mathematics

MDPI

Article

Secure HIGHT Implementation on ARM Processors

Hwajeong Seo *, Hyunjun Kim, Kyoungbae Jang, Hyeokdong Kwon, Minjoo Sim, Gyeongju Song, Siwoo Uhm and Hyunji Kim

Division of IT Convergence Engineering, Hansung University, Seoul 02876, Korea; amdjd0704@hansung.ac.kr (H.K.); starj1234@hansung.ac.kr (K.J.); hyeok@hansung.ac.kr (H.K.); alswntla@hansung.ac.kr (M.S.); thdrudwn98@hansung.ac.kr (G.S.); smile267@hansung.ac.kr (S.U.); 1594012@hansung.ac.kr (H.K.)

* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-760-8033

Abstract: Secure and compact designs of HIGHT block cipher on representative ARM microcontrollers are presented in this paper. We present several optimizations for implementations of the HIGHT block cipher, which exploit different parallel approaches, including task parallelism and data parallelism methods, for high-speed and high-throughput implementations. For the efficient parallel implementation of the HIGHT block cipher, the SIMD instructions of ARM architecture are fully utilized. These instructions support four-way 8-bit operations in the parallel way. The length of primitive operations in the HIGHT block cipher is 8-bit-wise in addition–rotation–exclusive-or operations. In the 32-bit word architecture (i.e., the 32-bit ARM architecture), four 8-bit operations are executed at once with the four-way SIMD instruction. By exploiting the SIMD instruction, three parallel HIGHT implementations are presented, including task-parallel, data-parallel, and task/data-parallel implementations. In terms of the secure implementation, we present a fault injection countermeasure for 32-bit ARM microcontrollers. The implementation ensures the fault detection through the representation of intra-instruction redundancy for the data format. In particular, we proposed two fault detection implementations by using parallel implementations. The two-way task/data-parallel based implementation is secure against fault injection models, including chosen bit pair, random bit, and random byte. The alternative four-way data-parallel-based implementation ensures all security features of the aforementioned secure implementations. Moreover, the instruction skip model is also prevented. The implementation of the HIGHT block cipher is further improved by using the constant value of the counter mode of operation. In particular, the 32-bit nonce value is pre-computed and the intermediate result is directly utilized. Finally, the optimized implementation achieved faster execution timing and security features toward the fault attack than previous works.

Keywords: efficient implementation; ARM Cortex-M4; HIGHT block cipher; fault attack detection

Citation: Seo, H.; Kim, H.; Jang, K.; Kwon, H.; Sim, M.; Song, G.; Uhm, S.; Kim, H. Secure HIGHT Implementation on ARM Processors. *Mathematics* **2021**, *9*, 1044. https://doi.org/10.3390/math9091044

Academic Editor: Angel Martín-del-Rey, Radi Romansky and Ioana Boureanu

Received: 27 March 2021
Accepted: 4 May 2021
Published: 6 May 2021

1. Introduction

Recently, advanced embedded platforms have supported data collection and data mining to generate useful information on Internet of Things (IoT) services. Since the data packet usually includes sensitive features in it, this should be securely encrypted before sending through the wireless network or saving into the database. However, the data encryption itself requires complicated computations and this is high overheads on low-end embedded platforms equipped with low computation capability, limited battery power, ROM storage, and RAM storage. For this reason, many works presented the efficient encryption on embedded processors by suggesting optimal computation routines of target block ciphers on target microcontrollers.

In this paper, we presented efficient and secure approaches for designs of the HIGHT block cipher on low-end Cortex-M4 embedded processors. We exploited parallel mechanisms, such as data-parallel, task/data-parallel, and task parallel methods to optimize the

speed on the target microcontroller. Together with the performance, the required storage in terms of ROM and RAM is reasonably small for the target microcontrollers.

Unlike previous works, this work utilized four-way ARM–SIMD instructions to execute four-way parallel computations. The HIGHT block cipher algorithm consists of 8-bit-wise computations, and 8-bit-wise computations are executed in a parallel way. To push the speed limit of the implementation, the assembly code of ARM Cortex-M4 microcontrollers is heavily exploited. Furthermore, general purpose registers are also allocated in an efficient manner.

An efficient fault attack-safe implementation is also explored. Proposed methods prevent the fault attack through the intra-instruction redundancy feature. In particular, we proposed two secure implementations by using task/data and data-parallel techniques. The task/data-parallel-based implementation is secure against fault injection models, including chosen bit pair, random bit, random byte and random word. The data-parallel implementation ensures all features of aforementioned secure implementation and the instruction skip model is also prevented. The implementation is further improved by using the unique feature of counter mode of operation. Finally, the proposed implementation of HIGHT block cipher obtained faster execution timing and security features against the fault attack than the state-of-art works. Since proposed methods for implementations are a generic approach, we can apply this technique to other works in a straightforward manner.

