATTACK ON BLOCK CIPHER

In this section, we provide a detailed description of the fault attack and the fault model for performing it. In addition, we present actual research cases to suggest the necessity of fault attack countermeasure. A fault attack is a practical side-channel attack that guesses the secret key by injecting faults with temperature, voltage, laser from the outside in an embedded device. The higher assumptions of the adversary, the more precise the faults can be injected. Through fault injection, it is possible to randomly change a value in a register at a specific location of a cryptographic algorithm in operation. The key in the fault attack is how to inject the faults at the correct location and to obtain the information of the secret key with lower complexity.

In order to carry out a fault attack, a fault model is usually used. Basically, the fault model can be divided into computation fault model and instruction fault model. First,

the computation fault is to randomly change the value of a specific word in the program by injecting a fault into the data. It can be classified into *Random-word*, *Random-byte*, and *Random-bit* according to the word, byte, and bit size of the changed value. The adversary with higher assumptions can inject finer faults. In addition, *Chosen bit pair* is a computation fault with the highest assumption of an adversary, and an adversary can change the chosen bit pair in a specific word in the program into a random value unknown to the adversary. Second, the instruction fault is a fault injection model in which the assumption of the adversary is higher than that of the computation fault, and it can replace the execution of specific instructions in the program with NOP (no-operation) instructions, thereby changing the flow of the program. As a representative example, there is an instruction skip fault model.

The fault attack was first introduced in 1996 by Boneh et al. [9] as a method of guessing secret information by injecting faults when operating the RSA-CRT signature algorithm. Since then, Biham and Shamir [10] further developed the fault attack and presented the possibility of applying the fault attack to block cipher through a DFA. DFA is an active side-channel attack that guesses the secret key with a differential between the normal ciphertext generated during

TABLE 2. Related work utilizing ARM/NEON processor (#Data parallelism: the number of encryptions performed at same time).

Optimization	Utilizing Processor		Target Block cipher	#Data parallelism
	ARM Processor	NEON Engine		
<i>Seo et al.</i> [22]	✓	✓	LEA	52
<i>Seo et al.</i> [30]	✓	✓	CHAM-64/128	112
<i>H.Fujii et al.</i> [31]	-	✓	AES	4
<i>Song et al.</i> [21]	-	✓	HIGHT, Revised CHAM	24, 16
<i>Song et al.</i> [23]	-	✓	LEA-CTR, HIGHT-CTR, Revised CHAM-CTR	24, 48, 48

the operation of the cryptographic algorithm and the error ciphertext obtained by injecting faults.

With DFA, research on fault attack in various block ciphers has been actively conducted [11]–[17]. In FutureTech'12 [16], The authors proposed a method to recover a secret key in seconds with 4 or more incorrect ciphertexts by applying a random byte fault model to the round function of HIGHT. In ICTEurAsia'15 [15], the authors recovered the secret key through 258 error ciphertexts by injecting random bit faults into the last and second-to-last rounds of LEA. In Korea Science'18 [17], 4 round keys in the last round of CHAM were recovered with a complexity of 2^4 through 24 pairs of ciphertext and error ciphertext for random word fault model. Therefore, when implementing block ciphers in an embedded environment, fault attack countermeasures should be applied to resist the threat to the DFA.

III. RELATED WORKS

A. OPTIMIZED SOFTWARE IMPLEMENTATIONS OF BLOCK CIPHERS ON ARM PROCESSORS

Since ARMv7 platforms, while supporting a powerful NEON engine capable of parallel processing in addition to ARM processor, optimization studies of various cryptographic algorithms have been conducted utilizing the ARM/NEON processor [21]–[23], [30], [31]. Table 2 shows related works optimizing the block cipher in ARM/NEON processor, and gives the explanation of the optimized cryptographic algorithm, utilized processor for optimization, and the number of encryptions processed at the same time. Since our target block cipher, PIPO, is the latest lightweight block cipher, there are no related works on optimization in ARM environment yet.

In ICISC'19 [31], the authors presented the optimized implementation of AES block cipher using ASIMD instruction set. A novel AES implementation has been provided that processes 4 blocks simultaneously by utilizing NEON engine. In particular, it has been further optimized through a novel and efficient formula in MixColumns. In IEEE ACCESS'20 [21], the secure and fast implementations of ARX-based block ciphers were proposed in ARMv8 platforms. The authors presented a method to efficiently apply the task and data parallelism techniques to HIGHT and revised CHAM family using the ASIMD instruction set. Through the proposed technologies, 24 encryptions of HIGHT were handled at the same time, and 16, 10, and 8 encryptions

of revised CHAM family were handled at the same time respectively, improving performance efficiently.

In MDPI Applied Science'21 [23], the parallel implementations of CTR mode optimization of ARX-based block ciphers (LEA, HIGHT, and revised CHAM) were proposed in ARMv8 platforms. For efficiently processing data in parallel using NEON engine, the authors presented the optimized data parallelism techniques and register scheduling which maximized the usage of vector registers. Using these optimized techniques, maximized encryption of LEA-CTR, HIGHT-CTR, and revised CHAM-CTR were performed simultaneously on NEON engine.

In WISA'16 [22], the authors presented the first Interleaved implementation for LEA block ciphers utilizing both ARM/NEON processors. Each implementation of ARM/NEON processor was optimized through a barrel shifter and efficient register scheduling that reduces vector registers required for round keys. In addition, each optimized implementation of ARM/NEON processor was combined. This Interleaved approach can improve performance by effectively hiding the latencies of the ARM operations into NEON overheads. By utilizing both ARM/NEON processors, the more maximized encryptions of LEA block cipher were performed simultaneously.

In WISA'18 [30], the authors presented the result of applying the Interleaved implementation technique of WISA'16 [22] proposed above to CHAM block cipher. Likewise above, performance was improved by effectively concealing the latencies of ARM operations into NEON overheads. As a result, because it utilizes both ARM/NEON processor, the more maximum encryption for CHAM block cipher were performed simultaneously.

Until now, most of the target block ciphers of the related works in ARM environment are based on 32-bit and 16-bit word units considering the 32-bit and 64-bit ARM environments. However, unlike the target block ciphers of related works, PIPO block cipher is optimized for the 8-bit AVR environment. Although PIPO block cipher provides high-speed performance in the 8-bit AVR environment, in 32-bit and 64-bit ARM processors, there are additional challenges to be considered unlike existing ciphers for providing the high-speed encryption of PIPO block cipher. For example, parallel rotate operation of R-Layer and transpose operation on ARM processor, where parallel processing is not

supported, and logic for efficiently parallel implementation of PIPO cipher in a vector environment.

In addition, looking at the above research results, when using ARM/NEON processor simultaneously rather than only using NEON engine, not only can more encryption be processed simultaneously, but also the latencies of ARM operations can be effectively hidden into NEON overheads. Thus, we propose the optimal implementation of PIPO block cipher co-designed with ARM processor and NEON engine. Furthermore, we present optimization methods to solve the additional challenges of PIPO block cipher in ARM processor and NEON engine, respectively. Finally, we present a method to minimize the performance overheads for countermeasures by applying proposed optimization techniques to our secure software.

B. FAULT ATTACK COUNTERMEASURES AND FAULT RESISTANCE IMPLEMENTATION OF ARM PROCESSORS

1) PREVIOUS FAULT ATTACK COUNTERMEASURES AND IMPLEMENTATION ON BLOCK CIPHER

As a number of studies on recovering keys through DFA in various block ciphers are in progress, studies on fault attack countermeasures have been actively conducted [18]–[21]. In general, software countermeasures against fault attack were based on time redundancy. However, this can be bypassed if the attacker injects the same fault sequentially. To mend this problem, In SAC'16 [18], the authors proposed various fault attack countermeasures for AES in a 32-bit environment. The first is *Intra-Instruction-Redundancy (IIR)* technique, which can detect faults within a single instruction based on software. In other words, computation fault can be detected through a comparison of the results of data and redundant data when completing the encryption. This method evolved the previous temporal redundancy into spatial redundancy. However, if an adversary with a high assumption injects the same faults into data and duplicate data, the same ciphertext is generated, which is not detected. An example of this is instruction fault. Second, the authors proposed *KC Slices* to detect instruction fault. It allocates *KC Slices* in the register and can detect instruction fault by comparing data, duplicate data, and *KC Slices* when encryption is complete. However, since *IIR* and *KC Slices* are allocated into registers, the number of simultaneous encryptions is further reduced. Finally, to make it difficult for attackers to inject faults, the authors proposed a concept of random shuffling. Since the designed structure was made of bit slicing, not only is additional overheads on operations such as transpose required for conversion of bit slicing in software, and the bit slicing structure is vulnerable to *Chosen bit pairs*.

