Research and Design of Block Cipher Special Instruction Set

Zhongxiang Chang, Zheng Dai, Wei Song, Hang Zhu, Mengfei Chen

College of Information and Communication, National University of Defense Technology, Wuhan, China

Corresponding author: changzhongxiang0@126.com

Abstract. The existing instructions are used to implement the block cipher algorithm with low efficient. So this paper classifies the operations in the cryptographic algorithm, and analyses the characteristics of bit width, combination and parallel mode in the cipher algorithm of different operations. Further develop the parallelism of S-box and shift instructions, improve the hybrid computing ability of arithmetic operation instructions, research and design new types of instructions for data position transformation, such as extract shift, shift insertion, data grouping and data mixing. At the last, this paper implements the cipher algorithm with the new instruction, such as DES, IDEA, BEST and other block cipher algorithms. The number of instructions is 14% ~ 35% of Haswell processor, and 40% to 68% of the COBRA cipher processor.

1. Introduction

Block ciphers are widely used in data protection and secure transmission because they are fast, easy to standardize, and easy to implement in hardware and software. With the explosive development of small and micro-intelligent mobile system terminals such as handheld devices and wearable devices, the traditional fixed algorithm encryption causes the information to have great security risks and cannot meet security requirements. General-purpose processor encryption can't meet high speed because the instructions are not specificity for block ciphers. How to simultaneously consider the high-performance and high-flexibility requirements of block cipher processing has become an urgent problem to be solved.

The main methods for improving cryptographic processing performance and supporting algorithms replaceable are instruction extension and coprocessor methods and special processors. But no matter which way to implement the cryptographic algorithm, its efficiency depends on the matching degree between the instructions and the coding of cipher. At present, some special instructions have appeared for different implementation methods, but there are some shortcomings. Such as PPERM, CROSS, BFLY, IBFLY and OMFLIP[1], which are designed for bit permutation, but the instructions are complicated, and the efficiency is only for fixed bits, which cannot meet the performance requirements of special replacement such as shifting and filtering, and cannot satisfy the larger width performance. The flexibility is very low for bit permutation. At the same time, it needs to calculate a large amount of control information in advance. ROTR, ROTL, SHR, SHL, GRP, PEX and PDEP, MUX and MIX[2]instructions are designed for some special permutation. Each instruction can implement a specific function. Although it can solve the efficiency problem, it integrates each instruction into one processor, where resource is unacceptable. The number of instructions has increased by multiple, and the processing efficiency has dropped dramatically when the instruction achieves larger width data processing. The S-box instructions use multiple resources to achieve parallelism, but they do not
consider the association between the S-box contents in the cipher algorithm. Arithmetic operation instructions also support multiple parallel executions, but they are not specifically supported for combination of XOR and modulo operations in the new lightweight block ciphers.

This paper analyzes the types of operations, data parallelism and data correlation of more than forty types of block cipher algorithms, studies and designs the S-box, arithmetic operation, shift, data group, data mix, extract-shift and shift-insert instructions, which greatly improves the degree of matching of the cipher algorithm, and reduces the number of instructions required to implement the cipher algorithms.

2. Block cipher feature analysis

2.1 Basic operations
The block cipher is a strong cryptographic function by iterating a weak function with a certain confusion and diffusion structure, and the basic operation in the algorithm is relatively simple. Based on this, this paper statistically analyzes the basic operational characteristics of more than forty block cipher algorithms such as DES, IDEA, AES, FOX, PP-2, HIGHT, LBlock and TWINE[3,4,5], As shown in table 1.

