Compact Implementation of CHAM Block Cipher on Low-End Microcontrollers

Hyeokdong Kwon, Hyunji Kim, Seung Ju Choi, Kyoungbae Jang, Jaehoon Park, Hyunjun Kim, and Hwajeong Seo*

IT Department, Hansung University, Seoul, South Korea,
{korlethean, bookingstore3, starj1023, p9595jh, khj930704, hwajeong84}@gmail.com

Abstract. In this paper, we presented an optimized implementation of CHAM block cipher on low-end microcontrollers. In order to accelerate the performance of the CHAM block cipher, the architecture of CHAM block cipher and the full specification of 8-bit AVR microcontrollers are efficiently utilized. First, the counter mode of operation for CHAM block cipher is optimized. A number of computations for round function are replaced to look-up table accesses. Second, multiple blocks of CHAM block cipher are computed in a parallel way for high throughput. With the parallel computation, we also presented the adopted encryption. This approach is efficient for long-length data handling. Third, the state-of-art engineering technique is fully utilized in terms of instruction level and register level. The partially unrolled 8-round based implementation is adopted, which avoids a number of word-wise rotation operations. With above optimization techniques, proposed CHAM implementations for counter mode of operation outperform the state-of-art implementations by 30.1%, 9.3%, and 10.0% for CHAM-64/128, CHAM-128/128, and CHAM-128/256, respectively.

Keywords: CHAM Block Cipher · Microcontroller · Counter Mode of Operation · Parallel Computation · Round based Encryption.

1 Introduction

Internet of Things (IoT) becomes feasible services as the technology of embedded processors are developed. In order to provide user-friendly services, IoT applications need to analyze data which should be securely encrypted before packet transmission. However, the encryption is expensive for resource-constrained IoT devices with limited computation, energy, and storage. For this reason, block cipher algorithms should be implemented in an efficient manner under certain limitations.

In this paper, we introduce optimization techniques for CHAM block cipher on 8-bit AVR microcontrollers. Optimized counter mode of CHAM block cipher and parallel computations are presented. To get compact results, we

* Corresponding Author

used platform-specific assembly-level optimizations for CHAM block ciphers (e.g. word size, number of registers, and instruction set). Proposed implementation techniques for CHAM block cipher can be used for other ARX based block ciphers, such as SPECK and SIMON, straightforwardly. Detailed contributions are as follows:

1.1 Contribution

Optimized counter mode of operation for CHAM block cipher The repeated input of counter mode of operation for CHAM block cipher can be optimized with the pre-computation. In total, 5 left rotation by 1-bit, 10 XOR, 3 ADD, and 3 left rotation by word-wise operations are replaced to 5 word-wise table accesses. The proposed method optimizes all CHAM parameters.

Parallel implementation of CHAM-64/128 The lightweight version of CHAM block cipher (i.e. CHAM-64/128) is accelerated with the parallel computation. By utilizing all registers, multiple blocks of CHAM block cipher are computed, simultaneously. We also presented an adopted encryption with the parallel computation, which is efficient when the data length is long enough.

Highly optimized source code and 8-round based implementation The state-of-art engineering technique is fully utilized with the available instruction sets and register files to achieve the high performance. The partially unrolled 8-round based implementation is adopted, which optimizes the left rotation by 8-bit wise operations.

The remainder of this paper is organized as follows. In Section 2, the basic specifications of CHAM block cipher and target AVR microcontrollers are described. In Section 3, the compact implementation of CHAM block cipher on AVR microcontrollers are described. In Section 4, the performance of proposed methods in terms of execution timing is evaluated. Finally, Section 5 concludes the paper.

2 Related Works

2.1 CHAM Block Cipher

Lightweight cryptography is a fundamental technology to optimize the hardware chip size and reduce the execution timing for low-end Internet of Things (IoT) devices. Recently, a number of block cipher algorithms have been designed for being lightweight features.

In ICISC'17, a family of lightweight block ciphers CHAM was announced by the Attached Institute of ETRI [1]. The family consists of three ciphers, including CHAM-64/128, CHAM-128/128, and CHAM-128/256. The CHAM block ciphers are of the generalized 4-branch Feistel structure based on ARX operations.

In ICISC'19, the revised version of CHAM block cipher was presented [2]. In order to prevent new related-key differential characteristics and differentials of CHAM using a SAT solver, the numbers of rounds of CHAM-64/128, CHAM-128/128, and CHAM-128/256 are increased from 80 to 88, 80 to 112, and 96 to 120, respectively.