Random-shuffling is performed to each round, so it has the most computational overheads that occur in fault attack countermeasures. Thus, the main purpose of studies of fault attack resistant implementation was to minimize the computational overheads incurred by random-shuffling. In WISA'17 [19],

the optimized implementation of random-shuffling for LEA block cipher using NEON engine was proposed. In addition, The authors designed the previous bit slicing structure to be a 32-bit word-wise structure to detect the *Chosen bit pair*. In ICISC'19 [20], optimized implementation and fault attack resistant implementation for HIGHT block cipher in ARM Cortex-M4 was proposed. The authors effectively applied task and data parallelism techniques to HIGHT. In addition, by applying these techniques to countermeasures against fault attack, computational overheads were reduced. In particular, the authors proposed the optimal technique of ARM-based random-shuffling. In IEEE ACCESS'20 [21], the authors proposed the fast and secure implementation of ARX-based block using ASIMD instructions on the ARMv8 platform. For fast implementation, data and task parallelism were efficiently applied to HIGHT and revised CHAM block ciphers. In addition, for fault attack countermeasures, the authors not only detected the computation fault and instruction fault by allocating *IIR* and *KC slices* in the vector register but also further optimized random-shuffling implementation by developing the previously masking-based method [19] into a simple table lookup method.

However, random-shuffling has still the largest computational overheads in countermeasures against fault attack, so studies on optimization implementation techniques that can further improve the performance are needed. In addition, most studies until now have proposed techniques to optimize random-shuffling using only one core (ARM or NEON). In this paper, we further minimize the computation overheads incurred by random-shuffling with ARM/NEON processors. In addition, by allocating both *IIR* and *KC slices* in the vector register, we present the more high-speed secure software of PIPO block cipher in ARMv8 platforms.

IV. PROPOSED FAST AND SECURE PIPO SOFTWARE IMPLEMENTATION

In this section, we give a detailed description of our fast and secure software for PIPO block cipher by utilizing both ARM processor and NEON engine. In our fast software, we present an optimal method, Interleaved way, to co-design with ARM/NEON processor for PIPO block cipher. This method can improve performance by hiding the latencies of ARM operations into NEON overheads through the interleaving technique. To further accelerate performance, we not only present a parallel logic of PIPO block cipher on ARM/NEON processor, respectively but also propose the optimization methods for PIPO block cipher in ARM/NEON processor, respectively. In our secure software, we present the fault attack resistant implementation of PIPO block cipher using *IIR*, *KC slices*, and random-shuffling which were the fault attack countermeasures proposed by [18]. Furthermore, for random-shuffling, which has the largest computational overheads in fault attack countermeasures, we propose an interleaved random-shuffling technique co-designed with ARM/NEON processor to minimize the computational overheads incurred by random-shuffling.

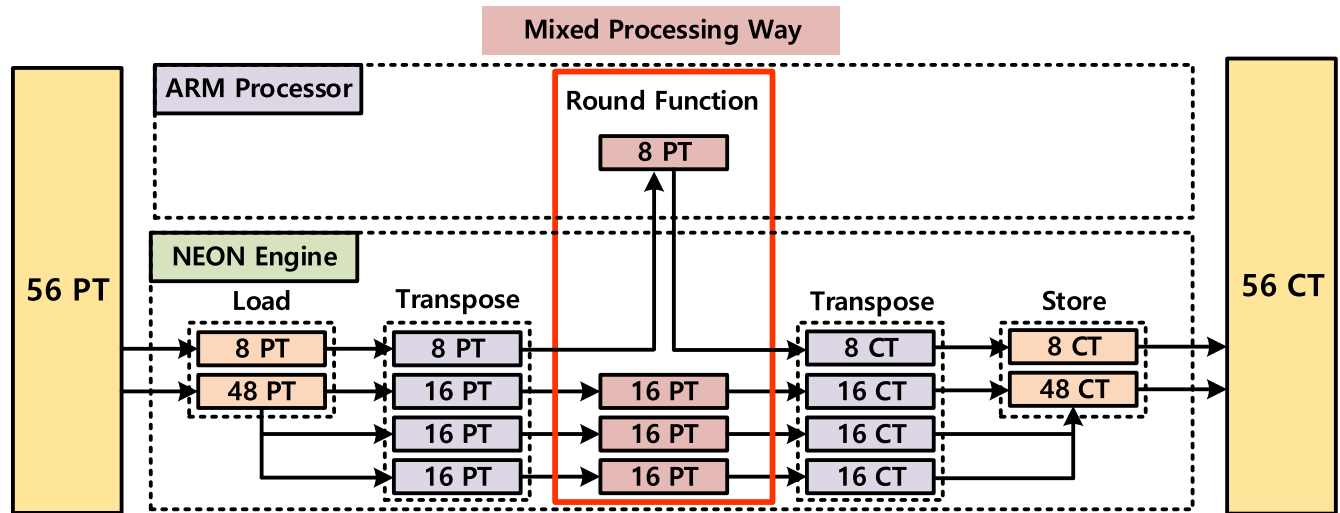


FIGURE 3. Overall design of interleaved way for PIPO block cipher utilizing both ARM/NEON processor (PT: Plaintext, CT: Ciphertext).

A. PROPOSED OPTIMIZATION APPROACHES FOR PIPO BLOCK CIPHER

1) PROPOSED CO-DESIGN TECHNIQUE OF PIPO BLOCK CIPHER UTILIZING ARM/NEON PROCESSOR

We propose a co-design technique, Interleaved way, for PIPO block cipher that utilize efficiently ARM/NEON processor. The proposed Interleaved way can be applied to other block ciphers composed of 8-bit word-wise such as PIPO cipher, and by changing the parallel unit in the NEON engine, It can also be extended to block ciphers composed of 8-bit, 16-bit, 32-bit, and 64-bit word-wise. The overall structure of the Interleaved way is shown in Fig. 3. The entire stage of the proposed Interleaved way consists of Load, Transpose, Round Function, Transpose, and Store. A total of 56 encryptions are simultaneously performed through a co-designed technique, 8 encryptions are performed simultaneously on ARM processor and 48 encryptions are simultaneously performed on NEON engine. In order to process multiple encryptions simultaneously, the transpose operation is essential.

However, moving values between registers is not easy on ARM processor, making it challenging to handle transpose efficiently. On the other hand, in the NEON engine, the transpose operation can be easily achieved with only a combination of *TRN1* and *TRN2* instructions. To solve this problem, we adopted the method of processing the transpose operation required by ARM processor in NEON Engine and then receiving it, so that ARM processor only performs the round operation. The detailed implementation of the transpose operation performed by NEON engine and the transfer process from NEON engine to ARM processor can be found in Appendix. After completing the transpose operation, when processing the round function of PIPO block cipher, the round function is performed simultaneously in ARM/NEON processor. As a result, we achieved maximum encryption simultaneously with two cores (ARM and NEON) through the interleaving technique.

TABLE 3. Difference between serial implementation and Interleaved way implementation (v0-v7: PT vector registers, v24: TEMP vector register, x10-x17: PT ARM registers, x18: TEMP ARM register).

Serial computations	Interleaved way
ARM Processor	ARM/NEON Processor
AND x18,x10,x11	AND.16b v24,v0,v1
EOR x12,x18,x12	AND x18,x10,x11
AND x18,x14,x12	EOR.16b v2,v24,v2
EOR x13,x13,x18	EOR x12,x18,x12
NEON Engine	AND.16b v24,v4,v2
AND.16b v24,v0,1	AND x18,x14,x12
EOR.16b v2,v24,v2	EOR.16b v3,v3,v24
AND.16b v24,v4,v2	EOR x13,x13,x18
EOR.16b v3,v3,24	EOR.16b v0,v0,v3
EOR.16b v0,v0,v3	

Since ARM processor and NEON engine exist as independent modules, they are calculated independently of each other. In other words, the sequential execution of ARM processor and NEON engine is the sum of the two execution times, but the interleaving approach of ARM processor and NEON engine can improve performance by effectively hiding the delay time of ARM operations into NEON overheads. Table 3 provides a detailed description of the difference between serial implementation and the Interleaved implementation of S-Layer. The rest of the R-Layer and Addround-key are also implemented through the interleaving approach as above, effectively hiding the latencies of ARM operations into NEON overheads.