Table 1. Basic operation type in block cipher.

| Operation type            | Operational characteristics |
|---------------------------|-----------------------------|
| S-box                     | 8-8, 6-4, 4-4               |
| Bit permutation           | 32-32, 32-48, 56-48, 64-64, 128-128 |
| Modular multiplication    | $2^8$, $2^{16}$, $2^{16}+1$, $2^{32}$ |
| Modular Addition          | $2^8$, $2^{16}$, $2^{32}$   |
| Finite field multiplication| GF($2^8$)                   |
| Logic operation           | 8, 16, 32, 64, 128          |
| Shift                     | 8, 16, 32, 64               |

x-y : x-bit input, y-bit output

The types of units in the cipher algorithm are relatively fixed, and the basic operations have logical operations (AND, OR, XOR), arithmetic operations (addition, multiplication) and shift. Complex operations include S-box, bit permutation and finite field multiplication. In addition to the above operations, there is also the existence of a linear feedback shift register in the key generation process of very few algorithms. The data widths corresponding to various operations in the algorithm are various, and different operations exhibit unique characteristics. For example, the data width of shift operation is 8, 16, 32, and 64. The data width of the bit permutation is 32, 48, 56, 64 and 128, and the data width is no more than 8 bits of the S-box and finite field multiplication.

2.2 Parallelism
The parallelism of operations in block ciphers refers to the data width in the horizontal direction. Regardless of the structural model, most of the data of large width (64, 128, 192) is split into sub-blocks with smaller width. The sub-block width is generally no more than 32 bits. The split sub-blocks can be operated as a whole or further grouped, and the width of the group is generally not more than 8 bits.

In the AES algorithm, the plaintext is 128 bits, the byte permutation is 8 bits in the S-box, the line shift is 32 bits and the column mixed is 8 bits. DES, CAST-128 and other algorithms divide the 64-bit plaintext into two parts, each part is 32 bits. Inside the sub-blocks, there are 32 bits bit permutation
and 6 bits S-box. RC6 internally has 32-bit shift, XOR and so on. The IDEA algorithm divides the data into four 16 bits, and the round operation includes 16 bits XOR, modular addition and modular multiplication.

In the block cipher algorithm, when further splitting data into smaller blocks, the split sub-blocks perform the same operation in most cases, but there are also some algorithms whose sub-blocks are completely different. In the IDEA algorithm, the plaintext is split into four 16 bits sub-blocks, the first and third sub-blocks are modular multiplication, and the second and fourth sub-blocks are modulo-add operations.

2.3 Correlation
The correlation of the operation means that the association between adjacent arithmetic units is considered from the vertical direction. There are a large number of related operations read-after-write in the cipher algorithm. This correlation exists between the rounds of the transition and between the operations in the round. However, the combination of different operations is also very limited because the types of operations are relatively fixed. After analyzing a large number of block cipher algorithms, it is found that the data-related operations have certain characteristics.

S-box, finite field multiplication, bit permutation, modular addition and modular multiplication are often accompanied by XOR operations, and XOR objects are usually key values, AES, DES, BEST, Twofish, SAFTER+, RC6 and other block ciphers have this situation. Shift is also accompanied by XOR, but the XOR object is another group, not the key value. The HIGHT algorithm has this situation.

There are also a few special combinations of bit permutation and S-box, modulo addition and shift, shift and S-box. Those can be found directly in the algorithm. In addition to the above, there are some implicit combinations. For example, bit permutation is implemented by the basic instructions that performance is difficult to meet the requirements, which requires the development of new instructions, supporting the serial combination of multiple basic instructions for improving the processing efficiency.

3. Specific instruction set research
At present, the traditional CISC processor integrates logic operations, arithmetic operations, shifting, parallel extraction and parallel insertion, etc. Parallelism and extensibility is very poor. It is very inefficient even with common cryptographic operations. Table 2 shows some example with 32-bit general purpose processors.

| Operation type | General instructions | Number |
|----------------|----------------------|--------|
| 64-bit cyclic shift | 4 logical shift, 2 xor | 6      |
| four 8-bit cyclic shift | 2 logical shift, 4 xor, 6 cyclic shift, 6 and | 18     |
| 64-bit bit permutation | N(N≤32) xor, N-1xor parallel extraction | 2N-1   |
| four 8-8 S-box | 4or, 3shift, 3xor, 4 load | 14     |