2.2 Previous Block Cipher Implementations on 8-bit AVR Microcontrollers

Table 1. Instruction set summary for efficient CHAM implementations on 8-bit AVR microcontrollers.

asm	Operands	Description	Operation	#Clock
ADD	Rd, Rr	Add without Carry	Rd ← Rd+Rr	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd+Rr+C	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd⊕Rr	1
LSL	Rd	Logical Shift Left	C|Rd ← Rd<<1	1
ROL	Rd	Rotate Left Through Carry	C|Rd ← Rd<<1||C	1
MOV	Rd, Rr	Copy Register	Rd ← Rr	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	1
LDI	Rd, K	Load Immediate	Rd ← K	1
LD	Rd, X	Load Indirect	Rd ← (X)	2
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	3
ST	Z, Rr	Store Indirect	(Z) ← Rr	2
PUSH	Rr	Push Register on Stack	STACK ← Rr	2
POP	Rd	Pop Register from Stack	Rd ← STACK	2

The low-end 8-bit AVR platform working at 8 MHz supports 8-bit instruction set, 128 KB FLASH memory, and 4 KB RAM. The number of available registers is 32. Among them, 6 registers (i.e R26 $\sim$ R31) are reserved for address pointers and the other registers are used for general purpose. The basic arithmetic instruction takes one clock cycle, while the memory access takes two clock cycles per byte. Detailed instruction set summary for efficient CHAM-CTR implementation is given in Table 1.

A number of works devoted to improve the performance of lightweight cryptography implementations on low-end microcontrollers (e.g. 8-bit AVR). The structure of block cipher is largely divided into two categories. First, Addition, Rotation, and eXclusive-or (ARX) based block ciphers were efficiently implemented on low-end microcontrollers.

In WISA'13, LEA block cipher was by the attached institute of ETRI [3]. First implementation of LEA-128 on the 8-bit AVR microcontroller achieved 190 clock cycles per byte for the encryption [3]. In WISA'15, speed-optimized and memory-efficient LEA implementations were presented [4]. In [5], the number of

general purpose registers and the instruction set of the AVR microcontroller were fully utilized to optimize the LEA block cipher implementation. In WISA'18, general purpose registers are efficiently utilized to cache intermediate results of delta variables during the key scheduling of LEA [6].

In CHES'06, HIGHT block cipher was introduced [7]. The basic implementation of HIGHT was firstly introduced in [8]. The execution timing for encryption and decryption is 2,438 and 2,520 clock cycles per byte, respectively. In [5], efficient rotation operations were suggested and achieved high performance. In [9], speed-optimized and memory-efficient HIGHT implementations were presented.

In ICISC'17, the original CHAM on 8-bit AVR microcontrollers achieved 172, 148, and 177 clock cycles per byte for CHAM-64/128, CHAM-128/128, and CHAM-128/256, respectively [1]. In [10], 2-round based memory-efficient implementation was suggested. The work achieved 211, 187, and 223 clock cycles per bytes for CHAM-64/128, CHAM-128/128, and CHAM-128/256, respectively, with reasonably small memory footprint. In ICISC'19, revised CHAM was presented by modifying the round of CHAM [2]. AVR implementations achieved 188, 203, and 219 clock cycles per byte for CHAM-64/128, CHAM-128/128, and CHAM-128/256, respectively.

Second, Substitution Permutation Network (SPN) based block ciphers were also actively investigated. Among them, AES implementations received the high attention since the block cipher is international standard.

In [11], S-box pointer was maintained in `Z` address pointer for the fast memory access. The mix-column computation was efficiently handled with the conditional branch skip. In ICISC'19, the compact implementation of AES-CTR on microcontrollers (i.e. FACE-LIGHT) was presented [12]. With the newly designed cache table for low-end microcontrollers, implementations of AES-CTR achieved 138, 168, and 199 clock cycles per byte for 128-bit, 192-bit, and 256-bit security levels, respectively.

3 Proposed Method

3.1 Optimized CHAM-CTR mode encryption

We present the optimized CHAM-CTR mode encryption on 8-bit AVR microcontrollers. The counter mode utilizes nonce and counter as an input value. The nonce is a fixed value, and the counter indicates the order of blocks. Generally, the length of counter is set to a quarter of plaintext. CHAM-64/128, CHAM-128/128, and CHAM-128/256 assigned 16-bit, 32-bit, and 32-bit counter values, respectively. The remainder is set to nonce (i.e. 48-bit, 96-bit, and 96-bit for CHAM-64/128, CHAM-128/128, and CHAM-128/256, respectively)[1]. By utilizing the fixed nonce value, operations for several rounds are replaced to look-up table. Unlike the nonce value, the counter value increases in each encryption,

[1] 32-bit counter can be used for CHAM-64/128. In this case, pre-computed part is slightly reduced to half but still this leads to performance improvements over basic implementation.

which updates the intermediate result. For this reason, some intermediate results cannot be cached. In Figure 1, the part affected by counter value is described.

