IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 13, 2019, accepted July 25, 2019, date of publication July 30, 2019, date of current version August 16, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2932129

Fast Implementation of LSH With SIMD

DONGYEONG KIM[1], YOUNGHOON JUNG[2], YOUNGJIN JU[1], AND JUNGHWAN SONG[1]
[1]Department of Mathematics, Research Institute for Natural Sciences, Hanyang University, Seoul 04763, South Korea
[2]The Affiliated Institute of ETRI, Daejeon 341293, South Korea

Corresponding author: Junghwan Song (camp123@hanyang.ac.kr)

This work was supported by the Institute for Information and Communications Technology Planning and Evaluation (IITP) funded by the Korean Government (MSIT) under Grant 2017-0-00267.

ABSTRACT In this paper, we propose a method of efficient software implementation for the cryptographic hash function LSH with single instruction multiple data (SIMD). The method is based on word-wise permutations of LSH. Using the modified functions $Step'_j = P \circ Step_j \circ P^{-1}$ and $MsgExp'$ instead of the original step function $Step_j$ and message expansion function $MsgExp$, where P is a permutation and P^{-1} is the inverse permutation of P, we show that the number of the SIMD instructions for implementing LSH is reduced. For efficient implementation of LSH in other environments (e.g., MIMD), various types of word permutations are listed.

INDEX TERMS Software implementation, word-wise permutation, SIMD, hash function, LSH, ARX.

I. INTRODUCTION

Cryptographic hash functions are necessary for constructing a system of information security. Generally, they are used for authentication, providing both data integrity and entity integrity [1]–[4]. A cryptographic hash function is a function that maps an input to a fixed output satisfying the following cryptographic resistance properties [5].

1. Preimage resistance: for essentially all pre-specified outputs, it is computationally infeasible to find an input that hashes to that output.

2. 2nd preimage resistance: it is computationally infeasible to find another input that hashes the same output as a specified input.

3. Collision resistance: it is computationally infeasible to find any two distinct inputs that hash to the same output.

From the output of a cryptographic hash function, it should be computationally difficult to find the corresponding input. Additionally, for a given input, it should be difficult to find another input that hashes to the same output. Because of these aspects, a cryptographic hash function is used in various fields, such as a message authentication code (MAC), key derivation function (KDF), and a pseudo-random number generator. LSH [6] is a cryptographic hash function that was designed by NSRI [7] (National Security Research Institute). SIMD [8] is a class of parallel computing. SIMD is an instruction set that performs the same operations on multiple data simultaneously. A core element of SIMD is a register. SIMD has registers in with various lengths such as 128, 256, and 512 bits. For a 128-bit resister, each resister has four 32-bit or two 64-bit sections of data. Thus, one operation for an 128-bit resister is equivalent to four 32-bit operations or two 64-bit operations. When implementing cryptographic algorithms, SIMD is used for various purposes such as resistance to side-channel attacks [9], [10] and efficient implementations [11]–[16]. There has not been any research on overcoming weakness using SIMD with a cipher For these reasons, BLAKE2 [17] and LSH are cryptographic algorithms having advantage of implementation with SIMD. BLAKE2, SIMON/SPECK [18], and LEA [19] were implemented using SIMD in [11]–[13]. However, to the best out knowledge, our research is the first to have an efficient implementation via SIMD by changing the representation of a cryptographic algorithm. In this paper, we show how to implement a cryptographic hash function LSH efficiently with SIMD by representing the LSH using P and P^{-1}, where P is a permutation and P^{-1} is the inverse of P. Note that complexity is considered as the number of SIMD instructions and their latency, not the number of XOR and modular additions. This metric is necessary for finding conditions that reduce the number of SIMD instructions and use SIMD instructions with low latency when an algorithm is implemented.

For example, a case in which a permutation in a register composed of four 32-bit words is implemented. If a word-wise permutation is operated in a register, then only a single SIMD instruction is needed, "_mm_shuffle_epi32". However, if the word-wise permutation is the identity, then there is no need for an SIMD instruction. In another example, assuming that two 64-bit words compose a register, if a word-wise permutation is operated

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

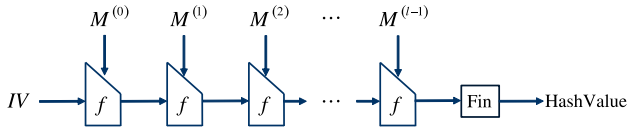

FIGURE 1. Wide-pipe merkle-damgard construction.

in two mixed registers, then three SIMD instructions are needed as follows: "_mm_unpacklo_epi64", "_mm_unpackhi_epi64", and "_mm_shuffle_epi32". The instruction "_mm_uppacklo_epi64" extracts the left 64-bit word in the two registers, and "_mm_unpackhi_epi64" extracts the right 64-bit word in the two registers. Additionally, the instruction "_mm_shuffle_epi32" is used to permute in 32-bit units.

If each of the two 64-bit words still remain in their same registers, then "_mm_unpacklo_epi64" and "_mm_unpackhi_epi64" are not needed for a word-wise permutation. In this paper, conditions to reduce the number of SIMD instructions and implement SIMD instructions with lower latency are found.

This paper is organized as follows. Section II shows the specifications of LSH. Section III provides an efficient implementation method via SIMD for LSH. We demonstrate the method and permutation conditions needed for efficient implementation. All permutations are categorized considering those conditions. In Section IV, we show an optimal permutation for the best performance with LSH. Concluding remarks on the implementation performance with an optimal permutation are given in Section V.

II. SPECIFICATION OF LSH

In 2014, a hash function LSH was published by D. Kim et al. at International Conference on Information Security and Cryptology, designed specifically to enhance software efficiency [6]. LSH was designed using wide-pipe Merkle Damgard construction (wide-pipe MD construction) [20]. The design of the compression function for LSH is based on ARX (Addition ($\boxplus$), Rotation ($\lll$), and XOR ($\oplus$)) [21]. The following describes the wide-pipe MD construction and compression function of LSH.

As shown in Fig. 1, the length of an internal state in a wide-pipe MD construction is $2n$ bits, which is twice the length of an output with n bits. Let w be the number of bits in a word. LSH-8w-n can represents any of the following LSHs: LSH-256-224, LSH-256-256, LSH-512-224, LSH-512-256, LSH-512-384, and LSH-512-512. Each has a different initializing value IV. The generating method of IV is given in [6].

The structure of the compression function f in an LSH is ARX-based. The number of bits for the input of f is $48w$, and that of the output is $16w$. The compression function f transforms 16-word and 32-word messages into 16-word messages. Each f contains the $MsgExp$ function, $Step_j$(for $j = 1, \cdots, N_s$), as well as the $MsgAdd$ function. The number of steps N_s is selected as follows:

$N_s = 26$ if the number of bits in w is 32,
$N_s = 28$ if the number of bits in w is 64.

A. NOTATIONS

W^t: Set of t-word arrays
$X \oplus Y$: Bit-wise exclusive-or of X and Y.
$X \boxplus Y$: $X + Y \bmod 2^w$.
$X^{\lll r}$: r bits left rotation of word X.
$M^{(i)} := (M^{(i)}[0], \cdots, M^{(i)}[31])$: The i-th 32-word array message block.
$M_j^{(i)} := (M_j^{(i)}[0], \cdots, M_j^{(i)}[15])$: The j-th 16-word array sub-message generated from the i-th message $M^{(i)}$.
$SC_j := (SC_j[0], \cdots, SC_j[7])$: The j-th 8-word array step constant.
$T := (T[0], \cdots, T[15])$: The 16-word array temporary variable used in a step function.
P: A word-wise permutation on 16 words

$$P(T) := P(T[0], \cdots, P[15]) := (T[P(0)], \cdots, T[P(15)])$$
$$P(T[i]) := T[P(i)]$$

P_i: A word-wise permutation on 4 words.