The detailed explanation of the overall operation process of Interleaved is as follows. First, 56 plaintexts are loaded into each vector register, and the transpose operation is performed in NEON engine. Second, 8 plaintexts with transpose operation applied are delivered from the NEON vector register to the ARM general-purpose register, and 48 encryptions

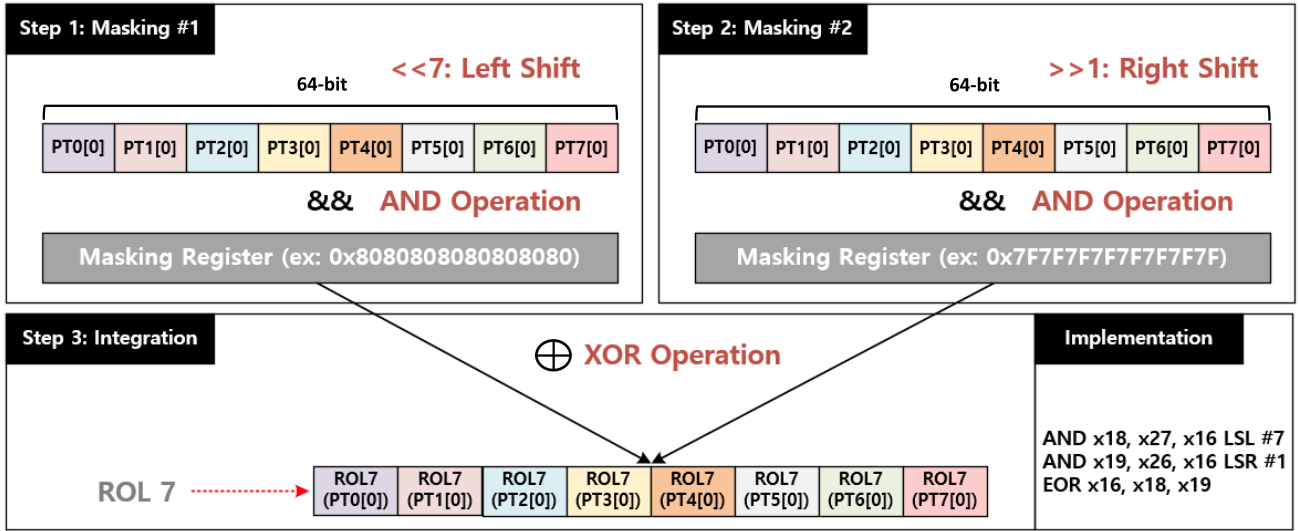


FIGURE 4. Proposed parallel rotate operation concept in ARM processor (ex: ROL7).

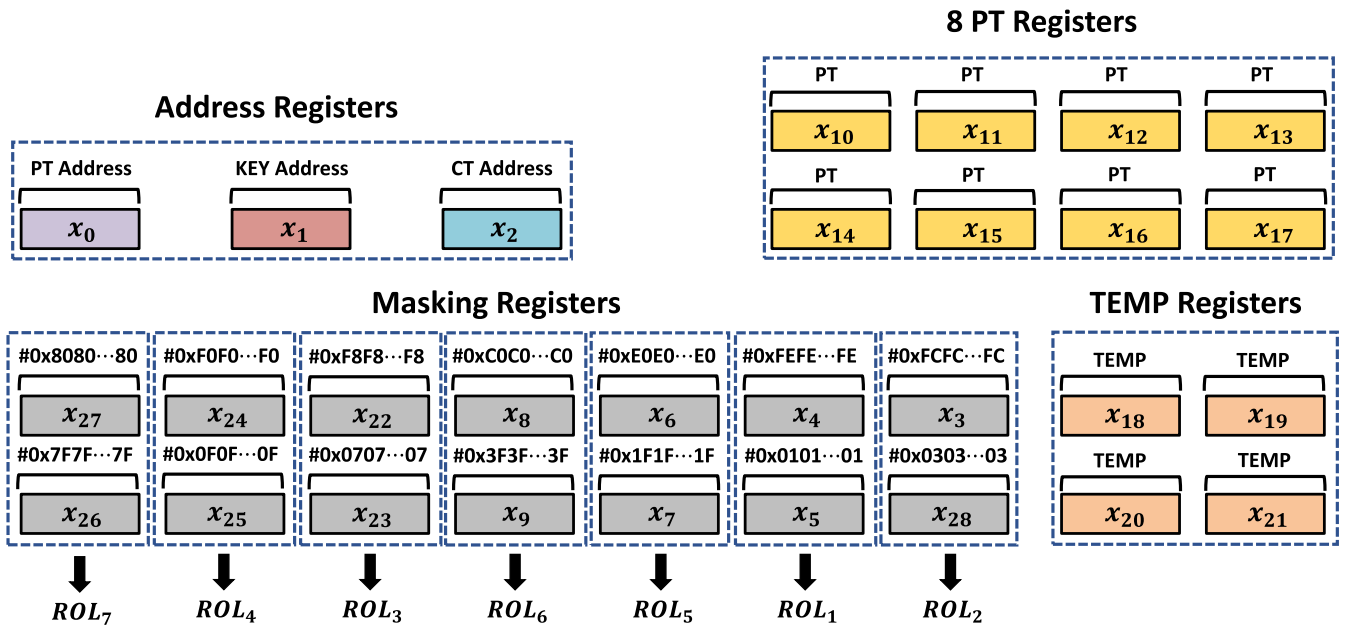


FIGURE 5. ARM register scheduling for optimization of PIPO block cipher (PT: Plaintext, KEY: Secret Key, CT: Ciphertext).

(resp. 8 encryptions) for PIPO block cipher are performed in NEON engine (resp. ARM processor). As a result, a total of 56 encryptions for PIPO block cipher are performed simultaneously by interleaving each implementation of ARM/NEON processor. Third, all values are transferred to the vector registers, and the transpose operation is performed in NEON engine. Finally, all 56 ciphertexts are stored in memory.

2) OPTIMIZED IMPLEMENTATION FOR PIPO BLOCK CIPHER IN ARM PROCESSOR

Compared to NEON engine, which supports parallel processing, ARM processor are not that powerful. However, ARM

processor support barrel shifters that can hide clock cycles for shift operations, and is functionally sufficient for small tasks. Since ARMv8-A supports a 64-bit ARM processor, and the word unit of PIPO block cipher is 8-bit, we present a parallel logic performing 8 encryptions simultaneously of PIPO block cipher on ARM processor. Since ARM processor does not support a parallel processing, parallelism for operations where carry propagates is challenging. However, for non-linear operations (XOR, AND, and OR), carry is not propagated, so parallel processing is easily possible in ARM processor. In PIPO block cipher, R-Layer consists of a rotate shift operation, which is a carry propagation operation. The remaining S-Layer and Addroundkey are composed of