In some special processors, the parallel processing capability of the instruction is considered. For example, the S-box instruction in the COBRA processor considers that one instruction supports four 8-8 S-box operations, and requires 1024 byte storage space. But in the AES algorithm, the byte substitution transform has only one fixed S-box, resulting in a waste of storage resources. The bit permutation can be implemented by one or two instructions. But it is necessary to calculate a large amount of configuration information in advance due to excessive emphasis on performance, and an additional configuration register is required. Especially when the processing data width is large, this method needs to continuously update the configuration register data, and the processing efficiency is greatly reduced.
Considering the requirements of performance and flexibility, the rapid implementation of cipher algorithms is studied. In the basic logic operation, the result is exactly the same as the non-group operation, and no special design is required, and the existing instruction can be used. The finite field multiplication is matrix multiplication on the GF($2^k$) domain. The instruction only needs to support four sets of byte multiplications to run in parallel. It is relatively simple and will not be described in detail. The following is a detailed analysis of arithmetic operation, shift, bit permutation and S-box.

### 3.1 Arithmetic operation

Arithmetic operations include modular multiplication and modular addition. The modulus required for modular addition is $2^8$ and $2^{16}$. The modulus required for modular multiplication is $2^8$, $2^{16}$ and $2^{16}+1$. For security purposes, it is usually necessary to use all zero sub-blocks as other data, such as the IDEA algorithm and 0 sub-block which should be replaced by $2^{16}$. Therefore, the instruction needs to support three functions of modulo $2^8$, $2^{16}$ and $2^{16}+1$. At the same time, a single block needs to support four-way modulo $2^8$ simultaneous operation or two-way modulo $2^{16}$ simultaneous operation or two modulo $2^{16}+1$ simultaneous operations, matching parallel requirements in the cryptographic algorithm. In addition to above, a single instruction also needs to support a combination of different arithmetic operations for some special forms. As shown in Figure 1, it supports modular plus and XOR to meet the requirements of BEST and other algorithms. It supports modular multiplication and modular plus combination operations for the requirements of IDEA algorithm. All of the above are horizontal splitting, which improves parallelism and does not reduce the working frequency. In the vertical direction, although there is also a combination of instruction shift and arithmetic operation, but the arithmetic operation usually has a large critical path, a single instruction supports the combination of shift and arithmetic operations, which greatly reduces the overall operating frequency of the processor. The combination of vertical functions is not supported.

![Fig. 1 Combination mode](image)

### 3.2 Shift

The shift includes logical shift and cyclic shift. In the following description, shift means cyclic shift and logical shift by default, data widths are 8, 16, 32, 64, etc. The width of the shift may be further increased. The shift instruction need to consider the horizontal parallelism and the scalability of the instruction. For example, the efficiency of multiple instructions to achieve greater width shift. A single instruction supports four 8-bit shifts or two 16-bit shift or one 32-bit shift for parallelism. However, it has low effectiveness when the number of shift steps is different in the BEST algorithm. So single instruction needs to support four bytes for independent shifting, but the required number of shift step control codes is relatively large, and it is unrealistic to use only the instruction encoding bits. Therefore, shifting need to support the four-byte shift function by means of the register indirect addressing function. As shown in Fig. 2(a), when $S1 = S2 = S3 = S4$, it is derived from the immediate value in the instruction code, when $S1 \neq S2 \neq S3 \neq S4$, it is derived from the register. In Fig. 2(b), it is same as Fig. 2(a). The hardware implementation of parallel shifting is design in the literature [6,7]. It can support the pre-XOR or post-XOR function.

![Fig. 2 Cyclic shift](image)
3.3 Bit permutation
The bit permutation in the cipher algorithm is more complicated, and it is divided into peer-to-peer permutation, extended permutation and compression permutation. The peer-to-peer permutation can implement others quickly. Therefore, the focus is on the peer-to-peer permutations.