Notice that we define a permutation P that has the same format for the input and output. If the input is an index i, then $P(i)$ is also an index. Similarly, if the input is a word $T[i]$, then the output is a word $P(T[i]) = T[P(i)]$.

B. MsgExp FUNCTION

The first two sub-messages $M_0^{(i)}$ and $M_1^{(i)}$ are defined as the first 16 words and the next 16 words of $M^{(i)}$, respectively. Then the next sub-messages $\{M_j^{(i)}\}_{j=2}^{N_s}$ are calculated by the following:

$$\begin{aligned} &\text{For } j = 2, 3, \cdots, N_s, \\ &M_j^{(i)}[l] \leftarrow M_{j-1}^{(i)}[l] \boxplus M_{j-2}^{(i)}[\tau(l)], \quad 0 \le l < 16 \end{aligned} \tag{1}$$

Here, the permutation τ is defined by Table 1.

TABLE 1. The permutation τ in *MsgExp*.

l	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\tau(l)$	3	2	0	1	7	4	5	6	11	10	8	9	15	12	13	14

C. $Step_j$ FUNCTION

$Step_j$ is used N_s times repeatedly in the compression function f. $Step_j$ is composed of three functions $MsgAdd$, Mix_j, and σ as

$$Step_j = \sigma \circ Mix_j \circ MsgAdd. \tag{2}$$

MsgAdd:

$$\begin{aligned} MsgAdd : W^{16} \times W^{16} &\rightarrow W^{16} \\ MsgAdd(T, M_j^{(i)}) &= (T[0] \boxplus M_j^{(i)}[0], \\ &\quad \cdots, T[15] \boxplus M_j^{(i)}[15]) \end{aligned} \tag{3}$$

Mix$_j$:

$$\begin{aligned} Mix_j : W^{16} &\rightarrow W^{16} \\ Mix_j(T) &= (T'[0], \cdots, T'[15]), \\ where \quad (T'[l], T'[l+8]) &\leftarrow Mix_{j,l}(T[l], T[l+8]), \\ l &= 0, \cdots, 7 \end{aligned} \tag{4}$$

TABLE 2. The number of bits in rotation: α_j, β_j, γ_i.

w	j	α_j	β_j	γ_0	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6	γ_7
32	even	29	1	0	8	16	24	24	16	8	0
	odd	5	17								
64	even	23	59	0	16	32	48	8	24	40	56
	odd	7	3								

TABLE 3. Permutation σ.

l	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma(l)$	6	4	5	7	12	15	14	13	2	0	1	3	8	11	10	9

$Mix_{j,l} : W^2 \to W^2$ for inputs X and Y is defined by (5). Here, the bit rotational amounts α_j, β_j, γ_j used in $Mix_{j,l}$ are shown in Table 2.

$$\begin{aligned} X &\leftarrow X \boxplus Y, \\ X &\leftarrow X^{\lll \alpha_j}, \\ X &\leftarrow X \oplus SC_j[l], \\ Y &\leftarrow X \boxplus Y, \\ Y &\leftarrow Y^{\lll \beta_j}, \\ X &\leftarrow X \boxplus Y, \\ Y &\leftarrow Y^{\lll \gamma_l} \end{aligned} \tag{5}$$

σ: The word permutation function σ permutes 16 words. The permutation σ is defined in Table 3, and the permutation of internal states are as follows:

$$(T[0], \cdots, T[15]) \leftarrow (T[\sigma(0)], \cdots, T[\sigma(15)]). \tag{6}$$

III. A FAST IMPLEMENTATION METHOD

In this section, we demonstrate the method in [22] of how to implement LSH efficiently using a word permutation and its inverse with repeated step functions.

A. OVERVIEW OF WORD-WISE PERMUTATION

In this subsection, we investigate the relation between a permutation P and the compression function LSH of LSH. We also show how to construct LSH', which is a modified representation of LSH using the permutation P. To represent LSH', we investigate the relation between the permutation P and operations in the step function of LSH.

For a permutation P and its inverse P^{-1}, we consider the function LSH' as in Fig. 2. Note that LSH' takes a message permuted by P as an input and outputs the permuted hash value of LSH by P. Thus, the output of LSH is equal to the permuted output of LSH' by P^{-1}.

We define LSH' as the following:

$$LSH = P^{-1} \circ LSH' \circ P. \tag{7}$$

Our goal is to implement LSH efficiently using fewer SIMD instructions with low latency. The followings are basic criteria for P for LSH.

Criterion 1: P is a word-wise permutation.

If P is not a word-wise permutation, then it is necessary to consider implementing modular addition, which increases the number of SIMD instructions. Thus, we do not need to consider permutations other than word-wise permutations.

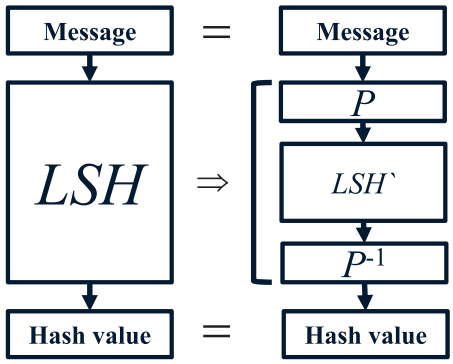

FIGURE 2. *LSH* and *LSH′*.

Criterion 2: $|P(i+8) - P(i)| = 8$ for $i = 0, \cdots, 7$.

Let the i-word array M be a message represented by $M = M[0]\|M[1]\|\cdots\|M[i-1]$. Then $P(M) = M[P(0)]\|M[P(1)]\|\cdots\|M[P(i-1)]$. In (5), two words $T[i]$ and $T[i+8]$ are input to the $Mix_{j,l}$ function as X and Y. For a register with SIMD, addition and XOR between two registers are operated in parallel between the corresponding words in order. Thus, if *Criterion 2* is not satisfied, then we cannot apply addition or XOR between the two registers before reordering the words in the registers. Thus, we limit the range of permutation P by *Criterion 2* since it needs many SIMD instructions for loading and storing. With the above two criteria, we have the following *Theorem 1*.

Theorem 1: If Criterion 1 and 2 hold, then there are relations between the internal state consisting of the 16 words $T[0], \cdots, T[15]$ in LSH and a word-wise permutation P as follows:

1) Modular addition($\boxplus$), XOR($\oplus$), and the permutation P are commutative with respect to each other.

$$\begin{aligned} T[P(i)] \boxplus T[P(i+8)] &= P(T[i] \boxplus T[i+8]), \\ T[P(i)] \oplus T[P(i+8)] &= P(T[i] \oplus T[i+8]) \end{aligned} \tag{8}$$

2) Left rotation by α_j, β_j and a permutation P are commutative with respect to each other.

$$\begin{aligned} T[P(i)]^{\lll \alpha_j} &= P(T[i]^{\lll \alpha_j}), \\ T[P(i)]^{\lll \beta_j} &= P(T[i]^{\lll \beta_j}) \end{aligned} \tag{9}$$

3) The following relation holds for the left rotation by γ_j and permutation P.

$$T[P(i)]^{\gamma_l} = P(T[i]^{\lll \gamma_{P^{-1}(l)}}) \tag{10}$$

Proof 1):

In LSH, there are steps: $T[i] \leftarrow T[i] \boxplus T[i+8]$ and $T[i+8] \leftarrow T[i] \boxplus T[i+8]$. By *Criterion 2*, since $|P(i+8) - P(i)| = 8$, we replace the above two steps with $T[P(i)] \leftarrow T[P(i)] \boxplus T[P(i+8)]$ and $T[P(i+8)] \leftarrow T[P(i)] \boxplus T[P(i+8)]$.