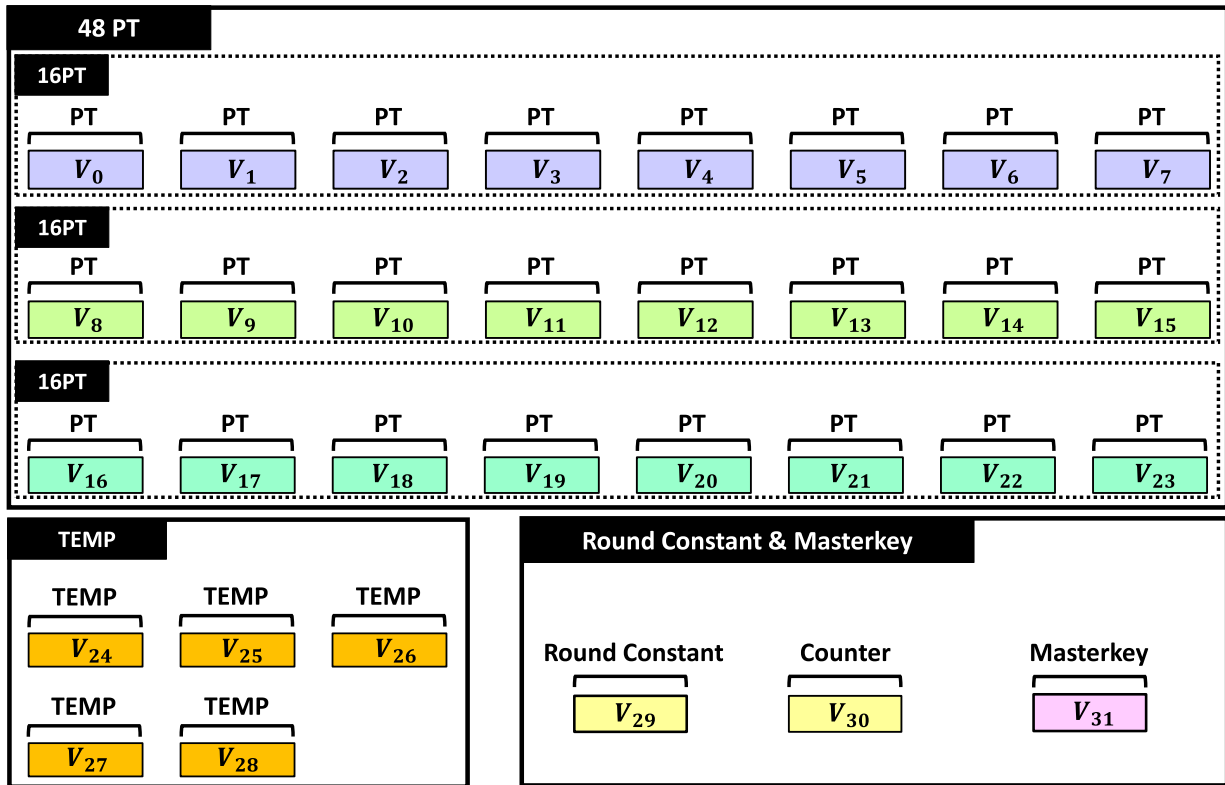


FIGURE 6. NEON vector register scheduling for optimization of PIPO block cipher (PT: Plaintext).

non-linear operations, so parallel processing can be easily implemented. However, R-Layer is not easy to implement in parallel on ARM processor due to carry propagation. Therefore, we propose an optimal method for efficient parallel implementation of R-Layer on ARM processor. The proposed concept of parallel rotate shift operation in ARM processor is shown in Fig. 4.

Fig. 4 shows a parallel rotate operation of ROL7 as an example on ARM processor. First, left shift operation ($\ll 7$) is performed, and the necessary part is extracted through the masking register. Second, the right shift operation ($\gg 1$) is performed, and the necessary part is extracted back to the masking register. Finally, by merging the two extracted result values into one through XOR operation, the rotate shift operation is easily performed in parallel on the ARM processor. This method eliminated the carry propagating through the masking register. In addition, shift operations are performed through a barrel shifter on ARM processor, eliminating clock cycles for shift operations. The implementation of R-Layer requires multiple masking registers for each ROL operation. Fig. 5 shows proposed register scheduling in ARM processor, and the detailed implementation of R-Layer in the ARM processor is in Appendix.

3) OPTIMIZED IMPLEMENTATION FOR PIPO BLOCK CIPHER IN NEON ENGINE

Unlike ARM processor, the NEON engine is a parallel module, so it is easy to implement computationally intensive

tasks such as cryptographic algorithms in parallel. Since the parallel processing is possible in units of 64-bit, 32-bit, 16-bit, and 8-bit within the 128-bit vector register, and the basic unit of the PIPO block cipher is 8-bit, we basically present parallel logic to simultaneously perform 16 encryptions of PIPO block cipher utilizing all the space in the vector register. To apply the data parallelism, a transpose operation should be required. In the NEON engine, it is easily possible with *TRN1*, *TRN2*, and *TBL* instructions, which is a widely used method. Furthermore, we present an optimal method to process 48 encryptions simultaneously by efficiently utilizing all vector registers from v_0 to v_{31} . The proposed NEON vector register scheduling is shown in Fig. 6.

In an embedded environment, efficient register scheduling to minimize memory access is very important. v_0 - v_7 , v_8 - v_{15} , v_{16} - v_{23} are used as vector registers to store each 16 plaintexts. The remaining v_{24} - v_{28} are utilized as TEMP registers to store intermediate values. Finally, v_{29} and v_{30} are used as Round Constants and counter to update them, and v_{31} is used as a master key. The R-Layer of PIPO block cipher consists of a cyclic shift operation. The rotate shift operation in parallel can be easily implemented with *SHL* and *SRI* instructions, which is a commonly used technique, on NEON engine. Since S-Layer and Addroundkey are composed of simple non-linear operations, parallel implementation is very simple on NEON engine. A more detailed implementation of R-Layer and Addroundkey is given in Appendix. Furthermore, we interleave each of the 16 encryptions of PIPO block

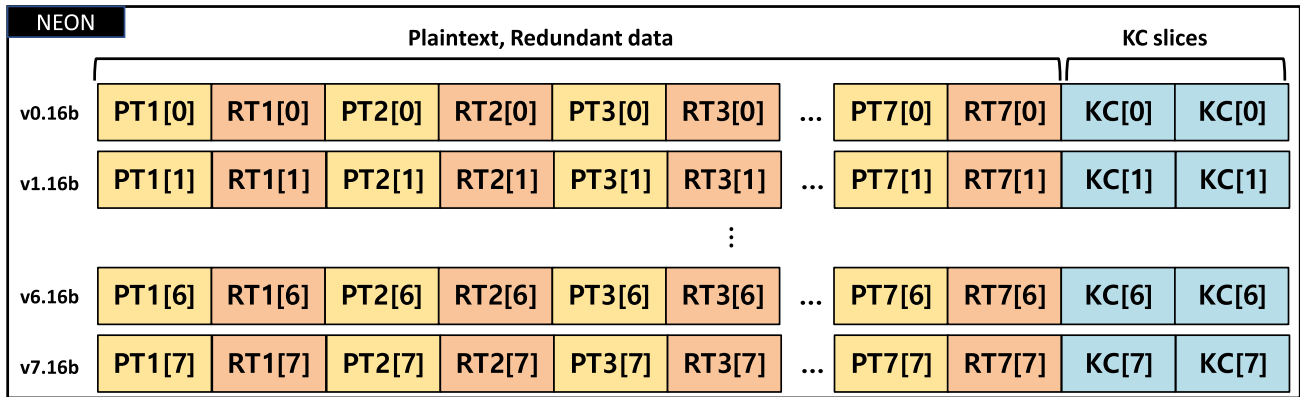


FIGURE 7. Structure of fault attack countermeasure in vector register.

cipher rather than sequentially performing the 16 encryptions of PIPO block cipher. This interleaving approach minimizes possible pipeline stalls on each implementation. A pipeline stall such as a data hazard causes performance degradation because it has to wait for the result to be received. Therefore, the proposed interleaving implementation performs better than the simple sequential implementation because it minimizes pipeline stalls.

B. PROPOSED FAULT ATTACK RESISTANT APPROACHES FOR PIPO BLOCK CIPHER

1) OVERALL DESIGN

In this section, we describe the overall design of the implementation of countermeasures for PIPO block cipher to counteract fault attacks using both ARM/NEON processors. First, we allocate *IIR* and *KC slices* in the vector register to respond to computation fault and instruction fault. Fig. 7 shows the structure in the vector register of the fault attack countermeasure implementation for the PIPO block cipher. It is the same in ARM processor, only the number of plaintexts is different. 21 (resp. 3) encryptions of PIPO block cipher are performed simultaneously in NEON engine (resp. ARM processor). Second, we utilize random-shuffling to each round so that make it difficult for an adversary to inject faults. However, since random-shuffling is the fault attack countermeasure with the largest computational overheads, we apply Interleaved way to random-shuffling to efficiently reduce the computational overheads.

Fig. 9 shows the overall design of the fault attack countermeasure implementation. First, the messages are loaded into vector registers. Next, the messages are duplicated to allocate the *IIR* into vector registers. It then performs the round function and applies random-shuffling at the end of each round. When an all-around function is over, last-random-shuffling is applied to shuffling the ciphertext to the correct position. Finally, compared with the ciphertext, *IIR*, and *KC slices* into vector registers, it checks whether there was fault injection by an aggressive adversary during encryption, and if there was no fault injection, the ciphertext is returned.