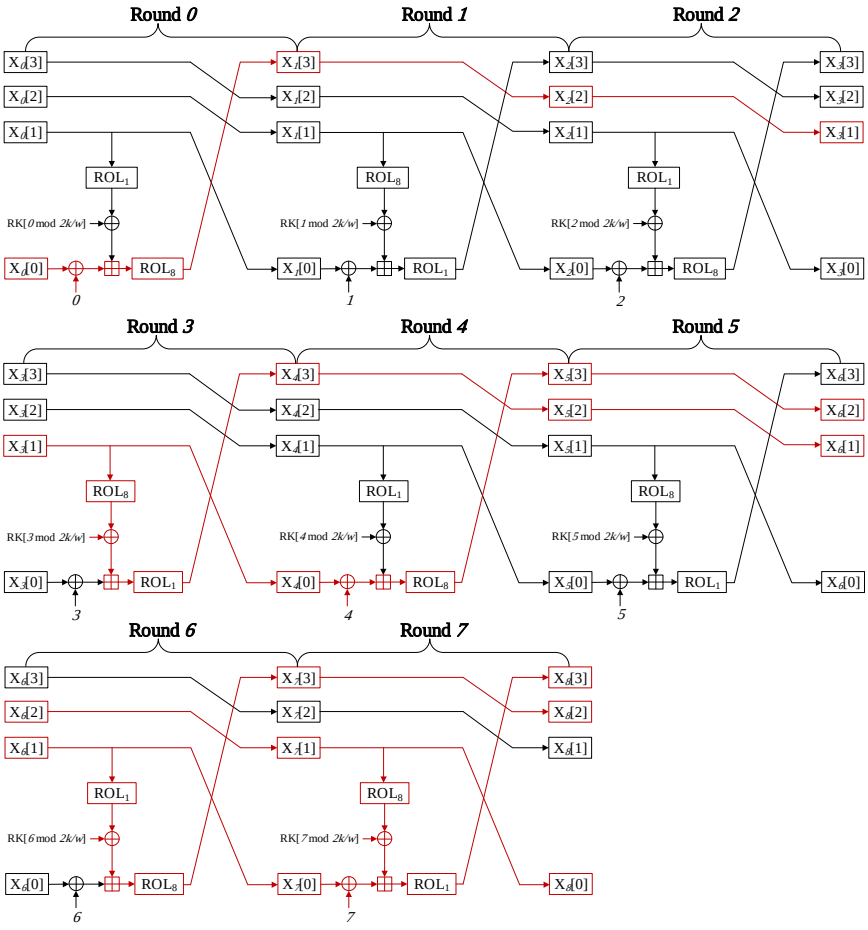

Fig. 1. Data and operation affected by counter value for CHAM round function.

Round 0 $X_0[0]$ block is counter value and the result cannot be pre-computed, while $X_0[1]$ block is nonce value and rotation to left by 1-bit and XOR operations can be pre-computed, which can be accessible through memory access. For better performance, `LDI` instruction is utilized, which directly assigns the 8-bit value in 1 clock cycles. This approaches reduces 2 clock cycles than memory access in each 8-bit assignment.

Round 1 Round 1 can be skipped because all inputs are fixed values, such as nonce, round key, and round counter. The result of this round is obtained through look-up table as $X_4[1]$ block of Round 4.

Round 2 Similar to Round 1, Round 2 is also skipped. The result is loaded to $X_5[1]$ of Round 5.

Round 3 Round 3 performs the following equation:

$$((X_3[0] \oplus i) \boxplus ((X_3[1] \lll 8) \oplus \text{RoundKey}[3]) \lll 1)$$

The i value is round counter. $X_3[0]$ block is from $X_0[3]$, which is the nonce value. The first part ($X_3[0] \oplus$ i) can be pre-calculated. This only requires word assignment. In case of CHAM-128/128, this reduces 8 clock cycles. On the other hand, $X_3[1]$ is from $X_0[0]$, which is counter value. The computation with $X_3[1]$ value cannot be pre-computed.

Round 4 CHAM requires one word shift in each round. After Round 3, all blocks are located in the initial word position. Computations with $X_4[1]$ (i.e. rotation left by 1-bit and add-round key) are optimized, while computations with $X_4[0]$ (i.e. counter-addition, two-word-addition, and rotation right by 8-bit) should be performed.

Round 5 Similar to Round 1 and Round 2, all operations can be skipped. Only loading $X_5[0]$ value for Round 8 is required. The operation takes 2 clock cycles.

Round 6 Round 6 performs the following equation:

$$((X_6[0] \oplus i) \boxplus ((X_6[1] \lll 1) \oplus \text{RoundKey}[6]) \lll 8)$$

$X_6[1]$ is from the counter value, while $X_6[0]$ is from the nonce value. The round counter addition operation is only optimized.

Round 7 Since all operations are originated from counter value, all operations should be implemented.

Round 8+ After Round 7, some operations are still pre-computed. In Round 8, $X_8[1]$ is not affected by counter value, which is represented in Figure 1. XOR operation between $X_8[1]$ and round key part can be pre-computed. The optimization is only 0.6, 0.7, and 0.7 clock cycles per byte for CHAM64/128, CHAM-128/128, and CHAM-128/256 respectively. For this reason, this case is not considered in this paper. The optimized CHAM-CTR is given in Figure 2. The green line indicates the pre-computed part. The proposed design shows that only 5 memory accesses for cache tables in Round 0, 3, 4, 6, and 7 are required.