Then, for $T[P(i)] \boxplus T[P(i+8)]$, the following equality holds:

$$\begin{aligned} &T[P(i)] \boxplus T[P(i+8)] \\ &\quad = P(T[P(P^{-1}(i))]) \boxplus P(T[P(P^{-1}(i+8))]) \\ &\quad = P(T[i]) \boxplus P(T[i+8]) \\ &\quad = P(T[i] \boxplus T[i+8]) \end{aligned} \tag{11}$$

For $\oplus$, the equation can be proved similarly.

Proof 2):

$$\begin{aligned} T[P(i)]^{\lll \alpha_j} &= P(T[P(P^{-1}(i))]^{\lll \alpha_j}) \\ &= P(T[i]^{\lll \alpha_j}) \end{aligned} \tag{12}$$

Proof 3):
For γ_l, $l = i - 8$, thus when i is permuted to $P(i)$, l is also permuted to $P(l)$. Then the following holds:

$$\begin{aligned} T[P(i)]^{\lll \gamma_l} &= P(T[P(P^{-1}(i))]^{\lll \gamma_{P^{-1}(l)}}) \\ &= P(T[i]^{\lll \gamma_{P^{-1}(l)}}) \end{aligned} \tag{13}$$

□

By Theorem 1, a word-wise permutation P commutes with the operations of LSH. That is, considering the rotation by γ and letting $\gamma' := (\gamma_{P^{-1}(0)}, \ldots, \gamma_{P^{-1}(7)})$, permutation by P after a left rotation by γ' is equal to left rotation by γ after a permutation by P. Let the compression function f' used in LSH' be defined as follows:

$$f' = P \circ f \circ P^{-1} \tag{14}$$

Then by substituting (14) into (7), we have

$$LSH' = f' \circ \cdots \circ f' \tag{15}$$

The compression function f consists of the message expansion $MsgExp$ and step function $Step_j$. Similarly to (14), $MsgExp'$ and $Step'$ are represented as follows:

$$MsgExp' = P \circ MsgExp \circ P^{-1} \tag{16}$$

$$\begin{aligned} Step'_j &= P \circ Step_j \circ P^{-1} \\ &= P \circ \sigma \circ Mix_j \circ MsgAdd \circ P^{-1} \end{aligned} \tag{17}$$

Similarly to (15),

$$\begin{aligned} &MsgExp \circ \cdots \circ MsgExp \\ &\quad = (P^{-1} \circ MsgExp' \circ P) \circ \cdots \circ (P^{-1} \circ MsgExp' \circ P) \\ &\quad = P^{-1} \circ MsgExp' \circ \cdots \circ MsgExp' \circ P \end{aligned} \tag{18}$$

$$\begin{aligned} &Step_0 \circ \cdots \circ Step_{N_s-1} \\ &\quad = (P^{-1} \circ Step'_0 \circ P) \circ \cdots \circ (P^{-1} \circ Step'_{N_s-1} \circ P) \\ &\quad = P^{-1} \circ Step'_0 \circ \cdots \circ Step'_{N_s-1} \circ P. \end{aligned} \tag{19}$$

As in (15), (18), and (19), all permutations P and P^{-1} are canceled out to the identity permutation except for the first P^{-1} and the last P. The following provides the details of constructing $MsgExp'$ and $Step'_j$. Using (1), the function $MsgExp'$ in the i-th compression function with inputs $M_0^{(i)}$ and $M_1^{(i)}$ in (16) is represented as follows:

$$\begin{aligned} &MsgExp'(M_{j-1}^{(i)}, M_{j-2}^{(i)}) \\ &\quad = P \circ MsgExp \circ P^{-1}(M_{j-1}^{(i)}, M_{j-2}^{(i)}) \\ &\quad = P(M_{j-1}^{(i)}[P^{-1}(0)] \boxplus M_{j-2}^{(i)}[\tau(P^{-1}(0))], \\ &\qquad \cdots, M_{j-1}^{(i)}[P^{-1}(15)] \boxplus M_{j-2}^{(i)}[\tau(P^{-1}(15))]]) \\ &\quad = (M_{j-1}^{(i)}[P \circ P^{-1}(0)] \boxplus M_{j-2}^{(i)}[P \circ \tau(P^{-1}(0))], \\ &\qquad \cdots, M_{j-1}^{(i)}[P \circ P^{-1}(15)] \boxplus M_{j-2}^{(i)}[P \circ \tau(P^{-1}(15))]) \\ &\quad = (M_{j-1}^{(i)}[0] \boxplus M_{j-2}^{(i)}[P \circ \tau(P^{-1}(0))], \\ &\qquad \cdots, M_{j-1}^{(i)}[15] \boxplus M_{j-2}^{(i)}[P \circ \tau(P^{-1}(15))]) \\ &\qquad \text{for} \quad j = 2, \ldots, N_s. \end{aligned} \tag{20}$$

Let τ' be

$$\tau' = P \circ \tau \circ P^{-1} \tag{21}$$

The function $MsgExp'$ is represented with τ' as follows:

$$\begin{aligned} &MsgExp'(M_{j-1}^{(i)}, M_{j-2}^{(i)}) \\ &= (M_{j-1}^{(i)}[0] \boxplus M_{j-2}^{(i)}[\tau'(0)], \ldots, M_{j-1}^{(i)}[15] \boxplus M_{j-2}^{(i)}[\tau'(15)]) \\ &\qquad \text{for} \quad j = 2, \ldots, N_s. \end{aligned} \tag{22}$$

By using (2), the function $Step'_j$ in (17) is represented as follows:

$$\begin{aligned} Step'_j &= P \circ \sigma \circ Mix_j \circ MsgAdd \circ P^{-1} \\ &= P \circ \sigma \circ (P^{-1} \circ P) \circ Mix_j \circ (P^{-1} \circ P) \\ &\quad \circ MsgAdd \circ P^{-1} \\ &= \sigma' \circ Mix'_j \circ MsgAdd' \end{aligned} \tag{23}$$

where $\sigma' = P \circ \sigma \circ P^{-1}$, $Mix'_j = P \circ Mix_j \circ P^{-1}$, and $MsgAdd' = P \circ MsgAdd \circ P^{-1}$.

In (23), $MsgAdd = MsgAdd'$. In $MsgAdd'$, the words are permuted by P^{-1}, and the added message is also permuted by P^{-1} in $MsgExp'$. Thus, $MsgAdd'$ is a function of modular addition between words in the same ordering as in the function $MsgAdd$. In Mix'_j, the modular additions of step constants and left rotation by γ_l are affected by the permutation P. Therefore step constants word-wise permuted by P^{-1} and left rotation by $\gamma_{P^{-1}(l)}$ should be in Mix'_j. Further, σ' is a word-wise permutation since P, P^{-1} and σ are all word-wise permutations. Consequently, $MsgAdd'$ is the same as $MsgAdd$, and Mix'_j is the same as Mix_j except for the values for left rotation and step constants. This implies that they do not affect to the performance when they are implemented with SIMD. However, the word-wise permutation τ' in $MsgExp'$ and σ' in $Step'_j$ affect the performance when implemented by SIMD instructions.

B. TYPES OF PERMUTATIONS FOR LSH

In this subsection, we investigate the conditions of P that improve performance via SIMD related to τ' and σ'.