2) PROPOSED FAULT ATTACK RESISTANT IMPLEMENTATION OF PIPO BLOCK CIPHER WITH INTERLEAVED WAY

We present an optimized method by applying Interleaved way to random-shuffling. Random-shuffling in ARM processor utilizes the method proposed in the previous work [20], and random shuffling in NEON engine utilizes the method proposed in the previous work [21]. The random bit is generated by using something like a random number generator. According to the random bit, if it is 1, the value in the register becomes swap, and if it is 0, no swap operation is applied.

Random-shuffling proposed in [20] and [21] are all constant-time implementations without branching statements. Table 4 shows the comparison of random-shuffling between serial implementation and Interleaved way. In the serial implementation of random-shuffling, the whole execution times are the sum of ARM and NEON instructions, but in the case of applying the Interleaved way, by interleaving each instruction of ARM/NEON processor, the latencies of ARM computations is effectively hidden into NEON overheads, thereby efficiently reducing the computational overheads incurred by random-shuffling. In addition, since the previously proposed random-shuffling techniques [20], [21] can be applied to all block ciphers composed of 8-bit, 16-bit, 32-bit, and 64-bit word-wise, the proposed interleaving random-shuffling technique can be expended to all block ciphers.

V. EVALUATION

We evaluate the performance of the proposed fast and secure implementation for PIPO block cipher in ARMv8 platform. The performance was measured on the Raspberry Pi 4B board [32]. Raspberry Pi 4B board supports 1.5 GHz quad-core Arm® Cortex-A72 CPU and NEON engine, and supports 4 GB internal memory. We used the AArch64-gnu-gcc compiler in Ubuntu 20.04 LTS OS environment and VS-Codium as the development environment. The reference implementation of PIPO block cipher uses the pure source code provided in github of specification [26], and was implemented in C language. The rest of the previous work and

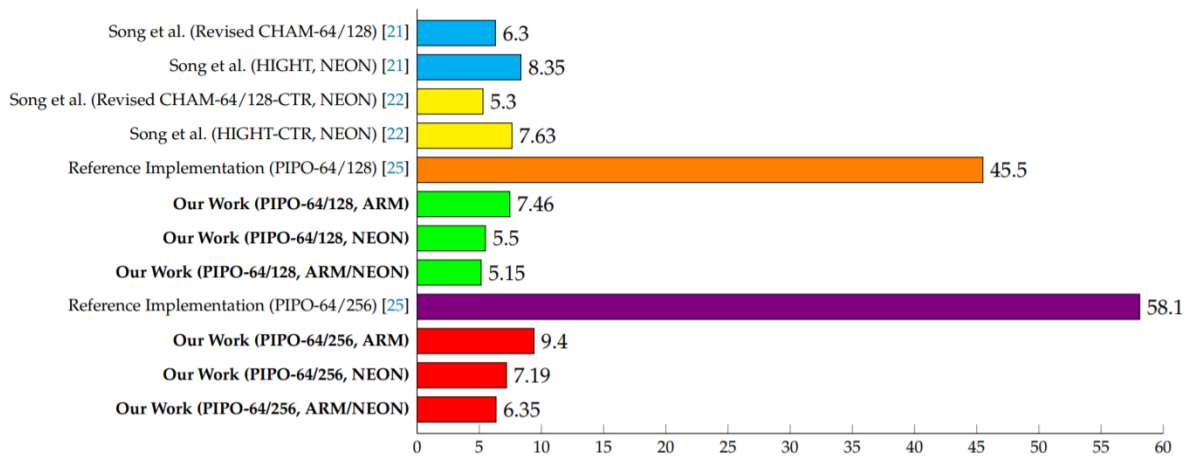


FIGURE 8. Comparison of performance of fast software for PIPO block cipher on ARMv8 platform.

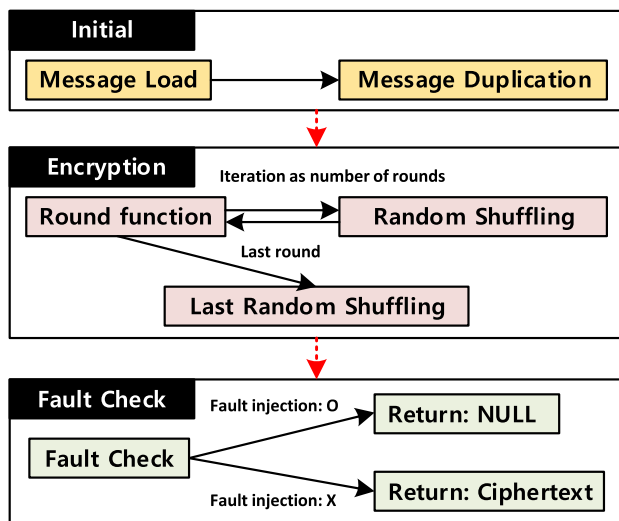


FIGURE 9. Overall design of fault attack countermeasures.

our work were implemented in assembly language. Also, the reference implementation applied the -O3 option at compile. In this section, we provide results of comparison with previous works by classifying it into fast software and secure software for PIPO block cipher. Since PIPO block cipher is the latest lightweight cipher, there is no related research in the IoT environment, so we compared the performance of our software with reference [26] and the existing work [21], [23] that optimize the block ciphers in ARMv8 environment.

A. PERFORMANCE COMPARISON OF FAST IMPLEMENTATIONS OF PIPO BLOCK CIPHER

Fig. 8 provides a detailed comparison of previous studies and our work in the ARMv8 platforms. In the case of PIPO-64/128, we compared the performance with related studies of HIGHT and revised CHAM-64/128, which provide the same security level, and in the case of PIPO-64/256, since there are no related studies that provide the same security

TABLE 4. Comparison of random-shuffling implementation between serial implementation and Interleaved way (x_0 :random-bit, x_{18} :temp, x_{20} :accumulator, $x_{10} - x_{17}$:PT, v_{24} :temp, v_{30} :random-table, $v_0 - v_7$:PT).

Serial computations	Interleaved Way
ARM Processor	ARM/NEON Processor
AND w18,w0,#1	AND w18,w0,#1
LSL w18,w18,#5	DUP v24.16b,w18
EOR w20,w18,w20	LSL w18,w18,#5
ROR x10,x10,x18	SHL.16b v24,v24,#3
ROR x11,x11,x18	EOR w20,w18,w20
ROR x12,x12,x18	EOR.16b v24,v24,v30
ROR x13,x13,x18	TBL.16b v0,v0,v24
ROR x14,x14,x18	ROR x10,x10,x18
ROR x15,x15,x18	TBL.16b v1,v1,v24
ROR x16,x16,x18	ROR x11,x11,x18
ROR x17,x17,x18	TBL.16b v2,v2,v24
NEON Engine	ROR x12,x12,x18
DUP v24.16b,w18	TBL.16b v3,v3,v24
SHL.16b v24,v24,#3	ROR x13,x13,x18
EOR.16b v24,v24,v30	TBL.16b v4,v4,v24
TBL.16b v0,v0,v24	ROR x14,x14,x18
TBL.16b v1,v1,v24	TBL.16b v5,v5,v24
TBL.16b v2,v2,v24	ROR x15,x15,x18
TBL.16b v3,v3,v24	TBL.16b v6,v6,v24
TBL.16b v4,v4,v24	ROR x16,x16,x18
TBL.16b v5,v5,v24	TBL.16b v7,v7,v24
TBL.16b v6,v6,v24	ROR x17,x17,x18
TBL.16b v7,v7,v24	-

level, we provided a comparison of the performance with the reference implementation.

In [21], The implementation of HIGHT and revised CHAM-64/128 achieved 6.3 cpb and 8.35 cpb, respectively, through applying data and task parallelism on ARM Cortex-A72 MCU (ARMv8). In [23], the authors achieved 5.3 and 7.63 cpb in HIGHT-CTR, revised CHAM-64/128-CTR on ARM Cortex-A72 MCU (ARMv8), which are

TABLE 5. Profiling of proposed secure PIPO block cipher and performance comparison of interleaving random-shuffling and related works [20], [21] (cc: Clock Cycles, #Data parallelism: the number of encryptions performed simultaneously).