1.1. Research Contributions

1. *Compact HIGHT implementations in task, data, and task/data-parallel methods:* By utilizing the four-way SIMD feature of 32-bit ARM Cortex-M4 microcontroller, we executed four 8-bit-wise operations at once. With this instruction, `F1` and `F0` functions are performed in the parallel way. With this proposed method, we suggested task, data, and task/data-parallel-based implementations.
2. *Fault attack safe implementations for HIGHT block cipher:* This paper presented the fault attack safe implementation for HIGHT block cipher on embedded processors. With the parallel feature of a 32-bit ARM Cortex-M4 microcontroller, the intra-instruction redundant feature is efficiently satisfied. We also suggested the random shuffling routine to prevent the guessing by the attacker.
3. *Detection on instruction skip attack:* In order to detect the instruction skip attack, known answer slots are assigned for intra-instruction redundant features of data-parallel implementations. Total four encryption operations are performed at once by fully utilizing the general purpose registers.
4. *Counter mode of operation for data-parallel based HIGHT block cipher:* The counter mode of operation for data-parallel-based HIGHT block cipher is optimized by using unique features of constant values. Furthermore, by skipping the packing step, the encryption routine is optimized.

1.2. Extended Version of ICISC'19

The previous work in ICISC'19 is extended in this paper [1]. In [1], efficient and secure implementations of the HIGHT block cipher on low-end ARM Cortex-M4 microcontrollers were investigated. This work presents the optimal random shuffling routine and fault attack resistance implementation against the instruction skip attack. Lastly, optimized implementations of the counter mode of operation for HIGHT block cipher are proposed.

The paper is constructed as below. In Section 2, we introduce the HIGHT block cipher together with former works on target embedded processors. In Section 3, we propose optimized designs of HIGHT block cipher on target embedded processors. This is the parallel implementation and achieved the feature of resistance against the fault attack. In Section 4, we evaluated the optimized method and presented the comparison with other works. In Section 5, the conclusion is given.

2. Related Works

2.1. HIGHT Algorithm

In CHES'06 [2], the HIGHT block cipher was presented. The HIGHT algorithm was selected as the international standard (i.e., ISO/IEC 18033-3). Since the HIGHT block cipher was designed for Internet of Things (IoT) environments, its implementation on low-end devices is suitable. Lengths of block and key are 64 bits and 128 bits, respectively. Each operation is performed in the 8-bit-wise operation and this is the ARX (addition–rotation–exclusive-or) structured block cipher. The number of rounds for encryption/decryption is 32. Every round requires a 64-bit round key and this indicates that the size of the full round key is 2048 bits. Descriptions of HIGHT algorithm are given in Figure 1.

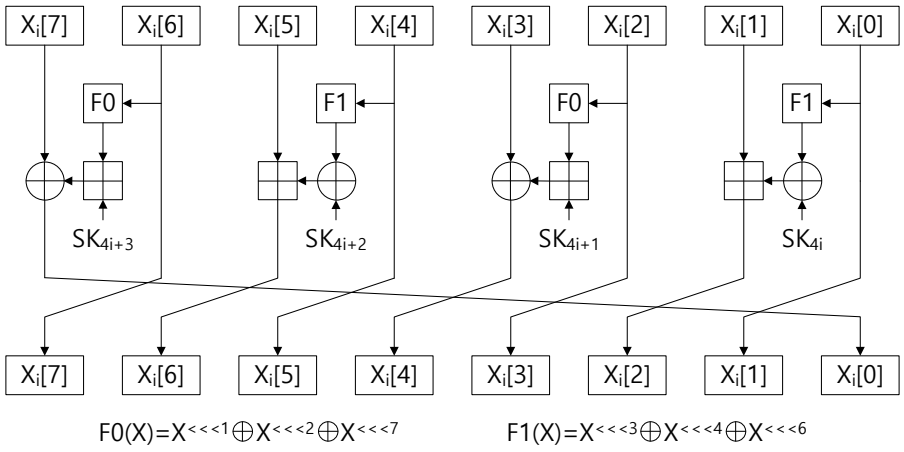

Figure 1. Encryption of the HIGHT algorithm; X and SK indicate plaintext and round key, respectively.

2.2. Previous Optimized Designs on Embedded ARM Processors

Designs of encryption algorithm on resource constrained 8-bit/16-bit IoT (Internet of Things)-embedded processors have been actively studied [3–7] . Recently, many works devoted to upgrade the execution timing of encryption operation on 32-bit ARM embedded processors. Since the word size is 32-bit wise, 32-bit operations are efficiently performed. Since the LEA algorithm relies on 32-bit wise computations, the performance of the algorithm outperforms the AES implementation on 32-bit ARM microcontrollers [8,9] In [10], the HIGHT block cipher, implementation is performed on embedded processors (i.e., Cortex-M3 microcontroller). Since the microcontroller does not support the SIMD instruction, they utilized the pseudo-SIMD approach to execute two encryption operations at once. In [11], the optimal register allocation is applied to the HIGHT implementation on the Cortex-M3 microcontroller. There are many public key cryptography implementations on Cortex-M4 microcontrollers [12–15].