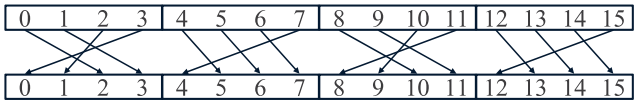

FIGURE 3. Permutation τ of *MsgExp*.

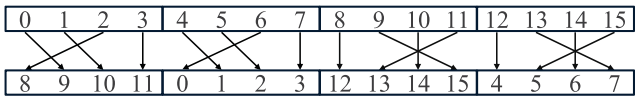

FIGURE 4. Permutation σ of $Step_j$.

Both the permutations τ and σ simultaneously permute four words with the some order as follows. The permutations τ and σ are represented by the compositions of two permutations, including internal permutations of four words and external permutations of a four-word arrays. Note that, in Fig. 3 and 4, four-word arrays permute to four-word arrays, regardless of the ordering of the four words in the four-word arrays.

To simply represent a permutation $P = (p_0 p_1 \cdots p_{15})$ that takes a word array of $(w_0 w_1 \cdots w_{15})$ to $(w_{p_0} w_{p_1} \cdots w_{p_{15}})$, we define external/internal permutations as follows. For a four-word array $W_i := (w_{4i} w_{4i+1} w_{4i+2} w_{4i+3})$, an external permutation denoted as $(a_0, a_1, a_2, a_3) \in \{0, 1, 2, 3\}^4$ is a permutation that permutes (W_0, W_1, W_2, W_3) to $(W_{a_0}, W_{a_1}, W_{a_2}, W_{a_3})$. An internal permutation P_j denoted as $(b_0 b_1 b_2 b_3)$, $b_i \in \{0, 1, 2, 3\}$ is a permutation that permutes 4 words $(w_{4j}, w_{4j+1}, w_{4j+2}, w_{4j+3})$ into 4 words $(w_{4j+b_0}, w_{4j+b_1}, w_{4j+b_2}, w_{4j+b_3})$. Using an external permutation and four internal permutations, a permutation P is represented as $(P_{a_0}, P_{a_1}, P_{a_2}, P_{a_3})$. For an example of the notation, the permutation σ of $Step_j$ is shown below.

In Fig. 5, the above four internal permutations are represented by $\sigma_0 = \sigma_1 = (2013)$ and $\sigma_2 = \sigma_3 = (0321)$. The external permutation is represented by (1, 3, 0, 2). Therefore σ is represented by $\sigma = (\sigma_1, \sigma_3, \sigma_0, \sigma_2)$. Similarly, $\tau = (\tau_0, \tau_1, \tau_2, \tau_3)$ where $\tau_0 = \tau_2 = (3201)$ and $\tau_1 = \tau_3 = (3012)$.

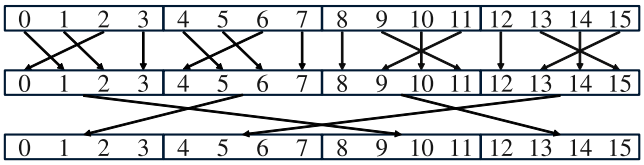

FIGURE 5. Representation of the permutation σ of $Step_j$ with internal permutations (top) and an external permutation (bottom).

Recall that SIMD instructions are operated in a register or among registers instead of via words. Thus, words in a register should be operated according to the same instructions. If some words in a register need to be partially operated, then the SIMD instructions for the register cannot be used, and the register should be divided. Composing and relieving of registers are done through SIMD instructions such as "load" and "store". Therefore, to reduce the number of SIMD instructions, words in a register need to be permuted simultaneously. Note that consecutive groups of four words are permuted by τ and σ simultaneously, and the same operations are applied to those consecutive groups of four words. It is enough to consider permutations of the form consisting of external and internal permutations. An external permutation in P provides no advantages for reducing the number of SIMD instructions, thus we fix the external permutation of P as the identity permutation. By *Criterion 2*, the permutation of the last eight words is determined by the permutation of the first eight words. Therefore, the total number of permutations we should consider is $(4!)^2 = 576$ for two internal permutations.

To find the optimal permutation P among the 576 internal permutation candidates, we define five types of permutations with respect to the following forms. Four are defined for internal permutations, and the other is defined for two internal permutations.

Definition 1: We define TYPE_i for $i = 1, \ldots, 5$ as a form of permutation in S_i as follows. Note that $*$ is an integer in $\{0, 1, 2, 3\}$.

$$
\begin{aligned}
S_1 &= \{(01**), (**01), (23**), (**23)\},\\
S_2 &= \{(10**), (**10), (32**), (**32)\},\\
S_3 &= \{(02**), (**02), (20**), (**20),\\
&\quad (13**), (**13), (31**), (**31)\},\\
S_4 &= \{(03**), (**03), (30**), (**30),\\
&\quad (12**), (**12), (21**), (**21)\},\\
S_5 &= \{(P_i, P_i)\}
\end{aligned}
$$

The following examples will be helpful for understanding the above TYPEs. For example, $P_i = (0132)$ has one TYPE1 as $(01**)$ in S_1 and one TYPE2 as $(**32)$ in S_2. Additionally, $P_i = (3102)$ has two TYPE3 as $(**02)$ and $(31**)$ in S_3 and none as TYPE1,2,4. For $(P_1, P_2) = ((1032), (1032))$, P_1 and P_2 are the same as an internal permutation (1032), thus it is TYPE5.

We find a good permutation by counting the number of TYPEs in τ' and σ'. Note that the total numbers for TYPE1 to TYPE4 for an internal permutation are always two. TYPE1 is represented as an internal permutation of two consecutive words permuted with the same ordering. In LSH-512, since the word size is 64 and the register size is 128, the register has two 64 bits words. If τ' or σ' has the form of TYPE1, then an SIMD instruction for permutation of the register positions '01' or '23' is not needed. For the other case of TYPE2, an SIMD instruction "_mm_shuffle_epi32" for changing word positions in a register is needed, then an SIMD instruction for this permutation is needed. For TYPE3, '02', '20', '13', and '31' are composed of words in different registers. Thus, those groupings require the SIMD instruction "_mm_unpacklo_epi64" or "_mm_unpackhi_epi64". For TYPE4, '03', '30', '12', and '21' are composed of words in different registers and different word positions(of left or right). Thus, two SIMD instructions are needed: "_mm_shuffle_epi32" to change word positions, one of "_mm_unpacklo_epi64" and

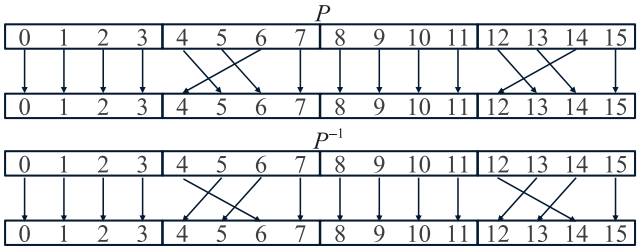

FIGURE 6. P and P^{-1} for LSH with SSE2, SSSE3, XOP, and LSH-512 with AVX2.

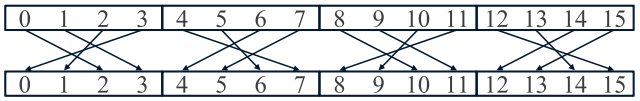

FIGURE 7. τ' for *LSH* with SSE2, SSSE3, XOP, and LSH-512 with AVX2.

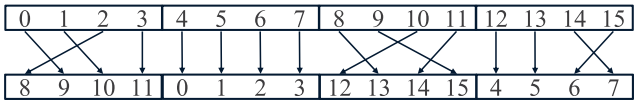

FIGURE 8. σ' for *LSH* with SSE2, SSSE3, XOP, and LSH-512 with AVX2.