Works	Transpose Operation	Round Operations			Fault Attack Countermeasures			Total
		AddKey	S – Layer	R – Layer	Duplication	Random – Shuffling	FaultCheck	
<i>Proposed PIPO-64/128</i>	153cc (5.34%)	283cc (9.88%)	1,270cc (44.34%)	601cc (20.98%)	57cc (1.99%)	430cc (15.01%)	70cc (2.44%)	2,864cc (100%)
<i>Proposed PIPO-64/256</i>	187cc (4.91%)	380cc (9.97%)	1,709cc (44.84%)	788cc (20.67%)	76cc (1.99%)	576cc (15.11%)	95cc (2.49%)	3,811cc (100%)

	ARM-Based Random-Shuffling [20]	NEON-Based Random-Shuffling [21]	Interleaving Random-Shuffling
<i>#Data Parallelism</i>	8	16	24
<i>Proposed PIPO-64/128</i>	72cc	430cc	430cc
<i>Proposed PIPO-64/256</i>	96cc	576cc	576cc

8.62% and 15.87% improvement of performance respectively, compared to the previous work [21]. The authors enhanced the performance of HIGHT and revised CHAM family by utilizing the data parallelism and characteristics of CTR mode that can be pre-computed in nonce parts. In [21] and [23], only NEON engine was utilized, and ARM processor was not utilized.

The reference implementations of PIPO block cipher are open source code of PIPO block cipher provided in [26]. It shows the performance of 45.5 cpb and 58.1 cpb, respectively, which is the result of performing one encryption. Our Work (ARM) (resp. NEON) was the result of parallel processing of PIPO block cipher in ARM processor (resp. NEON engine). The encryption of 8 (resp. 48) PIPO block cipher was simultaneously performed. Our Work (ARM/NEON) is the result of the proposed Interleaved way of PIPO block cipher. Through Interleaved way, 56 encryptions of PIPO block cipher were simultaneously performed and show the fastest performance among previous 64-bit block cipher studies. Taking advantage of the independent cores of ARM and NEON, we improved the performance by effectively hiding the latencies of ARM computations into NEON overheads. As a result, by utilizing both ARM/NEON processors, in the case of PIPO-64/128, it achieved the fastest performance compared with the previous best work, and in the case of PIPO-64/256, it achieved about 9.15 times faster performance than the reference implementation.

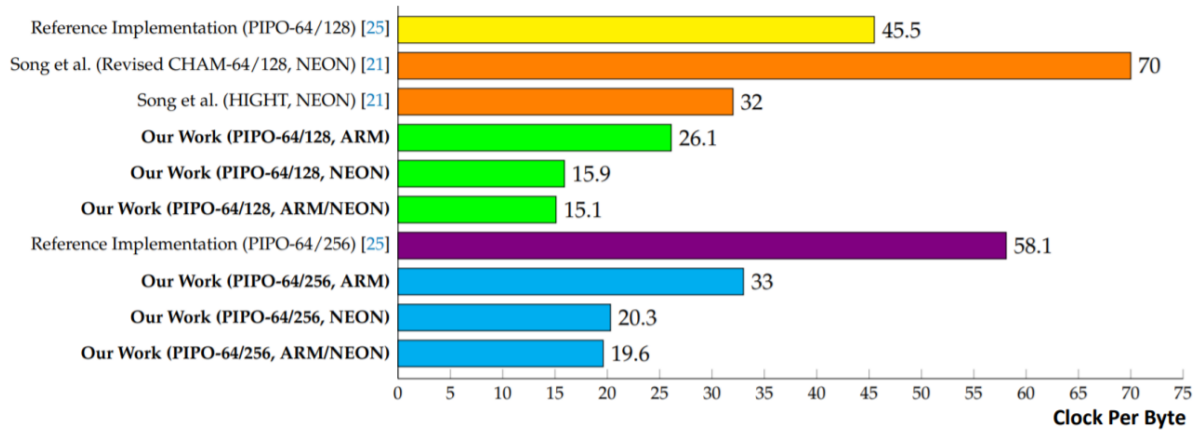
B. PERFORMANCE AND SECURITY COMPARISON OF SECURE IMPLEMENTATIONS OF PIPO BLOCK CIPHER

In this section, we provide the need for optimization of random-shuffling and evaluate the superiority of our random-shuffling compared to previous studies [20], [21]. In addition, we present the performance results comparing the secure PIPO implementation with the related work [21]. Table 5 provides a profiling result of the proposed secure

PIPO block cipher and a detailed comparison of interleaving random-shuffling and previous random-shuffling [20], [21]. From the profiling result, it can be seen that most of the overheads of the proposed secure PIPO block cipher implementation are round operations (about 73%). The overheads of secure implementation are about 19% of the total, and most of the overheads are generated by random-shuffling.

Random-shuffling is the part with the most overheads in secure implementation because it is performed for each round of the PIPO block cipher. Thus, it can be seen that optimizing random-shuffling to minimize the overloads of secure implementation is indispensable. In the performance comparison of random shuffling [20], [21], we applied the results of previous random-shuffling [20], [21] to secure PIPO implementation and measured them on the target platform. The data parallelism indicates the amount of data that can be processed at one time based on 8-bit, the basic unit of PIPO block cipher. The previous study [20] (resp. previous study [21]) can process 8 data (resp. 16 data) simultaneously. In addition, the previous studies [20], [21] optimized the random-shuffling by using only one core, ARM processor or NEON engine, and the fastest known random-shuffling implementation [21] shows 3.35 in terms of CPB (Clock Per Byte). The proposed interleaving random-shuffling shows that it can process more data in the same clock cycle compared to the previous best random-shuffling [21] by merging the two cores (ARM processor and NEON engine) well. The reason for this increase in throughput is that the interleaving implementation can improve performance by hiding the ARM operations into the NEON overheads. As a result, compared to the previous fastest random-shuffling [21], we achieved 2.23 in terms of CPB, which is a 50% improvement in performance.

Fig. 10 provides a detailed comparison of the performance and security analysis of secure software for PIPO block cipher. The fast software has the advantage of being fast, but it is vulnerable to fault attack because it is not equipped



Implementations	Computation Fault				Instruction Fault
	Rb	RB	RW	CbP	
Reference Implementation [25] (PIPO-64/128, PIPO-64/256)	-	-	-	-	-
Song et al. [21] (HIGHT)	✓	✓	✓	✓	✓
Song et al. [21] (Revised CHAM-64/128)	✓	✓	✓	✓	✓
Our Work (PIPO-64/128)	✓	✓	✓	✓	✓
Our Work (PIPO-64/256)	✓	✓	✓	✓	✓

FIGURE 10. Comparison of performance and security analysis of secure software for PIPO block cipher on ARMv8 platform, where Rb: Random-bit, RB: Random-Byte, RW: Random-Word, CbP: Chosen-bit-Pair.

Algorithm 1 Transpose Operation Process Utilizing ARM Processor

Require: v0-v7(PlainText)
Ensure: x10-x17(PlainText with transpose operation completed)

Loading the plaintext into NEON engine

```

1: LD4 {v0.8b-v3.8b},[x0],#32
2: LD4 {v4.8b-v7.8b},[x0]

```

Transpose operation

```

3: MOV v24.8b,v0.8b
4: TRN1 v0.8b,v0.8b,v4.8b
5: TRN2 v4.8b,v24.8b,v4.8b
6: MOV v24.8b,v1.8b
7: TRN1 v1.8b,v1.8b,v5.8b
8: TRN2 v5.8b,v24.8b,v5.8b

```

```

9: MOV v24.8b,v2.8b
10: TRN1 v2.8b,v2.8b,v6.8b
11: TRN2 v6.8b,v24.8b,v6.8b
12: MOV v24.8b,v3.8b
13: TRN1 v3.8b,v3.8b,v7.8b
14: TRN2 v7.8b,v24.8b,v7.8b

```

NEON Engine $\xrightarrow{MOV}$ ARM Processor

```

15: MOV x10,v0.d[0]
16: MOV x11,v1.d[0]
17: MOV x12,v2.d[0]
18: MOV x13,v3.d[0]
19: MOV x14,v4.d[0]
20: MOV x15,v5.d[0]
21: MOV x16,v6.d[0]
22: MOV x17,v7.d[0]

```

Algorithm 2 Parallel Implementation of R-Layer Utilizing ARM Processor

Require: x10-x17(PlainText)
Ensure: x10-x17(CipherText)