The optimized design of the HIGHT algorithm on ARM-embedded processors by using the four-way SIMD instruction sets is given in this paper. With this instruction, four-way parallelism is easily achieved. Afterward, the fault attack safe implementation is introduced through the intra-instruction redundant features. Lastly, the counter mode of operation is highly optimized. This skips many steps of HIGHT block cipher through the pre-computation.

2.3. 32-Bit ARM Embedded Processors

Thirty-two-bit ARM embedded processors provide the energy-efficient architecture together with high-performance. In particular, 32-bit ARM Cortex-M4 processors support both Thumb and Thumb-2 instruction sets. Basic operations take one clock cycle while memory-related operations take at least two clock cycles. The processor supports SIMD instructions, such as `UADD8` and `USUB8` instructions. These instructions perform byte-wise

operations in a parallel way. For the evaluation, we utilized the MK20DX256VLH7@72 MHz development board.

3. Proposed Methods

In this section, we present novel HIGHT algorithm implementations on 32-bit embedded processors (i.e., Cortex-M4 microcontroller). For the high security with reasonably fast computations, we present the fault attack resistance design by using parallel computations. Furthermore, the counter mode of operation is optimized with efficient packing and unpacking techniques for the parallel computation.

Largely, the parallel acceleration can be achieved through two ways, such as task and data-parallel ways. The data-parallel implementation is performing the single instruction on multiple data sets, which ensures high speed and high throughput. On the other hand, the task of parallel implementation executes multiple instructions with multiple information sets. This ensures low latency for specific algorithms. The optimized parallel implementations of the HIGHT algorithm on Cortex-M4 embedded processors are proposed in this paper. By using parallel features, the secure implementation is also efficiently achieved.

3.1. Key Scheduling

The key scheduling of HIGHT algorithm requires addition and rotation operations in a byte-wise fashion. Byte-wise rotation operations are not supported in the 32-bit ARM architecture. Only word-wise rotation operations are supported on the target processor. For this reason, the byte-wise rotation operation can be performed with the word-wise rotation and the masking/padding approach. This firstly performs the rotation operation. Afterwards, the correction with the masking/padding operation is performed. For the case of addition or subtraction operation in the byte-wise manner, the four-way SIMD instruction (UADD8 or USUB8) can be utilized and this does not incur overflow or underflow for byte-wise addition or subtraction operation, respectively. The embedded ARM processor provides fourteen registers for general purposes. For the key scheduling procedure, almost all registers (13 general purpose registers) are utilized to cache the intermediate result. Firstly, the master key pointer is assigned to the R0 register. Afterwards, the delta pointer is assigned to the R0 register. The R1 register keeps the round key pointer. Registers from R2 to R5 maintain delta variables. The R6 register keeps the loop counter. Registers from R7 to R8 are used for temporal variables. Registers from R9 to R12 are used for round keys. The register R14 is not utilized.

The detailed register allocation is given in Table 1. For the encryption in the task parallel way, two bytes of each round key is set in the word. This representation has two padding slots and two round keys in each word. For the encryption in the data and task parallel way, empty slots are used for the copy of the round key. For this reason, each word contains two bytes of round keys and two copied bytes. In other words, the word is fully utilized.

Table 1. Register assignment for key scheduling of HIGHT block cipher on the Cortex-M4 microcontroller.

Register	Assignment
R0	secret–key pointer → constant pointer
R1	session–key pointer
R2∼R5	constant variables
R6	routine counter
R7∼R8	extra registers
R9∼R12	session–key

3.2. Parallel Implementations

Three implementation approaches, including task parallel, task and data parallel, and data parallel are presented in this paper. Implementations with the task parallelism executes the single encryption by dividing the task into multiple sub tasks and executing them in parallel. The parallel implementation with task and data executes two blocks in a parallel way. The implementation in a data parallel executes four blocks in a parallel way. In Table 2, detailed assignments of general purpose registers are given. Since the decryption part of the HIGHT algorithm is similar to the encryption part, we only describe the implementation of the encryption part in detail.

Table 2. Register details of parallel and fault-safe implementations on target processors.

Register	Task-Parallel	Task/Data-Parallel	Data-Parallel	Fault Resistance (2-Way)	Fault Resistance (4-Way, CTR)
R0	plaintext pointer	plaintext pointer	text pointer → temporal#1	text pointer → random	text pointer → temporal#1
R1	round key pointer	round key pointer	temporal variable#2	round key pointer	temporal variable#2
R2	plaintext#1	plaintext#1	temporal variable#3	plaintext#1	temporal variable#3
R3	plaintext#2	plaintext#2	temporal variable#4	plaintext#2	temporal variable#4
R4	plaintext#3	plaintext#3	plaintext#1	plaintext#3	plaintext#1
R5	plaintext#4	plaintext#4	plaintext#2	plaintext#4	plaintext#2
R6	mask	mask	plaintext#3	mask	plaintext#3
R7	loop counter	temporal variable#1	plaintext#4	temporal variable	plaintext#4
R8	round key#1	round key#1	plaintext#5	round key#1	plaintext#5
R9	round key#2	round key#2	plaintext#6	round key#2	plaintext#6
R10	temporal variable#1	temporal variable#2	plaintext#7	temporal variable	plaintext#7
R11	temporal variable#2	temporal variable#3	plaintext#8	temporal variable	plaintext#8
R12	temporal variable#3	temporal variable#4	round key pointer	temporal variable	round key pointer
R14	–	loop counter	loop counter	loop counter	loop counter and random number