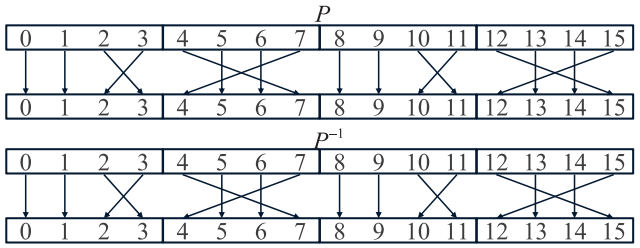

FIGURE 9. The permutation P and P^{-1} used for LSH-256 with AVX2.

''_mm_unpackhi_epi64''. Note that TYPE1 and TYPE2 do not share an internal permutation. By comparing the numbers of SIMD instructions for internal permutations, TYPE1 and TYPE2 are better than TYPE3 and TYPE4, and TYPE1 is the best.

TYPE5 is used for a 256-bit registers such as AVX2. For LSH-256, a register has eight words. If two internal permutations in a register are equal, the permutation can be operated using ''_mm256_shuffle_epi32'' instead of ''_mm256_permutevar8x32_ps''. The latter SIMD instruction has triple latency compared to that of the former. Thus, using the former is better than the latter for better latency performance.

TYPE1-5 are defined by considering an SIMD instruction set. Because there are many SIMD circumstances, if someone wants to use LSH with some specific SIMD, then the above TYPEs can be used to choose a permutation. All types of permutation considering TYPE1-5 are in Appendix A.

IV. PERMUTATION WITH THE BEST PERFORMANCE FOR LSH USING SIMD

In previous sections, we have shown how to find optimal permutations τ' and σ' with TYPEs. The optimal permutation is selected as TYPE1-4 except for LSH-256 with AVX2. For the case of LSH-256 with AVX2, since there are two internal permutations in the register, the optimal permutation is selected as TYPE5.

TABLE 4. Performance results with an intel CPU (Cycle/Byte).

Hash length (bits)	SIMD	implementation	The size of message		
			long message	4096 bytes	64 bytes
256	SSE2	[7]	6.67	6.88	15.25
		this paper	**4.59**	**4.74**	**11.06**
	SSSE3	[7]	3.75	3.88	9.19
		this paper	**3.34**	**3.46**	**8.69**
	AVX2	[7]	3.58	3.71	8.75
		this paper	**3.36**	**3.48**	**8.38**
512	SSE2	[7]	6.76	7.22	29.38
		this paper	**4.87**	**5.16**	**22.44**
	SSSE3	[7]	5.57	5.89	24.44
		this paper	**4.84**	**5.17**	**22.19**
	AVX2	[7]	2.44	2.61	11.38
		this paper	**2.18**	**2.33**	**10.31**

* With CPU(Intel Core i7-4790K CPU @ 4.00GHz), RAM(5GB), OS(Windows 7 Enterprise K SP1 64bit), Compiler(Visual Studio 2013, 64bit compiler)

TABLE 5. Performance results with an AMD CPU (Cycle/Byte).

Hash length (bits)	SIMD	implementation	The size of message		
			long message	4096 bytes	64 bytes
256	SSE2	[7]	9.41	9.73	21.23
		this paper	**6.11**	**6.33**	**14.31**
	SSSE3	[7]	5.10	5.26	12.39
		this paper	**4.73**	**4.90**	**11.36**
	XOP	[7]	3.88	4.00	9.97
		this paper	**3.23**	**3.35**	**8.39**
512	SSE2	[7]	6.62	7.07	29.72
		this paper	**4.63**	**4.95**	**21.11**
	SSSE3	[7]	5.28	5.63	24.67
		this paper	**4.39**	**4.70**	**20.28**
	XOP	[7]	4.42	4.74	20.81
		this paper	**3.26**	**3.49**	**15.61**

* With CPU(AMD FX-8350 CPU @ 4.00GHz), RAM(8GB), OS(Windows 7 Professional KN SP1 64bits), Compiler(Visual Studio 2013, 64bit compiler)

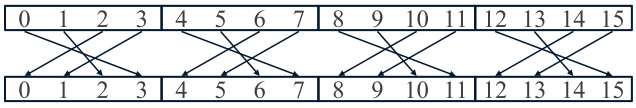

FIGURE 10. τ' for LSH-256 with AVX2.

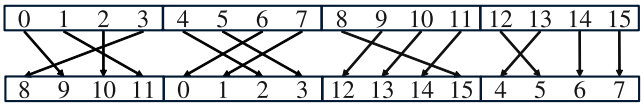

FIGURE 11. σ' for LSH-256 with AVX2.

A. THE PERMUTATION P FOR LSH WITH SSE2, SSSE3, XOP, AND LSH-512 WITH AVX2

The permutation P and its inverse P^{-1} for SSE2, SSSE3, XOP, and AVX2 in SIMD are described in Fig. 6. For AVX2,

TABLE 6. Permutation representation with english alphabet.

representation	A	B	C	D	E	F
permutation	(0123)	(0132)	(0213)	(0231)	(0312)	(0321)
representation	G	H	I	J	K	L
permutation	(1023)	(1032)	(1203)	(1230)	(1302)	(1320)
representation	M	N	O	P	Q	R
permutation	(2013)	(2031)	(2103)	(2130)	(2301)	(2310)
representation	S	T	U	V	W	X
permutation	(3012)	(3021)	(3102)	(3120)	(3201)	(3210)

TABLE 7. Permutations in TYPE1 and TYPE2.