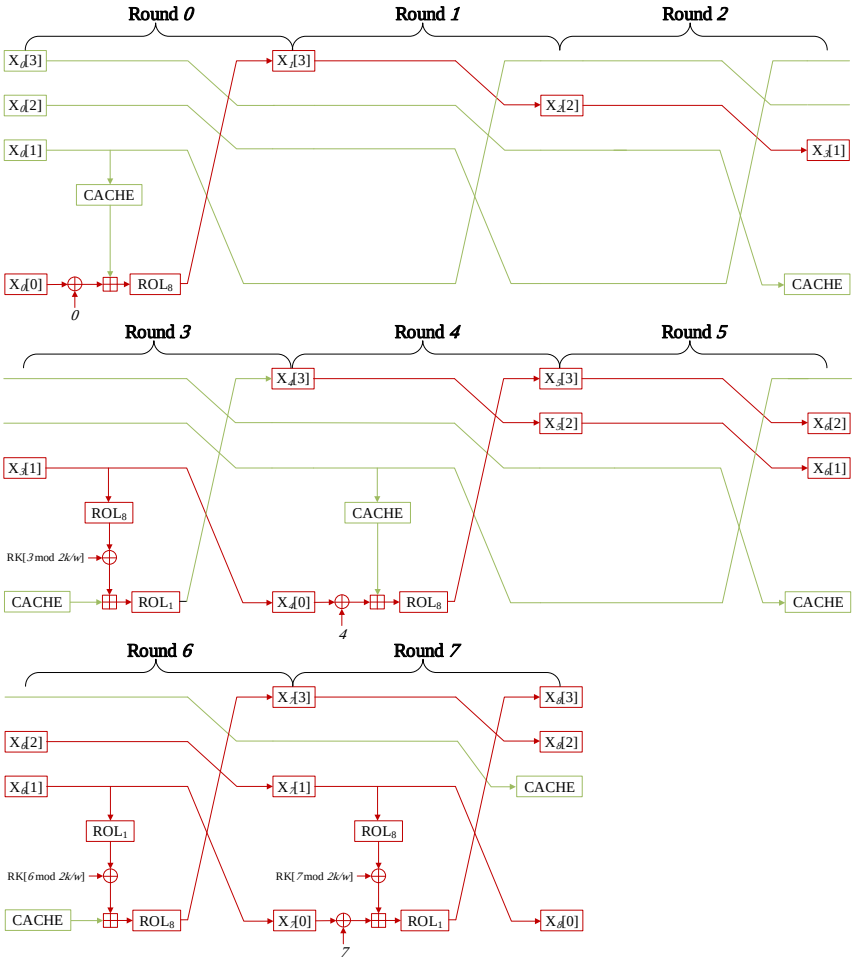

Fig. 2. Optimized CHAM-CTR mode encryption for round function.

32-bit counter for CHAM-64/128 Originally, CTR mode of operation using 32-bit counter, but in this paper we implemented 16-bit counter CHAM-64/128, since CHAM-64/128 defines the block size as 16-bit. For compatibility with existing mode of operation, it needs to implement 32-bit counter CHAM-64/128. It can be implemented by storing counter value in two blocks. In Figure 3, the counter value flow represented, the counter value is extended to $X_0[1]$ block as following figure. So we can see that some sections are no longer available pre-computation. It will result in slightly less performance than 16-bit counter CHAM-64/128.