3.2.1. Task Parallelism

For the task parallel implementation of the HIGHT block cipher, two bytes are paired to perform the single encryption in parallel way. In [10], `F1` and `F0` functions are executed with the rotation and masking/padding approach. For two combinations (exclusive-or operation after addition operation, and addition operation after exclusive-or operation), the special SIMD instruction (`UADD8`) is utilized. For the exclusive-or operation, the ordinary XOR operation is performed. The comparison between with and without SIMD instruction sets is given in Table 4 of [1].

3.2.2. Data Parallelism

In this paper, we investigated the two-way or four-way data-parallel implementation of the HIGHT algorithm. For the two-way data-parallel implementation, we combined the data-parallel and task parallel approaches. In the 32-bit word of ARM, 16 bits is allocated for data-parallel features and the remaining 16 bits is allocated for task parallel features. The order of data format is as follows: $\{X_i[4], X'_i[4], X_i[0], X'_i[0]\}$, $\{X_i[5], X'_i[5], X_i[1], X'_i[1]\}$, $\{X_i[6], X'_i[6], X_i[2], X'_i[2]\}$, and $\{X_i[7], X'_i[7], X_i[3], X'_i[3]\}$, where X represents the plaintext.

Both `F1` and `F0` functions need to perform rotation operations. However, the task and data-parallel implementation fully uses the 32-bit word and this does not allow padding or margin to prevent the overflow or underflow error. For this reason, additional steps

to avoid the overflow or underflow error are added. This is namely the correction parts in the implementation. Detailed procedures are described in Algorithm 1 of [1]. Firstly, the 16-bit data are extracted from the 32-bit data. Afterwards, the `F0` function is performed with the 16-bit data, which has a 16-bit padding or margin to avoid overflow or underflow error. The `F0` function on the remaining 16-bit data is executed similar way. This step also ensures the overflow-free condition through zero padding. Finally, both 16-bit results are masked to remove the overflow and added together to construct the 32-bit result.

Similarly, we investigated the 4-way data-parallel implementation. The register utilization of the 4-way data-parallel implementation is given in Table 2. This approach utilized 8 registers (`R4`∼`R11`) for plaintext. Four registers are used for temporal registers. Two registers maintain a round key pointer and loop counter. The computation is identical to the task and data-parallel-based implementation.

3.2.3. Fault Attack Resistance

The optimized implementation ensures high-speed and high-throughput. However, this is not enough when it comes to certain active attacks (e.g., fault attack). To prevent the fault attack, the cryptography implementation should equip fault-safe features in nature. In the previous section, we introduced the high-speed HIGHT implementation, which focused on the performance. In this section, we added security features to make the implementation more robust and secure against the active attack. The fault attack model can manipulate the instruction opcodes or data stream and change the program flow (e.g., `nop` instruction). These attacks can be generalized in instruction skip, chosen bit pair, random bit, random byte, and random word models [16,17].

In order to prevent these fault attack models, we introduced the intra-redundant-instruction-based fault attack detection mechanisms for the HIGHT algorithm on embedded processors. We first duplicated message packets. Afterwards, packets are randomly shuffled to make find attack points hard throughout executions. For the proper alignment of the round key and message, the shuffling process was also performed on the round key with the random seed. Detailed procedures are given as follows:

```
Data Loading →                    Data Copy →
Data / Round Key Mixing #1 →      Round #1 →
...
Data / Round Key Mixing #32 →     Round #32 →
Last Data Mixing →                Last Round Function →
Fault Attack Detection →          Result Storing
```

The duplication of the message was performed with the feature of barrel-shifter instruction. Thanks to the barrel-shifter instruction, many rotation operations are performed without additional costs. When registers (`R5, R4, R3, R2`) are paired in two bytes (i.e., {–, $X_i[4]$, –, $X_i[0]$}, {–, $X_i[5]$, –, $X_i[1]$}, {–, $X_i[6]$, –, $X_i[2]$}, and {–, $X_i[7]$, –, $X_i[3]$}), the copied of message is executed with the logical-or and barrel-shifter operations:

```
ORR R5, R5, R5, LSL#8 → ORR R4, R4, R4, LSL#8 →
ORR R3, R3, R3, LSL#8 → ORR R2, R2, R2, LSL#8 →
```

The random shuffling is performed with the swap operation. The instruction swaps the inner word whenever the target random value indicates the one. The number of rounds for the HIGHT block cipher is 32. If we perform the random shuffling in every round, the random seed should be 32 bits. Fortunately, 32-bit ARM Cortex-M4 microcontrollers support a 32-bit word and we only need to keep one random word in the register for efficient implementation. The message is shuffled in every round. However, the round key does not shuffle in every round since each round requires a different round key in the ordinary order. In order to properly update the shuffling order of the round key, we accumulated the shuffling order in the register and the shuffling for the round key is

performed at once in the last. This allows the synchronization of the order of the message and round key, which ensures the correct location of both values.