τ'	σ'	Permutations
(4,4)	(3,1)	(A,L), (A,M), (B,N), (B,V), (G,C), (G,K), (H,D), (H,U), (Q,L), (Q,M), (R,C), (R,K), (W,N), (W,V), (X,D), (X,U)
	(2,2)	(A,C), (A,K), (A,N), (A,V), (B,D), (B,L), (B,M), (B,U), (G,D), (G,L), (G,M), (G,U), (H,C), (H,K), (H,N), (H,V), (Q,C), (Q,K), (Q,N), (Q,V), (R,D), (R,L), (R,M), (R,U), (W,D), (W,L), (W,M), (W,U), (X,C), (X,K), (X,N), (X,V)
	(1,3)	(A,D), (A,U), (B,C), (B,K), (G,N), (G,V), (H,L), (H,M), (Q,D), (Q,U), (R,N), (R,V), (W,C), (W,K), (X,L), (X,M)
(2,2)	(3,1)	(C,D), (C,U), (D,C), (D,K), (K,D), (K,U), (L,N), (L,V), (M,N), (M,V), (N,L), (N,M), (U,C), (U,K), (V,L), (V,M)
	(2,2)	(C,C), (C,K), (C,N), (C,V), (D,D), (D,L), (D,M), (D,U), (K,C), (K,K), (K,N), (K,V), (L,D), (L,L), (L,M), (L,U), (M,D), (M,L), (M,M), (M,U), (N,C), (N,K), (N,N), (N,V), (U,D), (U,L), (U,M), (U,U), (V,C), (V,K), (V,N), (V,V)
	(2,0)	(A,F), (A,O), (B,E), (B,I), (G,P), (G,T), (H,J), (H,S), (Q,F), (Q,O), (R,P), (R,T), (W,E), (W,I), (X,J), (X,S)
	(1,3)	(C,L), (C,M), (D,N), (D,V), (K,L), (K,M), (L,C), (L,K), (M,C), (M,K), (N,D), (N,U), (U,N), (U,V), (V,D), (V,U)
	(1,1)	(A,E), (A,I), (A,P), (A,T), (B,F), (B,J), (B,O), (B,S), (E,C), (E,D), (E,K), (E,L), (E,M), (E,N), (E,U), (E,V), (F,C), (F,D), (F,K), (F,L), (F,M), (F,N), (F,U), (F,V),(G,F), (G,J), (G,O), (G,S), (H,E), (H,I), (H,P), (H,T), (I,C), (I,D), (I,K), (I,L), (I,M), (I,N), (I,U), (I,V), (J,C), (J,D), (J,K), (J,L), (J,M), (J,N), (J,U), (J,V), (O,C), (O,D), (O,K), (O,L), (O,M), (O,N), (O,U), (O,V), (P,C), (P,D), (P,K), (P,L), (P,M), (P,N), (P,U), (P,V), (Q,E), (Q,J), (Q,P), (Q,T), (R,F), (R,J), (R,O), (R,S), (S,C), (S,D), (S,K), (S,L), (S,M), (S,N), (S,U), (S,V), (T,C), (T,D), (T,K), (T,L), (T,M), (T,N), (T,U), (T,V), (W,F), (W,J), (W,O), (W,S), (X,E), (X,I), (X,P), (X,T)
	(0,2)	(A,J), (A,S), (B,P), (B,T), (G,E), (G,I), (H,F), (H,O), (Q,J), (Q,S), (R,E), (R,I), (W,P), (W,T), (X,F), (X,O)
	(0,0)	(A,A), (A,B), (A,G), (A,H), (A,Q), (A,R), (A,W), (A,X), (B,A), (B,B), (B,G), (B,H), (B,Q), (B,R), (B,W), (B,X), (G,A), (G,B), (G,G), (G,H), (G,Q), (G,R), (G,W), (G,X), (H,A), (H,B), (H,G), (H,H), (H,Q), (H,R), (H,W), (H,X), (Q,A), (Q,B), (Q,G), (Q,H), (Q,Q), (Q,R), (Q,W), (Q,X), (R,A), (R,B), (R,G), (R,H), (R,Q), (R,R), (R,W), (R,X), (W,A), (W,B), (W,G), (W,H), (W,Q), (W,R), (W,W), (W,X), (X,A), (X,B), (X,G), (X,H), (X,Q), (X,R), (X,W), (X,X)
(0,0)	(3,1)	(E,A), (E,B), (E,Q), (E,W), (F,A), (F,G), (F,Q), (F,R), (I,A), (I,B), (I,Q), (I,W), (J,B), (J,H), (J,W), (J,X), (O,A), (O,G), (O,Q), (O,R), (P,G), (P,H), (P,R), (P,X), (S,B), (S,H), (S,W), (S,X), (T,G), (T,H), (T,R), (T,X)
	(2,0)	(C,F), (C,O), (D,P), (D,T), (K,F), (K,O), (L,E), (L,I), (M,E), (M,I), (N,J), (N,S), (U,P), (U,T), (V,J), (V,S)
	(1,3)	(E,G), (E,H), (E,R), (E,X), (F,B), (F,H), (F,W), (F,X), (I,G), (I,H), (I,R), (I,X), (J,A), (J,G), (J,Q), (J,R), (O,B), (O,H), (O,W), (O,X), (P,A), (P,B), (P,Q), (P,W), (S,A), (S,G), (S,Q), (S,R), (T,A), (T,B), (T,Q), (T,W)
	(1,1)	(C,E), (C,I), (C,P), (C,T), (D,F), (D,J), (D,O), (D,S), (K,E), (K,I), (K,P), (K,T), (L,F), (L,J), (L,O), (L,S), (M,F), (M,J), (M,O), (M,S), (N,E), (N,I), (N,P), (N,T), (U,F), (U,J), (U,O), (U,S), (V,E), (V,I), (V,P), (V,T)
	(0,2)	(C,J), (C,S), (D,E), (D,I), (K,J), (K,S), (L,P), (L,T), (M,P), (M,T), (N,F), (N,O), (U,E), (U,I), (V,F), (V,O)
	(0,0)	the rest

TABLE 8. Permutations in TYPE3 and TYPE4.