ROL₇ and ROL₄ Operations

```

1: AND x18,x27,x16,LSL #7
2: AND x20,x24,x15,LSL #4
3: AND x19,x26,x16,LSR #1
4: AND x21,x25,x15,LSR #4
5: EOR x16,x18,x19
6: EOR x15,x20,x21

```

ROL₃ and ROL₆ Operations

```

7: AND x18,x22,x14,LSL #3
8: AND x20,x8,x13,LSL #6
9: AND x21,x9,x13,LSR #2

```

ROL₅ and ROL₁ Operations

```

10: AND x19,x23,x14,LSR #5
11: EOR x14,x18,x19
12: EOR x13,x20,x21
13: AND x18,x6,x14,x12 #5
14: AND x20,x4,x11,LSL #1
15: AND x19,x7,x12,LSR #3
16: AND x21,x5,x11,LSR #7
17: EOR x12,x18,x19
18: EOR x11,x20,x21

```

ROL₂ Operation

```

19: AND x18,x3,x10,x12 #2
20: AND x19,x29,x10,LSR #6
21: EOR x10,x18,x19

```

with fault attack countermeasures. To evaluate the resistance of fault attack, we performed the test from the viewpoint of computation fault and instruction fault, which are the same scenarios of fault attack in the previous works [18]–[20].

Since the fault attack countermeasures were not included in the reference implementation of PIPO block cipher, neither the computation fault nor the instruction fault was detected, and an adversary can easily exploit it in a variety of ways

through fault attack. In [21] and Our Work, various fault models based on Random-bit, Random-Byte, Random-Word, Chosen-bit-pair, and instruction fault were efficiently protected through fault attack countermeasures. Through *IIR*, the computation fault is easily detected by comparing the original data and the duplicated data in the vector register when completing the encryption. However, if an adversary injects a fault in the same position of the original data and

Algorithm 3 Parallel Implementation of R-Layer Utilizing NEON Engine

Require: v0-v23(48 PlainText)
Ensure: v0-v23(48 CipherText)

```

    Using TEMP register
1: MOV v24.16b,v6.16b
2: MOV v25.16b,v14.16b
3: MOV v26.16b,v22.16b

    ROL7 Operation
4: SHL v6.16b,v24.16b,#7
5: SHL v14.16b,v25.16b,#7
6: SHL v22.16b,v26.16b,#7
7: SRI v6.16b,v24.16b,#1
8: SRI v14.16b,v25.16b,#1
9: SRI v22.16b,v26.16b,#1

    Using TEMP register
10: MOV v24.16b,v5.16b
11: MOV v25.16b,v13.16b
12: MOV v26.16b,v21.16b

    ROL4 Operation
13: SHL v5.16b,v24.16b,#4
14: SHL v13.16b,v25.16b,#4
15: SHL v21.16b,v26.16b,#4
16: SRI v5.16b,v24.16b,#4
17: SRI v13.16b,v25.16b,#4
18: SRI v21.16b,v26.16b,#4

    Using TEMP register
19: MOV v24.16b,v4.16b
20: MOV v25.16b,v12.16b
21: MOV v26.16b,v20.16b

    ROL3 Operation
22: SHL v4.16b,v24.16b,#3
23: SHL v12.16b,v25.16b,#3
24: SHL v20.16b,v26.16b,#3
25: SRI v4.16b,v24.16b,#5
26: SRI v12.16b,v25.16b,#5
27: SRI v20.16b,v26.16b,#5

```

Algorithm 4 Roundkey Duplication and Parallel Implementation of AddRoundkey Utilizing NEON Engine

Require: v0-v23(48 PlainText)
Ensure: v0-v23(48 CipherText)

```

    0-th Byte ⊕ roundkey
1: DUP v24.16b,v31.b[8]
2: EOR v0.16b,v0.16b,v24.16b
3: EOR v8.16b,v8.16b,v24.16b
4: EOR v16.16b,v16.16b,v24.16b

    1-th Byte ⊕ roundkey
5: DUP v24.16b,v31.b[9]
6: EOR v1.16b,v1.16b,v24.16b
7: EOR v9.16b,v9.16b,v24.16b
8: EOR v17.16b,v17.16b,v24.16b

    2-th Byte ⊕ roundkey
9: DUP v24.16b,v31.b[10]
10: EOR v2.16b,v2.16b,v24.16b
11: EOR v10.16b,v10.16b,v24.16b
12: EOR v18.16b,v18.16b,v24.16b

    3-th Byte ⊕ roundkey
13: DUP v24.16b,v31.b[11]
14: EOR v3.16b,v3.16b,v24.16b
15: EOR v11.16b,v11.16b,v24.16b
16: EOR v19.16b,v19.16b,v24.16b

    4-th Byte ⊕ roundkey
17: DUP v24.16b,v31.b[12]
18: EOR v4.16b,v4.16b,v24.16b
19: EOR v12.16b,v12.16b,v24.16b
20: EOR v20.16b,v20.16b,v24.16b

    5-th Byte ⊕ roundkey
21: DUP v24.16b,v31.b[13]
22: EOR v5.16b,v5.16b,v24.16b
23: EOR v13.16b,v13.16b,v24.16b
24: EOR v21.16b,v21.16b,v24.16b

    6-th Byte ⊕ roundkey
25: DUP v24.16b,v31.b[14]
26: EOR v6.16b,v6.16b,v24.16b
27: EOR v14.16b,v14.16b,v24.16b
28: EOR v22.16b,v22.16b,v24.16b

    7-th Byte ⊕ roundkey
    round constant
29: DUP v24.16b,v31.b[15]
30: EOR v24.16b,v24.16b,v29.16b
31: EOR v7.16b,v7.16b,v24.16b
32: EOR v15.16b,v15.16b,v24.16b
33: EOR v23.16b,v23.16b,v24.16b

    Updating the Counter
34: EOR v29.16b,v30.16b,v29.16b

```

the duplicated data, the same ciphertext is generated, and the faults cannot be detected. This is called instruction skip. Thus, *KC slices* is required to detect the instruction skip.

Instruction skip is easily detected by comparing the original data, duplication data, and *KC slices* in the vector register when completing the encryption.

In addition, both [21] and Our Work applied random-shuffling for each round, making it difficult for the adversary to inject faults. In [21], the random-shuffling was more optimized than the previous work [19] through simply looking up the random table. However, random-shuffling is applied in each round and is the fault attack countermeasure with the largest computation overheads. We applied Interleaved way to random-shuffling to efficiently hide ARM operations into NEON overheads and processed multiple encryptions simultaneously through the data parallelism technique. In addition, the overheads incurred by random-shuffling were further minimized due to the advantage of providing security with the short number of rounds of PIPO block cipher. As a result, secure software for PIPO block cipher achieved a performance improvement of about 4.63 times and 2.11 times compared to the previous study [21], and achieved 3 times faster performance than unprotected reference implementations.

VI. CONCLUSION

In this paper, we have presented the first fast and secure software for PIPO block cipher co-designed with ARM/NEON processor. For accelerating execution time, we basically have applied data parallelism techniques and additional optimization methods in ARM/NEON processors, respectively. As an additional optimization method, in ARM processor, the parallel rotate shift operation has been further optimized. At this time, the clock cycles required for shift operation were effectively hidden through the barrel shifter. In NEON engine, we have presented the implementation that minimizes pipeline stalls and efficient register scheduling. In addition, we have presented Interleaved way that combines each implementation in ARM/NEON processor. Through the Interleaved way, maximum encryption was performed simultaneously, and the latencies of ARM computations were hidden into NEON overheads, improving the performance of our software. Thus, the fast software for PIPO-64/128 (resp. PIPO-64/256) not only outperformed the reference implementation by about 8.83 (resp. 9.15) times but also achieved improved performance results by up to about 1.62 times and 1.48 times compared to the previous studies [21], [23]. Furthermore, we applied the proposed optimization techniques to accelerate the performance of the secure implementation. For ensuring the security against fault attack, we have utilized *IIR*, *KC slices* and random-shuffling. Through *IIR* and *KC slices*, various fault models (computational fault and instruction fault) were efficiently detected. In addition, we made it difficult for the adversary to accurately inject faults by applying random-shuffling for each round. Although it can increase the complexity of the attack, random-shuffling is applied in each round, causing very large computational overheads. To solve these problems, we applied Interleaved way to secure software for PIPO block cipher to efficiently minimize the overheads incurred by random-shuffling. As a

result, compared to the previous study [21], HIGHT, and revised CHAM-64/128, we achieved a performance improvement of about 4.63 times and 2.11 times, respectively. Our research suggests various techniques to satisfy both speed and security levels simultaneously. As a result, the secure implementation achieved approximately 3 times faster performance than the reference implementation [26] without fault attack countermeasures. In the future, we plan to conduct research on optimization and implementation of side-channel countermeasures for various lightweight cryptography, such as SKINNY, SPECK, and SIMON, in a limited environment.