A peer-to-peer permutation can be thought of as the process of transforming a fixed ascending or descending sequence into an arbitrary sequence, which equivalent to the inverse of the process of transforming an arbitrary sequence into a fixed ascending or descending sequence. Then, the steps of the sorting algorithm can be simulated to improve the efficiency of the bit permutation. The merged sorting algorithm has a worst time complexity and is an ideal choice through analyzes various sorting methods.

There be an initial sequence \{7,6,5,4,3,2,1,0\}, the destination sequence \{0,1,2,3,4,5,6,7\}, and the sequence is sorted according to the algorithm of merge sorting. The process is as follows:

- **Initial sequence:** (7,6,5,4,3,2,1,0)
- **First merger:** (3,7) (2,6) (1,5) (0,4)
- **Secondary merger:** (1,3,5,7) (0,2,4,6)
- **Third merger:** (0,1,2,3,4,5,6,7)

That is to say, if the input is a fixed ascending or descending sequence, the inverse process of the merge sorting can be arranged in any order by log\_n operations. The inverse of the merge sort:

- **Initial sequence:** (0,1,2,3,4,5,6,7)
- **First group:** (1,3,5,7) (0,2,4,6)
- **Secondary group:** (3,7) (2,6) (1,5) (0,4)
- **Third group:** (7,6,5,4,3,2,1,0)

Through the reverse grouping process of the merge sorting algorithm, the n-bit peer-to-peer permutation operation can be completed up to log\_n times, which can improve the processing speed of the peer-to-peer permutation, which is an ideal method. In other words, the instruction need to support data grouping and data mixing. As shown in Fig. 3(a), the data group is grouped according to the control sequence RC, and the data sequence is mixed as shown in FIG. 3(b), and the source sequence RS is mixed according to the control sequence RC. The hardware implementation of data grouping and data mixing has been extensively studied in the literature [1, 6]. It supports pre-XOR and post-XOR function. However, if it is combined with S-box, modular multiplication, etc. The critical path is too long and is not considered for the time being.

![Fig.3 data group and data mix](image)

3.4 S-box
The most common S-box are 8-8, 4-4, and 6-4. The cipher algorithm usually divides the data into multiple groups to complete the S-box operation. For example, there are sixteen 8-8 S-box with the same content in AES. One S-box needs 256 bytes of storage space. There are eight 6-4 S-box in DES with different contents. The total of 128 bytes of storage space is needed. Therefore, a single instruction needs to support as many parallel operations as possible for the S-box. For a 32-bit processor, a single S-box instruction needs to support 4 channels 8-8, 8 channels 4-4, and 8 channels 6-4 parallel execution. At the same time, the S-box in most cipher algorithms are the same, only the S-box of very few algorithms require a large storage space, so it requires 512 bytes of storage space. As shown in Fig. 4, in order to maximize the parallelism and meet the algorithm requirements, the source data width is 48 bits in the 6-4 mode. The S-box is not a critical path and can support pre-XOR or post-XOR functions.
3.5 Extension

In the cipher algorithm, the width of existing shifts and bit permutation often appear to be larger than the maximum width that a single instruction can support, and there is a tendency to further increase. For arithmetic operations, the operation of each group is not fixed and the combination forms are various when divided into four groups of parallel operations. The input data must be quickly adjusted and then operated when the function is determined. Therefore, it is also necessary to consider the extended function of the instruction to ensure the processing efficiency of the larger bit width data.

The large width shift operation must be composed of a plurality of 32-bit data when the storage width of the data is 32. So the set instruction only needs to support two register cascade shifts. If the 128-bit left cyclic shift function is implemented, the initial data is stored in the R1, R2, R3, and R4 registers. R1 and R2 are cascade-shifted to take the value of R1. R2 and R3 are cascade-shifted to take R2. R3 and R4 are cascade-shifted to take R3. R4 and R1 are cascade-shifted to take R4.