Firstly, the bit is selected from the random word. If the bit indicates 16, the shift-offset becomes 16. If not, the shift-offset becomes 0. Then, the input data are randomly mixed by referring to the shift-offset. Afterwards, the updated random offset is extracted from the memory. The random bit in the register is updated to the random offset. The updated result is stored again in the memory. Finally, the updated offset value is utilized for the mixing of the round key.

Before finishing the operation, the procedure of fault attack detection is performed by checking the output. The pair of 4 bytes is grouped into two groups. Two results are exclusive-ored to compare the result. Then, the distinguished bits are accumulated. Lastly, the fault attack detection word (R0) is returned with the result.

Similarly, we also investigated the 4-way fault resistance implementation by using the 4-way data-parallel encryption model. Unlike the 2-way fault resistance implementation, this implementation is secure against the instruction set skip model by constructing the 32-bit packet with 28-bit known answer, one plaintext, and one copied plaintext. For the four-way computation, the message should be packed in 4-way before the encryption and unpacked after the encryption to be compatible with other systems. During the computation, the message and round key shuffling is executed and detailed procedures are described in Algorithm 1.

Algorithm 1 Message and Round Key Shuffling for 4-Way Fault Resistance Implementation

Input: message variables (R4∼R11), round key variable (R3), round key pointer (R12), temporal variable (R14), random number (R0).

Output: randomly mixed data variables (R4∼R11), randomly mixed round key variable (R3).

```
1: POP {R0}
2: AND R2, R0, #16
3: ROR R0, R0, #1
4: ROR R4, R4, R2
5: ROR R5, R5, R2
6: ROR R6, R6, R2
7: ROR R7, R7, R2
8: ROR R8, R8, R2
9: ROR R9, R9, R2
10: ROR R10, R10, R2
11: ROR R11, R11, R2
12: EOR R14, R14, R2
13: PUSH {R0}
14: LDM R12!, {R3}
15: ROR R3, R3, R14
```

3.2.4. Optimized Implementation of CTR

The CTR (counter mode) for the block cipher is actively utilized in TLS (transport layer security)/SSL (secure sockets layer) and SSH (secure shell) protocols. In the view of the implementation, the input of encryption consists of counter and nonce parts. The nonce part consists of a constant value. For this reason, this part can be pre-computed and the result is determined by the counter value [18–21]. By using this feature, many computations of the HIGHT algorithm, including 5 XOR, 5 addition, 3 F0, 3 F1, and 4 memory accesses, can be replaced in 6 memory accesses [22]. Detailed procedures are

drawn in Figure 2. Left figure shows the original structure of HIGHT block cipher. Lower parts of data (`X[0], X[1], X[2], X[3]`) painted in red color is the nonce part. Remaining parts (`X[4], X[5], X[6], X[7]`) painted in black color are the counter part. Before the computation with the counter value, the nonce part can be pre-computed since the nonce part is always the constant value. The pre-computation can be available until Round 4. In the right figure, the optimized implementation of HIGHT-CTR is given. Only memory accesses are required in Rounds 2, 3, 4, and 5.

By using the counter mode of operation, we further improved the fault resistance implementation. Firstly, we optimized half of the packing procedure by the pre-computation of packing for the nonce part, since every encryption has the same packed nonce parts. Secondly, the fault attack detection part is also simplified. The fault of the known answer part is directly detected in the packed format before the unpacking process. In Algorithm 2, detailed descriptions of fault attack detection are given. In Step 1∼7, all outputs (`R4, R5, R6, R7, R8, R9, R10, R11`) are exclusive-ored and this process generates the eight-bit accumulated result (`R0`). In Step 8, the result of known answer (`R1`) is loaded from the memory pointer (`R14`). In Step 9, the known answer part is checked. This step only verifies the half of word (i.e., known answer part). In Step 10, the duplicated part is checked by 8-bit shifting operations.