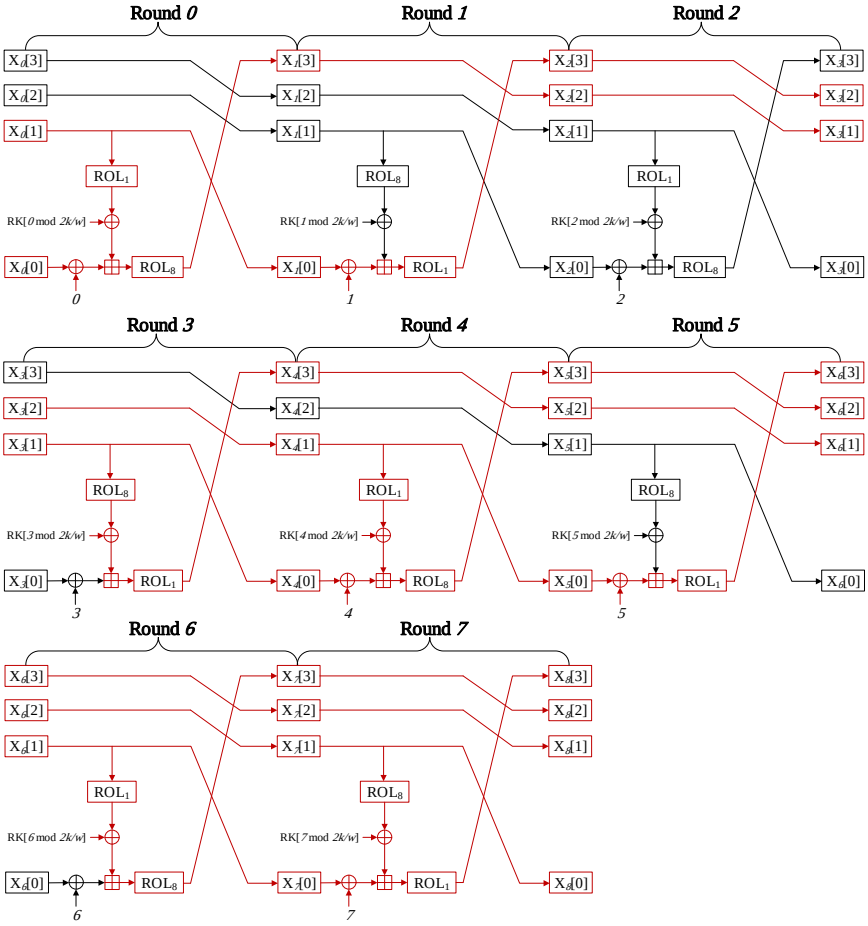

Fig. 3. The counter value flow of 32-bit counter CHAM-64/128.

Optimized memory access The round key is stored in `SRAM` and accessed through `LD` instruction. This requires 2 clock cycles per byte. By aligning the round key in 8-bit wise, the offset is only controlled with lower byte. In order to access the pre-computed value, `LDI` instruction is utilized. This requires 1 clock cycle per byte and does not require memory pointer setting.

Round counter and Pointer address optimization In the optimized version, the round counter is only used in Round 0, 4, and 7. The round key is used in Round 3, 6, and 7. Thus round counter or pointer address value is assigned directly in each round. With this approach, 8 `INC` instructions for round counter are replaced to 1 `INC` and 2 `LDI` instructions. Five `ADIW` instructions

for pointer address value are reduced to 2 `ADIW`. The `ADIW` instruction adds for pointer address in multiples of 2 and 4 for CHAM-64/128 and CHAM-128/128 or CHAM-128/256, respectively due to varied length of plaintext.

3.2 Parallel Implementations of CHAM block cipher

Parallel implementations generate multiple ciphertexts in one implementation. Two types of parallel implementations have been investigated (i.e. 2-parallel and 3-parallel). Since the implementation requires intermediate result caching, the register utilization should be optimized.

Target 8-bit AVR microcontrollers has 32 8-bit registers. For the CHAM-64/128 block cipher, 8 registers are needed. For the case of 2-parallel implementation, it requires 16 registers to save two plaintexts. In addition, the operation requires control variables, including round counter, round key, and address pointer for loading round keys and plaintexts storage. In Figure 4 (a), the register utilization for 2-parallel implementation is given. In this case, all values can be maintained in registers.

On the other hand, 3-parallel implementation requires more registers than 2-parallel implementation. Twenty four registers are used for plaintexts. Since there are not enough registers, `STACK` memory is utilized and some registers are used for multiple purposes. Detailed optimization techniques are as follows:

- **Total round variable**: By using `CPI` instruction, total round value is not maintained in the register.
- **Plaintext block** Only part of plaintext (i.e. $X_i[0]$, and $X_i[1]$) is required to perform the round. Other plaintext values are temporarily stored in `STACK` memory.
- **Address pointer**: Round key and plaintext address pointers are required in computations. However, the address pointer for plaintext is not used throughout computations. The address pointer is stored in `STACK`.
- **Round Key**: Each round key access requires word-wise memory access. By accessing byte by byte, only one register is utilized to access round key.
- **Round counter** Round counter is XORed with data in each round. After the XOR operation, the round counter is stored in `STACK`.
- **ZERO**: `R1` register is `ZERO` register. For 3-parallel implementation, `R1` register is assigned for plaintext. Some registers are temporarily initialized to act as `ZERO` register.

In Figure 4 (b), the register utilization for 3-parallel implementation is given.

The parallel computation of CHAM-64/128 is given in Algorithm 1. From Step 3 to 7, rotation operations are performed depending on the counter. In Step 8, round key addition is performed. With these round keys, multiple blocks are computed in parallel way. From Step 11 to 15, rotation operations are performed depending on the counter. From Step 16 to 19, the intermediate result is relocated by word-wise.