The original data of bit permutation needs to be quickly grouped and then replaced within the group, which can greatly improve the efficiency of peer-to-peer permutation. As shown in Fig. 5(a), the source data bit width (M) is larger than the processing width (N) of a single instruction. The source data is divided into 1, ..., ⌈M/N⌉ group according to each set of N-bit data. The data selected in each group to be stored in the first group of the target sequence and XOR for each data with extraction and shift operations. They do not affect each other in the subsequent permutation. If a single instruction can support the operation of shifting the data after the extraction, the consumption of the shift instruction during the grouping process can be reduced, and the speed of the data grouping is nearly doubled. As shown in Fig. 5(b), parallel extraction of the source data RS is completed according to the control sequence RC, and the shift function of the data after parallel extraction is completed according to shift steps S.

At the same time, it can be found that the data group function can be realized quickly by using the extract-shift. The inverse process of the extract-shift can quickly complete the data mix function, which only need two instructions. The result space of data grouping is only a subset of the two extract-shift instruction combinations. The result space of the data mixing is only a subset of the two shift-insert operation combinations. The hardware resources of these kinds of instructions can be shared, which can greatly reduce consumption of hardware resources.
The arithmetic operations performed on the data corresponding to each position after grouping are relatively fixed. The width of each group is a multiple of 8. The sequential adjustment can be regarded as a special bit permutation operation, which can utilize hardware resources required for extract-shift or shift-insert. Special byte permutation instructions and half-word permutation instructions can order adjustment of the input data quickly.

4. Specific instruction set design
Generally speaking, instructions in processor need two source registers and one destination register. Therefore, if you want to use correlation for improving the serial processing capability. You must consider the critical length and the source registers. It can be known from the above analysis that only the immediate shift instruction does not increase the number of registers when supporting the XOR function. Therefore, this section only lists the basic operation instructions. The instructions that support the serial function are not listed. Users can develop serial functions under the appropriate processor architecture according to actual needs.

4.1 Bit permutation instruction
Data group and Data mix Data group divides its data bits (Rs2) into two subsets depending on the corresponding control bits(Rs1): if a control bit is 1, that data bit is grouped right; if a control bit is 0, that data bit is grouped left. The two sets of data locations are exchanged by .S, or one set of data is split into two groups, and three sets of data are formed. As shown in Fig. 3(a). Data mix is the inverse process of the data group. The source data in Rs2 and the control sequence in Rs1.

Extract-shift and Shift-insert Extract-shift means the source data Rs2 is subjected to parallel extraction operation according to the control sequence Rs1, and the result of parallel extraction is cyclically shifted according to the shift step number .S. Shift-insert is the inverse process of the extract shift, the source data Rs2 according to the number of shift steps .S, and then performs parallel extraction operation on the shifted source data according to the control sequence Rs1.

Byte and half-word permutation Supports byte and half-word order conversion, but does not support all permutations. The result space is only a subset of the extract-shift.

Table 3. Bit permutation instruction

| Function                      | Format       |
|-------------------------------|--------------|
| Data group                    | PEXGRP<.S> Rd, Rs1, Rs2 |
| Data mix                      | PDEPMIX<.S>Rd, Rs1, Rs2 |
| Extract-shift                 | PEXR<.S> Rd, Rs1, Rs2 |
| Shift-insert                  | RPDEP<.S> Rd, Rs1, Rs2 |
| Byte and half-word permutation| PERB Rd, Rs1, #imm4   |
|                              | PERHW Rd, Rs1, #imm2   |
| < > optional                  |               |

4.2 Shift instruction
As shown in table 4, X represents the mode of shifting, and takes values of 8, 16, 32. J represents a cascade shift. I represents an immediate shift. [R,L] means pick one of two. The shift mode and the number of steps are identical When it is an immediate shift, which performing four 8-bit parallel shifts or two 16-bit parallel shifts,. The number of steps are different when it is a register shift, which
perform four 8-bit parallel shifts or two 16-bit parallel shifts. The instruction representation is based on #imm5 or Rs2. The lower 9 bits or the lower 13 bits in the middle perform the shift function.