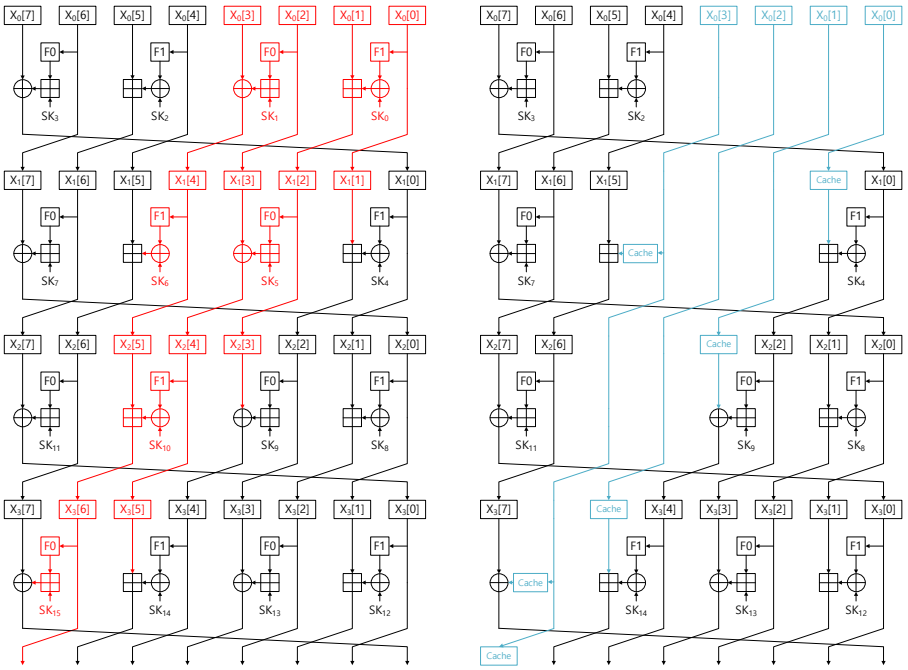

Figure 2. Counter mode of operation for HIGHT, (**left**) red colored routes represent the constant nonce part, (**right**) blue colored routes represent the optimized part.

Algorithm 2 Simplified Fault Attack Detection for Counter Mode of Operation for HIGHT

Input: message variables (R4∼R11), temporal variable (R0), random number pointer (R14).

Output: fault attack detection variable (R0).

```
1: EOR R0, R4, R5
2: EOR R0, R0, R6
3: EOR R0, R0, R7
4: EOR R0, R0, R8
5: EOR R0, R0, R9
6: EOR R0, R0, R10
7: EOR R0, R0, R11
8: LDM R14!, {R1}
9: EOR R0, R0, R1
10: EOR R0, R0, R0, LSR #8
```

4. Evaluation

We evaluated the proposed implementation of the HIGHT block cipher on embedded processors in terms of speed (clock cycles) and memory (RAM and ROM). The performance evaluation is given in Table 3. Since previously no HIGHT implementation on target processors was explored, our base comparison was done with the HIGHT implementation on Cortex-M3 microcontrollers. Compared with HIGHT implementation on Cortex-M3 microcontrollers, the Cortex-M4 provides high performance improvements. These improvements come from the optimized utilization of SIMD instruction and efficient Cortex-M4 architecture (e.g., pipelining).

Firstly, the execution timing of the task and data-parallel implementation shows better than the task parallel implementation because the task parallel only performs one encryption at once. However, the task/data-parallel implementation executes two encryption operations in the parallel way.

On the other hand, four-way data-parallel implementation shows the low performance than task parallel and task/data-parallel implementations. Since the parallel implementation assigned many registers for plaintext, the round key access in multiple ways is not available in target processors with small set of registers. The four-way data-parallel implementation is beneficial for the implementation of four-way fault resistance. In terms of the security model, we tested different fault attack scenarios, such as random word, random byte, random bit, and chosen bit pair, studied in previous works [16].

The two-way fault-safe implementation achieved the reasonable code size. The execution timing is longer than other plain implementations. The secure implementation executes a single HIGHT algorithm to detect fault attacks. Furthermore, it requires additional steps, such as message random shuffling. Compared with the task parallel implementation, the two-way fault-safe implementation consumes twice as large clock cycles.

The four-way fault resistance version requires more execution timing since it requires a greater number of shuffling, packing, unpacking, and memory accesses than that of the two-way version. The counter variant of four-way fault resistance implementation shows better performance than the original four-way fault resistance implementation, because it skips certain rounds of HIGHT algorithm and packing/unpacking routines. For the purpose of pre-computation, the RAM consumption of the counter version is larger than Electronic Code Book (ECB) version.

In terms of fault attack models, we considered random word, random byte, random bit, chosen bit pair, and instruction skip models described in [16].

Detailed fault-safe features are described in Table 4. The plain HIGHT implementation is vulnerable to all fault attack models since there is no fault detection mechanism. Previously, a bitslicing-based approach and an SIMD-based approach were investigated [16,17].

Table 3. Performance evaluation of HIGHT algorithm on ARM Cortex-M series in terms of RAM (bytes), execution time (clock cycles), and code size (bytes). EKS, ENC, DEC, and SUM represent encryption key scheduling, encryption, decryption, and summation, respectively.