τ'	σ'	Permutations
(8,0)	(8,0)	(D,B), (D,G), (D,R), (D,W), (L,B), (L,G), (L,R), (L,W), (M,B), (M,G), (M,R), (M,W), (U,B), (U,G), (U,R), (U,W)
	(4,4)	(F,E), (F,I), (F,P), (F,T), (J,E), (J,I), (J,P), (J,T), (O,E), (O,I), (O,P), (O,T), (8,0), (S,E), (S,I), (S,P), (S,T)
	(4,2)	(D,E), (D,I), (D,P), (D,T), (L,E), (L,I), (L,P), (L,T), (M,E), (M,I), (M,P), (M,T), (U,E), (U,I), (U,P), (U,T)
	(2,2)	(F,B), (F,G), (F,R), (F,W), (J,B), (J,G), (J,R), (J,W), (O,B), (O,G), (O,R), (O,W), (S,B), (S,G), (S,R), (S,W)
(4,4)	(8,0)	(E,E), (E,I), (E,P), (E,T), (I,E), (I,I), (I,P), (I,T), (P,E), (P,I), (P,P), (P,T), (T,E), (T,I), (T,P), (T,T)
	(4,4)	(C,B), (C,G), (C,R), (C,W), (D,A), (D,H), (D,Q), (D,X), (K,B), (K,G), (K,R), (K,W), (L,A), (L,H), (L,Q), (L,X), (M,A), (M,H), (M,Q), (M,X), (N,B), (N,G), (N,R), (N,W), (U,A), (U,H), (U,Q), (U,X), (V,B), (V,G), (V,R), (V,W)
	(4,2)	(C,E), (C,I), (C,P), (C,T), (K,E), (K,I), (K,P), (K,T), (N,E), (N,I), (N,P), (N,T), (V,E), (V,I), (V,P), (V,T)
	(4,0)	(E,B), (E,G), (E,R), (E,W), (I,B), (I,G), (I,R), (I,W), (P,B), (P,G), (P,R), (P,W), (T,B), (T,G), (T,R), (T,W)
	(2,4)	(D,F), (D,J), (D,O), (D,S), (L,F), (L,J), (L,O), (L,S), (M,F), (M,J), (M,O), (M,S), (U,F), (U,J), (U,O), (U,S)
	(0,8)	(F,F), (F,J), (F,O), (F,S), (J,F), (J,J), (J,O), (J,S), (O,F), (O,J), (O,O), (O,S), (S,F), (S,J), (S,O), (S,S)
	(0,4)	(F,A), (F,H), (F,Q), (F,X), (J,A), (J,H), (J,Q), (J,X), (O,A), (O,H), (O,Q), (O,X), (S,A), (S,H), (S,Q), (S,X)
(4,0)	(4,4)	(A,B), (A,G), (A,R), (A,W), (B,B), (B,G), (B,R), (B,W), (G,B), (G,G), (G,R), (G,W), (H,B), (H,G), (H,R), (H,W), (Q,B), (Q,G), (Q,R), (Q,W), (R,B), (R,G), (R,R), (R,W), (W,B), (W,G), (W,R), (W,W), (X,B), (X,G), (X,R), (X,W)
	(4,2)	(A,E), (A,I), (A,P), (A,T), (F,C), (F,K), (F,N), (F,V), (H,E), (H,I), (H,P), (H,T), (J,C), (J,K), (J,N), (J,V), (O,C), (O,K), (O,N), (O,V), (Q,E), (Q,I), (Q,P), (Q,T), (S,C), (S,K), (S,N), (S,V), (X,E), (X,I), (X,P), (X,T)
	(4,0)	(D,D), (D,L), (D,M), (D,U), (L,D), (L,L), (L,M), (L,U), (M,D), (M,L), (M,M), (M,U), (U,D), (U,L), (U,M), (U,U)
	(2,4)	(B,E), (B,I), (B,P), (B,T), (F,D), (F,L), (F,M), (F,U), (G,E), (G,I), (G,P), (G,T), (J,D), (J,L), (J,M), (J,U), (O,D), (O,L), (O,M), (O,U), (R,E), (R,I), (R,P), (R,T), (S,D), (S,L), (S,M), (S,U), (W,E), (W,I), (W,P), (W,T)
	(2,2)	(D,C), (D,K), (D,N), (D,V), (L,C), (L,K), (L,N), (L,V), (M,C), (M,K), (M,N), (M,V), (U,C), (U,K), (U,N), (U,V)
(0,8)	(4,4)	(E,F), (E,J), (E,O), (E,S), (I,F), (I,J), (I,O), (I,S), (P,F), (P,J), (P,O), (P,S), (T,F), (T,J), (T,O), (T,S)
	(2,4)	(C,F), (C,J), (C,O), (C,S), (K,F), (K,J), (K,O), (K,S), (N,F), (N,J), (N,O), (N,S), (V,F), (V,J), (V,O), (V,S)
	(2,2)	(E,A), (E,H), (E,Q), (E,X), (I,A), (I,H), (I,Q), (I,X), (P,A), (P,H), (P,Q), (P,X), (T,A), (T,H), (T,Q), (T,X)
	(0,8)	(C,A), (C,H), (C,Q), (C,X), (K,A), (K,H), (K,Q), (K,X), (N,A), (N,H), (N,Q), (N,X), (V,A), (V,H), (V,Q), (V,X)
(0,4)	(4,4)	(A,A), (A,H), (A,Q), (A,X), (B,A), (B,H), (B,Q), (B,X), (G,A), (G,H), (G,Q), (G,X), (H,A), (H,H), (H,Q), (H,X), (Q,A), (Q,H), (Q,Q), (Q,X), (R,A), (R,H), (R,Q), (R,X), (W,A), (W,H), (W,Q), (W,X), (X,A), (X,H), (X,Q), (X,X)
	(4,2)	(A,F), (A,J), (A,O), (A,S), (E,C), (E,K), (E,N), (E,V), (H,F), (H,J), (H,O), (H,S), (I,C), (I,K), (I,N), (I,V), (P,C), (P,K), (P,N), (P,V), (Q,F), (Q,J), (Q,O), (Q,S), (T,C), (T,K), (T,N), (T,V), (X,F), (X,J), (X,O), (X,S)
	(2,4)	(B,F), (B,J), (B,O), (B,S), (E,D), (E,L), (E,M), (E,U), (G,F), (G,J), (G,O), (G,S), (I,D), (I,L), (I,M), (I,U), (P,D), (P,L), (P,M), (P,U), (R,F), (R,J), (R,O), (R,S), (T,D), (T,L), (T,M), (T,U), (W,F), (W,J), (W,O), (W,S)
	(2,2)	(C,D), (C,L), (C,M), (C,U), (K,D), (K,L), (K,M), (K,U), (N,D), (N,L), (N,M), (N,U), (V,D), (V,L), (V,M), (V,U)
	(0,4)	(C,C), (C,K), (C,N), (C,V), (K,C), (K,K), (K,N), (K,V), (N,C), (N,K), (N,N), (N,V), (V,C), (V,K), (V,N), (V,V)
(0,0)	(4,0)	(A,C), (A,D), (A,K), (A,L), (A,M), (A,N), (A,U), (A,V), (H,C), (H,D), (H,K), (H,L), (H,M), (H,N), (H,U), (H,V), (Q,C), (Q,D), (Q,K), (Q,L), (Q,M), (Q,N), (Q,U), (Q,V), (X,C), (X,D), (X,K), (X,L), (X,M), (X,N), (X,U), (X,V)
	(0,4)	(B,C), (B,D), (B,K), (B,L), (B,M), (B,N), (B,U), (B,V), (G,C), (G,D), (G,K), (G,L), (G,M), (G,N), (G,U), (G,V), (R,C), (R,D), (R,K), (R,L), (R,M), (R,N), (R,U), (R,V), (W,C), (W,D), (W,K), (W,L), (W,M), (W,N), (W,U), (W,V)

the permutations in Fig. 6 are applied to only LSH-512 by the above argument.

The permutation P is (0123, 2013, 0123, 2013), which is represented by the symbol (A,M) in Appendix A. If P is used for τ', then there are four TYPE1 and four TYPE2. If P is used for σ', then there are three TYPE1, one TYPE2, and four TYPE3. Note that τ has two TPYE1, two TPYE2, and four TYPE4, and σ has four TYPE3 and four TYPE4.

$\tau' = P \circ \tau \circ P^{-1}$ is described in Fig. 7, and σ' is described in Fig. 8.

Four words are assigned in one register, as shown in Fig. 7 and Fig. 8. In this case, the number of SIMD operations of τ' is not reduced. However, there is a reduction in σ'. Because there is no need for SIMD instructions in the second internal permutation of σ', the identity permutation is considered.

B. THE PERMUTATION P FOR LSH-256 WITH AVX2

AVX2 uses a 256-bit register. Thus, the register contains eight words. Then by σ', the left-half and right-half of the register are divided into two different registers. Thus, there is no advantage to using σ' instead of σ. However, if τ' is of TYPE5, then we can implement τ' with "_mm256_shuffle_epi32" instead of "_mm256_permutevar8x32_ps". This reduces the latency by one third. The permutations P and P^{-1} used to implement LSH-256 using AVX2 are shown in Fig. 9. This permutation $P = (0132, 3120, 0132, 3120)$ is described as (B,V) in Appendix A.

τ' and σ' are described in Fig. 10 and Fig. 11. Fig. 10 shows that permutations of eight words in a register are of TYPE5.

TABLE 9. Permutations in TYPE5.

	Permutations
(2,0)	(A,C), (A,L), (A,N), (A,U), (B,D), (B,K), (B,M), (B,V), (C,A), (C,J), (C,Q), (C,S), (D,B), (D,I), (D,R), (D,T), (E,F), (E,H), (E,O), (E,X), (F,E), (F,G), (F,P), (F,W), (G,D), (G,K), (G,M), (G,V), (H,C), (H,L), (H,N), (H,U), (I,F), (I,H), (I,O), (I,X), (J,E), (J,G), (J,P), (J,W), (K,A), (K,J), (K,Q), (K,S), (L,B), (L,I), (L,R), (L,T), (M,E), (M,G), (M,P), (M,W), (N,F), (N,H), (N,O), (N,X), (O,B), (O,I), (O,R), (O,T), (P,A), (P,J), (P,Q), (P,S), (Q,D), (Q,K), (Q,M), (Q,V), (R,C), (R,L), (R,N), (R,U), (S,B), (S,I), (S,R), (S,T), (T,A), (T,J), (T,Q), (T,S), (U,E), (U,G), (U,P), (U,W), (V,F), (V,H), (V,O), (V,X), (W,C), (W,L), (W,N), (W,U), (X,D), (X,K), (X,M), (X,V)
(0,2)	(A,A), (B,B), (C,C), (D,D), (E,E), (F,F), (G,G), (H,H), (I,I), (J,J), (K,K), (L,L), (M,M), (N,N), (O,O), (P,P), (Q,Q), (R,R), (S,S), (T,T), (U,U), (V,V), (W,W), (X,X)
(0,0)	the rest

There is source code for a modified LSH implementation using (A,M) and (B,V) in GIT-HUB [23].