| Function                        | Format                                      |
|---------------------------------|---------------------------------------------|
| Left or Right cyclic shift      | RO[R,L]IX Rd, Rs1,#imm5                     |
|                                 | RO[R,L]X Rd, Rs1, Rs2                       |
|                                 | RO[R,L]J Rd, Rs1, Rs2, #imm5                |
| Left or Right logical shift     | SH [R,L]IX Rd, Rs1,#imm5                    |
|                                 | SH[R,L]X Rd, Rs1, Rs2                       |
|                                 | SH[R,L]J Rd, Rs1, Rs2, #imm5                |

[R,L] R means Right, L means Left

### Table 4. Shift instruction

4.3 Arithmetic operation instruction
As shown in table 5, X represents the bit width of the modulo addition and the modulo multiplication, and the values are 8, 16 or 32. ADD represents the modulo addition, MUL represents the modulo multiplication, MUL_ADD represents the modulo multiplication and modulo addition and MUL16_1 represents the modulo $2^{16}+1$, and the hybrid operation is performed. The operation corresponding to each set of data is shown in Fig. 1.

| Function                        | Format                                      |
|---------------------------------|---------------------------------------------|
| Modular Addition                | MODADDX Rd, Rs1, Rs2                       |
| Modular multiplication          | MODMULX Rd, Rs1, Rs2                       |
|                                 | MODMUL16_1 Rd, Rs1, Rs2                    |
| mix                             | MODMUL_ADDX Rd, Rs1, Rs2                   |

4.4 S-box instruction
As shown in table 6, mTn indicates the type of lookup table, which is divided into 8-8, 4-4, and 6-4. <> indicates optional, and appears only in 6-4 mode. PAGE indicates the page number of the lookup table. The number of different types of lookup table pages is different. 8-8 only has page 0, and the highest byte and the next highest byte share the lookup table space, and the lowest byte and the second byte share the lookup table space; 4-4 has pages 0–8, and 6-4 has pages 0–4.

| Function                        | Format                                      |
|---------------------------------|---------------------------------------------|
| Operation                       | SBOXmTn PAGE Rd, Rs1, < Rs2>                |
| Configuration                   | CFG_SBOXmTn PAGE Rd, Rs1, < Rs2>           |
5. Performance evaluation

The basic operation types in the cipher algorithm are limited. Only some typical cipher algorithms need to be selected, and the cryptographic operation types should be covered as much as possible to carry out key analysis. Based on this, this paper selects representative DES, IDEA and BEST. In order to evaluate the performance more objectively, the Intel Haswell[9,10] architecture instruction set and the COBRA coprocessor are compared respectively, and the basic processing width is limited to 32 bits when the function is unchanged.

| Algorithm | Number of instructions |
|-----------|------------------------|
| **DES 64** |                         |
| Initial   | Haswell: 45, COBRA: 6, My esign: 10 |
| Round     | Haswell: 87, COBRA: 18, My esign: 12 |
| Output    | Haswell: 87, COBRA: 18, My esign: 12 |
| **IDEA 64** |                         |
| Initial   | Haswell: 12, COBRA: 6, My esign: 4 |
| Round     | Haswell: 23, COBRA: 12, My esign: 8 |
| Output    | Haswell: 12, COBRA: 6, My esign: 4 |
| **BEST 64** |                         |
| Initial   | Haswell: 8, COBRA: 4, My esign: 4 |
| Round     | Haswell: 32, COBRA: 16, My esign: 6 |
| Output    | Haswell: 8, COBRA: 4, My esign: 4 |

As shown in the table, there are a large number of permutations and S-boxes in DES. This design can use the extract-shift and shift-insert to quickly group data, and then use the data group to complete the bit permutation function. COBRA[11] supports bit permutation with one instruction by static configuration. The efficiency drops sharply with the width increasing. At the same time, the configuration process needs to be calculated when the algorithm starts running, so the efficiency and flexibility are not high. The Haswell processor integrates parallel extraction and parallel insertion instructions, the number of instructions is increased by nearly three times when completing 64-bit permutation. DES has sixteen rounds. The number of instructions required for my design is only 68% of the COBRA processor and 14% of the Haswell processor.