Implementation	Code Size (Bytes)				RAM (Bytes)			Execution Time (Cycles per Byte)		
	EKS	ENC	DEC	SUM	EKS	ENC	DEC	EKS	ENC	DEC
32-bit ARM Cortex-M3										
w/LUT (2-way) [10]	316	860	896	1560	324	704	704	34	269	298
w/o LUT (2-way) [10]	316	344	384	1044	324	180	180	37	258	287
32-bit ARM Cortex-M4 (all implementations without LUT)										
Task parallel (1-way)	116	348	332	796	316	180	180	18	76	71
Task/data parallel (2-way)	160	592	544	1296	316	188	188	49	56	55
Data parallel (4-way)	196	596	548	1340	860	620	620	142	145	145
Fault resistance (2-way)	160	536	520	1216	316	188	188	98	143	143
Fault resistance (4-way)	196	780	784	1760	860	612	612	142	688	688
Fault resistance (4-way, CTR)	196	1864		2060	860	656	656	142	660	660

However, the HIGHT algorithm is an ARX-structured block cipher. For this reason, the bit-slicing is not an efficient approach. Furthermore, target embedded processors do not support 128-bit wise SIMD instructions. By considering the low-end microcontroller environments, we utilized the two-way or four-way data-parallel implementation to achieve the intra-redundancy. Both implementations utilized the random shuffling to make the fault attack complicated. The two-way implementation duplicates the plaintext and this allows to detect many fault injections including random byte, random bit, chosen bit pair, and instruction skip. For the random word attack, it is hard to prevent since the duplicated data can be altered together with the original data. This is protected in the four-way implementation since it contains the known answer part.

For the case of instruction skip attack, the known answer data can detect faults. This is available in the four-way parallel implementation. As we explored in Table 3, performance and security features have the trade-off relation. For this reason, the strength of fault resistance should be considered depending on the service or application.

Table 4. Comparison of fault detection capability depending on the implementation of HIGHT algorithm on target embedded processors. Red color and green color represent disadvantage and advantage of approaches, respectively.

Approach	Architecture	Shuffle	Random Word	Random Byte	Random Bit	Chosen Bit Pair	Instruction Skip
[10]	ARM	–	–	–	–	–	–
This work (2-way)	SIMD	–	√	√	√	√	–
This work (4-way)	SIMD	√	√	√	√	√	√

5. Conclusions

In this paper, the optimized HIGHT implementation on 32-bit embedded processors was investigated. For the high performance, several parallel HIGHT implementations were presented, such as task parallel, task/data parallel, and data parallel. In particular, primitive operations were fully utilized with power features of ARM processors such as barrel-shifter and four-way SIMD instruction. For the SIMD-like rotation, we utilized the padding and masking approach. In order to achieve the fault-safe implementation, the intra-instruction redundancy is utilized. The proposed implementation is secure against chosen bit pair, random bit, random byte, random word, and instruction skip attacks through four-way parallel encryption approaches.

This paper shows the new approach for fault detection on 32-bit ARM embedded processors. For this reason, we can exploit this approach for other block cipher implementations on embedded processors. Another research direction is that of optimized fault detection for high-end processors. They provide new instructions and features. These can be beneficial for securely computing on these platforms.

Author Contributions: Investigation, H.K. (Hyunjun Kim), K.J., H.K. (Hyeokdong Kwon), M.S., G.S., S.U., and H.K. (Hyunji Kim); Software, H.S. H.K. (Hyeokdong Kwon), and S.U.; Writing–original draft, H.S. and H.K. (Hyeokdong Kwon); Writing–review & editing, H.S., K.J., M.S., G.S., S.U. and H.K. (Hyunji Kim); All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services) and this work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2020R1F1A1048478). This research was financially supported by Hansung University for Hwajeong Seo.

Conflicts of Interest: The authors declare no conflicct of interest.