APPENDIX: DETAILED IMPLEMENTATION OF PIPO BLOCK CIPHER ON ARM/NEON PROCESSOR

See Algorithms 1–4.

REFERENCES

- [1] 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 Autom. Conf.*, San Francisco, CA, USA, Jun. 2015, pp. 175:1–175:6.
- [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.*, in Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Sep. 2017, pp. 321–345.
- [3] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *Proc. 36th Annu. Int. Cryptol. Conf. Adv. Cryptol.*, in Lecture Notes in Computer Science, vol. 9815, M. Robshaw and J. Katz, Eds. Santa Barbara, CA, USA: Springer, 2016, pp. 123–153.
- [4] 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.*, in Lecture Notes in Computer Science. Nara, Japan: Springer, Sep./Oct. 2011, pp. 342–357.
- [5] 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. Adv. Cryptol.* in Lecture Notes in Computer Science. Auckland, New Zealand: Springer, Nov./Dec. 2015, pp. 411–436.
- [6] B. Gérard, V. Grosso, M. Naya-Plasencia, and F. Standaert, "Block ciphers that are easier to mask: How far can we go?" in *Proc. 15th Int. Workshop Cryptograph. Hardw. Embedded Syst.* in Lecture Notes in Computer Science, G. Bertoni and J. Coron, Eds. Santa Barbara, CA, USA: Springer, 2013, pp. 383–399.
- [7] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Viskelson, "PRESENT: An ultra-lightweight block cipher," in *Proc. 9th Int. Workshop Cryptograph. Hardw. Embedded Syst.* in Lecture Notes in Computer Science. Vienna, Austria: Springer, Sep. 2007, pp. 450–466.
- [8] A. Baysal and S. Sahin, "Roadrunner: A small and fast bitslice block cipher for low cost 8-bit processors," in *Proc. 4th Int. Workshop, LightSec Lightw. Cryptogr. Secur. Privacy* (Lecture Notes Computer Science), T. Güneysu, G. Leander, and A. Moradi, Eds. Bochum, Germany: Springer Sep. 2015, pp. 58–76.
- [9] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults (extended abstract)," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol.* (Lecture Notes Computer Science), vol. 1233, Konstanz, Germany: Springer, May 1997, pp. 37–51.
- [10] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proc. 17th Annu. Int. Cryptol. Conf.* (Lecture Notes Computer Science), vol. 1294. Berlin, Germany: Springer, Aug. 1997, pp. 513–525.
- [11] G. Piret and J.-J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and Khazad," in *Proc. 5th Int. Workshop Cryptograph. Hardw. Embedded Syst.* Cologne, Germany: Springer, Sep. 2003, pp. 77–88.
- [12] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *Proc. 5th Int. Workshop Inf. Secur. Theory Pract. Secur. Privacy Mobile Devices Wireless Commun.* Heraklion, Greece: Springer, Jun. 2011, pp. 224–233.
- [13] W. Li, D. Gu, and J. Li, "Differential fault analysis on the aria algorithm," *Inf. Sci.*, vol. 178, pp. 3727–3737, 2008.
- [14] K. Jeong, Y. Lee, J. Sung, and S. Hong, "Differential fault analysis on block cipher seed," *Math. Comput. Model.*, vol. 55, pp. 26–34, Dec. 2012.
- [15] D. Jap and J. Breier, "Differential fault attack on LEA," in *Proc. 3rd Int. Conf. Inf. Commun. Technol.* Daejeon, South Korea: Springer, Oct. 2015, pp. 265–274.
- [16] Y. Lee, J. Kim, J. H. Park, and S. Hong, "Differential fault analysis on the block cipher hight," *Future Inf. Technol., Appl., Service*, vol. 1, pp. 407–416, Oct. 2012.
- [17] H. Kwon and J. Ha, "Fault injection attack on lightweight block cipher Cham," *J. Korea Inst. Inf. Secur. Cryptol.*, vol. 28, pp. 1071–1078, Mar. 2018.
- [18] C. Patrick, B. Yuce, N. F. Ghalaty, and P. Schaumont, "Lightweight fault attack resistance in software using intra-instruction redundancy," in *Proc. 23rd Int. Conf. Sel. Areas Cryptogr.* St. John's, NL, Canada: Springer, Aug. 2016, pp. 231–244.
- [19] H. Seo, T. Park, J. Ji, and H. Kim, "Lightweight fault attack resistance in software using intra-instruction redundancy, revisited," in *Proc. 18th Int. Conf. Inf. Secur. Appl.* Jeju Island, South Korea: Springer, Aug. 2017, pp. 3–15.
- [20] H. Seo and Z. Liu, "All the hight you need on cortex-M4," in *Proc. 22nd Int. Conf. Inf. Secur. Cryptol.* Jeju Island, South Korea: Springer, Dec. 2019, pp. 70–83.
- [21] J. Song and S. C. Seo, "Secure and fast implementation of ARX-based block ciphers using ASIMD instructions in ARMv8 platforms," *IEEE Access*, vol. 8, pp. 193138–193153, 2020.
- [22] H. Seo, T. Park, S. Heo, G. Seo, B. Bae, Z. Hu, L. Zhou, Y. Nogami, Y. Zhu, and H. Kim, "Parallel implementations of lea, revisited," in *Proc. 17th Int. Workshop Inf. Secur. Appl.* in Lecture Notes in Computer Science. Jeju Island, South Korea: Springer, Aug. 2016, pp. 318–330.
- [23] J. Song and S. C. Seo, "Efficient parallel implementation of CTR mode of ARX-based block ciphers on ARMv8 microcontrollers," *MDPI Appl. Sci.*, vol. 11, pp. 2548–2572, Dec. 2021.
- [24] *ARM Developer: A64 Instructions*. Accessed: May 10, 2021. [Online]. Available: <https://developer.arm.com/documentation/ddi0596/2020-12/Base-Instructions>
- [25] *ARM developer: ASIMD Vector Instructions*. Accessed: May 10, 2021. [Online]. Available: <https://developer.arm.com/documentation/ddi0596/2020-12/SIMD-FP-Instructions>
- [26] 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 *Proc. 23rd Int. Conf. Inf. Secur. Cryptol.* Seoul, South Korea: Springer, Dec. 2020, pp. 99–122.
- [27] 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. Yalçın, "PRINCE—A low-latency block cipher for pervasive computing applications—Extended abstract," in *Proc. 18th Int. Conf. Theory Appl. Cryptol. Inf. Secur. Adv. Cryptol.* in Lecture Notes in Computer Science, Beijing, China: Springer, Dec. 2012, pp. 208–225.
- [28] 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.* in Lecture Notes in Computer Science. Yokohama, Japan: Springer, Oct. 2006, pp. 46–59.
- [29] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw, "The LED block cipher," in *Proc. 13th Int. Workshop Cryptograph. Hardw. Embedded Syst.* in Lecture Notes in Computer Science. Nara, Japan: Springer, Sep./Oct. 2011, pp. 326–341.
- [30] H. Seo, K. An, H. Kwon, T. Park, Z. Hu, and H. Kim, "Parallel implementations of CHAM," in *Proc. 19th Int. Conf. Inf. Secur. Appl.* in Lecture Notes in Computer Science, vol. 11402. Jeju Island, South Korea: Springer, Aug. 2018, pp. 93–104.
- [31] H. Fujii, F. C. Rodrigues, and J. C. López-Hernández, "Fast AES implementation using armv8 ASIMD without cryptography extension," in *Proc. 22nd Int. Conf. Inf. Secur. Cryptol.* in Lecture Notes in Computer Science. Seoul, South Korea: Springer, Dec. 2019, pp. 84–101.
- [32] *Raspberry Pi 4B Specification*. Accessed: May 10, 2021. [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>

...