There are limitations to implement the parallel version of CHAM-128/128 and CHAM-128/256. CHAM-128/128 and CHAM-128/256 have twice much

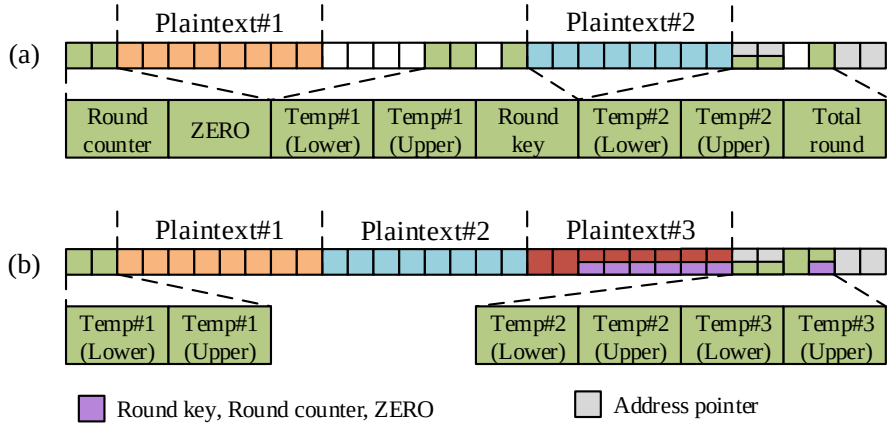

Fig. 4. Register alignment for (a) 2-parallel and (b) 3-parallel of CHAM-64/128 implementations. Each block represents one register. Two color in one register is used for various purposes.

longer plaintext then CHAM-64/128. For this reason, parallel implementations for these algorithms are not considered.

3.3 Adaptive Encryption of CHAM block cipher

The parallel computation is effective for huge data handling. The parallel computation can be applied to adaptive encryption [13]. The adaptive encryption performs parallel encryption operations when the length of data is long enough. When only one block is remained, single block encryption is performed. Detailed descriptions for 2-way adaptive encryption are given in Algorithm 2. The method performs 2-parallel computations. From Step 3 to 5, parallel computation is performed. Afterward, from Step 6 to 8, sequential computation is performed for remaining data.

3.4 Optimized implementations of primitive operations

The implementation is based on 8-round based implementation. In every 8 round, the 8-bit rotation operation on data for CHAM-64/128 is optimized away. For CHAM-128/128 and CHAM-128/256, still 16-bit wise rotation operation is required. This is performed in 16-bit wise move operation `MOVW`. The other optimized 16/32-bit word rotation operations are given in Table 2.

4 Evaluation

Proposed implementations of CHAM block cipher were evaluated on low-end 8-bit AVR microcontollers. The performance was measured in execution time

Algorithm 1 Parallel implementation of CHAM-64/128.

Input: Plaintext blocks $(X[0][0 \sim 3], ..., X[\#parallel - 1][0 \sim 3])$.
Output: Ciphertext blocks $(X[0][0 \sim 3], ..., X[\#parallel - 1][0 \sim 3])$.

for $i = 0$ to $\#round$ **do**
  for $j = 0$ to $\#parallel$ **do**
    if i mod $2 == 0$ **then**
      $tmp[j][0] \leftarrow ROL_1(X[j][1])$
    else
      $tmp[j][0] \leftarrow ROL_8(X[j][1])$
    end if

    $tmp[j][1] \leftarrow tmp[j][0] \oplus RK[i \bmod 16]$ `//round key access optimization`
    $tmp[j][2] \leftarrow X[j][0] \oplus i$
    $tmp[j][3] \leftarrow tmp[j][1]tmp[j][2]$

    if i mod $2 == 0$ **then**
      $tmp[j][4] \leftarrow ROL_8(tmp[j][3])$
    else
      $tmp[j][4] \leftarrow ROL_1(tmp[j][3])$
    end if

    $X[j][0] \leftarrow X[j][1]$
    $X[j][1] \leftarrow X[j][2]$
    $X[j][2] \leftarrow X[j][3]$
    $X[j][3] \leftarrow tmp[j][4]$
  end for
end for

Table 2. Optimized 16/32-bit word rotation operations on 8-bit AVR microcontroller.

16-bit ROL_1	16-bit ROL_8	32-bit ROL_1	32-bit ROL_8

`LSL LOW`
`ROL HIGH`
`ADC LOW, ZERO`
	`MOV TEMP, LOW`
	`MOV LOW, HIGH`
	`MOV HIGH, TEMP`
		`LSL R0`
		`ROL R1`
		`ROL R2`
		`ROL R3`
		`ADC R0, ZERO`
			`MOV TEMP, R3`
			`MOV R3, R2`
			`MOV R2, R1`
			`MOV R1, R0`
			`MOV R0, TEMP`

3 cycles	3 cycles	5 cycles	5 cycles

(clock cycles per byte). The software was implemented over `Atmel Studio 7` and the code was complied in `-O2` option. Comparison result of CHAM block cipher is given in Figure 5. Compared with previous works by [2], CHAM-64/128, CHAM-128/128, and CHAM-128/256 are improved by 11.7%, 6.5%, and 7.4%.

Algorithm 2 2-way adaptive encryption for CHAM64/128.

Input: Number of blocks N, Plaintext blocks $P \in \{P_1, P_2, ..., P_N\}$.
Output: Ciphertext blocks
$C \in \{C_1, C_2, ..., C_N\}$.