References

1. Seo, H.; Liu, Z. All the HIGHT You Need on Cortex–M4. In Proceedings of the International Conference on Information Security and Cryptology, Nanjing, China, 6–8 December 2019; pp. 70–83.
2. Hong, D.; Sung, J.; Hong, S.; Lim, J.; Lee, S.; Koo, B.S.; Lee, C.; Chang, D.; Lee, J.; Jeong, K.; et al. HIGHT: A new block cipher suitable for low-resource device. In Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems, Yokohama, Japan, 10–13 October 2006; pp. 46–59.
3. Eisenbarth, T.; Gong, Z.; Güneysu, T.; Heyse, S.; Indesteege, S.; Kerckhof, S.; Koeune, F.; Nad, T.; Plos, T.; Regazzoni, F.; et al. Compact implementation and performance evaluation of block ciphers in ATtiny devices. In Proceedings of the International Conference on Cryptology in Africa, Ifrance, Morocco, 10–12 July 2012; pp. 172–187.
4. Eisenbarth, T.; Kumar, S.; Paar, C.; Poschmann, A.; Uhsadel, L. A survey of lightweight-cryptography implementations. *IEEE Des. Test Comput.* **2007**, *24*, 522–533. [CrossRef]
5. Osvik, D.A.; Bos, J.W.; Stefan, D.; Canright, D. Fast software AES encryption. In Proceedings of the International Workshop on Fast Software Encryption, Seoul, Korea, 7–10 February 2010; pp. 75–93.
6. Beaulieu, R.; Shors, D.; Smith, J.; Treatman-Clark, S.; Weeks, B.; Wingers, L. The SIMON and SPECK block ciphers on AVR 8-bit microcontrollers. In Proceedings of the International Workshop on Lightweight Cryptography for Security and Privacy, Istanbul, Turkey, 1–2 September 2014; pp. 3–20.
7. Buhrow, B.; Riemer, P.; Shea, M.; Gilbert, B.; Daniel, E. Block cipher speed and energy efficiency records on the MSP430: System design trade-offs for 16-bit embedded applications. In Proceedings of the International Conference on Cryptology and Information Security in Latin America, Florianolis, Brazil, 17–19 September 2014; pp. 104–123.
8. Hong, D.; Lee, J.; Kim, D.; Kwon, D.; Ryu, K.H.; Lee, D.G. LEA: A 128-bit block cipher for fast encryption on common processors. In Proceedings of the International Workshop on Information Security Applications, Jeju Island, Korea, 19–21 August 2013; pp. 3–27.
9. Seo, H. High Speed Implementation of LEA on ARM Cortex-M3 processor. *J. Korea Inst. Inf. Commun. Eng.* **2018**, *22*, 1133–1138.
10. Seo, H.; Jeong, I.; Lee, J.; Kim, W. Compact Implementations of ARX-Based Block Ciphers on IoT Processors. *ACM Trans. Embed. Comput. Syst. (TECS)* **2018**, *17*, 60. [CrossRef]
11. Kim, B.; Cho, J.; Choi, B.; Park, J.; Seo, H. Compact Implementations of HIGHT Block Cipher on IoT Platforms. *Secur. Commun. Netw.* **2019**, *2019*, 5323578. [CrossRef]
12. Seo, H.; Azarderakhsh, R. Curve448 on 32-Bit ARM Cortex-M4. In Proceedings of the International Conference on Information Security and Cryptology, Seoul, Korea, 2–4 December 2020; pp. 125–139.
13. Seo, H.; Jalali, A.; Azarderakhsh, R. SIKE round 2 speed record on ARM Cortex-M4. In Proceedings of the International Conference on Cryptology and Network Security, Fuzhou, China, 25–27 October 2019; pp. 39–60.
14. Seo, H.; Anastasova, M.; Jalali, A.; Azarderakhsh, R. Supersingular isogeny key encapsulation (SIKE) round 2 on ARM Cortex-M4. *IEEE Trans. Comput.* **2020**. [CrossRef]
15. Seo, H. Memory efficient implementation of modular multiplication for 32-bit ARM Cortex-M4. *Appl. Sci.* **2020**, *10*, 1539. [CrossRef]
16. Patrick, C.; Yuce, B.; Ghalaty, N.F.; Schaumont, P. Lightwei ht Fault Attack Resistance in Software Using Intra-Instruction Redundancy. In Proceedings of the International Conference on Selected Areas in Cryptography, St. John's, NL, Canada, 10–12 August 2016; pp. 231–244.

17. Seo, H.; Park, T.; Ji, J.; Kim, H. Lightweight Fault Attack Resistance in Software Using Intra-instruction Redundancy, Revisited. In Proceedings of the International Workshop on Information Security Applications, Jeju Island, Korea, 24–26 August 2017; pp. 3–15.
18. Kwon, H.; Kim, Y.; Seo, S.C.; Seo, H. High-Speed Implementation of PRESENT on AVR Microcontroller. *Mathematics* **2021**, *9*, 374. [CrossRef]
19. Kwon, H.; An, S.; Kim, Y.; Kim, H.; Choi, S.J.; Jang, K.; Park, J.; Kim, H.; Seo, S.C.; Seo, H. Designing a CHAM Block Cipher on Low-End Microcontrollers for Internet of Things. *Electronics* **2020**, *9*, 1548. [CrossRef]
20. Kim, K.; Choi, S.; Kwon, H.; Kim, H.; Liu, Z.; Seo, H. PAGE-Practical AES-GCM Encryption for Low-End Microcontrollers. *Appl. Sci.* **2020**, *10*, 3131. [CrossRef]
21. Kim, K.; Choi, S.; Kwon, H.; Liu, Z.; Seo, H. FACE–LIGHT: Fast AES–CTR Mode Encryption for Low-End Microcontrollers. In Proceedings of the International Conference on Information Security and Cryptology, Seoul, Korea, 4–6 December 2019; pp. 102–114.
22. Kim, Y.; Kwon, H.; An, S.; Seo, H.; Seo, S.C. Efficient implementation of ARX-based block ciphers on 8-Bit AVR microcontrollers. *Mathematics* **2020**, *8*, 1837. [CrossRef]