For the IDEA algorithm, the modular multiplication and the modulo addition are dominant. This design can greatly reduce the consumption of the number of instructions at the beginning and end of the round operation. COBRA does not support mixed operations, so the clear and XOR are added in the data processing, and the performance of the upgrade is very limited. The Haswell processor does not support mixed and parallel operations. Each 16 bits must be operated separately, and special processing is required for $2^{16} + 1$. IDEA has eight rounds. The number of instructions required for my design is only 67% of the COBRA processor and 35% of the Haswell processor.

The BEST algorithm is mainly composed of 8-bit cyclic shift, XOR and modulo addition, and the number of cyclic shift steps is different. Each of the algorithms can be quickly completed by using register cyclic shift, mixed arithmetic operation and byte replacement. COBRA processor does not support the shift of the data after grouping, and which needs shift, clear and XOR instruction. Haswell processor can only be implemented with a combination of basic instructions with minimal efficiency. IDEA has 12 rounds. The number of instructions required for my design is only 40% of the COBRA processor and 20% of the Haswell processor.

6. Conclusion

Combining the operational characteristics and structural features of the block cipher algorithm, this paper designs data grouping, data mixing, extract-shift and shift-insert to support fast data sorting and grouping. Modular addition, modular multiplication, modular addition and XOR hybrid parallel operation instructions are design for arithmetic operation. Shift instructions are designed for different groups of individual shift. S-box instructions are designed to reduce the storage space while ensuring
the efficiency of the instruction. At the same time, this paper analyzes the situation that different instructions support the serial function, and minimizes the number of instructions required to implement the cipher algorithm. Finally, under the same conditions, the performance evaluation of the instructions designed in this paper is carried out. The results show that with the implementation of different cryptographic algorithms, the number of instructions designed in this paper is 14%~35% of the Haswell processor and 40%~68% of the COBRA processor.

The new instructions have a strong reference for improving the cryptographic processing capability of general-purpose processor and cipher processor. However, due to the limitation of the processor architecture, the utilization of correlation is low. Next, the serial function of the instruction and the architecture suitable for cryptographic processing are studied.

References
[1] R Lee, Z Shi, X Yang. IEEE MICRO 21, 56 (2001)
[2] Hilewitz Y, Lee R B. Princeton University Department of Electrical Engineering Technical Report CE-L2006-004 (2006)
[3] Jacob John. IOSR Journal of Computer Engineering, 16 91(2014)
[4] Zheng Gong, Svetla Nikova, Yee Wei Law. Lecture Notes in Computer Science. 7055, 1 (2011)
[5] Ray Beaulieu, Douglas Shors. Lightweight Cryptography for Security and Privacy, 3(2014)
[6] Hilewitz Y, Lee R B. IEEE Transactions on Computers 58, 1035(2009)
[7] Z X Chang, M F Chen. IOP Conference Series:Materials Science and Engineering. 439 042074 (2018)
[8] K. Bondalapati and V. K. Proceedings of the IEEE.90, 1201 (2002)
[9] Per Hammarlund, Alberto J. Martinez, Atiq A. Bajwa. IEEE Micro 34, 6 (2014)
[10] Jain, Tarush, and Tanmay Agrawal. International Journal of Computer Science and Information Technologies 4, 477 (2013)
[11] AJ Elbirt. IEEE Transactions on Parallel and Distributed Systems 16, 468 (2003)