1: $n \leftarrow \frac{N}{2}$
2: $m \leftarrow N \bmod 2$

3: **for** $i = 0$ to n **do**
4: $\quad \{C_{2 \cdot i+1}, C_{2 \cdot i}\} \leftarrow ENC(\{P_{2 \cdot i+1}, P_{2 \cdot i}\})$ `//parallel computation`
5: **end for**

6: **if** m **then**
7: $\quad C_{2 \cdot n+2} \leftarrow ENC(P_{2 \cdot i+2})$ `//sequential computation`
8: **end if**
9: **return** C

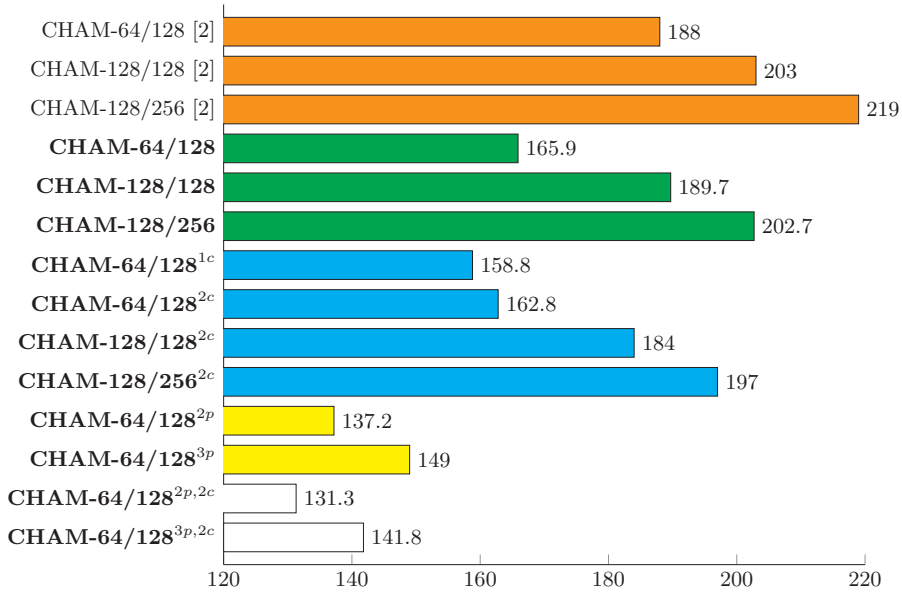

Fig. 5. Comparison of execution time for CHAM implementations on 8-Bit AVR microcontrollers under the fixed-key scenario in terms of clock cycles per byte, 1c: counter mode of operation (16-bit counter), 2c: counter mode of operation (32-bit counter), 2p: 2-parallel, 3p: 3-parallel.

The performance enhancement comes from the 8-round implementation and fast memory for round key. For the case of counter mode of operation, the implementation improved the performance further by 4.2%, 3.0%, and 2.8% for CHAM-64/128, CHAM-128/128, and CHAM-128/256, respectively. The implementation

utilized the repeated data (i.e. nonce) to improve the performance. The parallel implementation of CHAM-64/128 shows 137.2 and 149 clock cycles per byte for 2-parallel and 3-parallel versions, respectively. Due to limited number of registers, 2-parallel based implementation shows better performance than that of 3-parallel.

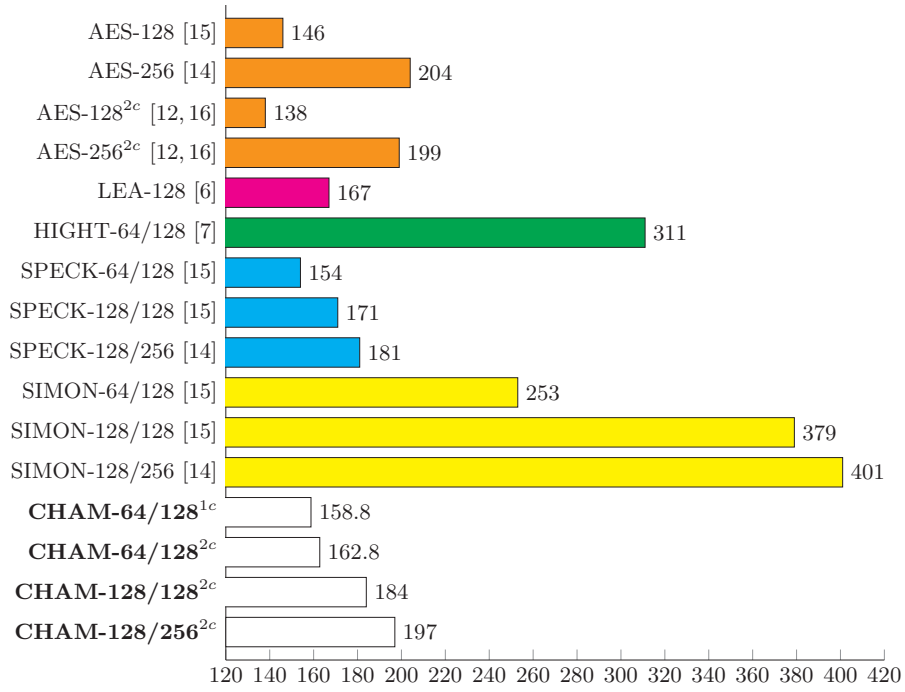

Fig. 6. Comparison of execution time for CHAM with other block ciphers on 8-Bit AVR microcontrollers in terms of clock cycles per byte, 1c: counter mode of operation (16-bit counter), 2c: counter mode of operation (32-bit counter).