V. CONCLUSION

We have shown how to implement LSH efficiently with SIMD using permutations. The performance results are summarized in Tables 4 and 5. On average, there is a 5% improved performance when using our method of permutations. Note that there is an additional 5% performance improvement after applying other optimization methods, such as the deployment of SIMD instructions; this is because the code in [7] has already been optimized by the creator of LSH. There is no security vulnerability when applying the proposed method, since the only changes are in orderings of words, circular rotations, and step constants in registers. Furthermore, we have defined five permutation types for LSH and SIMD, which classify all permutations of LSH. This classification can be used to implement LSH with a new SIMD instruction set for various register sizes or platforms.

APPENDIX

A. PERMUTATIONS

An internal permutation is represented using the English alphabet for readability. This representation is as shown in Table 6.

The different types of permutations are defined in IV. In Table 7, each pair in the first and second column represents the number of TYPE1 and TYPE2 permutations. Similarly in Table 8, each pair in the first and second column represents the number of TYPE3 and TYPE4 permutations. In Table 9, the first column represents the number of TYPE5 permutations in τ' and σ' respectively.

REFERENCES

[1] B. V. Sundaram, M. Ramnath, M. Prasanth, and J. V. Sundaram, "Encryption and hash based security in Internet of Things," in *Proc. 3rd Int. Conf. Signal Process., Commun. Netw. (ICSCN)*, Mar. 2015, pp. 1–6.

[2] M. H. Afifi, L. Zhou, S. Chakrabartty, and J. Ren, "HPMAP: A hash-based privacy-preserving mutual authentication protocol for passive iot devices using self-powered timers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kansas City, MO, USA, May 2018, pp. 1–6.

[3] T. D. Dang, D. Hoang, and P. Nanda, "A novel hash-based file clustering scheme for efficient distributing, storing, and retrieving of large scale health records," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Tianjin, China, Aug. 2016, pp. 1485–1491.

[4] R. Hamza, K. Muhammad, A. N, and G. Ramírez-González, "Hash based encryption for keyframes of diagnostic hysteroscopy," *IEEE Access*, vol. 6, pp. 60160–60170, 2018.

[5] *Information Technology–Security Techniques–Hash-Functions—Part 1: General*, 3rd ed., Standard ISO/IEC 10118-1:2016, 2016.

[6] D.-C. Kim, D. Hong, J.-K. Lee, W.-H. Kim, and D. Kwon, "LSH: A new fast secure hash function family," in *Information Security and Cryptology—ICISC* (Lecture Notes in Computer Science), vol. 8949. Cham, Switzerland: Springer, 2014, pp. 286–313.

[7] The Affiliated Institute of ETRI in Korea. (2015). *Korea Cryptography Forum*. [Online] Available: http://www.kcryptoforum.or.kr

[8] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. San Mateo, CA, USA: Morgan Kaufmann, 2013.

[9] B. Lac, A. Canteaut, J. J. A. Fournier, and R. Sirdey, "Thwarting fault attacks against lightweight cryptography using SIMD instructions," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, May 2018, pp. 1–5.

[10] A. Miyajan, Z. Shi, C.-H. Huang, and T. F. Al-Somani, "An efficient high-order masking of AES using SIMD," in *Proc. 10th Int. Conf. Comput. Eng. Syst. (ICCES)*, Cairo, Egypt, Dec. 2015, pp. 363–368.

[11] S. Neves and J.-P. Aumasson, "Implementing BLAKE with AVX, AVX2, and XOP," IACR Cryptol. ePrint Arch., Tech. Rep. 2012/275, 2012, p. 275.

[12] T. Park, H. Seo, and H. Kim, "Parallel implementations of SIMON and SPECK," in *Proc. Int. Conf. Platform Technol. Service (PlatCon)*, Jeju, South Korea, Feb. 2016, pp. 1–6.

[13] H. Seo, T. Park, S. Heo, G. Seo, B. Bae, Z. Hu, L. Zhou, Y. Nogami, Y. Zhu, and H. Kim, "Parallel implementations of LEA, revisited," in *Information Security Applications* (Lecture Notes in Computer Science), vol. 10144. Cham, Switzerland: Springer, 2016, pp. 318–330.

[14] K. C. Pabbuleti, D. H. Mane, A. Desai, C. Albert, and P. Schaumont, "SIMD acceleration of modular arithmetic on contemporary embedded platforms," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Waltham, MA, USA, Sep. 2013, pp. 1–6.

[15] C. Du, G. Bai, and H. Chen, "Towards efficient implementation of lattice-based public-key encryption on modern CPUs," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, Aug. 2015, pp. 1230–1236.

[16] W. Wang, J. Han, J. Wang, and X. Zeng, "A SIMD multiplier-accumulator design for pairing cryptography," in *Proc. IEEE 11th Int. Conf. ASIC (ASICON)*, Chengdu, China, Nov. 2015, pp. 1–4.

[17] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, smaller, fast as MD5," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science), vol. 7954. Berlin, Germany: Springer, 2013, pp. 119–135.

[18] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The SIMON and SPECK lightweight block ciphers," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jun. 2015, pp. 1–6.

[19] 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 *Information Security Applications* (Lecture Notes in Computer Science), vol. 8267. Cham, Switzerland: Springer, 2013.

[20] S. Lucks, "A failure-friendly design principle for hash functions," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science) vol. 3788. Berlin, Germany: Springer, 2005.

[21] R. P. Weinmann, "AXR-crypto made from modular additions, XORs, and word rotations," Dagstuhl Seminar 09031, 2009. [Online]. Available: http://www.dagstuhl.de/Materials/Files/09/09031/09031.WeinmannRalf Philipp.Slides.pdf

[22] Y. Jung, "Research on word permutation of ARX based cryptographic algorithm," Ph.D. dissertation, Dept. Maths, Hanyang Univ., Seoul, South Korea, 2016.

[23] GitHub. (2019). *LSH-SIMD-Implementation*. [Online]. Available: http://github.com/JuYoungjin/LSH-SIMD-Implementation

DONGYEONG KIM received the B.S. degree in mathematics from Hanyang University, Seoul, South Korea, in 2013, where he is currently pursuing the Ph.D. degree in mathematics, under the supervision of Prof. J. Song. His research interests include cryptanalysis of symmetric-key cyrptography, channel coding theory, and post quantum cryptography.

YOUNGHOON JUNG received the B.S. and Ph.D. degrees in mathematics from Hanyang University, Seoul, South Korea, in 2011 and 2016, respectively. He is currently with the Affiliated Institute of ETRI, as a Senior Researcher. His research interests include design, cryptanalysis, and implementation of symmetric key cryptography.

YOUNGJIN JU received the B.S. degree in mathematics from Hanyang University, Seoul, South Korea, in 2018, where he is currently pursuing the Ph.D. degree in mathematics, under the supervision of Prof. J. Song. His research interests include cryptanalysis of symmetric-key cryptography and post quantum cryptography.

JUNGHWAN SONG received the B.S. degree from Hanyang University, Seoul, South Korea, in 1984, the M.S. degree from Syracuse University, NY, USA, in 1989, and the Ph.D. degree from Rensselaer Polytechnic Institute, NY, USA, in 1993, all in mathematics. He was the Chairman of Korea Cryptographic Forum. He is currently a Professor with the Department of Mathematics, Hanyang University, Seoul. His current research interests include cryptanalysis of symmetric-key cryptography, mathematical optimization, and post quantum cryptography.