We also compared the result with other block ciphers in Figure 6. SPN based AES implementation achieved the highest performance among them, because AES is designed to fit into 8-bit word architecture. Among ARX implementations, SPECK shows the fastest performance. Second winner is proposed CHAM block cipher.

5 Conclusion

In this paper, we presented optimization implementations of lightweight CHAM block ciphers on low-end 8-bit AVR microcontrollers. Proposed techniques include pre-computed counter mode of operation, parallel implementation, and

optimized primitive operations. We evaluated our implementations in terms of execution time. The result shows that our implementations achieved fast execution timing for practical IoT applications.

Future work is applying proposed method to other ARX–based block ciphers, such as SIMON and SPECK. Furthermore, other microcontrollers including 16-bit MSP and 32-bit ARM will be investigated.

6 Acknowledgement

This work was partly supported as part of Military Crypto Research Center(UD170109ED) funded by Defense Acquisition Program Administration(DAPA) and Agency for Defense Development(ADD) and 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).

References

1. B. Koo, D. Roh, H. Kim, Y. Jung, D.-G. Lee, and D. Kwon, "CHAM: A family of lightweight block ciphers for resource-constrained devices," in *International Conference on Information Security and Cryptology (ICISC'17)*, 2017.
2. D. Roh, B. Koo, Y. Jung, I. W. Jeong, D.-G. Lee, D. Kwon, and W.-H. Kim, "Revised version of block cipher CHAM," in *International Conference on Information Security and Cryptology*, pp. 1–19, Springer, 2019.
3. D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, and D.-G. Lee, "LEA: A 128-bit block cipher for fast encryption on common processors," in *International Workshop on Information Security Applications*, pp. 3–27, Springer, 2013.
4. H. Seo, Z. Liu, J. Choi, T. Park, and H. Kim, "Compact implementations of LEA block cipher for low-end microprocessors," in *International Workshop on Information Security Applications*, pp. 28–40, Springer, 2015.
5. H. Seo, I. Jeong, J. Lee, and W.-H. Kim, "Compact implementations of ARX-based block ciphers on IoT processors," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 3, pp. 1–16, 2018.
6. H. Seo, K. An, and H. Kwon, "Compact LEA and HIGHT implementations on 8-bit AVR and 16-bit MSP processors," in *International Workshop on Information Security Applications*, pp. 253–265, Springer, 2018.
7. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, *et al.*, "HIGHT: A new block cipher suitable for low-resource device," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 46–59, Springer, 2006.
8. T. Eisenbarth, Z. Gong, T. Güneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni, *et al.*, "Compact implementation and performance evaluation of block ciphers in ATtiny devices," in *International Conference on Cryptology in Africa*, pp. 172–187, Springer, 2012.

9. B. Kim, J. Cho, B. Choi, J. Park, and H. Seo, "Compact implementations of HIGHT block cipher on IoT platforms," *Security and Communication Networks*, vol. 2019, 2019.
10. H. Seo, "Memory-efficient implementation of ultra-lightweight block cipher algorithm cham on low-end 8-bit avr processors," *Journal of the Korea Institute of Information Security Cryptology*, pp. 545–550, 2018.
11. D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, "Fast software AES encryption," in *International Workshop on Fast Software Encryption*, pp. 75–93, Springer, 2010.
12. K. Kim, S. Choi, H. Kwon, Z. Liu, and H. Seo, "FACE–LIGHT: Fast AES–CTR mode encryption for low-end microcontrollers," in *International Conference on Information Security and Cryptology*, pp. 102–114, Springer, 2019.
13. T. Park, H. Seo, S. Lee, and H. Kim, "Secure data encryption for cloud-based human care services," *Journal of Sensors*, vol. 2018, 2018.
14. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK block ciphers on AVR 8-bit microcontrollers," in *International Workshop on Lightweight Cryptography for Security and Privacy*, pp. 3–20, Springer, 2014.
15. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "Simon and speck: Block ciphers for the internet of things," *IACR Cryptology ePrint Archive*, vol. 2015, p. 585, 2015.
16. K. Kim, S. Choi, H. Kwon, H. Kim, Z. Liu, and H. Seo, "PAGE—practical AES-GCM encryption for low-end microcontrollers," *Applied Sciences*, vol. 10, no. 9, p. 3131, 2020.